AD-A107 914 RENSSELAER POLYTECHNIC INST TROY NY DEPT OF ELECTRIC--ETC F/G 9/2
PARALLEL COMPUTATION FOR DEVELOPING NONLINEAR CONTROL PROCEDURE--ETC(U)
JUL 81 H KAUFMAN, R TRAVASSOS AFOSR-77-3418

UNCLASSIFIED AFWAL-TR-81-3016 NL

1 OF 3
AD
A107914

AFWAL-TR-81-3016

AF WRIGHT AERONAUTICAL LABORATORIES

PARALLEL COMPUTATION FOR DEVELOPING NONLINEAR CONTROL PROCEDURES

Dr. Howard Kaufman
Dr. Richard Teavassos
Rensselaer Polytechnic Institute
Electrical and Systems Engineering Dept
Troy, New York 12181

July 1981

Final Report for Period 1 September 1977 - 30 September 1979

Approved for public release; distribution unlimited.

FLIGHT DYNAMICS LABORATORY
AIR FORCE WRIGHT AERONAUTICAL LABORATORIES
AIR FORCE SYSTEMS COMMARND
WRIGHT-PATTERSON AIR FORCE BASE, OHIO 45433

NOTICE

When Government drawings, specifications, or other data are used for any purpose other than in connection with a definitely related Government procurement operation, the United States Government thereby incurs no responsibility nor any obligation whatsoever; and the fact that the government may have formulated, furnished, or in any way supplied the said drawings, specifications, or other data, is not to be regarded by implication or otherwise as in any manner licensing the holder or any other person or corporation, or conveying any rights or permission to manufacture use, or sell any patented invention that may in any way be related thereto.

This report has been reviewed by the Office of Public Affairs (ASD/PA) and is releasable to the National Technical Information Service (NTIS). At NTIS, it will be available to the general public, including foreign nations.

This technical report has been reviewed and is approved for publication.

Keith B. Angus
KEITH B. ANGUS, Lt., USAF
Project Engineer
Control Systems Development Branch
Flight Control Division

Evard H Flinn
EVARD H. FLINN, Chief
Control Systems Development Branch
Flight Control Division

FOR THE COMMANDER

ROBERT C. ETTINGER, Colonel, USAF
Chief
Flight Control Division

"If your address has changed, if you wish to be removed from our mailing list, or if the addressee is no longer employed by your organization please notify AFWAL/FIGL, W-PAFB, OH 45433 to help us maintain a current mailing list".

Copies of this report should not be returned unless return is required by security considerations, contractual obligations, or notice on a specific document.

12

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER
AFWAL/TR-81-3016

2. GOVT ACCESSION NO.
AD-A109914

3. RECIPIENT'S CATALOG NUMBER

4. TITLE (and Subtitle)
PARALLEL COMPUTATION FOR DEVELOPING NONLINEAR CONTROL PROCEDURES

5. TYPE OF REPORT & PERIOD COVERED
Interim Technical Report
1 Sep 77 - 30 Sep 79

6. PERFORMING ORG. REPORT NUMBER

7. AUTHOR(s)
Howard Kaufman
Richard Travassos

8. CONTRACT OR GRANT NUMBER(s)
AFOSR 77-3418

9. PERFORMING ORGANIZATION NAME AND ADDRESS
Rensselaer Polytechnic Institute
Troy, New York 12181

10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
2307 03 1987 01

11. CONTROLLING OFFICE NAME AND ADDRESS
AFOSR (PMN)
Bolling AFB, DC 20332

12. REPORT DATE
July 1981

13. NUMBER OF PAGES
226

14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)
Flight Dynamics Laboratory (AFWAL/FIGLD)
Air Force Wright Aeronautical Laboratories

15. SECURITY CLASS. (of this report)
Unclassified

15a. DECLASSIFICATION DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for Public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Parallel Computation, Optimal Control, Flight Control, Optimization

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

In view of the potential benefits to be gained from the use of parallel computers for control system optimization; since September 1977 research has been conducted under AFOSR Grant No. AFOSR-77-3418 for the purpose of developing and evaluating control computation algorithms that employ a significant degree of parallelism.

In particular, consideration has been given to both gradient and non-gradient based parallel algorithms which search for the optimal initial costate vector.

Through a series of parallel objective and/or gradient function evaluations, a sequence of optimal search directions was determined. Results of the first year's efforts indicated that parallel computation offers the possibility for very rapid non-linear controller computation.

Subsequent efforts since 1 October 1978 have been concerned with the application of the developed parallel optimization procedures to the design of non-linear flight control systems. Consideration has been given to the advantages to be gained from the direct use of the non-linear equations of motion and to the potential for rapid on-line optimization in response to tracked parameter changes and/or changes in the mission objectives.

To date, based upon the non-linear longitudinal equations for the F-8, it has been shown that parallel optimization procedures alone can reduce the time required for optimization by at least 30% and that the direct use of the non-linear (rather than the linearized) equations of motion does indeed improve attitude control.

A study of further incorporation of parallelism into the basic utility routines such as integration and matrix manipulation has shown that the convergence time for a representative optimal control computation can be reduced by a factor of 20. Thus, it was concluded that incorporation of parallel computers might indeed permit online control computation and/or adaptation. To this effect possible architectural structures for such computers have also been studied.

PREFACE

The research reported herein was accomplished for the United States Air Force by Rensselaer Polvtechnic Institute (RPI), under Air Force Office of Scientific Research (AFOSR) grant number 77-3418. The grant was entitled "Parallel Computation For Developing Nonlinear Control Procedures." AFOSR at Bolling Air Force Base sponsored the program. The results of the first two years of this three year effort are reported.

Lt Keith B. Angus of AFWAL/FIGL was the Air Force Project Engineer. Dr Howard Kaufman of RPI was the principal investigator. Dr Richard Travassos did much of the work and wrote a great deal of this report in partial fulfillment of the requirements for the degree of Doctorate of Philosophy at RPI. The research was performed during the period from September 1977 to September 1979.

TABLE OF CONTENTS

Page

LIST OF TABLES

Table Page

LIST OF FIGURES

CHAPTER ONE

INTRODUCTION

During the last decade, considerable attention has been given to the design of parallel computers and the development of parallel numerical procedures for identification, estimation and control which are implementable on these machines. As an introduction to this topic, some background material is given in Section 1.1. A brief review of existing parallel algorithms for identification, estimation and control is presented in Section 1.2. The motivation and significance of the research reported in this report are discussed in Section 1.3. Finally, the objectives and contributions of this report are stated in Section 1.4.

1.1 Background

The design of an automatic control system generally involves the selection of additional components which usually have adjustable parameters such that the overall system meets a desired performance specification. For example, this performance specification may be formulated in terms of the minimization of an error criteria, settling time, energy constraint or it may simply require a stable response. The performance index provides a quantitative measure of system performance and is chosen to emphasize important system characteristics. This type of quantitative measure is very important for parameter idenfication, state estimation, and for the design of optimal and suboptimal control systems.

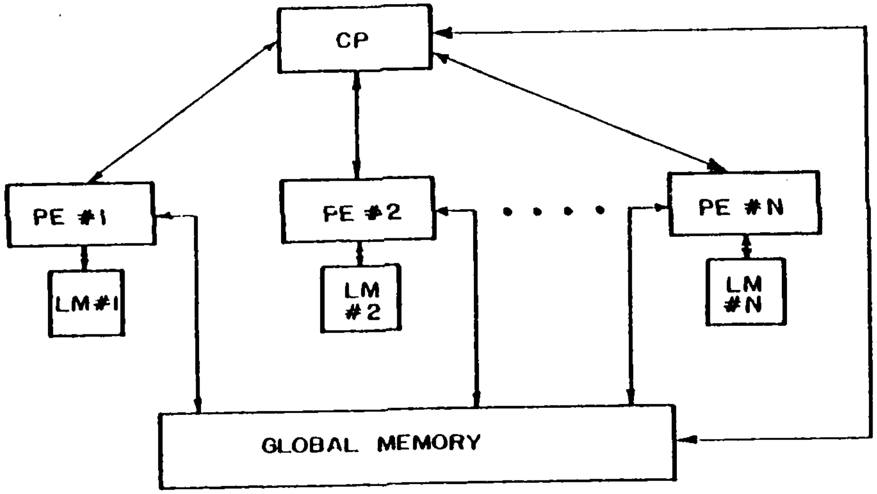
The early work in the area of parameter identification can be attributed to Nyquist [1] and Bode [2] in which frequency analysis methods were used in conjunction with logarithmic frequency diagrams to fit parametric models to data.

More recently, parametric models used in "modern" control theory have been formulated in terms of state equations. The need to determine such models from experimental data has led to a continual effort to improve parameter identification and state estimation procedures. Probably the oldest and most widely known methods for performing these tasks include: maximum likelihood techniques [3], Kalman filters [4], weighted least squares procedures [5], and stochastic approximation [6]. These sequential methods, however, may require a prohibitive amount of computer time to converge, if in fact, convergence occurs at all.

Since the introduction of time domain methods and the development of optimal control theory, a dramatic change in the design of automatic control systems has occurred. Because the use of optimal control theory generally results in the need to solve a highly nonlinear two-point boundary value problem (NTPBVP), much research has been conducted in the area of numerical methods for solving NTPBVP's. Currently, the most popular methods for solving NTPBVP's include iterative techniques such as shooting methods [7] and quasi-linearization [8] or non-iterative methods such as invariant imbedding [9]. These methods, however, suffer from the fact that convergence to the optimal solution is rather time consuming if these procedures are implemented on a conventional computer. One way to correct this problem might be to design faster computer systems, or simply develop more efficient algorithms. Apparently at this time, many felt that the numerical methods developed to date were rather efficient (although sequential in nature) and that the problem of excessive computation time could be most easily dealt with by the design of faster (parallel) computers.

In view of the need for faster computers, the computer industry has seen a significant change in the architecture of modern computers. This has led to the development of parallel computers which are capable of performing the same set of instructions on many data sets simultaneously (see Figure 1.1). Basically, a parallel computer can be viewed as a set of processing elements (PE's), each of which has its own local memory (LM) and a repertoire of arithmetic and logical instructions. The role of the central processor (CP) is to coordinate the efforts of each PE while the LM is used for temporarily storing intermediate results. Each PE is synchronized to perform the same instruction at the same time on data located in its own memory. When an instruction set has been completed by each processor, the results are transferred to global memory where the central processor interprets the results and decides whether to continue computations or halt. Note that if N processors are available and calculations are organized such that each PE is being fully utilized, then the speed of computation would be N times faster than the speed of a single PE. It is clear, however, that achieving this increase in speed requires great care in structuring computational algorithms.

Recognizing this fact, Larsen, et al. [10], [11] were the first to seriously consider using the parallel computing capabilities of modern parallel computers to allow the implementation of nonlinear



CP : Central Processor
PE : Processing Element
LM : Local Memory

FIGURE 1.1: A Parallel Computer Organization

estimation and control procedures. Much of the work in these papers was primarily concerned with restructuring dynamic programming so that many calculations could be performed by a computer with the facility for large scale parallel processing. Unfortunately, in many cases, the number of processors required by this method can be too extensive to be very practical.

For a number of years after the initial efforts of Larsen and Tse, it appeared that the development of parallel algorithms for nonlinear estimation and control had ceased. This occurrence might have been due to the many problems associated with the Illiac IV (the first truly parallel computer) [12].

Recently, however, there has been a renewed interest in parallel algorithms due to the successful development of many parallel computers (see Table 1.1). Due to the availability of these machines, many new nonlinear estimation and control algorithms have been proposed, the details of which, will be discussed in the following section.

1.2 A Survey of Parallelism in Identification, Estimation and Control

The idea of structuring estimation and control algorithms such that many operations may be performed simultaneously has only recently been considered. In fact, this area is so new that at the present time only a small number of parallel algorithms exist to perform such tasks. To survey these methods, we will consider the topics of identification, estimation and control separately in the remainder of this section.

Parallel Parameter Identification

In reference [13], Reid is concerned with identifying the parameters of a linear time invariant system from observing noise free

TABLE 1.1: A Comparison of Parallel Computers

Computer	Number of Processing Elements	Main Memory (words)	Word Length (bits)	Cycle Time (nsec)	Availability Date	Instruction Set	Cost
Illiac IV	64-256	2K	8-64	62.5	1972	Extensive	Very Fxpensive
Goodyear Aerospace STARAN	2K-8K	64K	32	200.0	1972	Primitive	Moderately Expensive
Texas Instruments ASC	2-5	16M	32	80.0	1972	Limited	Moderately Expensive
Carnegie-Mellon C_m^*	30-100	16M	16	380.0	1974	General	Inexpensive

output measurements. Basically, Reid's method employs an algebraic representation of the parameter sensitivity variables to efficiently and accurately obtain the system component matrices and component sensitivieties which are then used to form a linear system of equations and a small number of quadrature integrals. The parallel aspects of this algorithm lie in the fact that the quadrature integrals may be evaluated simultaneously. Also, parallelism can be exploited in solving the linear system of equations by utilizing the procedures discussed in references [14] and [15].

The structure of Reid's algorithm also leads to a natural set of conditions which are useful in determining if a system is identifiable. For example, for a system to be locally identifiable, the sensitivity matrix must have rank p where p is the number of unknown parameters. Other results on "structural identifiability," "excitation identifiability" and their "quality" are also reported by Reid in references [13] and [16]. Because Reid relies heavily on linearity and other properties of linear systems in the development of his method, it is restricted to linear dynamical systems. This, along with the fact that measurement noise and process noise are omitted from the problem formulation, seriously limits the application of this method.

Parallel State Estimation

The approach to the linear state estimation problem taken in reference [17] is to develop an explicit square-root algorithm which allows parallel processing with little communication between processors. The method is based upon a modification of the Kalman filter algorithm.

Basically, an interval of data is partitioned into N subintervals and state estimates are claculated based only on data within each subinterval. These calculations are performed simultaneously by N processors working independently. When each processor completes its task, some global calculations are performed and the results are combined to obtain an overall optimal estimate at the subinterval endpoints. At this point, the estimates at the subinterval interior points may be updated if desired.

The most expensive computations required by this procedure is estimating the states at the subinterval endpoints. Generally, this procedure requires about 14-40% more computations than a conventional Kalman filter but, because many of these computations can be performed in parallel, the actual execution time may indeed be much less. As pointed out by Kalaith [17], for this method to be faster than a single Kalman filter, the system should be high order, have a sparse system matrix and the state estimates must be desired infrequently compared to the number of data points. To help speed computations, a square-root doubling formula is introduced by Kalaith for calculating the steady-state covariance matrix of time invariant systems.

Although Kalaith argues that the parallel square-root algorithm can be more efficient than a single Kalman filter, this is not verified through simulation. Also, it should be noted that this method is only applicable to state estimation of linear systems.

Parallel Maximum Likelihood Estimation

In reference [11] Larsen and Tse restructure the dynamic programming method to estimate the maximum likelihood trajectory of a nonlinear discrete-time system. The approach taken by Larsen and Tse is to decompose the dynamic programming algorithm into three parts which consist of a parallel states algorithm, a parallel noises algorithm, and a parallel state and stages algorithm.

The major advantage of the parallel states algorithm is that the calculations performed by each processor are the same. Although this is highly desirable, there are several "overhead" calculations (such as binary search and compare) associated with the parallel noises algorithm which require moderate communication between processors and peripheral devices.

Also, the major shortcoming of this approach lies in the fact that the parallel states and stages algorithm can require a prohibitive number of processors. For example, for a problem with 10 states and 100 stages, then 10x100 = 1000 processing elements would be required. Although these processors need not be very sophisticated, such a large number of them may lead to reliability and synchronization problems.

Finally, Larsen and Tse do not discuss the application of their parallel dynamic programming algorithm to any problem of practical interest.

Parallel State and Parameter Estimation

The parallel state and parameter (SAP) estimation algorithm reported by Fargeon, *et al*. in reference [18], has been developed for discrete linear time invariant systems whose outputs are corrupted by a white Gaussian noise (WGN) process. This parallel Baysian algorithm employs many extended Kalman filters (EFK's) which simultaneously perform the SAP estimation functions. The integration of the a posteriori density function required by this method is approximated by a finite sum whose elements may be computed independently. Note that this method of integration is a variation of the rectangle rule, which is known to be extremely inaccurate unless many grid points are used. Since the number of parallel filters is equal to the number of grid points, it is entirely possible that a rather large number of filters may be required to implement this procedure; especially if accuracy is a major consideration.

As an indicator of performance, the parallel SAP estimation algorithm was tested by solving both a first and second order linear state estimation problem. For these simple examples, the simulations performed in reference [18] indicate that 8-20 parallel filters could be used without much loss in accuracy of the estimates. Although this is encouraging, it seems more appropriate to test this method on a low order, but highly nonlinear, process in which both the state and parameters of the process must be estimated. For this problem, it seems clear that a trade-off must be made between accuracy and the number of parallel filters required for the procedure to converge.

Parallel Control Algorithm

One of the first attempts to use parallelism to speed up optimal control computations was reported by Larsen and Tse in reference [10]. In this paper, a parallel dynamic programming algorithm for solving both deterministic and stochastic nonlinear control problems was developed.

Basically, their method is based upon a decomposition of the dynamic programming algorithm into a parallel states algorithm, a parallel state and stages algorithm, a parallel decisions algorithm, a parallel successive approximations algorithm, a parallel shift vector algorithm, and a parallel state increment dynamic programming algorithm.

Larsen and Tse's method suffers from the same problems as their maximum likelihood method previously discussed, as well as the fact that the parallel state increment dynamic programming algorithm employs Euler's method to integrate the right-hand side of a nonlinear system's equations of motion. Since it is well known that Euler's method is not very accurate unless extremely small steps are taken, it appears that this approach is of little practical value.

1.3 Motivation and Significance

Although nonlinear optimal control and estimation theory has been known for a number of years, the development of practical numerical methods based upon this theory has been relatively slow. As pointed out in the previous section, the major problems with existing methods have been a lack of accuracy and excessive computation time and that the use of parallelism has been proposed to alleviate such problems.

The survey of existing parallel estimation and control algorithms indicates that there exists a need to develop more efficient parallel procedures based upon modern nonlinear estimation and control theory. This fact has motivated an investigation of several parallel procedures with the hope that the computation time required for convergence of the new procedures could be significantly less than existing methods.

The development of more efficient parallel estimation and control algorithms is significant since:

- Parallelism should enable the design of nonlinear control systems without the need for approximating the behavior of highly nonlinear equations of motion by a linearized model.

- Parallel implementation could speed up computation time enough to permit modern nonlinear identification, estimation, and control algorithms to be executed in real time.

With this in mind, the objectives and contributions of this thesis will now be clearly stated.

1.4 Objectives and Contributions

In Section 1.2 a survey of some existing parallel identification, estimation, and control algorithms and an evaluation of their usefulness and drawbacks was made in terms of accuracy, speed, processor requirements and numerical efficiency. From this survey, it is clear that none of the existing parallel algorithms meet all of these requirements. In view of the above, the objectives of this report are to:

- Develop computationally efficient procedures for the identification, estimation and control of nonlinear dynamical systems which employ a high degree of parallelism but at the same time are not extravagant in the utilization of processing elements.

- Investigate both analytically and through computer simulation the convergence properties of the newly developed procedures.

- Examine the improvements obtainable through utilizing the nonlinear rather than linearized equations of motion when designing controllers for nonlinear dynamical systems.

- Study the robustness of the parallel algorithms and determine the values of certain algorithm parameters so that near optimum performance will result for a variety of problems.

- Propose a parallel computer architecture suitable for implementing the newly developed parallel methods.

The major contributions which resulted from conducting this research are:

- Development of a class of parallel rank-two quasi-Newton methods for unconstrained minimization.

- Establishment of a strategy for optimally selecting the number of subintervals and mesh points associated with the parallel shooting approach to solving nonlinear two-point boundary value problems.

- Development of a procedure which automatically adjusts the step size of a parallel predictor-corrector integration scheme to maintain a desired level of accuracy.

- Demonstration with representative examples that the newly developed parallel algorithms do indeed perform better than existing sequential methods in terms of speed, accuracy, and reliability.

- Application of the PQN, PVM, and CM methods to solving dynamic optimization problems (such as nonlinear estimation and control problems) rather than static optimization problems involving algebraic functions.

CHAPTER TWO

PARALLEL ALGORITHMS FOR THE IDENTIFICATION, ESTIMATION AND CONTROL OF NONLINEAR DYNAMICAL SYSTEMS

In this chapter, a collection of parallel algorithms are described which can be used to solve nonlinear estimation and control problems on a computer with the facility for large scale parallel processing. We shall begin our discussion of these techniques by formulating the nonlinear estimation and control problems in Sections 2.1.1 and 2.2.1 respectively. In Section 2.1, parallel methods for simultaneous state and parameter (SAP) estimation are presented while parallel control algorithms are discussed in Section 2.2. In Section 2.3, these methods are combined so that the estimation and control functions can be performed on-line in an adaptive fashion.

It should be emphasized that the goal of this chapter is to develop algorithms which possess a high degree of parallelism but at the same time do not require an excessive number of processors. This, along with the fact that the parallel algorithm presented in Sections 2.1, 2.2, and 2.3 should be capable of handling the nonlinear process equations directly, represent two of the contributions of this report.

Finally, one of the more subtle contributions developed in this chapter is an adaptive mesh selection algorithm which optimally places the mesh points needed by the method of parallel shooting. Note that the mesh point placement is optimal in the sense of minimizing the maximum local truncation error associated with integrating differential equations numerically.

2.1 Parameter Identification and State Estimation Algorithms

In this section, two methods are presented for simultaneously estimating the state and identifying the parameters of a nonlinear dynamical system. The first method is based upon solving a nonlinear two-point boundary value problem (NTPBVP) while the second method requires a direct minimization of a suitably defined cost function. Before the details of these methods are given, a formal statement of the state and parameter (SAP) estimation problem is in order.

2.1.1 Problem Statement

Consider the nonlinear dynamical system and measurement model represented by

$$\dot{x}(t) = f[x(t),t] + G[x(t),t]w(t) \qquad (2.1\text{-}1)$$

$$z(t) = h[x(t),t] + v(t) \qquad (2.1\text{-}2)$$

where

$x(t)\varepsilon R^{n}$ is an augmented state vector which includes the unknown parameters, $w(t)\varepsilon R^{p}$ is a process noise, and $z(t)\varepsilon R^{m}$ is the measurement vector which has been corrupted by the measurement noise $v(t)$.

It is assumed that:

- The initial state of this process is Gaussian with mean m_{xo} and covariance

$$E\{x(t_o)x^T(t_o)\}=\rho_{xo}$$

- The noise processes $w(t)$ and $v(t)$ are mutually independent zero-mean white Gaussian noise (WGN) processes with corresponding covariance matrices

$$E\{w(t)w^T(s)\} = Q(t)\delta(t-s) \qquad t_o \leq t,\ s \leq t_f$$

and

$$E\{v(t)v^T(s)\} = R(t)\delta(t-s) \qquad t_o \leq t,\ s \leq t_f$$

- $Q(t)$ and $R(t)$ are positive definite symmetric so that $Q^{-1}(t)$ and $R^{-1}(t)$ exist $\forall\ t\ \varepsilon\ [t_o,\ t_f]$.

Let us define $Z_t = \{z(\tau)\,|\,t_o \leq \tau \leq t\}$ as the accumulated noisy state measurements up to and including time t. The problem is to obtain an estimate of the augmented state vector x(t) at time t on the basis of the observations represented by Z_s. Our interest will be restricted to the case when s > t in which case $\hat{x}_{t|s}$ is referred to as a "smoothed" estimate of x(t).

By defining $p[x;t\ Z_t]$ as the a posteriori probability that the state vector assumes the value x at time t conditioned upon the measurement data represented by Z_t, the maximum-a-posteriori (MAP) estimate of $\hat{x}_{t|s}$ (denoted as $\hat{x}^{MAP}_{t|s}$)is defined by

$$p[\hat{x}^{MAP}_{t|s};t\,|Z_t] = \max_{x\varepsilon R^n} p[x;t\,|Z_t] \qquad t_o \leq t < s \leq t_f \qquad (2.1\text{-}3)$$

It has been shown (cf. [19], [20]) that the maximization indicated in eqn. (2.1-3) is equivalent to finding the deterministic signal, $\hat{w}(t)$ $t\varepsilon[t_o,t_f]$ which minimizes the functional

$$J = \tfrac{1}{2}||\hat{x}(t_o)-m_{x_o}||^2_{P^{-1}_{x_o}}$$

$$+ \tfrac{1}{2}\int_{t_o}^{t_f}\{||z(t)-h\{\hat{x}(t),t]||^2_{R^{-1}(t)} + ||\hat{w}(t)||^2_{Q^{-1}(t)}\}dt \qquad (2.1\text{-}4)$$

subject to the dynamic equality constraint given by

$$\dot{\hat{x}}(t)=f[\hat{x}(t),t]+G[\hat{x}(t),t]\;\hat{w}(t) \quad \forall\; t \in [t_o,\; t_f]\,. \qquad (2.1\text{-}5)$$

Note that the SAP estimation problem has been converted to a deterministic optimization problem and that once $\hat{w}(t)$ is found such that eqn. (2.1-4) is minimized, we may integrate eqn. (2.1-5) to obtain the MAP estimate of $\hat{x}(t)$ provided $\hat{x}(t_o)$ is known. Although this approach to the SAP estimation problem is not new, the use of parallelism to expedite the search for $\hat{w}(t)$ as well as in integrating eqn. (2.1-5) has not been considered before. These ideas are explored further in the next section.

2.1.2 Indirect State and Parameter Estimation Algorithm

In this section, a parallel numerical method based upon the calculus of variations is presented for simultaneously estimating the state and parameters of a nonlinear dynamical system. Although this method can be used to solve state and parameter (SAP) estimation problems which do not incorporate process noise into the state model, the method will be presented assuming that a noise process is used to account for modelling uncertainties. With this in mind, consider the optimization problem defined by eqns. (2.1-4) and (2.1-5).

To find $\hat{w}(t)$ using the calculus of variations, let the Hamiltonian be defined as

$$H = \tfrac{1}{2}\left[\,||z(t)-h[\hat{x}(t),t]||^2_{R^{-1}(t)} + ||\hat{w}(t)||^2_{Q^{-1}(t)}\right]$$
$$+\lambda^T(t)\{f[\hat{x}(t),t]+G[\hat{x}(t),t]\hat{w}(t)\} \quad \forall\; t\epsilon[t_o,\; t_f] \quad (2.1.2\text{-}1)$$

Then the necessary conditions for optimality become:

$$\frac{\partial H}{\partial \hat{w}} = 0 \Rightarrow \hat{w}(t) = -Q(t)G^T[\hat{x}(t),t]\lambda(t) \qquad (2.1.2\text{-}2)$$

$$\dot{\hat{x}}(t) = \frac{\partial H}{\partial \lambda} = f[\hat{x}(t), t] + G[\hat{x}(t), t]\, \hat{w}(t)$$

$$\triangleq \bar{f}\, [\hat{x}(t), \lambda(t), t] \qquad (2.1.2\text{-}3)$$

$$\dot{\lambda}(t) = \frac{\partial h^T[\hat{x}(t), t]}{\partial \hat{x}(t)}\, R^{-1}(t)\, \{z(t) - h[\hat{x}(t), t]\}$$

$$- \frac{\partial f^T[\hat{x}(t), t]}{\partial \hat{x}(t)}\, \lambda(t) - \frac{\partial\{\lambda^T(t)\, G[\hat{x}(t), t]\, \hat{w}(t)\}}{\partial \hat{x}(t)}$$

$$\triangleq g[\hat{x}(t), \lambda(t), t] \qquad (2.1.2\text{-}4)$$

The boundary conditions associated with eqns. (2.1.2-2)-(2.1.2-4) are given by

$$\lambda(t_o) = -P_{xo}^{-1}[\hat{x}(t_o) - m_{xo}] \qquad (2.1.2\text{-}5a)$$

$$\lambda(t_f) = 0 \qquad (2.1.2\text{-}5b)$$

In principle, the solution to the nonlinear two-point boundary value problem (NTPBVP) represented by eqns. (2.1.2-2), (2.1.2-3), (2.1.2-4) and (2.1.2-5) can be obtained using ordinary shooting [7] but, because computational problems can arise when integrating eqns. (2.1.2-3) and (2.1.2-4) forward in time, the following parallel shooting procedure, which is a modification of Keller's approach [21], is recommended.

Parallel Shooting Solution of Nonlinear SAP Estimation Problems

To illustrate the procedure eqns. (2.1.2-3) and (2.1.2-4) are concatenated as follows:

$$\begin{bmatrix} \dot{x}(t) \\ \hline \dot{\lambda}(t) \end{bmatrix} = \begin{bmatrix} f[x(t),\ \lambda(t),\ t] \\ \hline g[x(t),\ \lambda(t),\ t] \end{bmatrix} \triangleq s[x(t),\ \lambda(t),\ t] \qquad (2.1.2\text{-}6a)$$

$$A \begin{bmatrix} x(t_o) \\ \hline \lambda(t_o) \end{bmatrix} + B \begin{bmatrix} x(t_f) \\ \hline \lambda(t_f) \end{bmatrix} = \begin{bmatrix} \lambda(t_o) \\ \hline \lambda(t_f) \end{bmatrix} \triangleq \alpha \qquad (2.1.2\text{-}6b)$$

where α is a 2n vector of known boundary conditions and the elements of the 2nx2n matrices, A and B, are chosen such that eqn. (2.1.2-6b) is satisfied. By defining $y(t) = \begin{bmatrix} x(t) \\ \hline \lambda(t) \end{bmatrix}$ and partitioning the interval $[t_o, t_f]$ into N subintervals $t_o < t_1 < \ldots. < t_N = t_f$, the NTPBVP represented by eqn. (2.1.2-6) can be written as

$$\dot{y}_j = s_j(y_j,\ t) \qquad t \varepsilon [t_j,\ t_{j+1}] \qquad (2.1.2\text{-}7a)$$

$$Ay_o(t_o) + By_{N-1}(t_f) = \alpha \qquad j = 0,\ 1,\ \ldots,\ N-1 \quad (2.1.2\text{-}7b)$$

where

$$y_j(t) = \begin{cases} y(t) & t \varepsilon [t_j,\ t_{j+1}] \\ 0 & \text{otherwise} \end{cases}$$

Since y(t) is required to be continuous at the partition points, it is necessary that

$$y_{j-1}(t_j) = y_j(t_j) \qquad j = 1,2,\ \ldots,\ N-1 \qquad (2.1.2\text{-}8)$$

Combining eqns. (2.1.2-7) and (2.1.2-8) results in the following NTPBVP:

$$\dot{Y}(t) = F(Y(t),\ t) \qquad t \ \varepsilon\ [t_o,\ t_f] \qquad (2.1.2\text{-}9a)$$

$$PY_{\ell} + QY_r = \gamma \qquad (2.1.2\text{-}9b)$$

Here $Y(t)$, $F(Y(t),\ t)$, Y_{ℓ}, Y_r and γ are 2nN vectors defined as:

$$Y(t) \triangleq \begin{bmatrix} y_o(t) \\ \hdashline y_1(t) \\ \hdashline \vdots \\ \hdashline y_{N-1}(t) \end{bmatrix} \qquad F(Y(t),t) \triangleq \begin{bmatrix} s_o(t) \\ \hdashline s_1(t) \\ \hdashline \vdots \\ \hdashline s_{N-1}(t) \end{bmatrix}$$

$$Y_\ell \triangleq \begin{bmatrix} y_o(t_o) \\ \hdashline y_1(t_1) \\ \hdashline \vdots \\ \hdashline y_{N-1}(t_{N-1}) \end{bmatrix} \qquad Y_r \triangleq \begin{bmatrix} y_o(t_1) \\ \hdashline y_1(t_2) \\ \hdashline \vdots \\ \hdashline y_{N-1}(t_f) \end{bmatrix} \qquad \gamma \triangleq \begin{bmatrix} \lambda(t_o) \\ \lambda(t_f) \\ \hdashline 0 \end{bmatrix}$$

with P and Q being 2nNX2nN matrices of the form

$$P = \begin{bmatrix} A & 0 & \cdot & \cdot & \cdot & \cdot & 0 \\ 0 & I & 0 & \cdot & \cdot & \cdot & 0 \\ \cdot & & \cdot & & & & \cdot \\ \cdot & & & \cdot & & & \cdot \\ \cdot & & & & \cdot & & \cdot \\ \cdot & & & & & \cdot & 0 \\ 0 & \cdot & \cdot & \cdot & \cdot & 0 & I \end{bmatrix} \qquad Q = \begin{bmatrix} 0 & 0 & \cdot & \cdot & \cdot & \cdot & B \\ -I & 0 & \cdot & \cdot & \cdot & \cdot & 0 \\ 0 & -I & 0 & \cdot & \cdot & \cdot & 0 \\ \cdot & & \cdot & & & & 0 \\ \cdot & & & \cdot & & & \cdot \\ \cdot & & & & \cdot & & \cdot \\ 0 & \cdot & \cdot & \cdot & 0 & -I & 0 \end{bmatrix}$$

Note that if we could find the constant vector Y_ℓ which minimizes

$$E = \| PY_\ell + QY_r - \gamma \|^2 \qquad (2.1.2\text{-}10)$$

subject to the dynamic constraint (2.1.2-9a), we would have an estimate of the unknown states and parameters. Because the components of

(2.1.2-9a) are uncoupled, they may be integrated using a parallel integration algorithm by simultaneously shooting over each subinterval. Because each subinterval is much shorter than the interval $[t_o,t_f]$, it becomes less likely that the integration of the components of (2.1.2-9a) will diverge. As a result, convergence problems associated with excessively large intermediate values of $x(t)$ or $\lambda(t)$ may be alleviated by adopting parallel shooting.

In summary then, the parallel algorithm to be implemented would be:

Step 0: Record the noisy observations $z(t), t \varepsilon [t_o,t_f]$, arbitrarily choose the components of Y_ℓ, and set $\hat{x}(t_o)=m_{x_o}$ where m_{x_o} is the a priori mean value vector of the augmented initial state vector $x(t_o)$.

Step 1: Find $\lambda(t_o)$ by solving the linear system

$$\rho_{xo}\lambda(t_o) = (m_{x_o} - \hat{x}(t_o))$$

where ρ_{xo} is the a priori covariance matrix of $x(t_o)$.

Step 2: Using $\hat{x}(t_o)$, $\lambda(t_o)$ as the first 2n components of Y_ℓ and the remaining components of Y_ℓ, integrate the components of (2.1.2-9a) over each subinterval by employing a parallel integration method. Then record $z(t)$, and $x(t)$ $\forall$ $t \varepsilon [t_o, t_f]$.

Step 3: Evaluate the error function

$$E = ||PY_\ell + QY_r - Y||^2$$

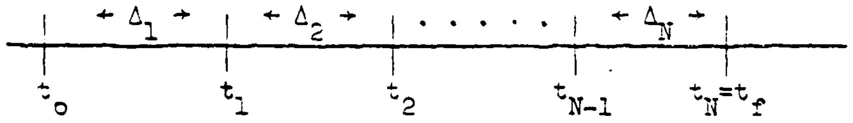
Step 4: If the error function is sufficiently small, the currently recorded values of $\hat{x}(t)$ are accepted as the "smoothed" estimates we desire. Hence, we may terminate the algorithm. Otherwise, using the recorded noisy measurements, compute a new value of Y_ℓ so as to minimize E by employing a parallel minimization method. Now return to Step 1.

Note that the parallel algorithm above reduces to a parallel version of ordinary shooting if only one subinterval is used.

Adaptive Mesh Selection

To implement the parallel algorithm discussed in the previous section, the mesh points $t_o<t_1<\ldots<t_N=t_f$ must be chosen. Given that N has been specified, the problem is to "optimally" select the mesh points. The technique we shall propose for optimizing the mesh is based upon using the local truncation error associated with any numerical method for integrating differential equations.* Upon convergence of the procedure, the mesh points will be optimal in the sense of minimizing the maximum local truncation error over each subinterval. Formally, the method is as follows:

Step 0: Let N be specified. Then partition the interval $[t_o,t_f]$ into N subintervals of length Δ_i $\forall i=1, 2, \ldots,N$.



Set $\ell=0$ and $E^{(0)}$ equal to some large positive real number.

Step 1: Integrate the components of (2.1.2-9a) over each subinterval and find

* The local truncation error is defined to be the norm of the difference between the computed solution and the exact solution of an initial-value problem. Techniques for estimating this quantity based on Taylor series approximations may be found in reference [22].

$$e_{\max}^{j+1} = \max_{t_j \leq t \leq t_{j+1}} e_L(t) \quad j=0,1,2,\ldots, N-1$$

where $e_L(t)$ is the local truncation error.

Step 2: Let $e=(e^1_{\max}, \ldots, e^N_{\max})^T$ and compute

$$E^{(\ell+1)} = ||e||^2$$

Step 3: If $|E^{(\ell+1)}-E^{(\ell)}|< \varepsilon$, the mesh points are accepted. Otherwise redistribute the mesh using a parallel minimization method so as to minimize $E^{(\ell+1)}$ as a function of

Δ_j, $j = 1, 2, \ldots, N$

subject to

$$\sum_{j=1}^{N} \Delta_j = t_f-t_o, \; \Delta_j > 0.$$

Set $\ell \leftarrow \ell + 1$ and go to Step 1.

Note that in Step 1 above the local truncation errors can be computed simultaneously by separate arithmetic processors. By combining the adaptive mesh selection algorithm and the parallel shooting algorithm described in the previous section, we will have a rapid method for accurately obtaining a MAP estimate of the states and parameters.

This is easily achieved by augmenting Y_ℓ with the unknown subinterval lengths $\Delta_1,\Delta_2, \ldots, \Delta_{N-1}$ and minimizing an error function of the form

$$E = ||PY_\ell + QY_r - Y||^2 + ||e||^2 \qquad (2.1.2\text{-}11)$$

subject to the constraints given by

$$\dot{Y}(t) = F[Y(t),t] \text{ and } \sum_{j=1}^{N}\Delta_j = t_f - t_o$$

2.1.3 Direct State and Parameter Estimation Algorithm

Suppose for the system defined by eqn. (2.1-1), $w(t) \equiv 0$ $\forall$ $t\varepsilon[t_o,t_f]$. That is, process noise is omitted from the state model. Then the fixed interval smoothed estimate of the unknown state and parameters may be found by searching for the vector $\hat{x}(t_o)$ which minimizes the functional

$$J = \tfrac{1}{2}||\hat{x}(t_o)-m_{x_o}||^2_{P^{-1}_{xo}}$$

$$+ \tfrac{1}{2}\int_{t_o}^{t_f}||z(t)-h[\hat{x}(t),t]||^2_{R^{-1}(t)}dt \qquad (2.1.3\text{-}1)$$

subject to

$$\dot{\hat{x}}(t)=f[\hat{x}(t),t] \quad \forall\ t\ \varepsilon\ [t_o,t_f]. \qquad (2.1.3\text{-}2)$$

The most direct method for solving this problem would be to initially set $\hat{x}(t_o)$ to m_{x_o}, integrate eqn. (2.1.3-2) forward in time over the interval $[t_o,t_f]$ and evaluate the performance index (2.1.3-1). By considering changes in the performance index due to changes in $\hat{x}(t_o)$, one may use this information to decide if this procedure should be repeated. Specifically, if the change in J is sufficiently small, then $\hat{x}(t_o)$ is accepted. Otherwise, the value of $\hat{x}(t_o)$ should be selected such that the performance index is minimized.

To speed computations, parallel integration methods may be used to integrate eqn. (2.1.3-2), while the selection of the next value of $\hat{x}(t_o)$ may be made using a parallel minimization method.

An example illustrating this procedure is given in Section 5.2.1 of this thesis. For now, however, let us devote our attention to developing parallel algorithms for optimal control computations.

2.2 Control Algorithms

This section is concerned with the development of parallel numerical methods for synthesizing controls for nonlinear dynamical systems. Specifically, an optimal control algorithm is presented in Section 2.2.2.1 which incorporates parallel shooting and adaptive mesh selection to solve the optimal control problem. This method is extended to accommodate problems in which the control is constrained in Section 2.2.2.2. In Section 2.2.2.3, a parallel algorithm is discussed which may be used to solve free terminal time problem. i.e., problems in which the terminal time is unspecified and as such must be optimized. Finally in Section 2.2.2.4, a suboptimal (or direct) control procedure is given to design feedback control laws for nonlinear dynamical systems.

Let us proceed by formally stating the optimal control problem.

2.2.1 Problem Statement

The physical process to be controlled is assumed to be a continuous nonlinear dynamical system which can be represented by

$$\dot{x}(t) = f(x(t),u(t),t) \qquad t\varepsilon[t_o,t_f] \qquad (2.2.1\text{-}1)$$

where

$x(t) \varepsilon R^n$ is the state of the system,

$u(t) \varepsilon R^r$ is the control and $r \leq n$.

The terminal time, t_f, may be either prescribed or be an unspecified problem parameter. It is assumed that the initial state vector $x(t_o)$ and M components, $0 \leq M \leq n$, of the final state vector, $x(t_f)$ are known, i.e., $x(t_o) = x_o$ and for $M > 0$, $x_i(t_f) = \sigma_i$, $1 \leq i \leq M$.

The optimal control problem is that of finding a control vector in some admissible set, $u(t) \in U$, which minimizes a performance index of the form:

$$J = \phi(x(t_f)) + \int_{t_o}^{t_f} L(x(t), u(t), t)\, dt \qquad (2.2.1\text{-}2)$$

subject to the differential constraints (2.2.1-1) and the above boundary conditions. Observe that, if M components of the final state vector are known, then these quantities need not be incorporated in the penalty function, $\phi(x(t_f))$, shown in eqn. (2.2.1-2). The solution, $u(t) = u^*(t)$, of the optimal control problem is called the optimal control and is assumed to exist and to be unique.

In practice, it is rather difficult to find the optimal control since to do so it is necessary to solve a highly nonlinear two-point boundary value problem (NTPBVP). Since the solution of NTPBVP's is often very time consuming, the role of parallelism might be to reduce the computational burden associated with solving NTPBVP's. This idea is pursued further in the next section.

2.2.2 Optimal Control Algorithms

For a control signal to be optimal, Pontryagin's Minimum Principle [23] indicates that the control must minimize the Hamiltonian defined as (2.2.2-1)

$$H(x(t), \lambda(t), u(t)) = L(x(t), u(t), t) + \lambda^T(t)\, f(x(t), u(t), t)$$

where $\lambda(t) \in R^n$ is the costate or adjoint vector. Let $u^*(t)$ be an element of U and let $x^*(t)$ be the solution of eqn. (2.2.1-1) which is dependent on $x^*(t_o) = x_o^*$ and $u(t) = u^*(t)$. In order for $u^*(t)$ to be the optimal control the following necessary conditions for optimality must be satisfied.

$$\dot{x}^*(t) = \frac{\partial H}{\partial \lambda}(x^*(t),\ \lambda^*(t),\ u^*(t)) \qquad (2.2.2\text{-}2)$$

$$\dot{\lambda}^*(t) = -\frac{\partial H}{\partial x}(x^*(t),\ \lambda^*(t),\ u^*(t)) \qquad (2.2.2\text{-}3)$$

$$x_i^*(t_f) = \sigma_i \qquad i = 1,\ 2,\ \ldots,\ M \qquad (2.2.2\text{-}4)$$

$$\lambda_i^*(t_f) = \phi_{x_i}(x^*(t_f)) \quad i = M+1,\ M+2,\ .\ .\ .N \qquad (2.2.2\text{-}5)$$

$$H(x^*(t),\ \lambda^*(t),\ u^*(t)) \leq H(x^*(t),\ \lambda^*(t),\ u(t)) \qquad (2.2.2\text{-}6)$$

$$\forall\ t\ \varepsilon\ [t_o, t_f] \text{ and } \forall\ u(t)\ \varepsilon\ U \qquad (2.2.2\text{-}7)$$

These necessary conditions may be used to solve many problems of interest in optimal control theory. In particular, our efforts will focus on developing parallel algorithms to solve specified terminal time problems, bounded control problems and free terminal time problems.

2.2.2.1 Specified Terminal Time Algorithm

Let us assume that the condition

$$\frac{\partial H}{\partial u}(x(t),\ \lambda(t),\ u(t)) = 0 \qquad (2.2.2.1\text{-}1)$$

can be explicitly solved for $u(t)$, the control is not subject to a magnitude constraint and the terminal time is specified.

Under these conditions eqn. (2.2.2-6) requires that

$$\frac{\partial H}{\partial u}(x^*(t),\ \lambda^*(t),\ u^*(t)) = 0 \Rightarrow u^*(t) = h(x^*(t),\ \lambda^*(t)) \qquad (2.2.2.1\text{-}2)$$

Now consider the system of equations

$$u(t) = h(x(t),\ \lambda(t)) \qquad (2.2.2.1\text{-}3)$$

$$\dot{x}(t) = \frac{\partial H}{\partial \lambda}(x(t), \lambda(t), u(t)) = f(x(t), u(t), t) \qquad (2.2.2.1\text{-}4)$$

$$\dot{\lambda}(t) = -\frac{\partial H}{\partial x}(x(t), \lambda(t), u(t)) = g(x(t), \lambda(t), u(t), t)$$

$$(2.2.2.1\text{-}5)$$

with boundary conditions

$$x(t_o) = x_o \qquad x_i(t_f) = \sigma_i \qquad i = 1, 2, \ldots, M$$

$$\lambda(t_o) = \lambda_o \qquad \lambda_i(t_f) = \phi_{x_i}(x(t_f)) \; i = M+1, M+2, \ldots, n$$

Let Ω represent that set of vectors for which the system (2.2.2-2) and (2.2.2-3), with $u(t)$ given by (2.2.2-6), has a unique solution $\forall\, t\, \varepsilon[t_o, t_f]$. Then for each $\lambda_o\, \varepsilon\, \Omega$, there corresponds a unique non-negative value for the scalar function:

$$E = \sum_{i=1}^{M} (x_i(t_f) - \sigma_i)^2 + \sum_{i=M+1}^{n} (\lambda_i(t_f) - \phi_{x_i}(x(t_f)))^2$$

$$(2.2.2.1\text{-}6)$$

The function E will be referred to as an error function.

Notice that if one could find $\lambda_o^*\, \varepsilon\, \Omega$ such that the forward integration of eqns. (2.2.2.1-4) and (2.2.2.1-5) leads to $E \equiv 0$, then the resultant solution of eqns. (2.2.2.1-4) and (2.2.2.1-5) subject to eqn. (2.2.2.1-3) would satisfy the necessary conditions (2.2.2-2), (2.2.2-3), (2.2.2-4), (2.2.2-5) and (2.2.2.1-2). The associated control vector, $u(t)$, as specified by eqn. (2.2.2.1-3) would, therefore, be taken as the optimal control for the original optimal control problem. Since $x(t_o)$ is assumed to be known, the problem of finding λ_o^* and hence of solving the optimal control problem is equivalent to the problem of minimizing the error function given by eqn. (2.2.2.1-6).

Since the initial state and terminal time are specified, this may be accomplished by iteratively updating the initial costate until the error function (2.2.2.1-6) is minimized. It may be, however, for any given initial costate the solution of eqns. (2.2.2.1-4) and (2.2.2.1-5) together with eqn. (2.2.2.1-3) may become excessive in magnitude for $t < t_f$. In order to cope with such situations, a technique used by Isaacs [24] or the method of parallel shooting can be adopted.

Isaacs Procedure:

If $\lambda_o \varepsilon \Omega$, then there will, in general, exist some $t'<t_f$ to the left of which the solution of eqns. (2.2.2.1-4) and (2.2.2.1-5) together with eqn. (2.2.2.1-3) remains computable. Consider then the optimal control problem which is identical to the original problem except that t_f is replaced by t'. Using λ_o, as the "priming guess" at the optimal initial costate for this modified optimal control problem, a solution can be obtained and the resulting initial costate, $\overline{\lambda}_c^*$, is taken as a candidate for membership in Ω. If $\overline{\lambda}_o^* \varepsilon \Omega$, t_f may be replaced by its original value and the solution to the original optimal control problem can be pursued.

If, however, $\overline{\lambda}_o^* \notin \Omega$, then there will, in general, exist t'', $t'<t''<t_f$, such that the solution of eqns. (2.2.2.1-4) and (2.2.2.1-5) together with eqn. (2.2.2.1-3) remains computable to the left of t''. A new optimal control problem in terms of t'' is then posed and the process is repeated.

Experience with Isaacs' method indicates that this method is particularly well suited only for problems with relatively short mission times. This is a result of the fact that, if the mission time is too long, the sensitivity of the solution to small changes in the initial costate becomes too excessive for the Isaacs method to overcome. For short, mission times, however, convergence to an element, $\lambda^*_o \in \Omega$, is quite rapid. Convergence may be accelerated still further if parallelism is introduced when integrating the state and costate equations forward in time. Also, the selection of the next value of the initial costate can be made using a parallel minimization procedure.

Parallel Shooting Solution of Optimal Control Problems

In some cases, the problem under consideration may be sensitive to small perturbations in the initial costate and, as a result, convergence to an optimal solution may be slow (if convergence occurs at all). In this situation, parallel shooting has proven to be very effective in alleviating such problems. By invoking the principle of duality, the parallel shooting procedure described in Section 2.1.2 to solve optimal estimation problems may be employed to solve optimal control problems also.

To illustrate the parallel shooting procedure for optimal control problems, eqns. (2.2.2.1-4) and (2.2.2.1-5) are concatenated:

$$\begin{pmatrix} \dot{x}(t) \\ \hline \dot{\lambda}(t) \end{pmatrix} = \begin{pmatrix} f(x(t), u(t), t) \\ \hline g(x(t), \lambda(t), u(t), t) \end{pmatrix} \underline{\Delta}\ s(x(t), \lambda(t), u(t), t) \tag{2.2.2.1-7a}$$

$$A \left[\begin{array}{c} x(t_o) \\ \hline \lambda(t_o) \end{array}\right] + B \left[\begin{array}{c} x(t_f) \\ \hline \lambda(t_f) \end{array}\right] = \left[\begin{array}{c} x(t_o) \\ \hline \lambda(t_f) \end{array}\right] \triangleq \alpha \qquad (2.2.2.1\text{-}7b)$$

where α is a 2n vector of known boundary conditions and the elements of the 2nx2n matrices, A and B, are chosen such that eqn. (2.2.2.1-7b) is satisfied. By defining $y(t) = \left[\begin{array}{c} x(t) \\ \hline \lambda(t) \end{array}\right]$ and partitioning the interval $[t_o, t_f]$ into N subintervals $t_o < t_1 < \ldots. < t_N = t_f$, the NTPBVP represented by eqn. (2.2.2.1-7) can be written as

$$\dot{y}_j = s_j(y_j, t) \qquad t \varepsilon [t_j, t_{j+1}] \qquad (2.2.2.1\text{-}8a)$$

$$Ay_o(t_o) + By_{N-1}(t_f) = \alpha \qquad j = 0, 1, \ldots, N\text{-}1 \quad (2.2.2.1\text{-}8b)$$

where

$$y_j(t) = \begin{cases} y(t) & t\varepsilon[t_j, t_{j+1}] \\ 0 & \text{otherwise} \end{cases}$$

Since y(t) is required to be continuous at the partition points, it is necessary that

$$y_{j-1}(t_j) = y_j(t_j) \qquad j = 1,2, \ldots, N\text{-}1 \qquad (2.2.2.1\text{-}9)$$

Combining eqns. (2.2.2.1-8) and (2.2.2.1-9) results in the following NTPBVP:

$$\dot{Y}(t) = F(Y(t), t) \qquad t \ \varepsilon \ [t_o, t_f] \qquad (2.2.2.1\text{-}10a)$$

$$PY_\ell + QY_r = \gamma \qquad (2.2.2.1\text{-}10b)$$

Here $Y(t)$, $F(Y(t), t)$, Y_ℓ, Y_r and γ are 2nN vectors defined as:

$$Y(t) \triangleq \left[\begin{array}{c} y_o(t) \\ \hline y_1(t) \\ \hline \vdots \\ \hline y_{N-1}(t) \end{array}\right] \qquad F(Y(t),t) \triangleq \left[\begin{array}{c} s_o(t) \\ \hline s_1(t) \\ \hline \vdots \\ \hline s_{N-1}(t) \end{array}\right]$$

$$Y_\ell \triangleq \left[\begin{array}{c} y_o(t_o) \\ \hline y_1(t_1) \\ \hline \vdots \\ \hline y_{N-1}(t_{N-1}) \end{array}\right] \qquad Y_r \triangleq \left[\begin{array}{c} y_o(t_1) \\ \hline y_1(t_2) \\ \hline \vdots \\ \hline y_{N-1}(t_f) \end{array}\right] \qquad \gamma \triangleq \left[\begin{array}{c} x(t_o) \\ \lambda(t_f) \\ \hline 0 \end{array}\right]$$

with P and Q being 2nNX2nN matrices of the form

$$P = \begin{bmatrix} A & 0 & \cdots & & 0 \\ 0 & I & 0 & \cdots & 0 \\ \vdots & & \ddots & & \vdots \\ \vdots & & & \ddots & \vdots \\ \vdots & & & & 0 \\ 0 & \cdots & & 0 & I \end{bmatrix} \qquad Q = \begin{bmatrix} 0 & 0 & \cdots & & B \\ -I & 0 & \cdots & & 0 \\ 0 & -I & 0 & \cdots & 0 \\ \vdots & & \ddots & & 0 \\ \vdots & & & \ddots & \vdots \\ \vdots & & & & \vdots \\ 0 & \cdots & 0 & -I & 0 \end{bmatrix}$$

In view of this formulation, the parallel variation of extremals algorithm considers the selection of Y_ℓ to minimize

$$E \triangleq || PY_\ell + QY_r - \gamma ||^2 \qquad (2.2.2.1\text{-}11)$$

subject to the dynamic constraint (2.2.2.1-10a). This defines a new optimization problem involving constant rather than time varying unknowns. Observe that finding the vector Y_ℓ such that $E \equiv 0$ is equivalent to satisfying the necessary conditions for optimality. Also, it is important to note that the components of eqn (2.2.2.1-10a) are uncoupled and hence may be integrated by simultaneously shooting over each subinterval.

In summary then, the procedure to be followed is:

Step 1: Arbitrarily choose the components of Y_ℓ.

Step 2: Integrate the components of eqn. (2.2.2.1-10a) simultaneously starting at Y_ℓ using a parallel integration scheme.

Step 3: Evaluate the error function $E \triangleq || PY_\ell + QY_r - \gamma ||^2$

Step 4: If the error function is sufficiently small, then Y_ℓ is accepted. Otherwise, update Y_ℓ such that E is minimized by using a parallel minimization procedure.

Note that the parallel algorithm above reduces to a parallel version of ordinary shooting if only one subinterval is used. In this case, however, the algorithm may still be considered a parallel method since the differential equations may be integrated using a parallel integration scheme. Also, the partition points required by this algorithm may be optimally selected via the adaptive mesh selection algorithm, discussed in Section 2.1.2.

2.2.2.2 <u>Bounded Control Algorithm</u>

The techniques described in the previous section can be extended to problems with control constraints of the form:

$$u_i(t) \leq B_i \quad \forall\, t \in [t_o, t_f] \qquad i = 1, 2, \ldots, r \qquad (2.2.2.2\text{-}1)$$

The method for handling constraints of this type is based on the fact that for each $t \varepsilon [t_o, t_f]$, $u^*(t)$ is either on its boundary or $\frac{\partial H}{\partial u}(x^*(t), \lambda^*(t), u^*(t)) = 0$. Consequently, in the evaluation of the error functions described in Section 2.2.2.1, u(t) is replaced by

$$u_i(t) = \begin{cases} h_i(x(t), \lambda(t)) & \text{for } h_i(x(t), \lambda(t)) \leq B_i \quad i = 1, 2, \ldots, r \\ B_i \operatorname{sgn} (h_i(x(t), \lambda(t)) & \text{otherwise} \end{cases} \qquad (2.2.2.2\text{-}2)$$

The approach can also be extended to some cases where $\frac{\partial H}{\partial u}(x(t), \lambda(t), u(t)) = 0$ can be explicitly solved for the control.* For example, consider

$$H(x(t), \lambda(t), u(t)) = F_1(x(t), \lambda(t)) + F_2(x(t), \lambda(t))u(t) \qquad (2.2.2.2\text{-}3)$$

where $\quad u_i(t) \leq B_i$, $i = 1, 2, \ldots, r \; \forall \; t\varepsilon[t_o, t_f]$

In this case, the extremization required by eqn. (2.2.2-6) is carried out directly. Thus, eqn. (2.2.2.1-3) is replaced by

$$u(t) = - B \operatorname{sgn} (F_2(x(t), \lambda(t))) \qquad (2.2.2.2\text{-}4)$$

Note that this technique requires that

$$F_2(x(t), \lambda(t)) \neq 0 \qquad \forall \; t\varepsilon[t_o, t_f]$$

since eqn. (2.2.2.2-4) would be undefined in this case. Note that if this occurs on the optimal trajectory, the problem is called a singular control problem.

2.2.2.3 Free Terminal Time Algorithm

To accommodate problems when t_f is free, we utilize the necessary condition

$$H(x^*(t), \lambda^*(t), u^*(t)) = 0 \; \forall \; t \; [t_o, t_f^*] \qquad (2.2.2.3\text{-}1)$$

* Note that the bang-bang control problems fall into this category.

In particular, since the Hamiltonian must be zero at $t = t_o$, the following constraint must be satisfied.

$$H(x(t_o), \lambda(t_o), u(t_o)) = 0 \qquad (2.2.2.3\text{-}2)$$

Incorporating the constraint given by eqn. (2.2.2.1-3) into eqn. (2.2.2.3-2), results in

$$H(x(t_o), \lambda(t_o), h(x(t_o), \lambda(t_o)) = 0 \qquad (2.2.2.3\text{-}3)$$

For computational purposes, we restrict our consideration to those cases where eqn. (2.2.2.3-3) can be explicitly solved for one of the components of the initial costate vector. For convenience, assume this to be the first component of $\lambda(t_o)$.† From eqn. (2.2.2.3-3), it can be seen that $\lambda_1(t_o)$ is a unique function of $x(t_o)$ and the remaining component of the initial costate vector.

Let this value of $\lambda_1(t_o)$ be defined by:

$$\lambda_1(t_o) = (x(t_o), \tilde{\lambda}(t_o)) \qquad (2.2.2.3\text{-}4)$$

where the (n-1)-vector $\tilde{\lambda}(t_o)$ is defined as

$$\tilde{\lambda}(t_o) \triangleq (\lambda_2(t_o), \lambda_3(t_o), \ldots, \lambda_n(t_o))^T$$

In view of eqn. (2.2.2.3-4) and the fact that t_f is unspecified, a suitable error function which must be minimized by selecting t_f and $\tilde{\lambda}(t_o)$ would be

$$E \triangleq \sum_{i=1}^{M} (x_i(t_f)-\sigma_i)^2 + \sum_{i=M+1}^{M} (\lambda_i(t_f) - \phi_{x_i}(x(t_f)))^2$$

(2.2.2.3-5)

Clearly, this error function can be viewed as a function of the

* If this is not the case, we simply reorder the components of the initial costate vector.

following n-vector:

$$(t_f, \tilde{\lambda}(t_o)) \triangleq (t_f, \lambda_2(t_o), \lambda_3(t_o), \ldots, \lambda_n(t_o))^T.$$

Observe that if one could find $(t_f, \tilde{\lambda}(t_o)) \in \Omega$ such that $E \equiv 0$ subject to the constraints of eqn. (2.2.2.1-3), (2.2.2.1-4), (2.2.2.1-5), (2.2.2.1-2) and (2.2.2.3-1), then the corresponding control vector as given by eqn. (2.2.2.1-3) would be the optimal control for the original optimal control problem. Notice that this is equivalent to minimizing the error function (2.2.2.3-5) subject to the constraints of eqns. (2.2.2.1-3), (2.2.2.1-4), (2.2.2.1-5) and (2.2.2.3-4).

In summary, free terminal time problems may be solved using the following parallel algorithm:

Step 1: Arbitrarily select the components of the n-vector $(t_f, \tilde{\lambda}(t_o))$.

Step 2: Using $x(t_o)$ and $\tilde{\lambda}(t_o)$, evaluate $\lambda_1(t_o)$ using eqn. (2.2.2.3-4).

Step 3: Compute $u(t_o)$ from eqn. (2.2.2.1-3) and use a parallel integration method to integrate the components of eqns. (2.2.2.1-4) and (2.2.2.1-5) starting with $x(t_o)$, $\tilde{\lambda}(t_o)$ and $\lambda_1(t_o)$ over the interval $[t_o, t_f]$.

Step 4. At time t_f, evaluate the error function given by eqn. (2.2.2.3-5).

Step 5: If the error function is sufficiently small, stop; otherwise use a parallel minimization algorithm to update $(t_f, \tilde{\lambda}(t_o))$ such that eqn. (2.2.2.3-5) is minimized.

2.2.3 Suboptimal Control Algorithm

In the previous section, various parallel methods were discussed which could be used to design optimal control systems. Although a controller designed using these methods is optimal in the sense of satisfying the necessary conditions for optimality, the resultant control system is open loop and, as such, may be very sensitive to environmental disturbances. Thus, it seems appropriate to consider methods for designing a closed loop control system to overcome such problems. With this in mind, the following parallel method is proposed.

Suppose the controller is constrained to be of the form

$$u(t) = h[x(t),t] \qquad \forall\ t\ \varepsilon[t_o,t_f] \tag{2.2.3-1}$$

where h[x(t),t] is assumed to be continuous and specified by the control system designer up to a set of constants. For example, to design a linear feedback controller one might select

$$u(t) = h[x(t),t] = kx(t) \tag{2.2.3-2}$$

where k is a gain matrix whose elements must be determined such that the closed loop system is stable. Once the structure of the controller has been specified, the problem is simply to find a finite number of constants which minimize:

$$J = \phi(x(t_f),t_f) + \int_{t_o}^{t_f} L(x(t),h[x(t),t],t)dt \tag{2.2.3-3}$$

subject to the dynamic constraint given by

$$\dot{x}(t) = f[x(t),h[x(t),t],t]. \tag{2.2.3-4}$$

If we let $K = (k_1, k_2, \ldots., k_m)$ be the vector of unknown constants to be optimized, then the optimal elements of K may be found as follows:

Step 0: Let u(t) = h[x(t),t] be specified and select $K^{(0)}$ such that the forward integration of eqn. (2.2.3-4) is stable over the interval $[t_o,t_f]$. Set $\ell = 0$.

Step 1: Given $K^{(0)}$ and the known initial state $x(t_o)$, integrate eqn. (2.2.3-4) forward in time over the interval $[t_o,t_f]$ using a parallel integration scheme.

Step 2: Evaluate the performance index:

$$J = \phi(x(t_f),t_f) + {}_{t_o}^{t_f}L(x(t),u(t),t)dt$$

Step 3: If $|J^{(\ell+1)} - J^{(\ell)}|<\varepsilon$, then the current value of K is accepted and the procedure is terminated. Otherwise, use a parallel minimization procedure to update K such that J is minimized. Then set $\ell \rightarrow \ell+1$ and return to Step 1.

Clearly, the simplicity of this method and the fact that the controller utilizes feedback makes this method very attractive. Also, by incorporating parallel algorithms in Steps 1 and 3 above, the computation time required for convergence can be significantly reduced. Finally, it should be noted that the direct gain optimization procedure above is the dual of the direct state and parameter estimation algorithm discussed in Section 2.1.3.

2.3 Adaptive Control and Estimation Algorithms

For many physical processes, variations in the environment necessitate major modifications in the control strategy to meet operating requirements. In such cases, an adaptive control system might be employed to provide near optimal control in spite of environmental disturbances. In this section, an explicit adaptive control scheme is described which employs parallel algorithms to generate a control signal in response to parameter changes tracked by an adaptive parameter identifier. Since, in many cases, the state variables required by the explicit adaptive control scheme are not accessible, the unknown states of the system must be estimated simultaneously with the parameters.

Due to the fact that the parameters may be rapidly varying, the role of parallelism would be to reduce the computation time needed to update the parameter values, state estimates and subsequently the control law. In particular, the state and parameters may be updated by using the parallel state and parameter estimation algorithms discussed in Section 2.1 on a "window" of measurement data and then the parallel control algorithms of Section 2.2 could be employed to update the control based upon the latest estimates of the states and parameters.

To this effect, this section will be concerned with developing parallel algorithms for rapidly performing nonlinear estimation and control in an adaptive fashion. Note that the major goal is to utilize these parallel algorithms in an explicit adaptive controller of the type shown in Figure 2.1. Hopefully, the use of parallelism will permit the on-line implementation of such a system. To this end, let us proceed by formally stating the adaptive control problem.

2.3.1 Problem Statement

Consider a stochastic nonlinear dynamical system and measurement model represented by

$$\dot{x}(t) = f[x(t),u(t),t] + G[x(t),t]w(t) \qquad (2.3.1\text{-}1)$$

$$z(t) = h[x(t),t] + v(t) \qquad (2.3.1\text{-}2)$$

where $x(t)$ is an augmented state vector which contains any unknown parameters, $u(t)$ is a control, and $z(t)$ is a measurement vector.

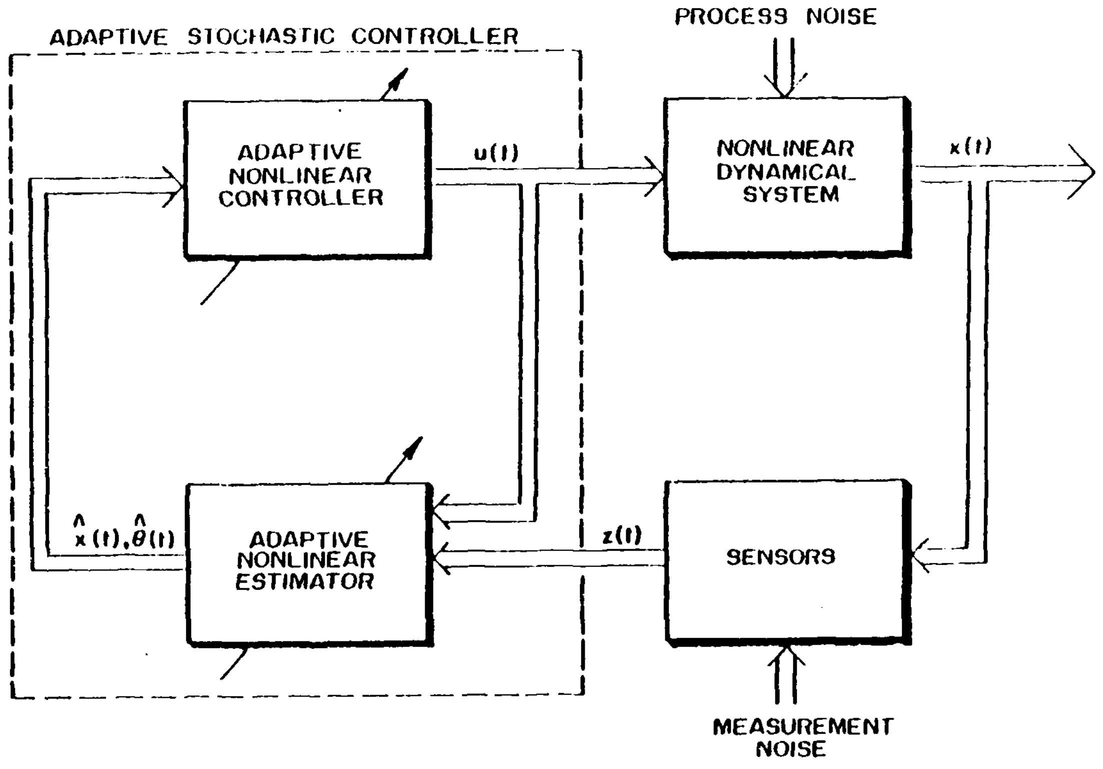

PROCESS NOISE
ADAPTIVE STOCHASTIC CONTROLLER
ADAPTIVE NONLINEAR CONTROLLER
u(t)
NONLINEAR DYNAMICAL SYSTEM
x(t)
$\hat{x}(t), \hat{\theta}(t)$
ADAPTIVE NONLINEAR ESTIMATOR
z(t)
SENSORS
MEASUREMENT NOISE

FIGURE 2.1: Explicit Adaptive Control of a Nonlinear Dynamical System

The noise processes w(t) and v(t) are mutually independent zero-mean white Gaussian noise processes with corresponding covariance matrices

$$E\{w(t)w^T(s)\} = Q(t)\delta(t-s) \qquad t_o \le t,\ s \le t_f$$

and

$$E\{v(t)v^T(t)\} = R(t)\ \delta(t-s) \qquad t_o \le t,\ s \le t_f$$

Also, it is assumed that the initial state, $x(t_o)$, is Gaussian and uncorrelated with w(t) and v(t). Furthermore, consider the performance criterion

$$J = E\{\phi[x(t_f),t_f] + \int_{t_o}^{t_f} L[x(t),u(t),t]dt\} \qquad (2.3.1\text{-}3)$$

where E {·} is an expectation operator. The objective is to determine the control, u(t), which minimizes eqn. (2.3.1-3) subject to the stochastic dynamic constraints given by (2.3.1-1) and (2.3.1-2).

The approach we shall take in solving this stochastic control problem is similar to that of Larsen and Tse [10] who proposed separating this problem into a deterministic control problem and a nonlinear estimation problem. Basically, the approach is as follows:

Suppose for a given system, the state and a nominal set of parameters which define the systems equations of motion are known at the initial time. Because the structure of the estimator and controller shown in Figure 2.1 is assumed to be known, we may set the appropriate parameters in the adaptive estimator and controller to their nominal values before the process to be controlled is started. When this initialization is complete, the process is started and the control is computed on-line and applied to the plant as the process evolves for all $t \ge t_o$.

To account for uncertainties in the plant parameters and disturbances, it may be necessary to adapt the control during the mission time interval $[t_o,t_f]$. This may be accomplished on-line by updating the control by employing the parallel control algorithms discussed in Section 2.2 at the adaptation times t_i where

$$t_i \varepsilon [t_o,t_f] \quad \forall\ i = 1, 2, \ldots$$

However, to use the algorithms of Section 2.2., an estimate of the process parameters and unknown state variables must be available at the time of adaptation. These estimates can be acquired by recording the noisy observations z(t) over the interval $[t_i,t_{i+1}]$ on-line and using the nonlinear SAP estimation algorithms discussed in Section 2.1.

The idea outlined above forms the basis for the explicit adaptive control scheme which is illustrated by the timing chart shown in Figure 2.2. With this background, we can proceed to the next section, in which the details of the adaptive control scheme are presented.

2.3.2 Direct Explicit Adaptive Control

In this section, an explicit adaptive control scheme is presented which utilizes the direct estimation and control algorithms discussed in Sections 2.1.3 and 2.2.3 respectively. In particular, consider the state and measurements models given by eqns. (2.3.1-1) and (2.3.1-2) with $w(t) \equiv 0 \ \forall\ t\ \varepsilon[t_o,\ t_f]$. That is, no process noise is present. It is assumed that the nominal initial state of this process is known and a nominal set of parameters which define the dynamics represented in eqn. (2.3.1-1) are given. Also, let us assume that the mission time $[t_o,t_f]$ is finite with the partition

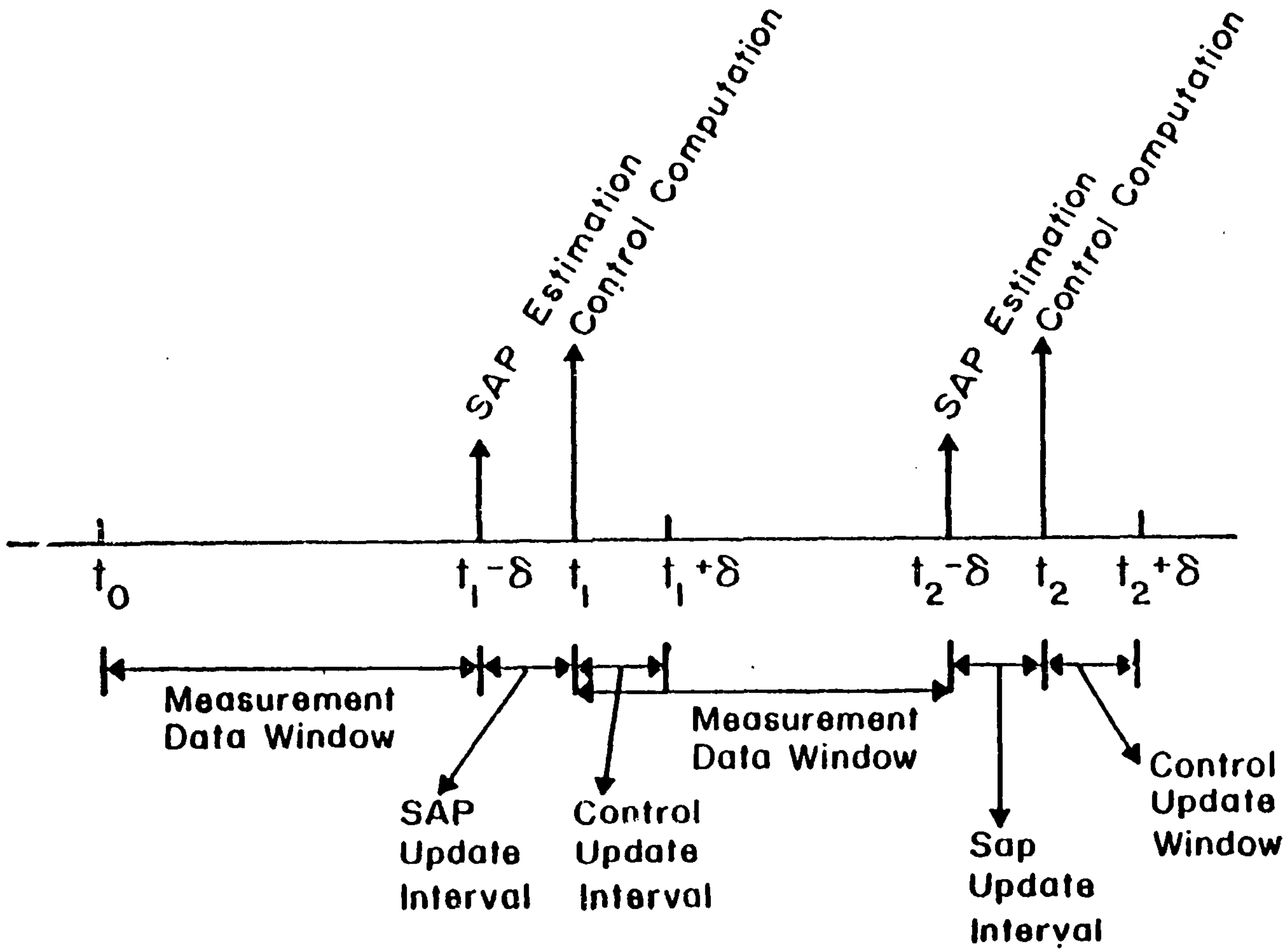


FIGURE 2.2: Adaptive Control Timing Chart

$t_o<t_1<t_2 \ldots .<t_{n-1}<t_N = t_f$

and that the control law to be optimized is linear in the state as follows:

$u(t) = Kx(t)$

To implement the explicit adaptive control scheme, a sequence of control and estimation problems must be solved. In particular, the following must be solved on-line:

Control Problem:

$$\min_{K} J_i = \int_{t_i}^{t_f} L[x(t),Kx(t),t]dt \qquad i = 0, 1, 2, \ldots \qquad (2.3.2\text{-}1)$$

subject to $\dot{x}(t) = f[x(t),Kx(t),t]$.

SAP Estimation Problem:

$$\min_{\hat{x}(t_i)} J_i = \tfrac{1}{2}\int_{t_i}^{t_{i+1}} ||z(t) - h[\hat{x}(t),t]||^2_{R^{-1}(t)} dt \quad i = 0,1,2,\ldots,N \qquad (2.3.2\text{-}2)$$

subject to

$\dot{\hat{x}}(t) = f[\hat{x}(t),K\hat{x}(t),t]$.

Note that $\hat{x}(t)$ is an augmented state vector which contains the unknown parameters to allow the simultaneous estimation of the states and parameters.

In view of the above problem formulation, the following parallel procedure might be employed to adapt the control in response to parameter changes detected on-line.

<u>Explicit Adaptive Control Algorithm - Direct Method</u>

Step 0: Initialize the estimator and controller with a nominal set of parameters and control gains. Start the process and apply the

control which is computed based upon the nominal quantities as the process evolves for $t \geq t_o$. Set $i = 0$.

Step 1: Throughout the interval $t\varepsilon[t_i, t_{i+1}]$ record the noisy measurements $z(t)$ and the control $u(t)$. At time t_{i+1}, use the recorded values to update the estimates of $x(t)$ by minimizing eqn. (2.3.2-2) using the direct SAP estimation algorithm discussed in Section 2.1.3.

Step 2: Reinitialize the controller with the updated estimates of $\hat{x}(t_{i+1})$ and reoptimize the control gains over the interval $[t_{i+1}, t_f]$ by minimizing eqn. (2.3.2-1) using the direct gain optimization procedure presented in Section 2.2.3.

Step 3: If $t_{i+1}<t_f$, apply the reoptimized control to the process for $t \geq t_{i+1}$, set $i \rightarrow i+1$ and go to Step 1. Otherwise, stop since the mission time has been exhausted.

The optimality of the control histories generated according to the above procedure primarily depends upon two items:

- The reliability of the state and parameter estimates obtained at the adaptation times.
- The ability of the parallel algorithms to reduce the performance criteria given by eqns. (2.3.2-1) and (2.3.2-2).

The stability of this algorithm depends mainly on how far the actual process parameters are from their nominal values when the process is started, the degree of parameter variation during the mission time and the frequency of adaptation.

Although the explicit adaptive control algorithm previously described employed the direct estimation and control procedures, it could just as easily employ the indirect methods discussed in Sections 2.1 and 2.2. In this case, these methods would be used to solve the NTPBVP's associated with the optimal control and estimation problems. Note that if parallel shooting is used, the adaptation times should correspond with the mesh points required by the parallel shooting method. It should be emphasized that no matter which parallel algorithm (direct or indirect) is employed to perform the estimation and control operations, the role of parallelism is to reduce the computation time enough to allow the on-line implementation of the explicit adaptive control scheme.

CHAPTER THREE

PARALLEL ALGORITHMS FOR REDUCING COMPUTATION TIME

To reduce the amount of computation time associated with the parallel algorithms described in Chapter Two, one may employ parallel minimization algorithms and parallel methods for integrating ordinary differential equations (ode's). Specifically, a reduction in computation time is possible because:

- Parallel minimization algorithms generally require fewer iterations to minimize a function compared with serial methods.

- Parallel integration procedures allow many of the arithmetic operations associated with integrating ode's to be performed simultaneously on separate processors.

In Section 3.1, a survey of parallel minimization procedures is presented. Also in this section, a class of parallel rank-two quasi-Newton methods are developed which is one of the major contributions of this thesis.

In Section 3.2, parallel integration methods are surveyed and one of the methods is extended so that the integration step size is automatically adjusted to maintain a desired level of accuracy while keeping the parallel structure of the algorithm. The development of such a parallel variable step size integration scheme is a significant contribution in its own right.

Finally, the advantages of utilizing the new parallel methods developed in this chapter are illustrated by comparing these methods

With regard to the monotone sequence, it has been observed that if this sequence is selected such that it approaches zero too rapidly, then the total number of function evaluations required to locate the minimum becomes needlessly large and as a result, the amount of time required for convergence increases significantly. On the other hand, if the monotone sequence approaches zero too slowly, these relatively large values may cause the Chazan-Miranker algorithm to become unstable.

Experience with the Chazan-Miranker method indicates that the performance of this method is highly dependent on the choice of algorithm parameters which is not very desirable.

Parallel Variable Metric Algorithm

Straeter has developed a gradient-based parallel variable metric (PVM) algorithm which can be implemented on modern parallel computers [26]. One of the properties of the PVM algorithm is that if the function being minimized is a quadratic in n variables, then the iterates will converge to the location of the minimum in one iteration provided n levels of parallelism are used. Also, Straeter has shown that for strictly convex functions on a finite dimensional space, the iterates converge to the minimum provided the metrics are uniformly bounded.

Straeter's PVM algorithm is a parallel version of Broyden's symmetric rank-one procedure [29], which requires at most n iterations to find the minimum of a quadratic function in n variables. Note that when minimizing a quadratic, the PVM algorithm is n times faster than the symmetric rank-one procedure. Although this is highly desirable and the major reason for developing a parallel minimization procedure, Straeter's method suffers from the same problems associated with Broyden's symmetric rank-one procedure.

with some existing minimization and integration algorithms.

3.1 Parallel Minimization Algorithms

In this section, methods for unconstrained minimization are discussed which are suitable for modern parallel computers. At the present time, only three algorithms have been reported which possess this feature. These methods include the nongradient algorithm of Chazan and Miranker [25], the parallel variable metric (PVM) algorithm reported by Straeter [26], and the parallel Jacobson-Oksman (PJO) procedure developed by Straeter and Markos [27].

These parallel procedures are described in some detail in Section 3.1.1 to motivate the discussion of a class of parallel quasi-Newton (PQN) methods which is developed in Section 3.1.2. In Section 3.1.3, the PQN method is tested by minimizing a standard set of test functions and the performance of this new method is demonstrated by comparing it with some popular minimization algorithms currently in use.

3.1.1 A Survey of Parallel Algorithms for Unconstrained Minimization

In this section, three parallel algorithms for unconstrained minimization are discussed to provide an indication of the state-of-the-art in this area of research. The methods to be considered include the nongradient algorithm of Chazan and Miranker [25] and the gradient-dependent algorithms developed by Straeter [26], [27]. The mathematical details of each parallel algorithm may be found in the Appendix, while a brief review of their properties and shortcomings is given in the remainder of this section.

Chazan-Miranker Algorithm

Chazan and Miranker have developed a parallel nongradient algorithm for unconstrained minimization which is suitable for execution on an array of parallel processors [25]. It can be shown that this algorithm will converge for strictly convex, twice continuously differentiable functions. Moreover, if the function to be minimized is a quadratic in n variables, the procedure will require at most n^2 one-dimensional minimizations to converge. Since these one-dimensional minimizations can be performed simultaneously using n levels of parallelism, at most n iterations would be needed. Note that this is significantly faster than the serial Zangwill-Powell nongradient method [28], which requires approximately n^2 sequential one-dimensional minimizations to find the minimum of a quadratic in n variables. This implies that the speed-up due to parallelism increases linearly with the number of processors when minimizing a quadratic by the Chazan-Miranker algorithm.

The Chazan-Miranker algorithm is based on the properties of conjugate directions. In fact, it can be shown that the search direction vectors generated by this algorithm form a set of conjugate directions. By searching along these directions, convergence is guaranteed (at least when the function being minimized is convex). The rate of convergence, however, depends primarily on the accuracy of each line search and a monotone decreasing sequence tending to zero.

With regard to the line search, provisions must be made for allowing both positive and negative values of the linear search parameter because the search directions generated are not necessarily descent directions. Note that this complicates the line search algorithm to some degree.

The most noteable problem with all rank-one algorithms (serial or parallel) is that the update rule used to construct the inverse Hessian is numerically unstable. That is, the update is undefined when certain vectors are orthogonal. Unfortunately, this occurs quite often when applying Straeter's method to nonquadratic functions and generally results in a nonpositive definite update. Also, Straeter's PVM algorithm requires accurate gradient information for the method to converge. Since the gradient of highly complex functions is difficult at best to compute numerically, this problem may seriously limit the application of Straeter's method.

Parallel Jacobson-Oksman Procedure

Another gradient-dependent method for unconstrained minimization which exploits the parallel computing capabilities of modern parallel computers is the parallel Jacobson-Oksman (PJO) procedure reported by Straeter and Markos [27]. This algorithm is a modification of the sequential Jacobson-Oksman (SJO) procedure [30] which assumes that the function being minimized is homogeneous. Because the class of homogeneous functions contains the quadratics as a subclass, homogeneous functions are therefore richer than the quadratics. Moreover, functions which have a singular Hessian at the minimum can be more accurately approximated by a homogeneous model.

At each iteration of the PJO algorithm, a linear system of n+2 equations must be solved. Straeter has shown that if the solution of this linear system exists, and the function being minimized is homogeneous in n variables, then the PJO algorithm will converge to the minimum in one iteration provided n+2 levels of parallelism are used.

By comparison, the SJO procedure requires n+2 iterations to minimize a homogeneous function in n variables. Straeter also shows that the PJO algorithm will converge to the location of the minimum of any function with a continuous, uniformly positive definite matrix of second partial derivatives.

Although the PJO algorithm is relatively efficient, if has been reported in [27] that in practice the PJO algorithm may not perform better than the SJO algorithm. Straeter also indicates that the major problem associated with the PJO algorithm is its limited robustness. The term robustness used by Straeter refers to the relative insensitivity of the PJO algorithm to the magnitude of the basis vectors needed by the PJO algorithm. In fact, if the magnitude of the basis vectors is too small, the linear system which must be solved at each iteration may not have full rank or may be very close to being singular. The problems cited above are not very encouraging and seem to indicate that much care must be taken when using the PJO algorithm.

In view of the problems associated with the parallel minimization algorithms discussed in this survey, it appears that there exists a need to develop a more robust and dependable method for minimizing a function of several variables. In the next section, a class of parallel rank-two quasi-Newton methods are presented which are shown to be more robust and dependable than currently existing procedures.

3.1.2 A Class of Parallel Double-Rank Quasi-Newton Methods

In the previous section, a survey of parallel minimization methods was presented and the shortcomings of these methods were cited. Since the time of their development, new results have appeared in the literature which may be amenable to parallel computation. In particular, Broyden [29] has introduced a family of variable metric formulae which are useful for function minimization and have the desirable property of quadratic termination provided accurate line searches are used. Imbedded in Broyden's class of quasi-Newton methods is the Davidon-Fletcher-Powell (DFP) method [31], the Broyden-Fletcher-Shanno (BFS) method [32] and the symmetric rank-one (SR1) method [29].

Analytical and empirical studies by Dixon [33] and [34], and Himmelblau [35] indicate that the BFS rule is generally preferable to the DFP and SR1 updates because of its reliability of convergence for a wide class of problems. In view of these results, the remainder of this section is concerned with restructuring Broyden's class of quasi-Newton methods such that the modified procedure posses a high degree of parallelism. A particularly interesting outcome of this work is a class of parallel double-rank quasi-Newton methods (such as a Parallel Davidon-Fletcher-Powell (PDFP) method and Parallel Broyden-Fletcher-Shanno (PBFS) method, as well as a parallel version of the symmetric rank-one method. It is felt that this new class of parallel quasi-Newton methods potentially can be far superior to the parallel methods surveyed in Section 3.1.1.

3.1.2.1 The Parallel Quasi-Newton Method

In this section, a gradient-dependent parallel algorithm which employs a rank-two correction to approximate the inverse Hessian matrix associated with Newton's method is developed. One of the desirable properties of this new parallel minimization algorithm is that if the function being minimized is a quadratic in n variables, then the inverse Hessian can be constructed exactly in one iteration, provided n+1 levels of parallelism are used. This property of the parallel quasi-Newton (PQN) method will be proven later in this chapter. At this time, however, it seems appropriate to formally present the method.

Parallel Quasi-Newton Method

Given $x^{(0)}$, $H^{(0)}$, and $\Sigma = (\sigma_1, \sigma_2, \ldots, \sigma_n) = c\, I_n$; $c > 0$, let $\ell = 0$, $m = 2$, and perform the following steps:

Step 1:

a. Let $x_j = x^{(\ell)} + \sigma_j$. Then simultaneously compute:

$$g(x^{(\ell)}) \quad \text{and} \quad g_j = g(x_j)$$

$$\forall\ j = 1, 2, \ldots, n$$

b. Simultaneously compute the gradient differences:

$$y_j = g_j - g(x^{(\ell)}) \qquad j = 1, 2, \ldots, n$$

Step 2:

Let $d_1 = \sigma_1$ and solve the following linear system for c_{m1}, c_{m2}, $\ldots$, $c_{m,m-1}$:

$$\begin{bmatrix} y_1^T d_1 & y_2^T d_1 & \cdots\cdots & y_{m-1}^T d_1 \\ y_1^T d_2 & y_2^T d_2 & \cdots\cdots & y_{m-1}^T d_2 \\ \vdots & & & \vdots \\ y_1^T d_{m-1} & y_2^T d_{m-1} & \cdots\cdots & y_{m-1}^T d_{m-1} \end{bmatrix} \begin{bmatrix} c_{m1} \\ c_{m2} \\ \vdots \\ c_{m,m-1} \end{bmatrix} = \begin{bmatrix} -y_m^T d_1 \\ -y_m^T d_2 \\ \vdots \\ -y_m^T d_{m-1} \end{bmatrix}$$

Then construct the direction vector:

$$d_m = \sigma_m + \sum_{j=1}^{m-1} c_{mj} \sigma_j$$

If $m < n$, set $m \to m+1$ and repeat this step. Otherwise, go to Step 3.

Step 3:

a. Compute "n+1" gradients of $f(x)$ at "n+1" distinct points in parallel:

$$g(x^{(\ell)}) \qquad \text{and} \qquad g_j = g(x^{(\ell)} + d_j) \qquad j = 1, 2, \ldots, n$$

b. Compute the gradient difference in parallel:

$$y_j = g_j - g(x^{(\ell)}) \qquad j = 1, 2, \ldots, n$$

Step 4:

Update H using "n" rank-two corrections. Let $H_o^{(\ell+1)} = H^{(\ell)}$, $\phi \in [0, 1]$ and compute:

$$H_j^{(\ell+1)} = H_{j-1}^{(\ell+1)} + \frac{d_j\, d_j^T}{d_j^T\, y_j} - \frac{(H_{j-1}^{(\ell+1)}\, y_j)\,(H_{j-1}^{(\ell+1)}\, y_j)^T}{y_j^T\, H_{j-1}^{(\ell+1)}\, y_j}$$

$$+ \phi\, v_j\, v_j^T \qquad\qquad j = 1, 2, \ldots, n$$

where

$$v_j = \left[y_j^T \left(H_{j-1}^{(\ell+1)} y_j\right)\right]^{\frac{1}{2}} \left[\frac{d_j}{d_j^T y_j} - \frac{H_{j-1}^{(\ell+1)} y_j}{y_j^T H_{j-1}^{(\ell+1)} y_j}\right]$$

Then set $H^{(\ell+1)} = H_n^{(\ell+1)}$

Step 5:

Perform a line search in the direction s as follows:

$$\min_{\alpha} \; f(x^{(\ell)} + \alpha \, s)$$

where

$$s = -H^{(\ell+1)} \, g(x^{(\ell)})$$

and set $x^{(\ell+1)} = x^{(\ell)} + \alpha \, s$.

If $|f(x^{(\ell+1)}) - f(x^{(\ell)})| < \varepsilon$, stop; otherwise, set $\ell \rightarrow \ell+1$, set $\sigma_j = d_j \; \forall \; j = 1, 2, \ldots, n$ and go to Step 2.

It should be pointed out that a fundamental need of the PQN method is the solvability of the linear system of equations shown in Step 2 of the algorithm. The issue of solvability will be analyzed in the next section assuming the function being minimized is quadratic. However, a rank test should be incorporated into the linear equation solver to test for solvability at each step of the iteration.

3.1.2.2 Properties and Convergence of the PQN Method

In this section, an analysis of the PQN method will be conducted to demonstrate the properties of this algorithm and show that the algorithm will converge in only one iteration to the minimum of a quadratic function. If the reader is not particularly interested in the mathematical details of the convergence proof presented in this section, but is more interested in the performance of the PQN algorithm, he should move on to Section 3.1.3 since the rest of this report may be read without an understanding of the following analysis.

To begin our study of the PQN method, the following definitions are in order:

Definition: A function $f\colon R^n \to R^1$ is said to be quadratic if f is of the form:

$$f(x) = \tfrac{1}{2} x^T A x + b^T x + c$$

where the A matrix is positive definite symmetric (pds).

Definition: Let A be pds. Then a finite set of vectors $d_1, d_2, \ldots, d_n$ is said to be mutually conjugate if

$$d_i^T A d_k = 0 \qquad \forall\ i \neq k.$$

At this time, a number of propositions will be stated and proved which summarize the properties of the PQN method. Ultimately, these results will be used to prove convergence of the PQN method.

Proposition 3.1: Let $f\colon R^n \to R^1$ be quadratic and b_j, $j = 1, 2, \ldots, n$ be an arbitrary vector. If $\tilde{x}_j = x + b_j$, and $y_j = g(\tilde{x}_j) - g(x)$, then $y_j = A\ b_j$.

Proof: Since $f(x) = \tfrac{1}{2} x^T Ax + b^T x + c$, we have

$$y_j = A(x + b_j) - Ax = Ab_j, \quad \forall\ j = 1, 2, \ldots, n$$

At this point, it will be shown that the direction vectors generated according to Step 2 of the PQN method form a set of mutually conjugate directions.

Proposition 3.2: Let $f\colon R^n \to R^1$ be quadratic and suppose

$$\Sigma = (\sigma_1, \sigma_2, \ldots, \sigma_n) = c\ I_n; \quad c > 0.$$

If the sequence of linear systems shown in Step 2 of the PQN method is solvable when $d_1 = \sigma_1$ and

$$d_i = \sigma_i + \sum_{j=1}^{i-1} c_{ij}\, \sigma_j \qquad i = 2, \ldots, n$$

then

$$d_i^T A\, d_k = 0 \qquad \forall\ i \neq k$$

Proof: This proposition will be proven in two steps. First, the result will be shown for the first iteration of the PQN method and secondly, for all other iterations.

Since f is quadratic, the result of Proposition 3.1 implies that $y_j = A\, \sigma_j \quad \forall\ j = 1, 2, \ldots, n$ in Step 1 of the PQN method. By direct computation,

$$d_i^T A\, d_k = \sigma_i^T A\, d_k + \sum_{j=1}^{i-1} c_{ij}\, \sigma_j^T A\, d_k$$

$$= y_i^T d_k + \sum_{j=1}^{i-1} c_{ij}\, y_j^T d_k \qquad (3.1.2.2\text{-}1)$$

$\forall\ i = 2, 3, \ldots, n$ and $k = 1, 2, \ldots, i-1$

In view of the structure of the linear system shown in Step 2 of the PQN method, it should be clear that the linear system of equations may be written as follows:

$$y_i^T d_k + \sum_{j=1}^{i-1} c_{ij}\, y_j^T d_k = 0 \qquad (3.1.2.2\text{-}2)$$

$\forall\ i = 2, 3, \ldots, n$ and $k = 1, 2, \ldots, i-1$

By the solvability hypothesis, however, the c_{ij}'s can be found to satisfy eqn. (3.1.2.2-2). But this implies that

$$d_i^T A d_k = 0 \qquad \forall i \neq k$$

Hence, the direction vectors generated during the first iteration of the PQN method are mutually conjugate.

Now consider all other iterations.

In Step 5 of the PQN method, $\sigma_j = d_j$, $\forall j = 1, 2, \ldots, n$, i.e., the basis vectors are set to the most recent set of mutually conjugate directions. Hence, $y_j = A d_j$ for all remaining iterations. Now let $\hat{d}$ denote the updated value of d. Then

$$\hat{d}_i^T A \hat{d}_k = (d_i^T + \sum_{j=1}^{i-1} c_{ij} d_j) A (d_k + \sum_{\ell=1}^{k-1} c_{k\ell} d_\ell) = 0$$

$$\forall i = 2, 3, \ldots, n \quad \text{and} \quad k = 1, 2, \ldots, i-1$$

since the d_j's are mutually conjugate.

The next result shows that if the function being minimized is quadratic, then the linear system shown in Step 2 of the PQN method will be solvable for all iterations provided it is solvable on the first iteration. To show this and other results, the following assumption is needed.

<u>Assumption A.1</u>: Henceforth in this section we will assume that the algorithm is solvable on the first iteration.

Proposition 3.3: If f: $R^n \to R^1$ is quadratic and $\{d_i\}_{i=1}^{n}$ is a set of mutually conjugate directions generated according to Step 2 of the PQN method, then after one iteration of the PQN method the coefficient matrix

$$C_{m-1} = \begin{bmatrix} y_1^T d_1 & \cdots & y_{m-1}^T d_1 \\ \vdots & \ddots & \vdots \\ y_1^T d_{m-1} & \cdots & y_{m-1}^T d_{m-1} \end{bmatrix}$$

becomes a positive definite diagonal matrix for all other iterations.

Proof: Using the result of Proposition 3.1, and the fact that the d's are mutually conjugate, we have

$$y_i^T d_j = d_i^T A d_j = 0 \qquad \forall\ i \neq j \qquad (3.2.2.2\text{-}4)$$

Also, it should be clear that

$$y_i^T d_i = d_i^T A d_i > 0 \quad \text{since}$$

A is positive definite symmetric. Since the off diagonal terms of C_{m-1} are zero and the diagonal terms are positive, C_{m-1} is clearly a positive definite diagonal matrix after the initial iteration.

The next issue to be considered is positive definiteness of the update. That is, if we initialize the PQN method with a pds approximation to the inverse Hessian, can it be guaranteed that the updated inverse Hessian is pds? The answer to this question is resolved by Proposition 3.4 below.

Proposition 3.4: If $H_{j-1}^{\ell+1}$ is positive definite symmetric with $\phi \geq 0$, then $H_j^{\ell+1}$ is positive definite symmetric

$$\Leftrightarrow d_j^T y_j > 0 \qquad \forall j = 1, 2, \ldots, n$$

Proof: The symmetric property is obvious from the form of the update rule below:

$$H_j^{(\ell+1)} = H_{j-1}^{(\ell+1)} + \frac{d_j d_j^T}{d_j^T y_j} - \frac{(H_{j-1}^{(\ell+1)} y_j)(H_{j-1}^{(\ell+1)} y_j)^T}{y_j^T H_{j-1}^{(\ell+1)} y_j}$$

$$+ \phi v_j v_j^T \qquad (3.1.2.2\text{-}5)$$

where

$$v_j = \left[y_j^T H_{j-1}^{(\ell+1)} y_j\right]^{\frac{1}{2}} \left[\frac{d_j}{d_j^T y_j} - \frac{H_{j-1}^{(\ell+1)} y_j}{y_j^T H_{j-1}^{(\ell+1)} y_j}\right]$$

To show positive definiteness, the result is proved for $\phi = 0$ and then for $\phi > 0$. By direct computation, it is easy to show that when $\phi = 0$:

$$x^T H_j^{(\ell+1)} x = x^T H_{j-1}^{(\ell+1)} x + \frac{(x^T d_j)^2}{d_j^T y_j} - \frac{(x^T H_{j-1}^{(\ell+1)} y_j)^2}{y_j^T H_{j-1}^{(\ell+1)} y_j}$$

$$(3.1.2.2\text{-}6)$$

Let $a = [H_{j-1}^{(\ell+1)}]^{\frac{1}{2}} x$ and $b = [H_{j-1}^{(\ell+1)}]^{\frac{1}{2}} y_j$.

Substituting these quantities into eqn. (3.1.2.2-6), we have

$$x^T H_j^{(\ell\div1)} x = \frac{(a^T a)(b^T b) - (a^T b)^2}{b^T b}$$

$$+ \frac{(x^T d_j)^2}{d_j^T y_j} \qquad (3.1.2.2\text{-}7)$$

What we must show is that $x^T H_{j-1}^{(\ell+1)} x > 0 \quad \forall x \neq 0$. The first term in eqn.(3.1.2.2-7) can be shown to be positive semi-definite using the Schwartz inequality [36] as follows:

$$(a^T a)\, b^T b - (a^T b)^2 \geq 0$$

$$\frac{(a^T a)\,(b^T . b) - (a^T b)^2}{b^T b} \geq 0$$

Also, it is clear that:

$$\frac{(x^T d_j)^2}{d_j^T y_j} \geq 0 \Leftrightarrow d_j^T y_j > 0$$

Thus, when $\phi = 0$, we have shown that:

$$x^T H_j^{(\ell+1)} x \geq 0 \qquad \forall x \neq 0$$

To show strict inequality we must show that:

$$\frac{(a^T a)\,(b^T b) - (a^T b)^2}{b^T b} \quad \text{and} \quad \frac{(x^T d_j)^2}{d_j^T y_j}$$

do not vanish simultaneously. Note that

$$\frac{(a^T a)\,(b^T b) - (a^T b)^2}{b^T b} = 0$$

only if a and b are colinear. But this implies that x and y_j are colinear, i.e.,

$$x = \beta\, y_j \qquad \forall \beta \neq 0.$$

In this case, however,

$$x^T d_j = d_j^T x = d_j^T \beta\, y_j = \beta\, d_j^T y_j \neq 0$$

since

$$d_j^T y_j > 0 \quad \forall j = 1, 2, \ldots, n$$

Therefore, both

$$\frac{(a^T a)(b^T b) - (a^T b)^2}{b^T b} \quad \text{and} \quad \frac{(x^T d_j)^2}{d^T y_j}$$

can't vanish simultaneously, Hence,

$$x^T H_j^{(\ell+1)} x > 0 \qquad \forall\, x \neq 0.$$

Now suppose $\phi > 0$. In this case, the matrix $\phi\, v_j\, v_j^T$ is at least positive semidefinite. Since the update given by eqn.(3.1.2.2-5) consists of the sum of a positive definite matrix and atleast a positive semidefinite matrix, the update is positive definite.

Corollary: If $H^{(\ell)}$ is pds and $\phi \geq 0$, then $H^{(\ell+1)}$ is pds

$$\Leftrightarrow d_j^T y_j > 0 \quad \forall\, j = 1, 2, \ldots, n$$

Proof: Since $H^{(\ell+1)}$ is obtained from a finite sum of pds matrices, $H^{(\ell+1)}$ is pds.

The next result shows that the set of mutually conjugate directions generated by the PQN algorithm are also linearly independent.

Proposition 3.5: Let $\Sigma = (\sigma_1, \sigma_2, \ldots, \sigma_n) = c\, I_n$; $c > 0$. If $f(x): R^n \to R^1$ is quadratic with $d_1 = \sigma_1$ and Assumption A.1 holds, then for the PQN algorithm the set of vectors $d_1, d_2, \ldots, d_n$ are linearly independent.

Proof: Suppose there exist α_i $i = 1, 2, \ldots, n$ such that

$$\alpha_1\, d_1 + \ldots \alpha_n\, d_n = 0.$$

Then $\alpha_1\, d_i^T A\, d_1 + \ldots + \alpha_n\, d_i^T A\, d_n = \alpha_i\, d_i^T A\, d_i = 0$, in view of the fact that the d's are mutually conjugate by proposition 3.2. But since $d_i^T A\, d_i > 0$ due to the positive definiteness of A, α_i must be zero. But this is precisely what is required for the d's to be linearly independent.

The next two propositions are particularly useful in proving the convergence of the PQN method.

Proposition 3.6: Let $\phi \geq 0$ and $H_j^{(\ell+1)}$ be given by eqn. (3.1.2.2-5). Then $H_j^{(\ell+1)} y_j = d_j \quad \forall\ j = 1, 2, \ldots, n$

Proof: By direct computation,

$$H_j^{(\ell+1)} y_j = H_{j-1}^{(\ell+1)} y_j + \frac{d_j\, d_j^T\, y_j}{d_j^T\, y_j} - \frac{H_{j-1}^{(\ell+1)}\, y_j\, y_j^T\, H_{j-1}^{(\ell+1)}\, y_j^T}{y_j^T\, H_{j-1}^{(\ell+1)}\, y_j} + \phi\, v_j\, v_j^T\, y_j$$

$$= H_{j-1}^{(\ell+1)}\, y_j + d_j - H_{j-1}^{(\ell+1)}\, y_j + \phi\, v_j\, v_j^T\, y_j$$

$$= d_j + \phi\, v_j\, v_j^T\, y_j$$

$$= d_j \Leftrightarrow \phi v_j\, v_j^T\, y_j \equiv 0 \qquad \forall\ j = 1, 2, \ldots, n$$

Thus, the proposition will be established provided we can show $\phi v_j\, v_j^T\, y_j = 0$. If $\phi = 0$, the result is trivially true. Therefore, suppose $\phi > 0$. Then

$$\phi\, v_j\, v_j^T\, y_j = \phi\, y_j^T\, H_{j-1}^{(\ell+1)}\, y_j \left[\frac{d_j}{d_j^T\, y_j} - \frac{H_{j-1}^{(\ell+1)}\, y_j}{y_j^T\, H_{j-1}^{(\ell+1)}\, y_j}\right] \cdot \left[\frac{d_j}{d_j^T\, y_j} - \frac{H_{j-1}^{(\ell+1)}\, y_j}{y_j^T\, H_j^{(\ell+1)}\, y_j}\right]^T y_j$$

$$= \frac{\phi}{d_j^T\, y_j}\left[\frac{y_j^T\, H_{j-1}^{(\ell+1)}\, y_j\, d_j\, d_j^T}{d_j^T\, y_j} - d_j\, y_j^T\, H_{j-1}^{(\ell+1)} - H_{j-1}^{(\ell+1)}\, y_j\, d_j^T\right] y_j + \frac{H_{j-1}^{(\ell+1)}\, y_j\, y_j^T\, H_{j-1}^{(\ell+1)}\, y_j}{y_j^T\, H_{j-1}^{(\ell+1)}\, y_j}$$

After some manipulation, it is easy to show that:

$$\phi \, v_j \, v_j^T \, y_j = \frac{\phi}{d_j^T \, y_j} \left[y_j^T \, H_{j-1}^{(\ell+1)} \, y_j \, d_j - d_j \, y_j^T \, H_j^{(\ell+1)} \, y_j - H_{j-1}^{(\ell+1)} \, y_j \, d_j^T \, y_j \right] + H_{j-1}^{(\ell+1)} \, y_j = 0$$

Proposition 3.7: If f(x): $R^n \to R^1$ is quadratic, $\phi \geq 0$, $\exists \; w \ni$

$A^{-1} w = H_{j-1}^{(\ell+1)} w$, and

$$B = H_{j-1}^{(\ell+1)} + \frac{d_j \, d_j^T}{d_j^T \, y_j} - \frac{(H_{j-1}^{(\ell+1)} \, y_j)\,(H_{j-1}^{(\ell+1)} \, y_j)^T}{y_j^T \, (H_{j-1}^{(\ell+1)} \, y_j)}$$

$$+ \, \phi \, v_j \, v_j^T \Leftrightarrow d_j^T \, y_j > 0 \qquad j = 0, 1, \ldots, n$$

where

$$v_j = \left[y_j^T \; H_{j-1}^{(\ell+1)} \; y_j \right]^{\frac{1}{2}} \left[\frac{d_j}{d_j^T \, y_j} - \frac{H_{j-1}^{(\ell+1)} \, y_j}{y_j^T \, H_{j-1}^{(\ell+1)} \, y_j} \right]$$

then

$$A^{-1} \, w = B \, w$$

Proof: Since f(x) is quadratic, from Proposition 3.1 we have:

$$A^{-1} \, y_j = d_j \qquad \forall \; j = 1, \ldots, n$$

By hypothesis,

$$(B - A^{-1}) \, w = \left[\frac{d_j \, d_j^T}{d_j^T \, y_j} - \frac{(H_{j-1}^{(\ell+1)} \, y_j)\,(H_{j-1}^{(\ell+1)} \, y_j)^T}{y_j^T \; H_{j-1}^{(\ell+1)} \; y_j} + \phi \, v_j \, v_j^T \right] w \qquad (3.1.2.2\text{-}8)$$

Also, the assumption that $\exists w \ni$

$$A^{-1} w = H_{j-1}^{(\ell+1)} w \Rightarrow A^{-1} . H_{j-1}^{(\ell+1)}$$

Substituting $d_j = A^{-1} y_j$ and $H_{j-1}^{(\ell+1)} = A^{-1}$ into eqn.(3.1.2.2-8) leads to:

$$(B - A^{-1}) w = 0$$

$$\Rightarrow A^{-1} w = B w$$

Corollary 1: If $H_{j-1}^{(\ell+1)} w = A^{-1} w$ for some $w \in R^n$ and $f(x): R^n \to R^1$ is quadratic, then $H_j^{(\ell+1)} w = A^{-1} w$.

Corollary 2: (Fundamental Property of H)

If $f(x): R^n \to R^1$ is quadratic, then

$$H_j^{(\ell+1)} y_k = A^{-1} y_k = d_k \qquad \forall k \leq j = 1, 2, \ldots, n$$

The proof of Corollary 1 is obvious from Proposition 3.7 when $B \equiv H_j^{(\ell+1)}$. However, the proof of Corollary 2 is more subtle. To prove Corollary 2, we shall use mathematical induction. Note that since f(x) is quadratic, we may invoke Corollary 1 with $w \equiv y_k$ and Proposition 3.1 to obtain

$$H_j^{(\ell+1)} y_k = A^{-1} y_k = d_k \text{ for any k and } j = 1, 2, \ldots, n$$

However from Proposition 3.6, we have

$$H_j^{(\ell+1)} y_j = d_j \qquad \forall j = 1, 2, \ldots, n.$$

Now let us assume $H_{j-1}^{(\ell+1)} y_j = d_k \quad \forall k \leq j-1$. Also, by Proposition 3.6

$$H_j^{(\ell+1)} y_k = d_k \text{ for } k \equiv j = 1, 2, \ldots, n.$$

However, using Corollary 1 of Proposition 3.6 the fact that $d_k = A^{-1} y_k$, and the inductive hypothesis, we have

$$H_j^{(\ell+1)} y_k = A^{-1} y_k = d_k \qquad \forall k \leq j = 1, 2, \ldots, n$$

At this time, we are in a position to prove two very important convergence theorems. The first result shows that the PQN method converges exactly to the inverse Hessian of a quadratic function by performing Steps 1, 2, 3, and 4, while the second result indicates that the PQN method will minimize a quadratic function in only one iteration (Steps 1, 2, 3, 4, and 5).

Theorem 3.1: If $f(x): R^n \to R^1$ is quadratic and Assumption A.1 holds, then $H^{(1)} = A^{-1}$.

Proof: Let $x, z \in R^n$ and suppose $x = Az$. Then since $f(x)$ is quadratic with A psd, A^{-1} exist so that

$$z = A^{-1} x \qquad (3.1.2.2\text{-}9)$$

From Proposition 3.5, the d_j's $j = 1, 2, \ldots, n$ are linearly independent so they form a basis in R^n. Hence, $\exists\ \beta_j \ni$

$$z = \sum_{j=1}^{n} \beta_j d_j$$

Since f is quadratic, we may write:

$$x = A z = \sum_{j=1}^{n} \beta_j A d_j = \sum_{j=1}^{n} \beta_j y_j$$

From the fundamental property of H, we have

$$H_n^{(1)} y_j = d_j \qquad j = 1, 2, \ldots, n$$

But by definition, $H_n^{(1)} \equiv H^{(1)}$. Therefore,

$$H^{(1)} x = \sum_{j=1}^{n} \beta_j H^{(1)} y_j = \sum_{j=1}^{n} \beta_j d_j = z \qquad (3.1.2.2\text{-}10)$$

using the fact that $H^{(1)} y_j = d_j$. But eqns.(3.1.2.2-9) and (3.1.2.2-10) imply that

$$H^{(1)} x = z = A^{-1} x$$

Hence, $H^{(1)} = A^{-1}$.

Theorem 3.2: If $f(x)$: $R^n \rightarrow R^1$ is quadratic, then the PQN algorithm will converge to the location of the minimum of $f(x)$ in one iteration provided Assumption A.1 holds.

Proof: Let $f(x) = \frac{1}{2} x^T Ax + b^T x + c$. From Theorem 3.1, it is clear that after performing Steps 1-4 of the PQN method,

$$H^{(1)} = A^{-1}.$$

Using this result in Step 4 of the PQN method, results in a single line search of the form:

$$\min_{\alpha_0} f\,(x^{(1)} + \alpha_0 s)$$

where

$$s = -H^{(1)}\, g(x^{(1)})$$
$$= -A^{-1}(Ax^{(1)} + b) = -x^{(1)} - A^{-1}\, b$$

Hence, α_0 must be found to minimize

$$f(x^{(1)} - \alpha_0\, x^{(1)} - \alpha_0\, A^{-1}\, b) \qquad (3.1.2.2\text{-}11)$$

Since $f(x)$ is quadratic, the minimum of $f(x)$ is located at $x = -A^{-1}b$. Clearly, $\alpha_0 \equiv 1$ minimizes eqn.(3.1.2.2-11) and the updated solution is $x^{(2)} = -A^{-1}b$. Hence the procedure converges in one iteration.

The analysis presented in this section indicates that one of the major attributes of the PQN algorithm is that convergence will result after one iteration when the function being minimized is quadratic. This is significant because most highly efficient serial procedures (such as the DFP method) can require at most n iterations to converge in such cases.

It should be noted that the convergence results derived in this section assume that the function being minimized is quadratic. For nonquadratic functions, however, the convergence properties of the

PQN algorithm will be demonstrated in the usual way by testing this new algorithm on a set of standard test functions. This aspect is considered in the next section.

3.1.3 Test Function Performance

When a new minimization algorithm, such as the PQN method discussed in the previous section, is developed, it is a common practice to compare it with existing methods using a standard set of test functions. Some of the most common test functions used by researchers in this area includes the quadratic function, Rosenbrock's function, Powell's function, Wood's function, and the Helical Valley function [26, 34]. These functions and their properties are summarized below:

- Quadratic Function

$$f(x_1, x_2, x_3) = x_1^2 + 2x_2^2 + 5x_3^2 - 2x_1 x_2$$

Exact Solution: (0 , 0 , 0)

Starting Approximation: (1 , 1 , 1)

This function is rather easy to minimize and is included to verify the finite step convergence property of quasi-Newton methods.

- Rosenbrock's Function

$$f(x_1, x_2) = 100 (x_2 - x_1^2)^2 + (1 - x_1)^2$$

Exact Solution: (1 , 1)

Starting Approximation: (-1.2 , 1)

Rosenbrock's function is particularly difficult to minimize since the minimization must travel along the parabolic valley $y = x^2$.

- Powell's Function

$$f(x_1, x_2, x_3, x_4) = (x_1 + 10x_2)^2 + 5(x_3 - x_4)^2 + (x_2 - 2x_3)^4 + 10(x_1 - x_4)^4$$

Exact Solution: $(0 , 0 , 0 , 0)$

Starting Approximation: $(3 , -1 , 0 , 1)$

This function is difficult for a variable metric algorithm to minimize because at the minimum the Hessian is singular.

- Wood's Function

$$f(x_1, x_2, x_3, x_4) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2 + 90(x_4 - x_3^2)^2$$
$$+ (1 - x_3)^2 + 10.1 [(x_2 - 1)^2 + (x_4 - 1)^2]$$
$$+ 19.8(x_2 - 1)(x_4 - 1)$$

Exact Solution: $(1 , 1 , 1 , 1)$

Starting Approximation: $(-3 , -1 , -3 , -1)$

This function is difficult to minimize because the quadratics $x_1^2 - x_2$ and $x_3^2 - x_4$ form a set of level curves which are banana shaped.

- Helical Valley

$$f(x_1, x_2, x_3) = 100 [(x_3 - 100)^2 + (\sqrt{x_1^2 + x_2^2} - 1)^2] + x_3^2$$

where

$$2 \pi \Theta = \begin{cases} \tan^{-1} x_2/x_1 & \text{for } x_1 \geq 0 \\ \pi + \tan^{-1} x_2/x_1 & \text{for } x_1 < 0 \end{cases}$$

Exact Solution: $(1 , 0 , 0)$

Starting Approximation: $(-1 , 0 , 0)$

This function is rather difficult to minimize because the minimum is located at the bottom of a helical valley.

The test functions described above were used to study the convergence properties of the PQN algorithm. In particular, the PDFP [illegible] methods were employed to minimize these test functions. Also, [illegible] method, as well as the PVM method, were employed to minimize the test functions previously described. The resulting performance of each method is summarized in Tables 3.1-3.5.

The results indicate:

- Quadratic Function

In this case, the parallel algorithms converge in only one iteration while the serial methods converged after three iterations. Note that these results are consistant with theoretical results which indicate that the PVM, PDFP and PBFS methods must converge in one iteration (see Table 3.1).

- Rosenbrock's Function

For this function, the PVM and PBFS algorithms converge significantly faster than the serial methods but the PDFP method required more iterations to converge than the serial DFP method (see Table 3.2). Since each gradient evaluation requires approximately the same time as two function evaluations, in this case, the equivalent number of function evaluations required by each method is:

26 x 2 + 63 = 115 for PDFP

107 + 42 = 149 for DFP

Because the PDFP method requires fewer equivalent function evaluations compared to the serial DFP method, the PDFP method will actually converge faster than the DFP method even though more iterations are required.

- Helical Valley

From the results shown in Table 3.3, the parallel methods require fewer iterations to converge than the serial methods. Note that the PVM methods converged the fastest in this case.

TABLE 3.1: Minimization Algorithm Performance: Quadratic Function

$\hat{x}_0 = (1 , 1 , 1)$ and $x_0 = (0 , 0 , 0)$

Minimization Algorithm	Total Number of Iterations	Total Number of Function Evaluations	Total Number of Gradient Evaluations	Function Value at Convergence
DFP	3	25	3	0.780×10^{-31}
BFS	3	24	3	0.607×10^{-31}
PVM	1	4	1	0.963×10^{-34}
PDFP	1	9	2	0.148×10^{-50}
PBFS	1	9	2	0.141×10^{-51}

TABLE 3.2: Minimization Algorithm Performance: Rosenbrock's Function

$\hat{x}_0 = (-1.2 \quad 1)$ and $x = (1 \quad 1)$

Minimization Algorithm	Total Number of Iterations	Total Number of Function Evaluations	Total Number of Gradient Evaluations	Function Value at Convergence
DFP	21	107	21	0.242×10^{-16}
BFS	24	80	24	0.123×10^{-18}
PVM	14	31	14	0.777×10^{-22}
PDFP	25	63	26	0.346×10^{-19}
PBFS	16	36	17	0.379×10^{-29}

TABLE 3.5: Minimization Algorithm Performance: Helical Valley

$\hat{x}_0 = (-1 , 0 , 0)$ and $x = (1 , 0 , 0)$

Minimization Algorithm	Total Number of Iterations	Total Number of Function Evaluations	Total Number of Gradient Evaluations	Function Value at Convergence
DFP	21	83	21	0.101×10^{-13}
BFS	21	72	21	0.105×10^{-15}
PVM	13	31	13	0.358×10^{-14}
FDFP	17	39	18	0.111×10^{-16}
PBFS	15	32	16	0.278×10^{-14}

- Wood's Function

In this case, the parallel algorithms once again converged more rapidly than the serial methods. In fact, the PVM method is nearly 50% faster than the serial DFP method. Also, note that the PBFS method is competitive with the PVM method this time (see Table 3.4).

- Powell's Function

As seen from Table 3.5, each of the parallel minimization procedures convergsd more rapidly than the serial methods. All of the parallel methods performed equally well on this test function.

In summary, the results indicate that without question the parallel minimization procedures converge more rapidly than serial methods. Also, it appears that the PBFS method is preferable to the PDFP procedure.

Before a recommendation can be made as to which parallel algorithm should be generally used, a robustness study should be conducted. The term robustness used here is a measure of the relative insensitivity of a parallel algorithm to the magnitude of $||\sigma_i||$ $i = 1, 2, \ldots, n$. The issue of robustness will be addressed by varying the weighting parameter, c, associated with the set of linearly independent vector

$$\Sigma = (\sigma_1, \sigma_2, \ldots, \sigma_n) = c\ I_n;\ c > 0$$

required by the PVM, PDFP, and PBFS algorithms. In particular, the robustness of these algorithms is demonstrated in Figures 3.0-3.3, which were obtained by solving the set of standard test functions described earlier with $10^{-9} \leq c \leq 10^{-3}$.

TABLE 3.4: Minimization Algorithm Performance: Wood's Function

$\hat{x}_0 = (-3\ ,\ -1\ ,\ -3\ ,\ -1)$ and $x = (1\ ,\ 1\ ,\ 1\ ,\ 1)$

Minimization Algorithm	Total Number of Iterations	Total Number of Function Evaluations	Total Number of Gradient Evaluations	Function Value at Convergence
DFP	47	246	47	0.586×10^{-16}
BFS	36	121	36	0.107×10^{-16}
PVM	26	60	26	0.156×10^{-22}
PDFP	42	94	43	0.776×10^{-20}
PBFS	27	60	28	0.252×10^{-19}

TABLE 3.5: Minimization Algorithm Performance: Powell's Function

$\hat{x}_0 = (3 , -1 , 0 , 1)$ and $x = (0 , 0 , 0 , 0)$

Minimization Algorithm	Total Number of Iterations	Total Number of Function Evaluations	Total Number of Gradient Evaluations	Function Value at Convergence
DFP	19	85	19	0.205×10^{-10}
BFS	18	69	18	0.932×10^{-10}
PVM	13	29	13	0.199×10^{-10}
PDFP	13	32	14	0.153×10^{-10}
PBFS	13	32	14	0.477×10^{-10}

The results indicate:

- For Rosenbrock's function, the PDFP and PBFS algorithms are more robust than the PVM algorithm, although the PVM algorithm may require fewer iterations to converge (see Figure 3.1).
- For the Helical Valley function, the PBFS and PVM algorithms are more robust than the PDFP algorithm. Again the PVM algorithm converges more rapidly than the other methods (see Figure 3.2).
- For Wood's function, the PBFS algorithm possesses the highest degree of robustness. Note that for the PVM algorithm the total number of iterations required for convergence increases very rapidly if c is chosen too large. Also observe that the PDFP algorithm is more robust than the PVM algorithm even though more iterations are required for convergence (see Figure 3.3).
- For Powell's function, the PDFP method is the most robust, although the parallel minimization procedures all require approximately the same number of iterations to converge over a wide range of c; $10^{-9} \leq c \leq 10^{-5}$ (see Figure 3.4).

In summary, the robustness study conducted here indicates that the parallel rank-two quasi-Newton methods (PDFP, PBFS) generally are more robust than the rank-one PVM algorithm. The results also indicate that the PBFS algorithm might be preferable to the PDFP method. Although the PVM algorithm generally required fewer iterations to converge, the PBFS algorithm might be preferred in view of its superior robustness characteristics. Finally, the results obtained clearly show that parallel rank-two methods are more robust than parallel rank-one methods which was one of the major motivations for developing the PQN method presented in Section 3.1.

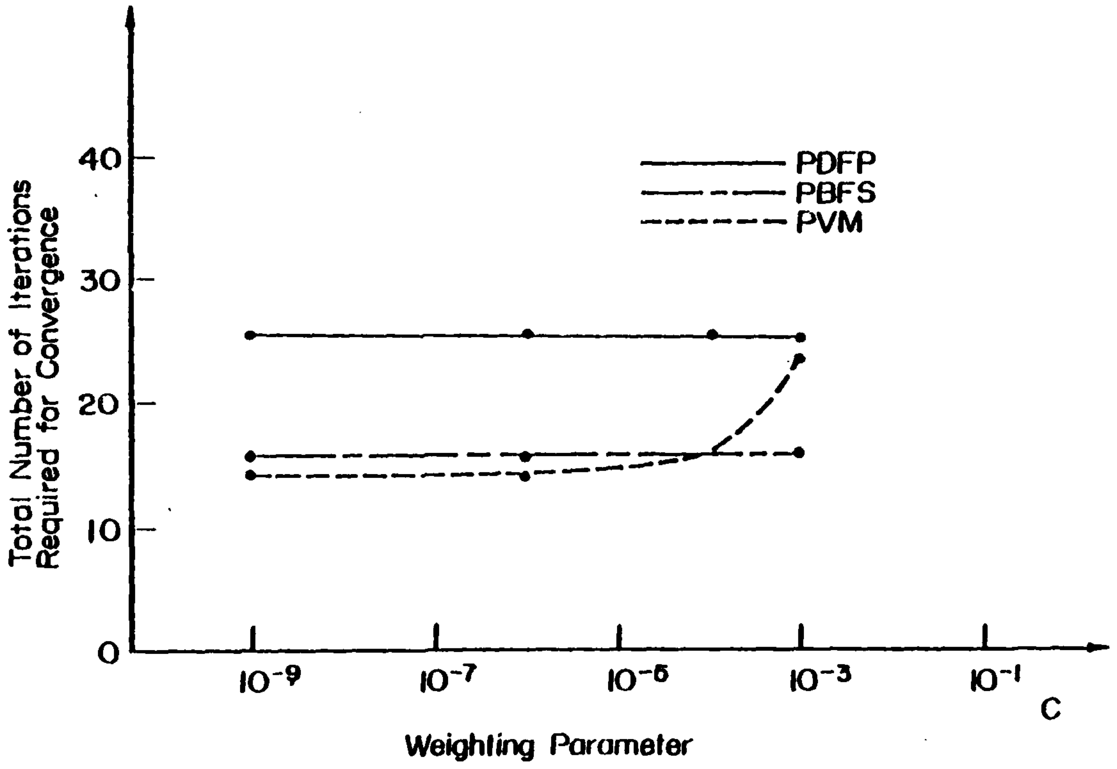


FIGURE 3.1: Robustness of Parallel Minimization Algorithms: Rosenbrock's Function

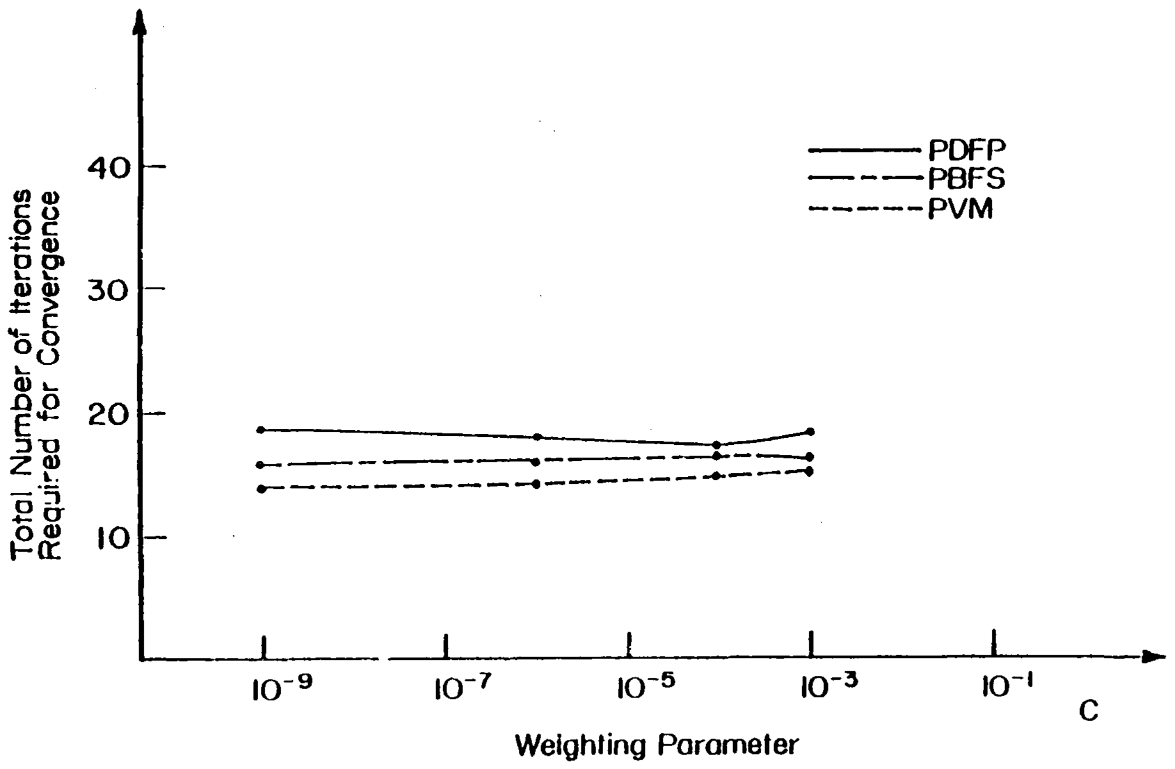


FIGURE 3.2: Robustness of Parallel Minimization Algorithms: Helical Valley

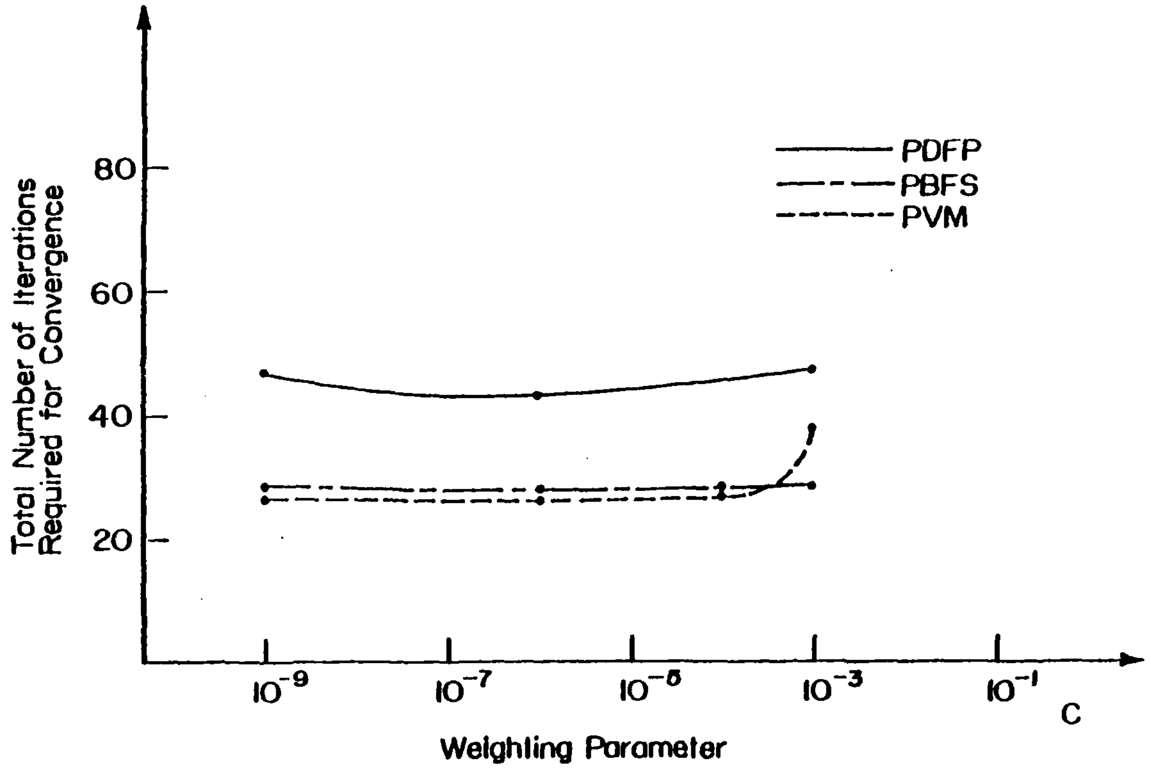


FIGURE 3.3: Robustness of Parallel Minimization Algorithms: Wood's Function

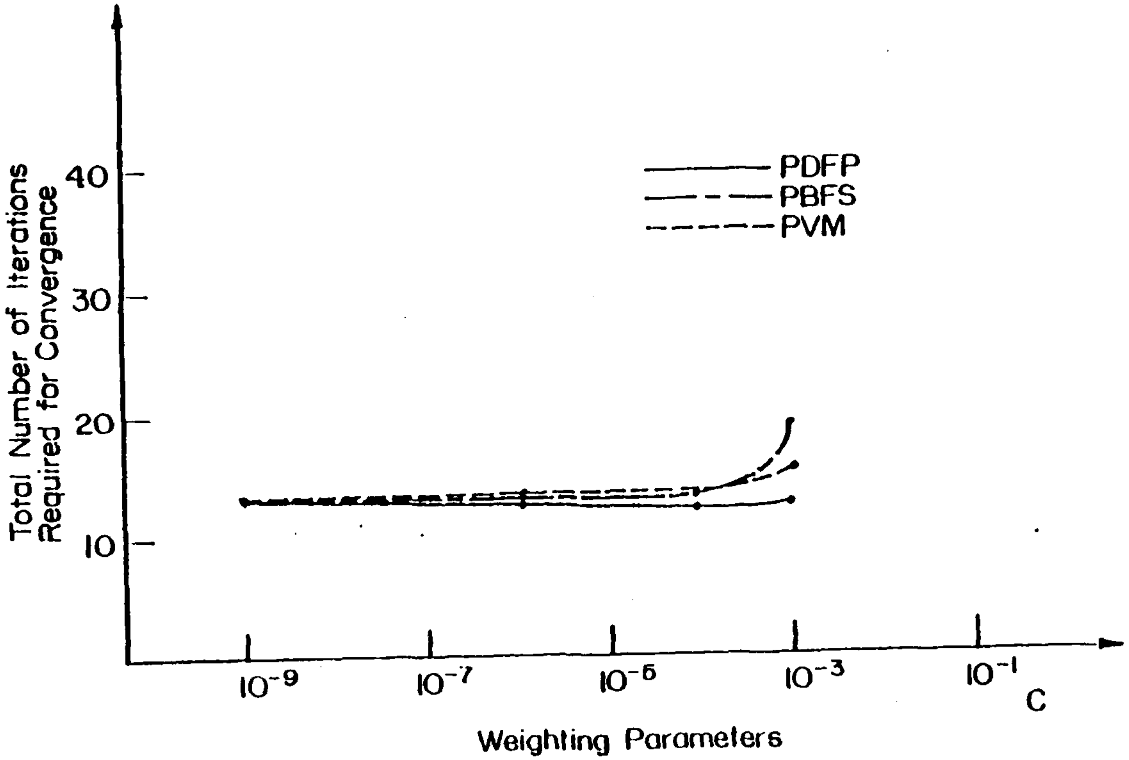


FIGURE 3.4: Robustness of Parallel Minimization Algorithms: Powell's Function

3.2 Parallel Methods for Integrating Ordinary Differential Equations

In this section, parallel methods for integrating ordinary differential equations are discussed and compared. In particular, Downs' method [37] and the Miranker-Liniger method [38] are discussed in Section 3.2.1. In Section 3.2.2, the Miranker-Liniger method is extended to allow the integration step size to be automatically adjusted so as to maintain a desired level of accuracy. To more fully appreciate the speed and accuracy of the parallel variable step size integration method, it is compared with existing integration methods in Section 3.2.3.

3.2.1 A Survey of Parallel Integration Algorithms

Before discussing parallel procedures for solving initial-value problems, let us first define the underlying problem and briefly mention some possible approaches to its solution. Therefore, consider the initial-value problem:

$$\dot{y}(t) = f[y(t), t] \qquad t \geq t_o \qquad (3.2.1\text{-}1)$$
$$y(t_o) = y_o$$

where the initial time, t_o, and the initial condition, y_o, are assumed to be known.

It is assumed that $f\colon\ R^n \rightarrow R^n$ is continuous and differentiable. The exact solution to this problem is only known for special choices of the function $f[y(t), t]$. In general, however, the right-hand side (RHS) of eqn.(3.2.1-1) is so complex that only approximate solutions may be found.

At the present time, many numerical procedures have been proposed to solve initial-value problems. Some of these methods include: Euler's method, Runge-Kutta methods, and predictor-corrector methods [22].

AD-A107 914   RENSSELAER POLYTECHNIC INST TROY NY DEPT OF ELECTRIC--ETC   F/G 9/2
PARALLEL COMPUTATION FOR DEVELOPING NONLINEAR CONTROL PROCEDURE--ETC(U)
JUL 81   H KAUFMAN, R TRAVASSOS   AFOSR-77-3418

UNCLASSIFIED   AFWAL-TR-81-3016   NL

2 OF 3
AD
A107914

Currently, the fourth-order Adam's predictor corrector and the Runge-Kutta-Fehlberg methods are among the most efficient procedures for solving initial-value problems [39]. These methods, however, are sequential in nature and as such are not suitable for parallel computers.

Although many parallel computers exist at the present time, only two parallel methods for solving initial-value problems currently exist. These methods, due to Downs [37] and Miranker & Liniger [38], surprisingly were developed nearly a decade ago. Apparently, this area of research may be reconsidered in the near future, but, for now, let us discuss Downs' method.

Downs' Method

One of the first parallel methods for numerically solving an initial-value problem was reported by Downs in reference [37]. This method was originally designed for use on the Illiac IV although it can be executed on any parallel computer with N processors which are capable of operating simultaneously.

To begin our discussion of this method, let $\pi_N = [t_0, t_1, \ldots, t_{N-1}, t_f]$ be a time partition of the interval $[t_0, t_f]$. Associated with π_N is a sequence of functions which will be denoted by

$$Y \triangleq (y'(t), y^2(t), \ldots, y^{N+1}(t))$$

Basically, the approach taken by Downs is to construct a sequence $(y^k(t))_{k=1}^{N+1}$ in a recursive manner such that in the limit, the sequence approaches the exact solution of the initial-value problem under consideration.

In reference [37], Downs gives two methods for computing the recursion on a parallel computer. The first method is based upon computing a partial sum in 2 $\log_2 N$ steps. Downs indicates that although this method is quite efficient on a parallel processor, such as the Illiac IV, convergence can be slow. The second method proposed by Downs requires more complicated computations but usually leads to much better convergence. This technique is based on a first-order Taylor series which also requires only 2 $\log_2 N$ steps to execute. Downs shows that his procedure converges linearly to the exact solution of an initial-value problem provided that the initial approximation to the solution is sufficiently good.

The major problem with Downs' method is that the number of processors needed to implement his procedures may indeed become prohibitive. This is especially true if the RHS of the initial-value problem is highly nonlinear since, in this case, the number of partition points (or processors) associated with the time partition π_N must be relatively large to ensure accuracy. This, along with the fact that Downs does not present an example illustrating the performance of his procedure, may cause one to be reluctant to use his method.

Miranker and Liniger's Method

Miranker and Liniger's class of parallel predictor-corrector integration methods is based upon decoupling the predictor-corrector equations such that the calculations required by the predictor and corrector can be performed simultaneously on separate processors [38]. This may be achieved by forcing the corrector to lag the predictor by one time step. In fact, Miranker and Liniger have shown that if

$$t_i = t_0 + ih \qquad i = 0, 1, 2, \ldots$$

where h is an integration step size parameter and y_i, y_i^p, y_i^c, f_i^p, and f_i^c are denoted as the value of $y(t_i)$, the predicted value of $y(t_i)$, the corrected value of $y(t_i)$, the value of $f[y^p_i, t_i]$, and the value of $f[y^c_i, t_i]$ respectively, then the following predictor-corrector pairs may be derived:

Parallel Trapezoidal Rule:

$$y^p_{i+1} = y^c_{i-1} + 2h\, f^p_i \tag{3.2.1-2a}$$

$$y^c_i = y^c_{i-1} + (h/2)\,(f^p_i + f^c_{i-1}) \tag{3.2.1-2b}$$

Parallel Adams-Moulton (3rd order):

$$y^p_{i+1} = y^c_{i-1} + (h/3)\,(7\, f^p_i - 2\, f^c_{i-1} + f^c_{i-2}) \tag{3.2.1-3a}$$

$$y^c_i = y^c_{i-1} + (h/12)\,(5\, f^p_i + 8\, f^c_{i-1} - f^c_{i-2}) \tag{3.2.1-3b}$$

Parallel Adams-Moulton (4th order):

$$y^p_{i+1} = y^c_{i-1} + (h/3)\,(8\, f^p_i - 5\, f^c_{i-1} + 4\, f^c_{i-2} - f^c_{i-3}) \tag{3.2.1-4a}$$

$$y^c_i = y^c_{i-1} + (h/24)\,(9\, f^p_i + 19\, f^c_{i-1} - 5\, f^c_{i-2} + f^c_{i-3}) \tag{3.2.1-4b}$$

It is clear from the structure of the parallel predictor-corrector pairs above that the predictor and corrector equations may be evaluated at the same time if two processors are available. Also, note that the computation time may be reduced by a factor of two if one of these methods were used rather than a conventional (serial) predictor-corrector method.

Miranker and Liniger extend this idea of parallel operation on two processors to parallel operation on any even number of processors. They also analyze the stability and convergence of their class of methods by studying the root condition and local truncation error associated with the theory of classical multistep methods.

Since the integration step size, h, is fixed for all time in the parallel methods above, the accuracy and efficiency of these procedures may be greatly influenced by the choice of h. For example, if the step size is too large, then the resultant solution will be inaccurate. On the other hand, if the step size is too small, then the efficiency of the algorithm is reduced since too many integration steps would be taken. Thus, the problem of step size selection is crucial to the successful application of this method. This observation has led to a modification of the basic method which is the topic of the next section.

3.2.2 PPC Integration With Variable Step Size

In the previous section, a number of parallel predictor-corrector (PPC) methods for integrating ordinary differential equations (ode's) due to Miranker and Liniger were presented. The primary advantage of using these methods over other procedures is the speed-up of computation. Although the computations required by a PPC method may be done extremely rapidly on separate processors, the accuracy of the solution may suffer if the step size parameter, h, is not chosen properly. Thus, it seems appropriate to modify Miranker and Liniger's methods such that a prescribed level of accuracy can be maintained while keeping the parallel feature of these methods.

This modification might be realized by using the predictor and corrector values at the same time step to estimate the local truncation error and use this quantity to vary the step size to achieve a prescribed accuracy. However, since the corrector lags the predictor by one time step in the PPC pairs described in the previous section, one might suspect problems with this approach. Indeed, this is true and has been verified through simulation.

In view of the above, it appears that a better method might be to use the predictor at step i+1 and the corrector at step i to estimate the local truncation error at step i which could be used to automatically vary the step size.

In the remainder of this section, the ideas discussed above will be used to extend Miranker and Liniger's class of parallel predictor-corrector methods such that a desired level of accuracy is maintained. Although each member of Miranker and Liniger's class of PPC methods can be extended, the basic procedure will be demonstrated for the parallel 4th order Adams-Moulton method given by eqns.(3.2.1-4a) and (3.2.1-4b).

It is well known that the local truncation error associated with the Adams-Moulton corrector (eqn. 3.2.1-4b) is given by [19]:

$$d_{i,c} \triangleq y(t_i) - y_i^c = \frac{-19}{720} h^5 y^{(5)}(\zeta_i) \qquad \zeta_i \varepsilon [t_o, t_f] \tag{3.2.1-5}$$

Due to the form of eqn.(3.2.1-4a), we must estimate the local truncation error of the predictor. Since eqn.(3.2.1-4a) is accurate to $O(h^4)$, we will assume an exact solution of the form:

$$y(t) = t^5 \qquad t \geq t_0 \tag{3.2.1-6}$$

so that $y' = f(y, t) = 5t^4 \qquad \forall\, t \geq t_0.$ (3.2.1-7)

By definition, the local truncation error associated with the predictor is given by:

$$d_{i+1,p} \triangleq y(t_{i+1}) - y_{i+1}^p \tag{3.2.1-8}$$

Substituting eqns.(3.2.1-6) and (3.2.1-7) into (3.2.1-4a) and evaluating (3.2.1-8) at $t_{i-3} = 0$, $t_{i-2} = h$, $t_{i-1} = 2h$, $t_i = 3h$, $t_{i+1} = 4h$, the desired result is obtained:

$$d_{i+1,p} = \frac{232}{720} h^5 y^{(5)} (\zeta_2) \qquad \zeta_2 \varepsilon [t_0, t_f] \qquad (3.2.1\text{-}9)$$

Note that although the estimates of the local truncation errors given by eqn.(3.2.1-5) and (3.2.1-9) are mathematically correct, they are of little use numerically. This is due to the fact that explicit calculation of $y^{(5)}(\zeta)$ is clearly problem dependent. To circumvent such problems, we need a better method of estimating the local truncation error.

To obtain such a method, we proceed by subtracting eqn. (3.2.1-5) from (3.2.1-9) assuming $\zeta_1 \simeq \zeta_2 = \zeta$, to get:

$$y(t_{i+1}) - y^p_{i+1} + (y^c_i - y(t_i)) = \frac{251}{720} h^5 y^{(5)} (\zeta) \qquad (3.2.1\text{-}10)$$

Using this result and eqn.(3.2.1-5), it is easy to show that:

$$d_{i,c} = \frac{-19}{251} [y^c_i - y^p_{i+1} + \delta_y] \qquad (3.2.1\text{-}11)$$

where

$$\delta_y \triangleq y(t_{i+1}) - y(t_i)$$

Using the Schwartz and triangle inequalities [36] on eqn. (3.2.1-11), we obtain the upper bound on the local truncation error

$$| d_{i,c} | \leq \frac{19}{251} \{|y^c_i - y^p_{i+1}| + |\delta_y|\} \qquad (3.2.1\text{-}12)$$

This result is particularly interesting because it indicates that $|d_{i,c}|$ is directly proportional to $|\delta_y|$. This implies that to maintain a small local error at the corrector, the step size cannot be too large since to predict too far in advance may cause $|\delta_y|$ to be large. Although this result is clearly what is needed, eqn.(3.2.1-12) is not very useful as is because the $|\delta_y|$ is unknown. To overcome this difficulty, we may write a Taylor series approximation for y(t) evaluated at $t = t_{i+1}$ as follows:

$$y(t_{i+1}) = y(t_i) + h_i\ f\,[y(t_i),\ t_i\,] + O(h_i{}^2)$$

where

$$h_i \triangleq t_{i+1} - t_i > 0.$$

Thus, we have

$$|\delta_y| \le h_i\ |f\ [y(t_i),\ t\ ]|\ +\ |O(h_i^2)| \qquad (3.2.1\text{-}13)$$

If we substitute eqn (3.2.1-13) into eqn.(3.2.1-12), we have a means of estimating the local truncation error of the corrector numerically as follows:

$$|d_{i,c}| \le \frac{19}{251}\ \{|y_i^c - y_{i+1}^p| + h_i\ |f(y_i^c,\ t_i)|\} \qquad (3.2.1\text{-}14)$$

Finally, this result can be utilized to automatically vary the step size, h_i, until a desired accuracy is obtained. More specifically, this may be achieved by performing the following steps:

Step 1: Simultaneously evaluate the predictor and corrector eqns.

$$y_{i+1}^p = y_{i-1}^c + \frac{h_i}{3}\,(8f_i^p - 5f_{i-1}^c + 4f_{i-2}^c - f_{i-3}^c) \qquad (3.2.1\text{-}15a)$$

$$y_i^c = y_{i-1}^c + \frac{h_i}{24} + (9f_i^p + 19f_{i-1}^c - 5f_{i-2}^c + f_{i-3}^c) \qquad (3.2.1\text{-}15b)$$

Step 2: Estimate the local truncation error:

$$d_{i,c} = \frac{19}{251}\,\{|y_i^c - y_{i+1}^p| + h_i\ |f_i^c|\}$$

Step 3:

a. If $\varepsilon_{min} \le d_{i,c} \le \varepsilon_{max}$, the step is accepted so set $i \leftarrow i+1$ and go to Step 1.

b. If $d_i > \varepsilon_{max}$, then the local truncation error is too large. Therefore, replace $h_i \leftarrow h_i/2$, restart method and go to Step 1.

c. If $d_i < \varepsilon_{min}$, the solution is more accurate than desired. Therefore, replace $h_i \leftarrow 2h_i$, restart method, set $i \leftarrow i+1$, and go to Step 2.

The sequence of computations required by eqns.(3.2.1-15a) and (3.2.1-15b) is illustrated in Figure 3.5 below.

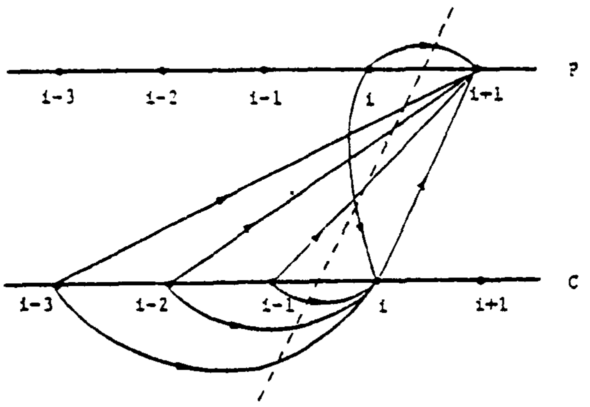


Figure 3.5 - The Sequence of Computations of a PPC Integration Procedure

The upper line represents the progress of the computation at the mesh points for the predictor while the lower line shows the progress of the corrector. The dashed line is referred to as a computation front. The arrows in Figure 3.5 indicate that the computations at the mesh points ahead of the computation front only depend on information behind the front which is characteristic of a parallel integration algorithm. The method can be implemented by simultaneously evaluating the following quantities in separate processors:

$$. . . . \rightarrow y^p_{i+1} \rightarrow f^p_{i+1} \rightarrow . . . .$$

$$. . . . \rightarrow y^c_i \rightarrow f^c_i \rightarrow . . . .$$

Before leaving this section, a few words should be said about starting the parallel variable step size integration scheme. Note that to start this method the following quantities must be available to the parallel processors:

$$y_0^c,\ f_0^c,\ y_1^c,\ f_1^c,\ y_2^c,\ f_2^c,\ y_3^p \text{ and } f_3^p$$

These values may be obtained by taking four forward integration steps of a standard Runge-Kutta (RK) integration method since RK methods are self-starting. The values of $y(t)$ and $f[y(t), t]$ computed over the first three time steps may be used by the corrector while the value of $y(t)$ and $f[y(t), t]$ at the fourth time step can be used to initialize the predictor. At this point, enough information is available to begin processing in parallel using the parallel variable step size integration scheme.

3.2.3 Comparison of Methods

In order to determine the effectiveness of the parallel integration procedures discussed in the previous sections, these methods were used to find the solution of the forced Van der Pol equation [37]:

$$\ddot{x}(t) + a(t)\,(1 - x^2(t))\,\dot{x}(t) - x(t) + u(t) = 0 \qquad (3.2.3\text{-}1)$$

where

a(t) is a parameter which defines a particular systems dynamics and u(t) is a forcing function or control.

To solve this problem by the parallel integration procedures previously discussed, we must write eqn (3.2.3-1) as a system of first-order differential equations. If we let $x_1(t) = x(t)$ and $x_2(t) = \dot{x}(t)$, then eqn (3.2.3-1) may be written as:

$$\dot{x}_1(t) = x_2(t) \qquad (3.2.3\text{-}2a)$$

$$\dot{x}_2(t) = a(t)\,(1 - x_1^2(t))x_2(t) - x_1(t) + u(t) \qquad (3.2.3\text{-}2b)$$

In the simulations, the control was selected as:

$$u(t) = \sin \frac{1}{\sqrt{2}}\, t \qquad t \geq 0$$

since this input will cause x(t) to be uniformly asymptotically stable [40]. Also, in the simulations a(t) was set to zero and the initial conditions were chosen as:

$$x_{10} = 1 \text{ and } x_{20} = 1 + \sqrt{2} \; .$$

In this case, the exact solution of eqn (3.2.3-2) is given by:

$$\begin{aligned} x_1(t) &= \cos t + \sin t + 2 \sin \frac{1}{\sqrt{2}} t \\ & \qquad\qquad\qquad\qquad\qquad t \geq 0 \\ x_2(t) &= \cos t - \sin t + \frac{2}{\sqrt{2}} \cos \frac{1}{\sqrt{2}} t \end{aligned}$$

At this point, Miranker and Liniger's fourth-order parallel integration method (PPC42) and the parallel variable step integration procedure (PPC42V) were used to obtain a numerical solution to Van der Pol's equation when $a(t) \equiv 0$. Let us denote the computed solution as $\hat{x}(t)$ and the exact solution as x(t).

In Tables 3.6 and 3.7, x(t), $\hat{x}(t)$, and the error $x(t) - \hat{x}(t)$ are shown over a five second interval. The results indicate

- By using the PPC42 procedure with a fixed step size of h = 5./200 = 0.025, the computed solution is accurate to about 6 digits (see Table 3.6) which substantiates the claim that the PPC42 procedure is accurate to $O(h^4)$.
- The PPC42V integration procedure can indeed vary the step size to meet a 5-6 digit accuracy requirement imposed by the user (see Table 3.7). To obtain the computed solution shown in Table 3.7, the PPC42V procedure only took 140 integration steps while the PPC42 procedure required 200 integration steps.

The second example which was considered is when $a(t) \equiv 1$ $\forall\, t \in [0, 5]$. In this case, an analytical solution to the Van der Pol

Table 3.6: PPC42 Solution of Van der Pol's Equation when
$a(t) \equiv 0 \quad \forall\, t \in [0, 5]$

t	$x_1(t)$	$\hat{x}_1(t)$	$x_1(t) - \hat{x}_1(t)$
0.0	0.1000000000E+01	0.1000000000E+01	-0.0
0.5000000000E+00	0.2049475287E+01	0.2049474716E+01	0.5707559656E-06
0.1000000000E+01	0.2681047167E+01	0.2681046486E+01	0.6811311495E-06
0.1500000000E+01	0.2813588250E+01	0.2813588142E+01	0.1076905718E-06
0.2000000000E+01	0.2468682482E+01	0.2468681335E+01	0.1146074653E-05
0.2500000000E+01	0.1758656377E+01	0.1758655548E+01	0.8286384690E-06
0.3000000000E+01	0.8556285315E+01	0.8556280136E+00	0.5178779003E-06
0.3500000000E+01	-0.5041817443E-01	-0.5041858926E-01	0.4148348849E-06
0.4000000000E+01	-0.7943026224E+00	-0.7943028212E+00	0.1987446135E-06
0.4500000000E+01	-0.1269079672E+01	-0.1269079208E+01	-0.4633928843E-06
0.5000000000E+01	-0.1442923584E+01	-0.1442922592E+01	-0.9918124615E-06

t	$x_2(t)$	$\hat{x}_2(t)$	$x_2(t) - \hat{x}_2(t)$
0.0	0.2414213560E+01	0.2414213181E+01	0.3794580077E-06
0.5000000000E+00	0.1724899120E+01	0.1724898338E+01	0.7819634014E-06
0.1000000000E+01	0.7739795403E+00	0.7739793658E+00	0.1744257939E-06
0.1500000000E+01	-0.2362028580E+00	-0.2362030745E+00	0.2164758168E-06
0.2000000000E+01	-0.1104906572E+01	-0.1104906082E+01	-0.4901708046E-06
0.2500000000E+01	-0.1676376551E+01	-0.1676376343E+01	-0.2084020718E-06
0.3000000000E+01	-0.1870935547E+01	-0.1870935440E+01	-0.1073054323E-06
0.3500000000E+01	-0.1697040180E+01	-0.1697039604E+01	-0.5756756729E-06
0.4000000000E+01	-0.1242271760E+01	-0.1242270470E+01	-0.1290101930E-05
0.4500000000E+01	-0.6463259809E+00	-0.6463252306E+00	-0.7402529024E-06
0.5000000000E+01	-0.6330324004E-01	-0.6330257654E-01	-0.6634964014E-06

Table 3.7: PPC42V Solution of Van der Pol's Equation when

$a(t) \equiv 0 \qquad \forall\ t \in [0, 5]$

t	$x_1(t)$	$\hat{x}_1(t)$	$x_1(t) - \hat{x}_1(t)$
0.0	0.1000000000E+01	0.1000000000E+01	-0.0
0.5000000000E+00	0.2049475287E+01	0.2049475670E+01	-0.3829183508E-06
0.1000000000E+01	0.2681047167E+01	0.2681047440E+01	-0.2725431669E-06
0.1500000000E+01	0.2813588250E+01	0.2813587189E+01	0.1061364888E-05
0.2000000000E+01	0.2468682482E+01	0.2468680382E+01	0.2099748969E-05
0.2500000000E+01	0.1758656377E+01	0.1758652687E+01	0.3689661418E-05
0.3000000000E+01	0.8556285315E+00	0.8556241393E+00	0.4392179811E-05
0.3500000000E+01	-0.5041817443E-01	-0.5042218789E-01	0.4013465313E-05
0.4000000000E+01	-0.7943026224E+00	-0.7943050265E+00	0.2404116470E-05
0.4500000000E+01	-0.1269079672E+01	-0.1269079208E+01	-0.4633928843E-06
0.5000000000E+01	-0.1442923584E+01	-0.1442920685E+01	-0.2899161094E-05

t	$x_2(t)$	$\hat{x}_2(t)$	$x_2(t) - \hat{x}_2(t)$
0.0	0.2414213560E+01	0.2414213181E+01	0.3794580077E-06
0.5000000000E+00	0.1724899120E+01	0.1724898338E+01	0.7819634014E-06
0.1000000000E+01	0.7739795403E+00	0.7739778757E+00	0.1664541913E-05
0.1500000000E+01	-0.2362028580E+00	-0.2362051606E+00	0.2302638384E-05
0.2000000000E+01	-0.1104906572E+01	-0.1104907990E+01	0.1417177828E-05
0.2500000000E+01	-0.1676376551E+01	-0.1676377296E+01	0.7452722446E-06
0.3000000000E+01	-0.1870935547E+01	-0.1870934486E+01	-0.1060979749E-05
0.3500000000E+01	-0.1697040180E+01	-0.1697036743E+01	-0.3436698622E-05
0.4000000000E+01	-0.1242271760E+01	-0.1242266655E+01	-0.5104799196E-05
0.4500000000E+01	-0.6463259809E+00	-0.6463202834E+00	-0.5697438419E-05
0.5000000000E+01	-0.6330324004E-01	-0.6329768896E-01	-0.5551077273E-05

equation is impossible to obtain. Nevertheless, an approximate solution can be obtained by employing the parallel integration methods.

For this example, the results indicate that the solution obtained using the PPC42 procedure (h = 0.005 and fixed) and the variable step PPC42V method agree to about 5 digits (see Tables 3.8 and 3.9).

It is interesting to count the number of integration steps required by each procedure. Clearly, for the fixed step size method 5/0.005 = 1000 integration steps were needed. For the PPC42V method, however, the number of integration steps needed depends largely on the behavior of the solution x(t). Note that in regions where x(t) varies rapidly many integration steps were required while relatively few steps were taken when x(t) was nearly constant (see Figure 3.6 and Table 3.9). But this is precisely what we would expect a good variable step size method to do.

TABLE 3.8: PPC42 Solution of Van der Pol's Equation
$a(t) \equiv 1 \qquad \forall\ t\ \varepsilon\ [0, 5]$

TIME (sec)	$x_1(t)$	$x_2(t)$
0.0	0.1000000000E+01	0.2414213181E+01
0.5000000000E+00	0.1858656883E+01	0.8029198647E+00
0.1000000000E+01	0.1956999779E+01	-0.1791329980E+00
0.1500000000E+01	0.1808356285E+01	-0.3609815836E+00
0.2000000000E+01	0.1620203972E+01	-0.3831909299E+00
0.2500000000E+01	0.1427042961E+01	-0.3920248151E+00
0.3000000000E+01	0.1222183228E+01	-0.4368848205E+00
0.3500000000E+01	0.9764278531E+00	-0.5651698112E+00
0.4000000000E+01	0.6299955845E+00	-0.8582303524E+00
0.4500000000E+01	0.6659322977E+01	-0.1457375526E+01
0.5000000000E+01	-0.8489770889E+00	-0.2094655037E+01

TABLE 3.9: PPC42V Solution of Van der Pol's Equation when $a(t) \equiv 1 \quad \forall\, t\ \varepsilon\ [0, 5]$

TIME (sec)	$x_1(t)$	$x_2(t)$	Number of Integration Steps
0.0	0.1000000000E+01	0.2414213181E+01	
0.5000000000E+00	0.1858656883E+01	0.8029198647E+00	183 t ε [0.0, 0.5]
0.1000000000E+01	0.1956999779E+01	-0.1791330576E+00	203 t ε [0.5, 1.0]
0.1500000000E+01	0.1808356285E+01	-0.3609815836E+00	53 t ε [1.0, 1.5]
0.2000000000E+01	0.1620203972E+01	-0.3831909895E+00	53 t ε [1.5, 2.0]
0.2500000000E+01	0.1427042961E+01	-0.3920248747E+00	53 t ε [2.0, 2.5]
0.3000000000E+01	0.1222183228E+01	-0.4368848205E+00	53 t ε [2.5, 3.0]
0.3500000000E+01	0.9764278531E+00	-0.5651698112E+00	53 t ε [3.0, 3.5]
0.4000000000E+01	0.6299955845E+00	-0.8582303524E+00	99 t ε [3.5, 4.0]
0.4500000000E+01	0.6659322977E-01	-0.1457375526E+01	71 t ε [4.0, 4.5]
0.5000000000E+01	-0.8489770889E+00	-0.2094654083E+01	203 t ε [4.5, 5.0]

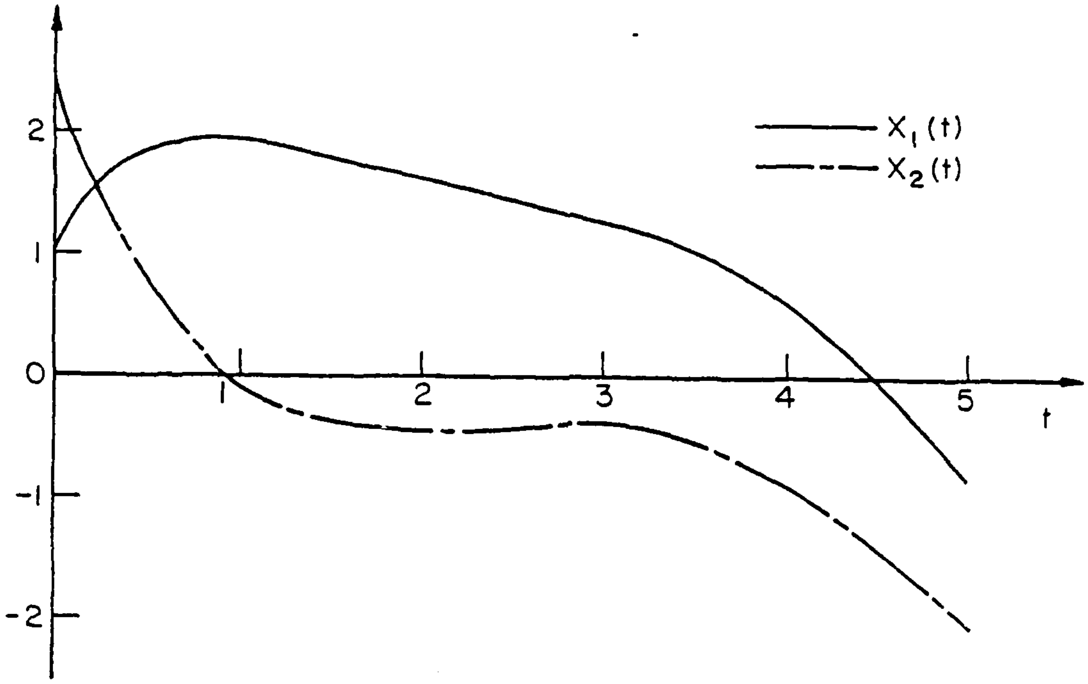


FIGURE 3.6: Solution of Van der Pol's Equation

CHAPTER FOUR

IMPLEMENTATION CONSIDERATIONS

Thus far, this thesis has primarily been concerned with the development of efficient numerical methods for solving nonlinear estimation and control problems which are suitable for modern parallel computers. In this chapter, the implementation of these methods is considered. In particular, a parallel computer architecture is proposed in Section 4.1 which utilizes three levels of parallelism to allow the implementation of the parallel algorithms discussed in Chapters Two and Three. In Section 4.2, the execution time of the parallel algorithms is estimated and compared with that of currently used sequential methods. Finally, the speedup due to parallelism is estimated using the timing equations given in Sections 4.2.3.

4.1 A Parallel Computer Architecture

Whereas most existing computer systems (parallel or serial) have been designed as general purpose machines, the parallel computer proposed in this section may be considered a special purpose machine for implementing the parallel algorithms discussed in Chapters Two and Three. The architecture of the proposed computer utilizes many independent processors capable of operating simultaneously such that more processing power would be possible than a single central processing unit with traditional architecture. Although there is no reason to believe that the architecture of the proposed parallel computer is an "optimal" implementation of the parallel algorithms described in Chapters Two and Three, it may be viewed as a "natural" implementation.

In view of the structure of the parallel algorithms developed in Chapter Two and Three and the availability of low-cost microprocessor systems (available on a single $8\frac{1}{2}$ x 11 inch card), the proposed parallel computer is organized into three levels of parallelism; namely:

- Level I (Minimization Level)
- Level II (Shooting Level, if applicable)
- Level III (Integration Level)

To reduce the possibility of coordination and synchronization problems with each level, the proposed computer should be synchronous and utilize a single-instruction-multiple-data (SIMD) stream to efficiently implement the parallel algorithms. Other considerations which should be of interest include:

- Timing requirements for real time control computations (Specifying processor add, multiply and transfer times such that the execution time is rapid enough to permit the nonlinear estimation and control computations to be done in real time.)
- Memory and peripheral requirements
- Effects of wordsize
- Communication and interconnection among processors
- Feasibility of implementation and, of course, cost.

With these considerations in mind, the organization of each level of the proposed parallel computer will now be described.

4.1.1 Minimization Level

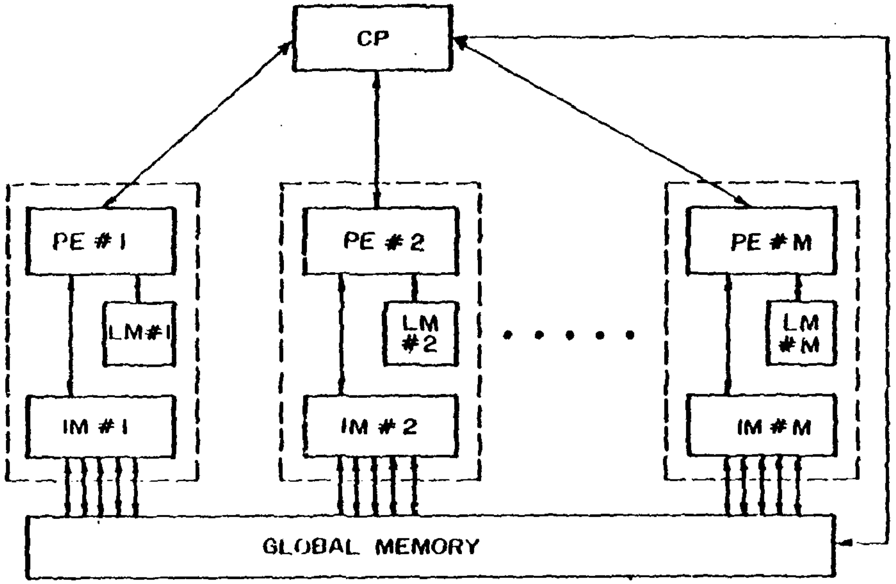
The minimization level (Level I) is the upper level of the architecture in which a finite-dimensional minimization problem, and a search for the appropriate unknowns (parameters or boundary conditions) is initiated. If M is the number of unknowns to be optimized, then the search for the unknowns can be performed simultaneously by M optimization modules. Each optimization module might consist of a processing element (PE), a local memory (LM) element, and an integration module (IM). Because we want the searches to be performed simultaneously, the structure at this level might be organized as shown in Figure 4.1. Notice that this parallel structure is ideally suited for evaluating a function and its gradient at M distinct points simultaneously which is precisely what is required to implement the parallel minimization algorithms discussed in Section 3.1.

At this level, the role of the central processor (CP) is to:

- Initialize each processor
- Control the operation of each processor
- Monitor the status of each processor
- Watch the clock and controls to keep the processors synchronized during a given iteration

As indicated above, the role of the optimization modules is to implement the minimization phase of the parallel algorithms discussed in Chapter Two. Because the mathematical computations required by the parallel minimization algorithm discussed in Section 3.1 are relatively sophisticated, the processors at this level should be also. In fact, the PE's should be pipelined so that the required vector-matrix operations can be performed very rapidly.

Since speed is a primary concern, cache memories might be applicable for the local memory units at this level.



CP : Central Processor
PE : Processing Element
LM : Local Memory
IM : Integration Module

FIGURE 4.1: Parallel Structure at Level I

As shown in Figure 4.1, the connection of adjacent processors is not necessary since, by design, the parallel algorithms of Chapters Two and Three allow nearly all computations to be performed independently of the others. Note that this indicates that relatively little (if any) communication is required among the processors at this level.

Finally, because of the large dynamic range of the computations required by quasi-Newton methods, the wordlength required should be relatively long.

4.1.2 Shooting Level

If parallel shooting is used to aid the search for the unknown boundary conditions, then the mission time interval $[t_o, t_f]$ would be divided into N subintervals using the partition

$$t_o < t_1 < \ldots . < t_N = t_f$$

The task of the processors at this level is to implement the parallel shooting phase of the parallel algorithms discussed in Sections 2.1 and 2.2 by finding the solution to the appropriate initial-value problem over each subinterval simultaneously in parallel. This phase of the algorithm might be implemented using the parallel structure shown in Figure 4.2.

At the shooting level (Level II), the role of each PE is to:

- Initialize the integration module
- Monitor the status of each integration module
- Communicate with the processors at Level I and Level II to keep computations synchronized.

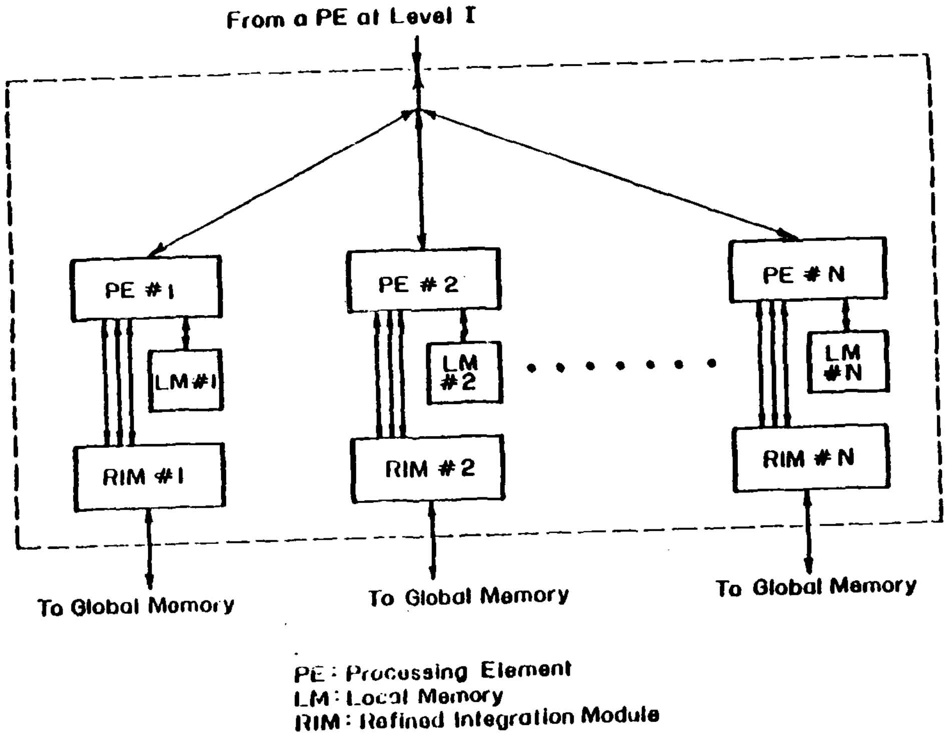


FIGURE 4.2: Parallel Structure at Level II

Since the processors at this level are mainly used for "bookkeeping" and "status checks," the PE's need not be very sophisticated. Also, the local memory units shown in Figure 4.2 might be relatively small in view of the primitive operations performed at this level. Note that if parallel shooting is not used, then this level is not necessary. In this case, the integration module required at Level I simply consists of a refined integration module which is discussed next.

4.1.3 Integration Level

At the integration level (Level III), the processors are dedicated to the task of integrating ordinary differential equation (ode's) over a subinterval using a parallel integration scheme such as the methods presented in Section 3.2.1. These methods are suggested for the numerical solution of the initial-value problems (IVP's) over each subinterval since computations can be sped up significantly by utilizing more than one processor operating in parallel on each ode. To further speed computations, a parallel integration method could be used to integrate each component of the right-hand side (RHS) of an IVP. If "L" processors are available for integrating each component of the RHS and the IVP is n^{th} order, then this phase of the parallel algorithm might be implemented as shown in Figure 4.3. Note that when L = 2 the structure shown in Figure 4.3 is ideally suited to implement the parallel predictor-corrector pairs presented in Section 3.2.

With regard to processor and memory requirements, the processors at this level must be somewhat sophisticated, due to the

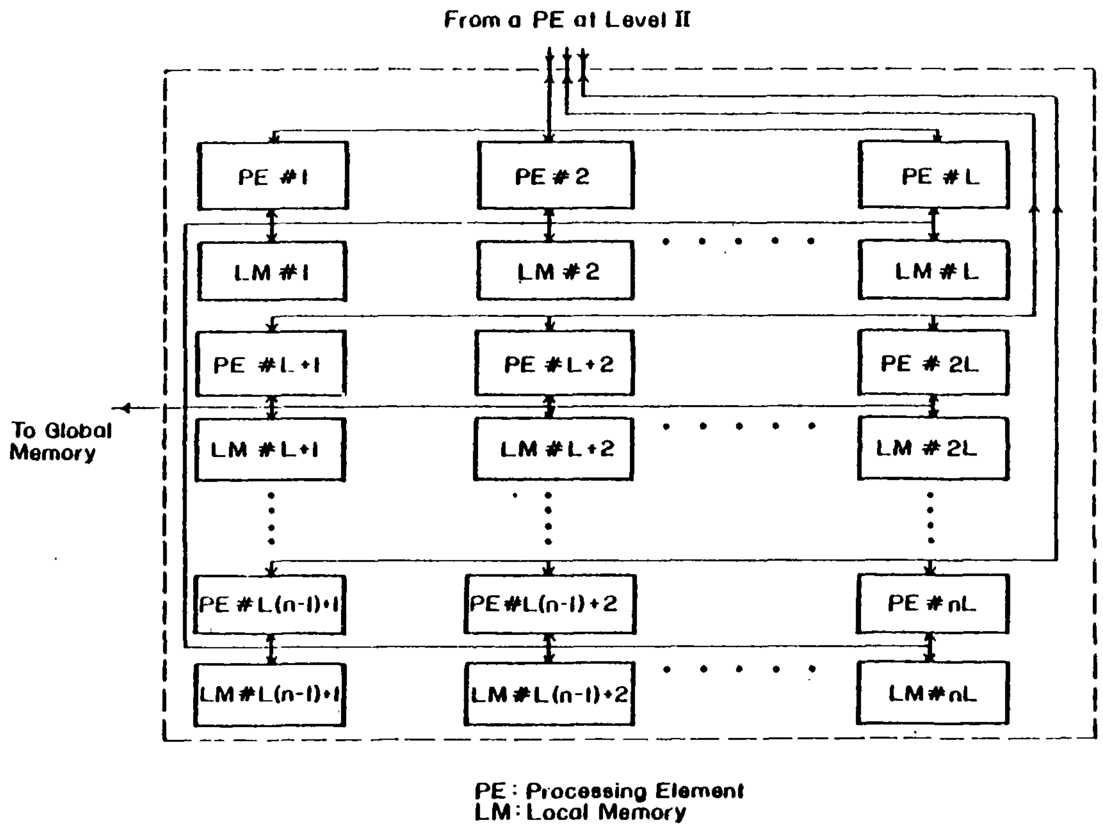


FIGURE 4.3: Parallel Structure at Level III

mathematical computations involved when evaluating the RHS of an IVP. The processors, however, need not be pipelined because the computations required by the parallel integration procedures do not warrant it. Nevertheless, the processor add and multiply time should be as small as possible since the integration phase generally is the most time consuming phase of the parallel algorithms discussed in Chapter Two. Again, since speed is crucial here, cache memories should be employed at this level.

Because the numerical solution of an IVP involves knowing the solution at many points, the solution stored in the local memories should be accessible to all processors. Generally, in situations like this, two or more processors may attempt to access the same memory module during a memory cycle. This phenomenon is called "memory contention" and is usually rectified by providing the system with a "memory lock."

The function of the memory lock is to preclude access by other processors once a processor has initiated a memory access. Since only one access can be made per memory cycle, one of the requests must wait. For the system to be efficient, however, the wait time should be no more than one or two memory cycles.

4.1.4 Coordination of Each Level

Now that the structure and function of each level has been discussed, the operation of the entire system will be briefly described.

Basically, the parallel processing would begin at Level I and proceed to Levels II and III as follows. At Level I, the central processor would initialize the PE's with an initial approximation of the state and costate variables at the partition points of the parallel shooting algorithm. This information, along with the initial and final times associated with each subinterval, would be transferred to Level II where the parallel shooting phase of the algorithm would be initiated. When a function and/or gradient evaluation is required, the processors at Level III would be activated. At Level III, each processor would simultaneously integrate its assigned initial-value problem over its assigned subinterval and use its local memory as temporary storage for intermediate results. When the integration phase is complete, the results would be transferred to global memory which then would be accessed by the central processor for the values needed to evaluate an appropriate error function. Finally, the central processor would evaluate the error function and decide whether to continue computations or halt.

Although it appears that, while one group of processors are busy at a given level, the remaining processors are idle, this is not the case because the idle processors are really performing status checks and other utility functions.

4.2 Parallel Algorithm Execution Time

The goal of this section is to analytically determine a set of timing equations which can be used to estimate the execution time of the parallel algorithms discussed in Chapters Two and Three. The timing equations are also used to compare the execution times of different algorithms, as well as to estimate the speed-up due to parallelism.

One way to estimate the execution time of the parallel algorithms would be to compute the time required per iteration by taking into account that many arithmetic operations would be performed in parallel. This can be done by counting the total number of additions, multiplications, function evaluations, and gradient evaluations for a given iteration and multiplying the operation count by representative execution times for these operations. By adopting this approach, one may estimate the time required for convergence by using a serial computer (such as an IBM 360) to determine the number of iterations needed and multiply this by the estimated time per iteration. Note that this estimate, however, would be problem dependent and that communication time between processors and memory is ignored.

The assumption that processor-memory communication time can be ignored is realistic since for many nonlinear estimation and control problems the mathematical computations performed by each processor would be significantly more time consuming than memory access time.

The execution time of the parallel algorithms will be estimated by deriving a set of timing equations for the minimization phase and integration phase separately. At the end of this section, the timing equations are combined to provide an estimate of the execution time of the entire parallel algorithm. At this time, it should be pointed out that the timing equations given in this section are derived assuming the parallel algorithms are executed on a parallel computer whose architecture is consistent with that discussed in Section 4.1.

4.2.1 Minimization Phase

In this section, the total number of arithmetic operations required by the CM, PVM, PDFP and PBFS procedures discussed in Section 3.1 are counted assuming many of these operations can be performed in parallel. To form a basis for comparison, an operation count is also given for the serial ZP, DFP, and BFS procedures assuming these algorithms are executed sequentially. To derive the operation count for each of these methods at this time would be very time consuming and repetitious. Therefore, only the operation count for the PQN algorithm will be derived at this time.

Since step 1 of the PQN algorithm can be considered an initialization step, one iteration of this method essentially consists of steps 2, 3, 4 and 5. Therefore, the operation count will only include the arithmetic operations required to perform steps 2 - 5. The operation count will be derived by counting the arithmetic operations required by each step, one at a time, and combined at the end to obtain an overall operation count.

Starting with step 2 then, observe that a sequence of linear systems of equations must be solved during this step. To solve each linear system as rapidly as possible, any one of the following algorithms discussed in references [14], [15], and [41] may be employed. Among these methods, the procedure reported by Pease [41] is preferable since it only requires n processors and, as such, could be implemented on the parallel computer proposed in Section 4.1.

Basically, to solve a general linear system of the form $ax = b$ where $a \varepsilon R^{nxn}$ and $x,b \varepsilon R^{nx1}$ using Pease's algorithm, we must augment the "a" matrix with the vector "b" by placing "b" in the n+1 column as follows:

$$A = [a \vdots b]$$

and solve $x = a^{-1}b$ by performing the following operations in parallel on the rows of A.

```
for j = 1 step 1 until n do
     begin $t_i \leftarrow a_{ij}/a_{jj}$   $i=1,2,\ldots,n$   $i \neq j$
          for k=j+1 step 1 until n+1 do
               $a_{ik} \leftarrow a_{ik} - t_i a_{jk}$   $i=1,2,\ldots,n$   $i \neq j$
     end;
$x_i \leftarrow a_{i,n+1}/a_{ii}$   $i=1,2,\ldots,n$
```

If the arithmetic operations required by Pease's algorithm are counted assuming they are done in parallel, then only $n(n+1)/2$ additions and $n(n+3)/2$ multiplications are needed by this method. Note that this is significantly faster than a serial Gaussian-Elimination procedure which required $O(n^3)$ multiplications and additions. In summary, if Pease's algorithm is used to solve the linear systems required in step 2 of the PQN method, then $n(n+1)/2$ additions and $n(n+3)/2$ multiplication must be performed for each system of equations. Note that the linear systems in step 2 are increasing in size and step 2 must be executed n-1 times. Therefore, the total number of operations which must be performed during step 2 is given by:

$$\sum_{i=1}^{n-1}\{\frac{i(i+1)}{2}A + \frac{i(i+3)}{2}M\}$$

$$= \frac{1}{2}\sum_{i=1}^{n-1}\{i^2(A+M) + i(A+3M\}$$

$$= \frac{A+M}{2}\{\frac{n(2n^2-3n+1)}{6}\} + \frac{A+3M}{2}\{\frac{n(n-1)}{2}\}$$

$$= \frac{n(n^2-1)}{6}A + \frac{n(n^2+3n-4)}{6}M$$

where A and M represent the addition and multiplication operations respectively.

In step 3a. of the PQN algorithm n+1 gradients must be evaluated but if n+1 processors are available to perform the gradient evaluations (ge's) simultaneously then equivalently only one ge is performed. Also, the n vector addition can be performed in parallel. Similarly, in step 3b, the n gradient differences may be done simultaneously. Hence, 2n additions and 1ge are required in step 3.

In step 4., the inverse Hessian, H, is updated using n rank-two corrections in which the vector-matrix operations can be performed in parallel. By straightforward evaluation, it can be shown that $(8n^2+11n)$ multiplications and $(7n^2+2n-2)$ additions must be performed in step 4 of the PQN algorithm assuming "n" processors are utilized for each vector-matrix operation and $\phi > 0$. If $\phi = 0$, then the update is somewhat simpler and as a result only $(4n^2+4n)$ multiplications and $(3n^2+n)$ additions would be needed.

In step 5, a single line search is required. If we assume "L" function evaluations are performed during the line search and

$f(x^{(\ell+1)})$ is evaluated, then L+1 function evaluations are performed during this step.

To obtain the overall operation count, the operation counts obtained for steps 2 - 5 are combined. Hence, the overall operation count for one iteration of the PQN algorithm is given by:

$$\begin{aligned}\text{PQN:}\quad & (7n^2+4n+n(n^2-1)/6-2)A \\ & + (8n^2+11n+n(n^2+3n-4)/6)M \\ & + (L+1)FE+1GE\end{aligned}$$

n denotes the dimensionality of the problem,

A denotes additions

M denotes multiplications

FE denotes function evaluations

GE denotes gradient evaluations,

and L denotes the number of function evaluations during a line search.

Using the operation count above, it is a simple matter to estimate the execution time for one iteration of the PQN algorithm. Specifically, if t_a, t_m, t_{fe}, and t_{ge} denote the processor add time, processor multiply time, the time required for one function evaluation, and the time required for one gradient evaluation respectively, then by multiplying the operation count above by these quantities the execution time for one iteration of the PQN may be obtained as follows:

$$\begin{aligned}\text{PQN:}\quad & (7n^2+4n+n(n^2-1)/6-2)At_a \\ & + (8n^2+11n+n(n^2+3n-4)/6)Mt_m \\ & +(L+1)FEt_{fe}+1GEt_{ge} \qquad (4.2.1\text{-}2)\end{aligned}$$

Recall that, when $\phi = 0$ ($\phi = 1$) in the PQN algorithm, the PDFP (PBFS) method is obtained. Observe if $\phi = 0$, then the update in step 4 is somewhat simpler and the time per iteration is reduced to:

$$\text{PDFP:}\quad t_{pi} = (2n^2 + 3n + n(n^2-1)/6)\ At_a + (4n^2 + 4n + n(n^2 + 3n - 4)/16)\ Mt_m + (L + 1)\ FEt_{fe} + 1\ GEt_{ge} \tag{4.2.1-3}$$

Also, note that the time for one iteration of the PBFS method is the same as the PQN algorithm since the PBFS method is only a special case of the PQN algorithm when $\phi > 0$.

Following a similar procedure as outlined above, the execution time of one iteration of the CM and PVM procedures can be shown to be:

$$\text{CM:}\quad t_{pi} = (5 + n)\ Mt_m + (3 + 2n)\ At_a + (L + 1)\ FEt_{fe} \tag{4.2.1-4}$$

$$\text{PVM:}\quad t_{pi} = (2n^2 + n[4 + 3(n-1)/2] - 1)\ At_a + (2n^2 + 3n[1 + (n-1)/2] + 2)\ Mt_m + (L + 1)\ FEt_{fe} + 1\ GEt_{ge} \tag{4.2.1-5}$$

As indicated earlier, it is useful to estimate the execution time for one iteration of the ZP,DFP and BFS procedures assuming all computations are done sequentially. By performing an operation count and multiplying it by representative execution times, it is straightforward to show that the execution time of one iteration of the serial methods are:

$$\text{ZP:}\quad t_{pi} = (n^2 + 3n + 2)\ Mt_m + (n^2 + 4n + 1)\ At_a + [(n + 2)\ L + 1]\ FEt_{fe} \qquad (4.2.1\text{-}6)$$

$$\text{DFP:}\quad t_{pi} = (6n^2 + 3n)\ Mt_m + (4n^2 + 2n - 2)\ At_a + (L + 1)\ FEt_{fe} + 1\ GEt_{ge} \qquad (4.2.1\text{-}7)$$

$$\text{BFS:}\quad t_{pi} = (7n^2 + 4n + 2)\ Mt_m + (4n^2 + 5n - 4)\ At_a + (L + 1)\ FEt_{fe} + 1\ GEt_{ge} \qquad (4.2.1\text{-}8)$$

Note that if the number of additions and multiplications performed during one function and gradient evaluation were known, then the execution time could be estimated as a function of processor add and multiply times. In the next section, t_{fe} and t_{ge} will be estimated in terms of t_a and t_m which will allow the speed-up due to parallelism to be estimated in Section 4.2.3.

4.2.2 Integration Phase

In the previous section, the execution times of the minimization phase of the parallel algorithms discussed in Chapter Two were estimated in terms processor add time, multiply time and the time required for a function and gradient evaluation. In the context of the parallel algorithms described in Chapter Two, a function (or gradient) evaluation consists of integrating a set of differential equations over an appropriate time interval and evaluating a suitably defined error function.

Thus, the function (or gradient) evaluation time depends primarily on the numerical integration method employed and the complexity of the differential equations being integrated. This will be made more precise by estimating the execution time required by the parallel predictor-corrector (PPC) method discussed in Section 3.2 and comparing it with the execution time required by a serial Adam's predictor-corrector (APC) method.

To begin, consider the PPC integration method defined by:

$$y_{i+1}^{p} = y_{i-1}^{c}+\frac{h}{3}(8f_{i}^{p}-5f_{i-1}^{c}+4f_{i-2}^{c}-f_{i-3}^{c}) \qquad (4.2.2\text{-}1)$$

$$y_{i}^{c} = y_{i-1}^{c}+\frac{h}{24}(9f_{i}^{p}+19f_{i-1}^{c}-5f_{i-2}^{c}+f_{i-c}^{c}) \qquad (4.2.2\text{-}2)$$

As described in Section 3.2, the predictor and corrector equation above may be evaluated simultaneously on two separate processors. Also, the right-hand-side (RHS) of the initial-value problem (IVP) can be evaluated simultaneously since the sequence of computations is

$$\text{PE\#1:} \; . \; . \; . \; . \rightarrow y_{i+1}^{p} \rightarrow f_{i+1}^{p} \rightarrow . \; . \; . \; .$$

$$\text{PE\#2:} \; . \; . \; . \; . \rightarrow y_{i}^{c} \rightarrow f_{i}^{c} \rightarrow . \; . \; . \; .$$

By performing the computations in this manner, effectively only one RHS evaluation is performed at each step. Also, when evaluating eqn. (4.2.2-1) and (4.2.2-2) four additions and five multiplications must be performed. Therefore, the total number of arithmetic operations which must be carried out per step is:

$$4A+5M+1RHS \qquad (4.2.2\text{-}3)$$

Now suppose N steps are taken to solve an IVP over some interval of time. Then the time for one function evaluation may be estimated using the following expression:

$$t_{fe} = 4NAt_{a}+5NMt_{m}+1NRHSt_{rhs} \qquad (4.2.2\text{-}4)$$

where t_{rhs} is the time required to perform one RHS evaluation. Finally, if there are A_{rhs} additions and M_{rhs} multiplications when evaluating the RHS of an IVP, then t_{fe} may be estimated as follows:

$$t_{fe} = (4+1A_{rhs})NAt_a+(5+1M_{rhs})NMt_m \qquad (4.2.2\text{-}5)$$

Note that, if the function being evaluated is a function of n variables, then the gradient of such a function has n components. If we assume that each component of the gradient requires approximately the same time to evaluate as the function itself, then the time required to evaluate the gradient is approximately:

$$t_{ge} = nt_{fe} \qquad (4.2.2\text{-}6)$$

To form a basis for comparison, it is instructive to count the total number of arithmetic operations which must be performed by the serial Adam's predictor-corrector (APC) pair:

(4.2.2-7)

$$y^p_{i+1} = y^c_i + \frac{h}{24}(55\ f^p_i - 59\ f^c_{i-1} + 37\ f^c_{i-2} - 9\ f^c_{i-3})$$

$$y^c_{i+1} = y^c_i + \frac{h}{24}(9f^p_{i+1} + 19\ f^c_i - 5\ f^c_{i-1} + f^c_{i-2}) \qquad (4.2.2\text{-}8)$$

From eqns. (4.2.2-1) and (4.2.2-8), it is straightforward to show that 8A + 11M + 2RHS operations must be carried out sequentially to implement the serial APC pair.

As before, if A_{rhs} additions and M_{rhs} multiplication are performed during a RHS evaluation, then the time required for one function and gradient evaluation is simply:

$$t_{fe} = (8+2A_{rhs})NAt_a+(11+2M_{rhs})NMt_m \qquad (4.2.2\text{-}9)$$

and

$$t_{ge} = nt_{fe} \qquad (4.2.2\text{-}10)$$

Note that two RHS evaluations must be performed at each step of the serial APC method while only one RHS evaluation was required by the parallel APC. Since a RHS evaluation is the most time consuming operation that must be performed, the PPC method will execute nearly twice as fast as the serial APC method. Ideally then, the speed-up due to parallelism would be approximately two in this case.

4.2.3 Overall Execution Time

In Section 4.2.1 and 4.2.2, the execution time of the minimization phase and integration phase of the parallel algorithms discussed in Chapter Two was estimated. Also, in these sections, the timing equations for some widely used serial procedures were given. In this section, these results will be utilized to estimate the execution time of the overall algorithm, as well as the speed-up due to parallelism.

To estimate the overall execution time of the parallel algorithm discussed in Chapter Two, we simply substitute eqns. (4.2.2-5) and (4.2.2-6) into one of the timing equations for the parallel minimization algorithms. Similarly, the overall execution time of a completely serial method may be obtained by substituting eqns. (4.2.2-9) and (4.2.2-10) into one of the timing equations associated with the serial minimization procedures.

For example, if the serial ZP or DFP method were used with the serial APC method, the overall execution time for one iteration is:

ZP/APC:

$$\begin{aligned} t_{pi} = \{&n^2+3n+2+[(n+2)L+1](11+2M_{rhs})N\}Mt_m \\ &+\{n^2+4n+1+[(n+2)L+1](8+2A_{rhs})N\}At_a \end{aligned} \qquad (4.2.2\text{-}11)$$

DFP/APC:

$$t_{pi} = \{6n^2+3n+(L+n+1)(11+2M_{rhs})N\}Mt_m$$
$$+\{4n^2+2n-2+(L+n+1)(8+2A_{rhs})N\}At_a \qquad (4.2.2\text{-}12)$$

On the other hand, if the CM or PVM algorithm were used with the PPC integration method, the time per iteration of the overall algorithm would be:

CM/PPC:

$$t_{pi} = \{5+n+(L+1)(5+M_{rhs})N\}Mt_m$$
$$+\{3+2n+(L+1)(4-A_{rhs})N\}At_a \qquad (4.2.2\text{-}13)$$

PVM/PPC:

$$t_{pi} = \{2n^2+3n+2+\tfrac{3}{2}n(n-1)+(5+M_{rhs})N\}Mt_m \qquad (4.2.2\text{-}14)$$
$$+\{2n^2+4n-1+\tfrac{3}{2}n(n-1)+(L+n+1)(4+A_{rhs})N\}At_a \qquad (4.2.2\text{-}14)$$

Note that if the processor add time, t_a, and multiply time, t_m, were known along with M, L, N, A_{rhs} and M_{rhs}, then the time per iteration, t_{pi}, could be estimated using eqn. (4.2.2-11), (4.2.2-12), (4.2.2-13) or (4.2.2-14). Also, the timing equations may be used to estimate the speed-up due to parallelism, the details of which are given in the following theorem:

Theorem 4.1: Let $N > > n^2$ and $2(M_{rhs}+A_{rhs}) > > 19$. Then the speed-up due to parallelism, S, satisfies the following inequalities:

i. $$S \underline{\Delta} \frac{t_{pi} \text{ of the DFP/APC method}}{t_{pi} \text{ of the PVM/PPC method}} > 2$$

ii. $$S \underline{\Delta} \frac{t_{pi} \text{ of the ZP/APC method}}{t_{pi} \text{ of the CM/PPC method}} > \frac{2[(n+2)L+1]}{L+1}$$

Proof: To prove i.: Dividing eqn. (4.2.2-12) by (4.2.2-14) and using the fact that $N >> n^2$, we have

$$S = \frac{19+2(M_{rhs}+A_{rhs})}{9+M_{rhs}+A_{rhs}}$$

But since $2(M_{rhs}+A_{rhs}) > 19$, S may be bounded from below as follows:

$$S > \frac{19+19}{9+19/2} = 2.054 > 2$$

To prove ii.: Dividing eqn. (4.2.2-11) by eqn. (4.2.2-13) and recalling that $N >> n^2$, we have

$$S = \frac{(n+2)L+1}{L+1} \{\frac{19+2(M_{rhs}+A_{rhs})}{9+M_{rhs}+A_{rhs}}\}.$$

But $2(M_{rhs}+A_{rhs}) > 19$ implies that

$$S > \frac{2.054[(n+2)L+1]}{L+1} > \frac{2[(n+2)L+1]}{L+1}$$

To illustrate the impact of parallelism, as well as the tightness of the bounds given by Theorem 4.1, suppose t_a = 200 nsec, t_m = 1000 nsec, n=9, L=4, N=100, A_{rhs}=33 and M_{rhs}=43. Then by substituting these quantities into eqns. (4.2.2-11), (4.2.2-12), and (4.2.2-14), we have:

t_{pi} = 0.4503 seconds for ZP/APC

t_{pi} = 0.1432 seconds for DFP/APC

t_{pi} = 0.02721 seconds for CM/PPC

t_{pi} = 0.0710 seconds for PVM/PPC

Observe that the results above indicate that one iteration of the parallel algorithms require significantly less time to execute compared with the serials methods. This, along with the fact that the parallel algorithms generally require fewer iterations to converge compared with serial methods (this was demonstrated in Section 3.1.3), makes the parallel methods very attractive. Furthermore, the speed-up due to parallelism in this case is:

GRADIENT-DEPENDENT METHODS:

$$S = \frac{0.1432}{0.0710} = 2.017 > 2.0$$

NONGRADIENT METHODS:

$$S = \frac{0.4503}{0.02721} = 16.548 > 16.4$$

Note that the lower bounds above, which were obtained from Theorem 4.1, are rather close to the actual speed-up calculations.

Finally, from eqns. (4.2.2-11), (4.2.2-12),(4.2.2-13) and (4.2.2-14), it can be observed that if many integration steps are needed to solve a given initial-value problem, then the integration phase of the parallel algorithm requires much more time to execute compared with the minimization phase. Thus, it is very important to use an efficient, yet accurate, integration method. This observation was the primary reason for developing the parallel variable step size integration method discussed in Section 3.2.3.

CHAPTER FIVE

PERFORMANCE OF PARALLEL ALGORITHMS

In this chapter, the performance of the parallel identification, estimation and control algorithms devised in this thesis will be evaluated. In Section 5.1, the robustness of the parallel variable metric algorithm discussed in Section 3.1.1 is demonstrated by solving an optimal control problem associated with a Van der Pol process. All aspects of the parallel algorithms are tested including parallel shooting and adaptive mesh selection. In Section 5.2, the performance of the parallel state and parameter estimation algorithms is evaluated by identifying the aerodynamic parameters and the initial state of the lateral equations of motion for a T-33 aircraft. The indirect and direct control algorithms developed in Section 2.1 are then utilized in Section 5.3 to design a controller for controlling the longitudinal equations of motion for a F-8 Crusader aircraft. Finally, in Section 5.4, the performance of the parallel adaptive control algorithms discussed in Section 2.3 is evaluated for the F-8 aircraft.

To establish a basis for comparison, the serial Davidon-Fletcher-Powell (DFP), Broyden-Fletcher-Shanno (BFS), and Zangwill-Powell (ZP) methods, along with Straeter's parallel variable metric (PVM) algorithm, the Chazan-Miranker (CM) method, and the parallel quasi-Newton (PQN) method will be employed to minimize the appropriate error functions described in Chapter Two. The gradients required by the variable metric algorithms were obtained by finite-differencing and the line searches were implemented by fitting a quadratic function through three points. All of the parallel algorithms were coded in

FORTRAN IV and executed on an IBM 3033 (a serial computer) since a parallel computer was not accessible.

5.1 Evaluation of Parallel Algorithm Performance

In this section, the Van der Pol system presented in Section 3.3 will be used to study the following:

- The robustness of the parallel variable metric algorithm discussed in Section 3.1.
- The convergence properties of the indirect control algorithm discussed in Section 2.2 when different integration methods are employed.
- The effect the number of subintervals has on convergence of the parallel shooting algorithm presented in Section 2.3.
- The convergence properties of the adaptive mesh selection algorithm described in Section 3.1.

To begin this study, recall from Section 3.3 that the Van der Pol system may be written as follows:

$$\dot{x}_1(t) = x_2(t) \qquad (5.1\text{-}1)$$

$$\dot{x}_2(t) = a(t)\,[1 - x_1^2(t)]\,x_2(t) - x_1(t) + u(t) \qquad (5.1\text{-}2)$$

where $x(0) = x_0$ is known, $a(t)$ is a parameter which reflects a particular system's dynamics, and $u(t)$ is a control variable.

The problem considered in this section is to obtain the optimal control, $u(t)$, which minimizes the cost functional:

$$J = \int_0^5 [x_1^2(t) + x_2^2(t) + u^2(t)]\,dt \qquad (5.1\text{-}3)$$

subject to the satisfaction of eqns. (5.1-1) and (5.1-2). From the necessary conditions of optimality given in Section 2.3, it is easy to show that the nonlinear two-point boundary value problem (NTPBVP) associated with the Van der Pol system is given by:

State Equations

$$\dot{x}_1(t) = x_2(t) \qquad (5.1\text{-}4)$$

$$\dot{x}_2(t) = a(t)\,[1 - x_1^2(t)]\,x_2(t) - x_1(t) + u(t) \qquad (5.1\text{-}5)$$

Costate Equations

$$\dot{\lambda}_1(t) = \lambda_2(t)\,[2\,a(t)\,x_1(t)\,x_2(t) + 1] - 2\,x_1(t) \qquad (5.1\text{-}6)$$

$$\dot{\lambda}_2(t) = -\lambda_1(t) - \lambda_2(t)\,a(t)\,[1 - x_1^2(t)] - 2\,x_2(t) \qquad (5.1\text{-}7)$$

Boundary Conditions

$$x(0) = [x_{10}, x_{20}]^T \qquad (5.1\text{-}8a)$$

$$\lambda(5) = [0, 0]^T \qquad (5.1\text{-}8b)$$

Optimal Control

$$u(t) = -\tfrac{1}{2}\,\lambda_2(t) \qquad (5.1\text{-}9)$$

If ordinary shooting is used to solve the NTPBVP above, then an appropriate error function which must be minimized is simply:

$$E = ||\lambda(5)||^2 \qquad (5.1\text{-}10)$$

To demonstrate the effectiveness of ordinary shooting and the parallel algorithms developed in Chapters Two and Three, these methods were used to determine the initial costate which causes eqn. (5.1-10) to be minimized, given that

$$a(t) \equiv 1 \quad \forall\, t \in [0,\ 5]$$

and

$$x(0) = [0 \quad 1]^T.$$

The initial costate vector was chosen as $\lambda(0) = [0.5 \quad 5.0]^T$ which caused the forward integration of eqns.(5.1-4), (5.1-5), 5.1-6) and (5.1-7) to be well defined. As a result, the NTPBVP could be easily solved for the optimal unconstrained control which is shown in Figure 5.1.

In many instances, due to physical constraints, the optimal control may be bounded. To accommodate problems of this type, the bounded control algorithm described in Section 2.3 can be employed. To demonstrate the effectiveness of this procedure, the optimal control problem above was solved assuming $|u(t)| \leq 0.8 \quad \forall\, t \in [0,\ 5]$. The optimal bounded control for this example is shown in Figure 5.1.

The ability of the parallel initial costate algorithm to reduce the error function defined by eqn (5.1-10) is shown in Tables 5.1-5.4. Since the parallel algorithms were simulated on an IBM 3033 (a serial computer), the total number of function and gradient evaluations shown in Tables 5.1-5.4 reflect the fact that an advanced computer with n = 2 or n + 1 = 3 levels of parallelism should be used to execute the Chazan-Miranker method or Straeter's method, respectively. The results indicate:

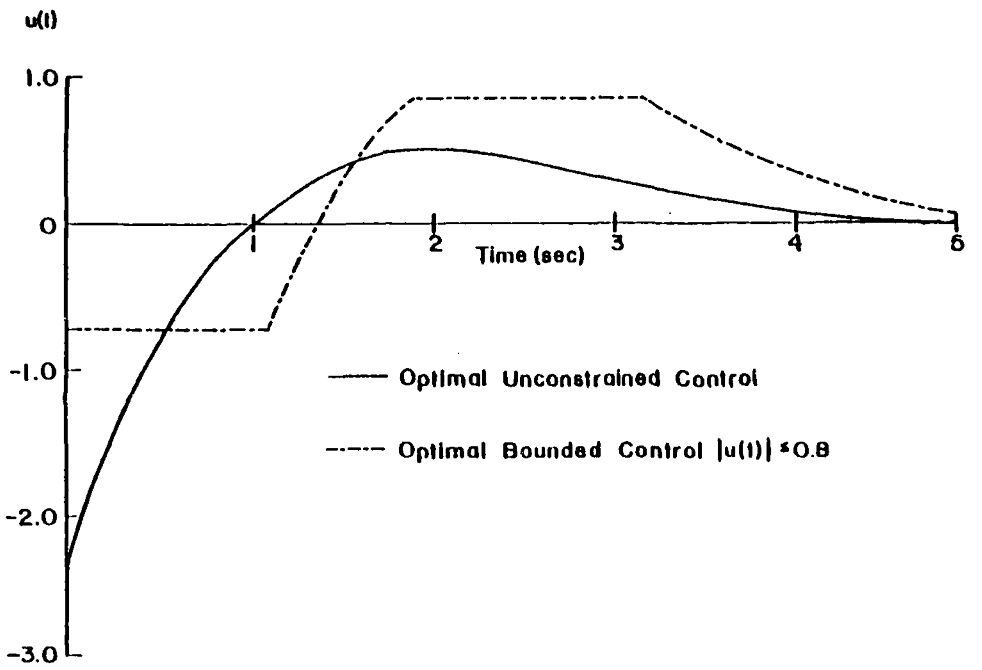


FIGURE 5.1: Control Trajectories: Van der Pol System

- For initial values of $\lambda(t_o)$ close to optimum, the Chazan-Miranker procedure is capable of speeding up convergence by nearly a factor of two over the DFP and ZP methods. Straeter's method is much more effective than the DFP method in reducing the error function for about the same number of function and gradient evaluations (Table 5.1).

- For initial values of $\lambda(t_o)$ far from optimal, the parallel algorithms are capable of reducing the error function by several orders of magnitude compared with the DFP method. Although the ZP method managed to reduce the error function the most, this was achieved at the expense of many function evaluations (Table 5.2).

- When the control is constrained ($|u(t)| \leq 0.8$) and the initial value of $\lambda(t_o)$ is selected near the optimum, all procedures converged. Note that the ZP method required a relatively large number of function evaluations to do so however (Table 5.3).

- If u(t) is constrained, Straeter's method is clearly superior to the DFP method when $\lambda(t_o)$ is initially far from optimal. Observe that although the CM method required many function evaluations, convergence did result, whereas the ZP method failed to locate the minimum in this case (Table 5.4).

5.1.1 Robustness of Parallel Minimization Algorithms

To study the robustness of the parallel minimization algorithms and the effect different integration schemes have on convergence, ordinary shooting was used to solve the NTPBVP represented by eqns.

Table 5.1: Algorithm Performance Near the Optimum

When $u(t)$ is Unconstrained

$x_o = (0. \quad 1.0)^T$ and $\lambda_o = (0.5 \quad 5.0)^T$

Minimization Algorithm	Total Number of Function Evaluations	Total Number of Gradient Evaluations	Total Number of Iterations	Resultant Error Function	Optimal Initial Costate Vector
ZP	53	-	4	0.102×10^{-6}	0.43019 5.1156
DFP	38	5	5	0.153×10^{-3}	0.4308 5.1152
CM	22	-	3	0.47×10^{-4}	0.43020 5.1156
FVM	37	5	5	0.412×10^{-7}	0.43017 5.1156

Table 5.2: Algorithm Performance Far From the Optimum

When u(t) is Unconstrained

$x_o = (0. \quad 1.0)^T$ and $\lambda = (5.0 \quad 1.0)^T$

Minimization Algorithm	Total Number of Function Evaluations	Total Number of Gradient Evaluations	Total Number of Iterations	Resultant Error Function	Optimal Initial Costate Vector
ZP	147	--	10	0.892×10^{-8}	0.43018 0.51156
DFP	85	10	10	0.186×10^{-3}	0.42904 5.1151
CM	94	--	10	0.474×10^{-6}	0.4302 5.1156
FVM	58	7	7	0.512×10^{-6}	0.43013 5.1156

Table 5.3: Algorithm Performance Near the Optimum

When $|u(t)| \leq 0.8$

$x_o = (0. \quad 1.0)^T$ and $\lambda_o = (-3. \quad 10.)^T$

Minimization Algorithm	Total Number of Function Evaluations	Total Number of Gradient Evaluations	Total Number of Iterations	Resultant Error Function	Optimal Initial Costate Vector
ZP	52	--	4	0.775×10^{-6}	-3.5703 9.8759
DFP	28	4	4	0.674×10^{-6}	-3.3668 9.8739
CM	33	--	4	0.853×10^{-6}	-3.5667 9.8738
FVM	29	4	4	0.235×10^{-8}	-3.5671 9.8745

Table 5.4: Algorithm Performance Far From the Optimum

When $|u(t)| \leq 0.8$

$x_o = (0. \quad 1.0)$ and $\lambda_o = (6.0 \quad 1.0)^T$

Minimizati Algorithm	Total Number of Function Evaluations	Total Number of Gradient Evaluations	Total Number of Iterations	Resultant Error Function	Optimal Initial Costate Vector
ZP	Failed to Locate Minimum				
DFP	80	9	9	0.539×10^{-5}	-3.5699 9.8741
CM	141	--	14	0.539×10^{-9}	-3.5701 9.8754
FVM	68	8	8	0.320×10^{-7}	-3.5702 9.8754

(5.1-4) - (5.1-8). As in the previous example, the error function which must be minimized is $E = ||\lambda(5)||^2$. The methods used to integrate eqns.(5.1-4) - (5.1-7) forward in time were the Adam's predictor-corrector (APC) pair given by eqns. (4.2.2-7), (4.2.2-8), the parallel predictor-corrector (PPC42) pair given by eqns. (3.2.1-4a), (3.2.1-4b), and the parallel predictor-corrector variable step size (PPC42V) method, discussed in Section 3.2.2.

Because of the expense incurred when simulating the parallel algorithms, only two of the parallel minimization methods were considered; namely the CM and PVM algorithms. As shown in Section 3.1.3, the rate of convergence of this the PVM algorithm depends primarily on a weighting parameter which defines a set of linearly independent vectors denoted by:

$$\Sigma \triangleq (\sigma_1, \sigma_2, \ldots, \sigma_n) = c\, I_n$$

where

I_n is a n x n identity matrix and

c is a scalar weighting parameter which is fixed for all iterations.

By varying the weighting parameter over a large range (say $10^{-9} \leq c \leq 10^{-1}$) the robustness of the PVM method can be evaluated. Also, integration effects can be measured by using one of the integration methods cited earlier. Because the APC and PPC42 integration methods are fixed step size procedures, the step size must be selected a priori. Since the step size, h, must be sufficiently small to assure accurate results without an excessive number of integration steps, h was set to 5/100 = 0.05. For the variable step size method (PPC42V), a 5-6 digit accuracy requirement was requested which meant that the step size must be varied to maintain the local truncation error below 10^{-5}. Initially, the integration step size was set to h = 0.05, but in order to meet the accuracy requirement imposed, the step size was immediately reduced to h = 0.025. After this initial adjustment, the step size remained at h = 0.025 for the remainder of the integration interval.

At this point, simulations were performed to determine the convergence properties of the indirect control algorithm when the integration methods described above are employed. The results are summarized in Figure 5.2.

The results indicate:

- The robustness of the PVM algorithm is enhanced the most when the PPC42 integration scheme is employed, i.e., for a wide range of c ($10^{-7} \leq c \leq 10^{-3}$), the performance of the PVM algorithm is insensitive to the specific value of the weighting parameter.

- The parallel integration methods enhance the robustness of the PVM algorithm more than the serial APC method; although the PVM method could be tuned to converge the fastest when the APC method was employed (see Figure 5.2).

Before leaving this section, a few words should be said about the robustness of the Chazan-Miranker (CM) algorithm. To use the CM algorithm, a monotone decreasing sequence must be selected. In optimal control applications, it has been determined empirically that a reasonable choice of the monotone decreasing sequence is:

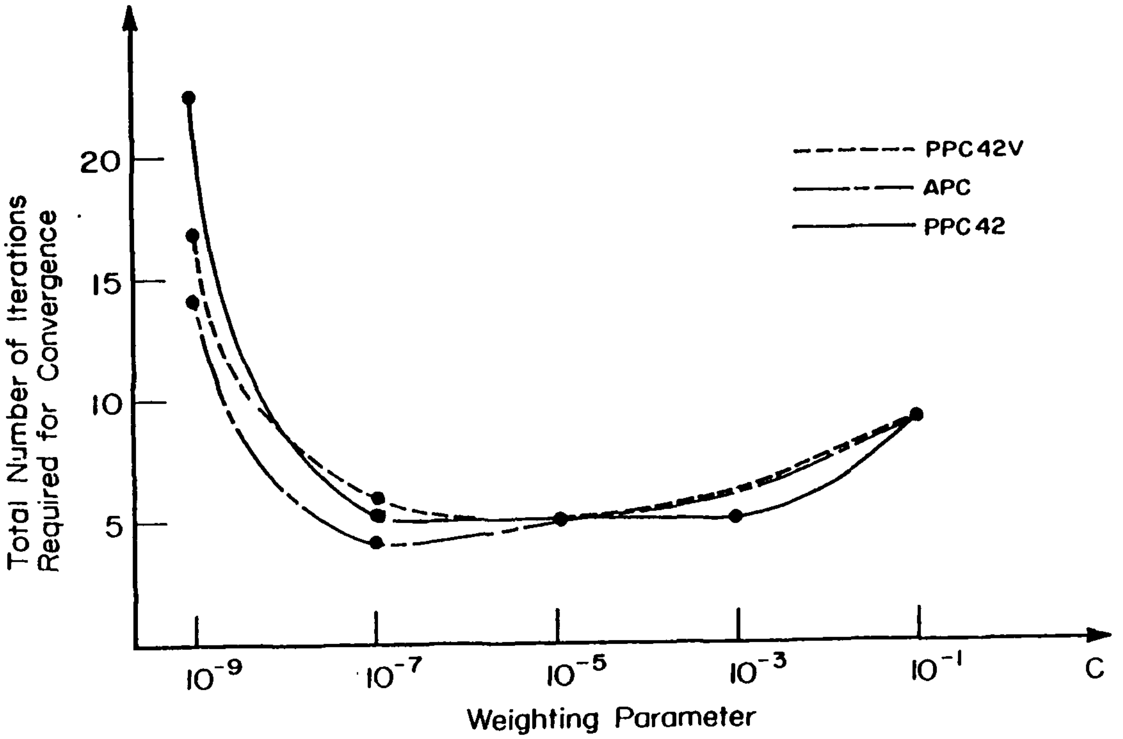


FIGURE 5.2: Robustness of PVM Algorithm

$$\beta_\ell = c \exp (-\ell)$$

where

c is a weighting parameter and ℓ denotes the iteration number.

In the context of the CM method, the term robustness refers to the relative insensitivity of the method to the particular value of c. By varying the weighting parameter c over a wide range, say $10 \leq c \leq 10^{-2}$, the robustness of the CM method can be measured. To this effect, the CM method was used with PPC integration to solve the NTPBVP associated with the Van der Pol system. The results obtained are shown in Figure 5.3.

The results indicate that the CM algorithm was not very robust at all. In fact, if the weighting parameter was greater than 10, the CM method didn't converge because the excessively large value of c caused the forward integration of eqns. (5.1-4) - (5.1-7) to become unstable. In view of these undesirable results, no further simulations were considered.

5.1.2 Parallel Shooting

In this section, the convergence of the parallel shooting algorithm presented in Section 2.3 will be studied by dividing the mission time interval into many subintervals and determining if convergence may be accelerated or not. Recall from Chapter Two that as more subintervals are introduced, the original NTPBVP becomes a multipoint boundary value problem and the number of unknown boundary conditions increase from 2n to n(2N-1) where n is the system order, and N is the number of subintervals. Despite this fact, the major advantage

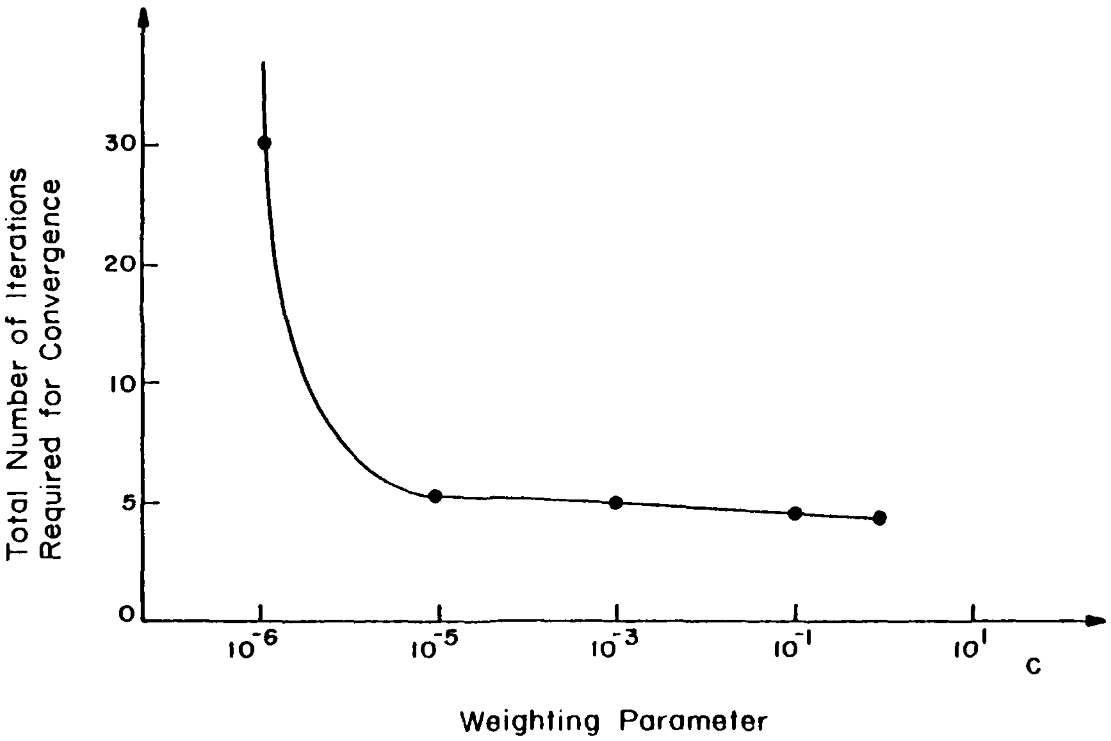


FIGURE 5.3: Robustness of CM Algorithm

of using parallel shooting is that the sensitivity problems associated with the forward integration of the state and costate equations is significantly reduced.

In view of the above, a natural question then is "How should N be chosen?" Although the parallel shooting method has been known for a number of years, the question raised here has not been answered satisfactorily.

One way to answer the above question, which is the approach taken here, is to select a value of N, solve the resultant multipoint boundary value problem using parallel shooting, and record the number of iterations required for convergence as a function of N. To determine if this procedure would indeed provide some insight into how to choose the number of subintervals, the mission time interval was divided into 2, 3, and 5 subintervals and the parallel shooting algorithm discussed in Section 2.3 was employed to solve the resultant multi-point boundary value problem. To illustrate the procedure, the two subinterval case will be briefly described.

For the two subinterval case, the mission time interval [0, 5] was arbitrarily partitioned into two subintervals of length $\Delta_1 = 1.6$ and $\Delta_2 = 3.4$. Note that the sum of Δ_1 and Δ_2 is equal to the length of the mission time interval since this is a necessary constraint. The "reduced" error function which must be minimized subject to the dynamic constraints given by eqns (5.1-4) to (5.1-7) is given by:*

*Since $x(t_0) = x_0$ is known, there is no need to include it in Y_ℓ. Observe that this reduces the number of unknowns in Y_ℓ by a factor of n.

$$E = ||PY_\ell + QY_r - \gamma||^2 \qquad (5.1.2\text{-}1)$$

where

$$P = \left[\begin{array}{cc|cccc} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{array}\right] \qquad Q = \left[\begin{array}{cccc|cc} 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ \hline -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 \end{array}\right]$$

$$Y_\ell = [\lambda_1(0)\ \lambda_2(0)\ x_1(1.6)\ x_2(1.6)\ \lambda_1(1.6)\ \lambda_2(1.6)]^T$$

$$Y_r = [x_1(1.6)\ x_2(1.6)\ \lambda_1(1.6)\ \lambda_2(1.6)\ \lambda_1(5.0)\ \lambda_2(5.0)]^T$$

and

$$\gamma = [\lambda_1(5.0)\ \lambda_2(5.0)\ 0\ 0\ 0\ 0]^T$$

To start the parallel shooting method, Y_ℓ was initially set to a zero vector but upon convergence Y_ℓ was determined to be:

$$Y_\ell = (0.43019 \quad 5.1156 \quad 0.3189 \quad -0.2012 \quad 1.871 \quad -0.7814)^T$$

Since the goal of this section is to determine how to choose the number of subintervals, the above procedure was repeated by dividing the mission time into three subintervals of length, $\Delta_1 = 1.0$, $\Delta_2 = 1.0$, and $\Delta_3 = 3.0$, and then into five subintervals of length, $\Delta_1 = \Delta_2 = \Delta_3 = \Delta_4 = \Delta_5 = 1.0$. Observe that, in each case the constraint that the sum of the subinterval lengths must be equal to the mission time was imposed at all times. Finally, to be consistent with the two subinterval case, Y_ℓ was initially set to zero for both the three and five subinterval cases.

To this effect, the simulations were performed. The results shown in Figure 5.4 indicate:

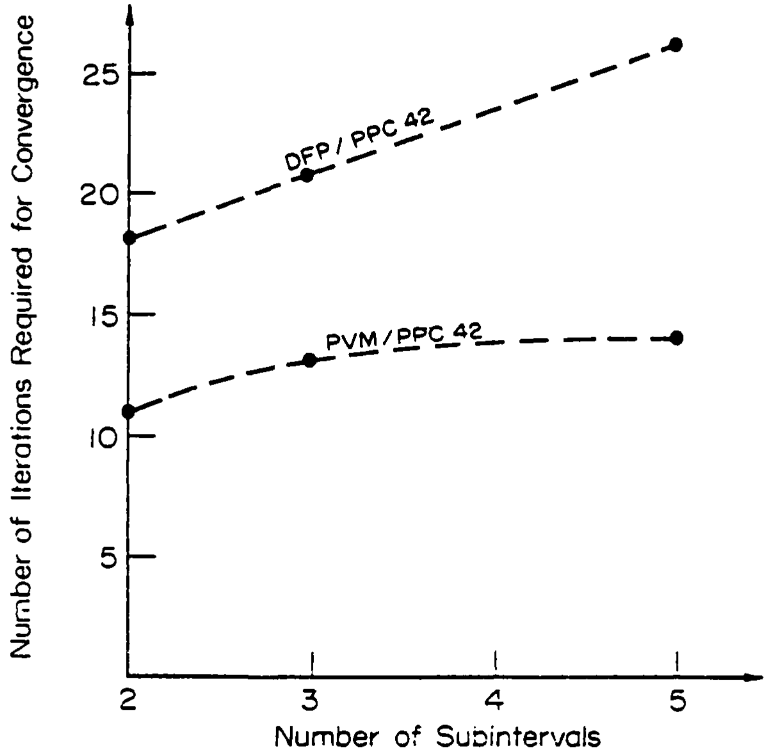


FIGURE 5.4: Convergence Properties of the Parallel Shooting Algorithm

- The integration interval should be partitioned into as few as possible subintervals since the number of iterations required for convergence, as well as the number of unknowns, will be reduced (see Figure 5.4).

- The number of iterations required for convergence appears to increase linearly with the number of subintervals for the DFP/PPC42 method while for the PVM/PPC42 method, the number of iterations required for convergence tends to level off as the number of subintervals is increased.

The last observation may be directly related to the accuracy of the solution to the initial-value problems over each subinterval. As the number of subintervals increase, the step size employed by the PPC42 integration method decreases because the number of integration steps taken over a given subinterval was held fixed. As a result, a more accurate integration of the appropriate differential equations was obtained over each subinterval. Thus, the gradients required by the PVM and DFP methods were obtained more accurately as the number of subintervals was increased. But since the PVM algorithm is known to perform better when accurate gradients are utilized, this may explain why the number of iterations required by the PVM/PPC42 method does not increase very rapidly. On the other hand, since the performance of the DFP method is generally not affected very much by inaccurate gradient information, this may explain the linear trend shown in Figure 5.4 for the DFP/PPC42 method.

It is instructive to note that the dimension of Y_ℓ in each case was $n(2N - 1)$. Since $n = 4$, this implies that 6, 10, and 18 unknowns must be found when 2, 3, and 5 subintervals were used respectively. Since the parallel algorithm converged in all cases, this indicates that this method may be employed to solve high order optimization problems.

Finally, it should be noted that the results obtained in this section are not very meaningful as is, because no insight has been gained as to where the mesh points should be placed. However, if the number of subintervals has been specified, then the adaptive mesh selection algorithm discussed in Section 2.2.2.1 can be utilized to allocate the mesh points in an optimal fashion. An example illustrating the use of the adaptive mesh selection algorithm is presented in the next section.

5.1.3 Adaptive Mesh Selection

In this section, the adaptive mesh selection (AMS) algorithm described in Section 2.2.2.1 will be employed to optimize the mesh points required by the parallel shooting algorithm. In view of the results obtained in the previous section, the mission time interval $[0, 5]$ was divided into only two subintervals. Hence, the problem was to simultaneously find the optimal values of the subinterval lengths (Δ_1 and Δ_2) and the solution to the resulting multi-point boundary value problem subject to the dynamic constraints given by eqns.(5.1-4) to (5.1-8), as well as the static constraints, $\Delta_1 > 0$, $\Delta_2 > 0$, and $\Delta_1 + \Delta_2 = 5$.

To use the AMS algorithm, one must decide which numerical integration method to use because the local truncation error associated with the integration method selected is required. Since the parallel predictor-corrector (PPC) method given by eqn.(3.2.1-4a) and (3.2.1-4b) is accurate to $O(h^4)$, this method was selected to integrate the required initial-value problems.

To illustrate the use of the AMS algorithm, consider the NTPBVP represented by eqns.(5.1-4) - (5.1-8) and the error function below:

$$E = ||PY + QY_r - \gamma||^2 + ||e||^2 \qquad (5.1.3\text{-}1)$$

where

$$P = \left[\begin{array}{cc|cccc} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{array}\right] \qquad Q = \left[\begin{array}{cccc|cc} 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ \hline -1 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 \end{array}\right]$$

$$Y_\ell = [\lambda_1(0)\ \lambda_2(0)\ x_1(\Delta_1)\ x_2(\Delta_1)\ \lambda_1(\Delta_1)\ \lambda_2(\Delta_2)\ \Delta_1]^T$$

$$Y_r = [x_1(\Delta_1)\ x_2(\Delta_1)\ \lambda_1(\Delta_1)\ \lambda_2(\Delta_1)\ \lambda_1(5.0)\ \lambda_2(5.0)]^T$$

and

$$\gamma = [\lambda_1(5.0)\ \lambda_2(5.0)\ \ 0\ \ 0\ \ 0\ \ 0]^T.$$

The first term shown in eqn (5.1.3-1) is the usual error function associated with the parallel shooting method while the second term is included to allow the mesh points to be optimized. For the problem under consideration, the error vector, e, has two components, namely:

$$e^{j+1}_{\max} = \max_{t_j \leq t \leq t_{j+1}} e_L(t) \qquad j = 0, 1 \qquad (5.1.3\text{-}2)$$

where $e_L(t)$ is the local truncation error associated with the PPC42 integration method. Recall from Section 3.2.2 that the norm of $e_L(t)$ was derived to be:

$$|e_L(t)| = \frac{19}{251} \left(|y_i^p - y_{i+1}^c| + h_i \; |f_i^c| \right) \qquad (5.1.3\text{-}3)$$

where in this case $y = (x_1 \;\; x_2 \;\; \lambda_1 \;\; \lambda_2)$ and f consists of the right-hand side (RHS) of $\dot{x}_1$, $\dot{x}_2$, $\dot{\lambda}_1$ and $\dot{\lambda}_2$.

In view of the above problem formulation, the parallel algorithms discussed in Chapter Three were used to minimize eqn.(5.1.3-1) subject to the dynamic constraints given eqns.(5.1-4) – (5.1-8) and the static constraint $\Delta_1 > 0$, $\Delta_2 > 0$, and $\Delta_1 + \Delta_2 = 5$. Based upon previously obtained solutions, Y_ℓ was initially selected as:

$$Y_\ell = (0.4 \quad 5.0 \quad 0.25 \quad -0.2 \quad 1.4 \quad -0.9 \quad 2.5)$$

which resulted in an initial error function value of $E = 0.9157$ and $||e|| = 0.01004$. Based upon the definition of E, these values indicate that initially most of the error was due to the choice of x and λ at the partition points rather than the choice of the partition points themselves. To reduce E, the DFP was used in conjunction with the PPC42 integration method. After ten iterations, the following values of Y_ℓ, E and $||e||$ were obtained.

$$Y_\ell = (0.4192 \quad 5.1201 \quad 0.1505 \quad -0.2093 \quad 0.9562 \quad -0.8581 \quad 2.334)$$

$$E = 0.0093 \quad \text{and} \quad ||e|| = 0.00929$$

Observe that these results indicate that the error in the solution at the partition points is only $0.0093 - 0.00929 = 0.00001$ and that the norm of the local truncation error, $||e||$, dominates over the solution error. Also, note that after ten iterations, the subinterval lengths had converged to $\Delta_1 = 0.2334$ and $\Delta_2 = 5 - \Delta_1 = 4.7666$.

To determine if the local truncation error could be reduced still further, the AMS algorithm was allowed to execute for a total of 50 iterations. After 50 iterations, the norm of the local truncation error had been reduced to $||e|| = 0.007402$ while the total error was $E = 0.007724$. Note that $||e||$ had been reduced by about 23% from its initial value of 0.01004 which is encouraging. The value of Y_ℓ obtained after 50 iterations was:

$$Y_\ell = (0.3285 \;\; 5.0683 \;\; 0.3354 \;\; -0.2084 \;\; 2.0047 \;\; -0.7364 \;\; 1.4378)$$

which indicates that the subinterval lengths should be selected as $\Delta_1 = 1.4378$ and $\Delta_2 = 3.5622$.

In summary, it can be concluded from the results presented in this section that by using the AMS algorithm to optimally select the mesh points required by the parallel shooting algorithm, the local truncation error can indeed be minimized, although many iterations may be required. Observe, if the integration interval [0, 5] is divided into two subintervals of lengths $\Delta_1 = 1.6$ and $\Delta_2 = 3.4$ and if the parallel shooting algorithm is used, then it was shown in the previous section that the optimal value of Y_ℓ is given by:

$$Y_\ell = (0.43019 \quad 5.1156 \quad 0.3189 \quad -0.2012 \quad 1.871 \quad -0.7814)^T$$

In this case, the norm of the local truncation error, $||e||$, is $||e|| = 0.00814$ which is nearly 8% greater than the error obtained using the adaptive mesh selection algorithm.

5.2 Evaluation of Estimator Performance

The performance of the parallel state and parameter estimation algorithms was evaluated using simulated measurement data characterizing the lateral motion of a T-33 aircraft. The equations of motion used in the simulations were given by [42]:

$$\dot{x}(t) = \begin{bmatrix} Y_\beta & \alpha_0 & -1 & 0.05467 \\ L_\beta & L_\rho & L_r & 0 \\ N_\beta & N_\rho & N_r & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} x(t) + \begin{bmatrix} Y_{\delta_a} \\ L_{\delta_a} \\ N_{\delta_a} \\ 0 \end{bmatrix} \delta_a(t) \qquad (5.2\text{-}1)$$

where the state variables $x_1(t)$, $x_2(t)$, $x_3(t)$, and $x_4(t)$ represent the sideslip angle, roll rate, yaw rate and roll angle respectively. The aileron deflection angle,δ_a,was selected as a one degree step command for the purposes of identification.

The measurement model used in the simulation was selected as:

$$z(t) = x(t) + v(t) \qquad (5.2\text{-}2)$$

where v(t) is a zero-mean WGN process with covariance

$$Q(t) = \begin{bmatrix} .004 & 0 & 0 & 0 \\ 0 & 4.0 & 0 & 0 \\ 0 & 0 & .017 & 0 \\ 0 & 0 & 0 & 3.4 \end{bmatrix}$$

This value of Q(t) was selected since a ratio of signal variance to noise variance of approximately two was desired.

The objective was to simultaneously estimate the four state variables and the aerodyanmic parameter vector

$$\Theta = (Y_\beta, \alpha_0, L_\beta, L_\rho, L_r, N_\beta, N_\rho, N_r, Y_{\delta_a}, L_{\delta_a}, N_{\delta_a})^T$$

However, to identify all of the aerodynamic parameters would be impractical since the computational cost would be too high. Therefore, a sensitivity study was conducted to determine those parameters which most affect system performance and therefore are most important to explicitly identify. More specifically, the state sensitivities:

$$\Delta x_i(t) = \frac{\partial x_i(t)}{\partial \Theta_j(t)} \Delta \Theta_j(t) \qquad \begin{array}{l} i = 1, 2, 3, 4 \\ j = 1, 2, \ldots, 11 \end{array} \qquad (5.2\text{-}3)$$

were calculated and displayed to aid the decision process (see Figure 5.5). From Figure 5.5, it can be concluded that α_0, L_ρ, L_{δ_a}, and N_{δ_a} are the most important parameters to identify since the state sensitivities associated with these parameters are much larger in magnitude than the remaining parameters. Therefore, the remaining parameters can be considered less important and set to their nominal values. Thus, we shall be concerned with solving the reduced SAP estimation problem in the remainder of this section.

5.2.1 Direct SAP Estimation

To find the unknown states and parameters associated with the T-33 aircraft, the following performance index was considered:

$$J = \tfrac{1}{2} ||\hat{x}(0) - m_{x_o}||^2_{\rho^{-1}_{xo}} + \tfrac{1}{2} \int_0^1 ||z(t) - x(t)||^2_{Q^{-1}(t)} \, dt \qquad (5.2.1\text{-}1)$$

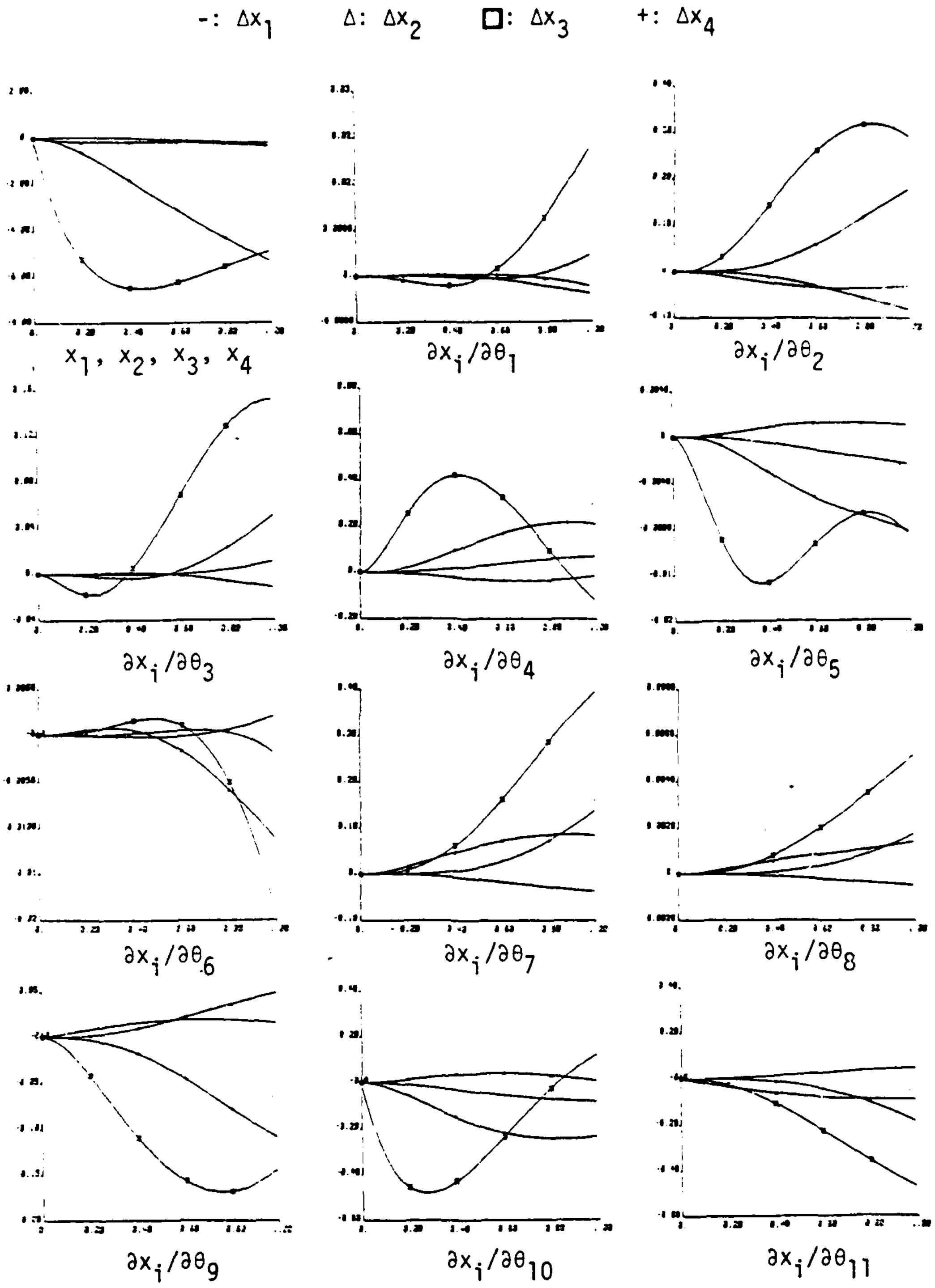
where

$$m_{x_o} = (0 \quad 0 \quad 0 \quad 0 \quad 0.142 \quad -6.51 \quad -44.4 \quad -1.8)^T$$

and

$$\rho_{x_o} = 0.01 \; I_8.$$

The parallel algorithms discussed in Chapters Two and Three were employed to find $\hat{x}(0)$ which minimizes eqn.(5.2.1-1) subject to eqn (5.2-1). To start the algorithm, the estimate of the augmented state vector was selected to be $\hat{x}(0) \equiv 0$ which resulted in an initial performance of $J = 1.008 \times 10^6$.



x_1, x_2, x_3, x_4 $\quad \partial x_i/\partial\theta_1$ $\quad \partial x_i/\partial\theta_2$

$\partial x_i/\partial\theta_3$ $\quad \partial x_i/\partial\theta_4$ $\quad \partial x_i/\partial\theta_5$

$\partial x_i/\partial\theta_6$ $\quad \partial x_i/\partial\theta_7$ $\quad \partial x_i/\partial\theta_8$

$\partial x_i/\partial\theta_9$ $\quad \partial x_i/\partial\theta_{10}$ $\quad \partial x_i/\partial\theta_{11}$

FIGURE 5.5: State Sensitivities

As indicated in Table 5.5, the parallel SAP estimation algorithm managed to reduce the performance index more rapidly than the serial DFP algorithm. In fact, the results in Table 5.5 indicate that the serial DFP method did not converge after eight iterations while the parallel method converged after seven iterations. Upon convergence of the PVM method, the initial state and parameter estimates compare very favorably with the true values which were

$$x_1 = 0., \; x_2 = 0., \; x_3 = 0., \; x_4 = 0.$$

$$\alpha_0 = 0.142, \; L_\rho = -6.51, \; L_{\delta_a} = -4.44 \text{ and } N_{\delta_a} = 1.8.$$

Using the estimated initial state and parameter vector, a simulation of the T-33 aircraft was performed. The resulting trajectories along with the simulated measurement data are shown in Figures (5.6-5.9). Note that the estimated roll rate and roll angle trajectories are essentially indistinguishable from the true trajectories while there is only a small error associated with the sideslip and yaw rate trajectories (see Figures 5.6 - 5.9).

5.2.2 Indirect SAP Estimation

To assess the performance of the indirect SAP estimation algorithm discussed in Section 2.1.1, it was used to find the unknown states and parameters of the T-33 aircraft. To use the indirect method, the SAP estimation problem defined by:

$$J = \tfrac{1}{2} \, ||\hat{x}(t_o) - m_{x_o}||^2_{P_{xo}^{-1}} + \tfrac{1}{2}\int_{t_o}^{t_f} (||z(t) - h\,[x(t), t]||^2_{Q^{-1}(t)} + ||\hat{w}(t)||^2_{R^{-1}(t)}) \;\; dt$$

TABLE 5.5: T-33 AIRCRAFT

Direct State and Parameter Estimation

$x_o = (0\ 0\ 0\ 0\ 0\ 0\ 0\ 0)$ and $J^o = 0.1008829 \times 10^6$

Minimization Algorithm	Total Number of Function Evaluations	Total Number of Gradient Evaluations	Number of Iterations	Estimated Initial State Vector	Estimated Parameter Vector	Resultant J
DFP	38	8	8	-0.1723 0.0285 -0.1747 0.0878	0.1157 -6.6755 -44.363 -2.0444	17.8238
PVM	34	7	7	-0.0065 -0.0001 0.0025 -0.0003	0.1421 -6.5119 -44.401 -1.8058	4.2031
ZP	306	-	10	-0.0059 0.0023 -0.0023 -0.0059	0.1419 -6.5101 -44.397 -1.8083	4.2052
CM	90	-	19	-0.0142 0.0012 0.0292 -0.0043	0.1414 -6.5021 -44.409 -1.8161	4.2802

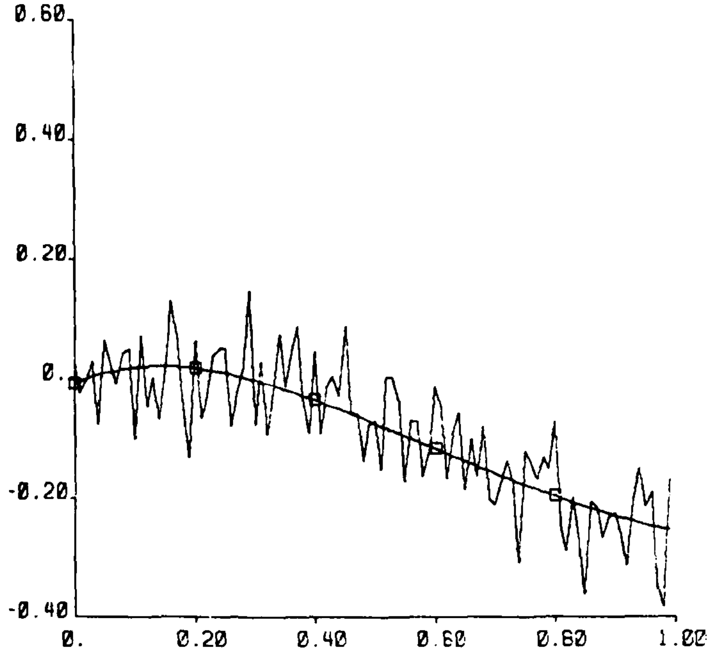


FIGURE 5.6: Sideslip Angle Estimation

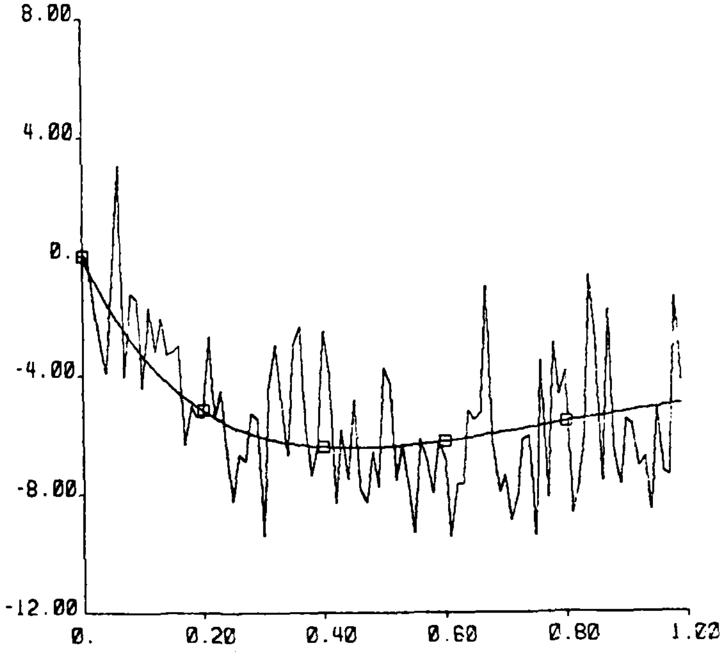


FIGURE 5.7: Roll Rate Estimation

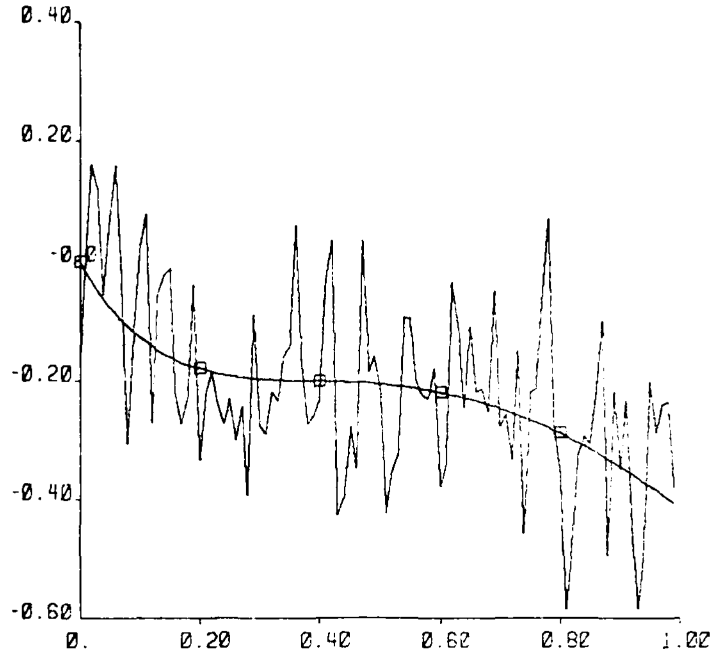


FIGURE 5.8: Yaw Rate Estimation

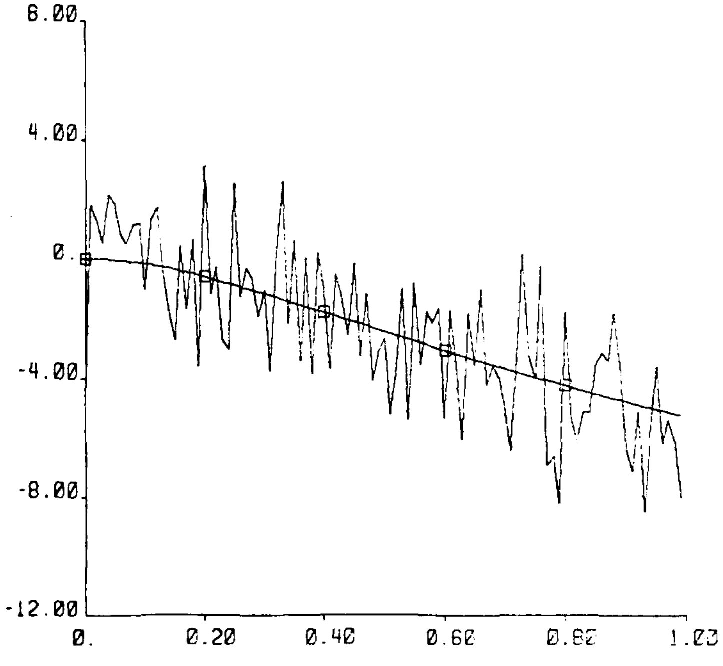


FIGURE 5.9: Roll Angle Estimation

and $\dot{x}(t) = f\,[x(t),\, t] + G[x(t),\, t]\,\hat{w}(t)$

must be reduced to a nonlinear two-point boundary value problem (NTPBVP). Using eqns. (2.1.2-2), (2.1.2-3), (2.1.2-4) it is easy to show that the NTPBVP associated with the T-33 aircraft is given by:

$$\dot{x}(t) = a\,x(t) + b\,\delta_a \qquad (5.2.2\text{-}1)$$

$$\dot{\lambda}(t) = R^{-1}(t)\,(z(t) - x(t)) - a^T(t)\,\lambda(t) \qquad (5.2.2\text{-}2)$$

where

$$a = \left[\begin{array}{cccc:c} Y_\beta & \alpha_0 & -1 & 0.05467 & \\ L_\beta & L_\rho & L_r & 0 & \\ N & N & N_r & 0 & 0_4 \\ 0 & 1 & 0 & 0 & \\ \hdashline & & 0_4 & & 0_4 \end{array}\right] \quad \text{and } b = \begin{bmatrix} Y_{\delta_a} \\ L_{\delta_a} \\ N_{\delta_a} \\ \underline{0} \end{bmatrix}$$

The boundary conditions associated with eqns. (5.2.2-1) and (5.2.2-2) are:

$$\lambda(0) = -\rho_{xo}^{-1}\,[\hat{x}(0) - m_{x_o}] \qquad (5.2.2\text{-}3a)$$

$$\lambda(1) = 0$$

In the problem formulation above, the augmented state vector, x(t), includes the unknown parameters α_0, L_ρ, L_{δ_a}, and N_{δ_a}.

To solve the NTPBVP represented by eqns. (5.2.2-1), (5.2.2-2) (5.2.2-3a) and (5.2.2-3b), ordinary shooting was used initially. However, due to the sensitivity of the costate eqn. (5.2.2-2) to small changes in $\hat{x}(0)$, convergence was rather slow.

On the basis of these results, it was decided that parallel shooting should be considered. Thus, the integration interval [0, 1] was partitioned into two subintervals ($\Delta_1 = 0.4$ and $\Delta_2 = 0.6$) and the parallel shooting algorithm discussed in Section 2.1.2 was employed to minimize the reduced error function:*

$$E = ||PY_\ell + QY_r - \gamma||^2 \qquad (5.2.2\text{-}4)$$

where

$$P = \left[\begin{array}{c|c} 0_8 & 0_8 \\ \hline 0_{16} & 0_{16} \end{array}\right] \qquad Q = \left[\begin{array}{c|c} 0_8 & I_8 \\ \hline -I_{16} & 0_8 \end{array}\right]$$

$$Y_\ell = [\hat{x}^T(0) \quad \lambda^T(0.4) \quad x^T(0.4)]^T$$

$$Y_r = [\lambda^T(0.4) \quad x^T(0.4) \quad \lambda^T(1.0)]^T$$

and

$$\gamma = 0$$

Note that by using parallel shooting, the dimension of the problem is artificially increased from $2n = 16$ to $n(2N - 1) = 24$. Although the problem now appears more formidable to solve, this is not the case because the sensitivity of the solution to the selection of $\hat{x}(0)$ should be reduced which might help convergence.

In fact, the results shown in Tables 5.6 and 5.7 indicate that convergence can occur using parallel shooting; however, the number of iterations required still was rather large. Note that the

* Since $\lambda(0) = \rho_{xo}^{-1} [\hat{x}(0) - m_{xo}]$ is known once $\hat{x}(0)$ is given, there is no need to include it in Y_ℓ. Observe that this reduces the number of unknowns in Y_ℓ by a factor of n.

TABLE 5.6: T-33 AIRCRAFT

Indirect SAP Estimation Using Ordinary Shooting

$$\hat{x}(o) = (0. \quad 0. \quad 0. \quad 0. \quad 0.14 \quad -6.5 \quad -44.3 \quad -1.9)$$

Initial Error Function = 0.2369×10^5

Minimization Algorithm	Total Number of Function Evaluations	Total Number of Gradient Evaluations	Total Number of Iterations	Estimated Initial State Vector	Estimated Parameter Vector	Resultant Error Function
ZP	1535	-	46	-0.0076 0.0003 -0.0015 -0.0002	0.1422 -6.5103 -44.3999 -1.8050	0.3581×10^{-3}
DFP	603	46	46	-0.0077 0.0003 -0.0012 0.0	0.1424 -6.5104 -44.3999 -1.8048	0.1073×10^{-9}

TABLE 5.7: T-33 AIRCRAFT

Indirect SAP Estimation Using Parallel Shooting

$\hat{x}(o) = (0.\ \ 0.\ \ 0.\ \ 0.\ \ 0.142\ \ -6.51\ \ -44.4\ \ -1.8)$

$\hat{\lambda}(0.4) = (0.\ \ 0.\ \ 0.\ \ 0.\ \ 0.\ \ 0.\ \ 0.\ \ 0.\ )$

$\hat{x}(0.4) = (0.\ \ -6.4\ \ 0.\ \ -1.8\ \ 0.142\ \ -6.51\ \ -44.4\ \ -1.8)$

$\Lambda_1 = 0.4$ and $\Lambda_2 = 0.6$ Initial Error Function = 0.5532×10^3

Minimization Algorithm	Total Number of Function Evaluations	Total Number of Gradient Evaluations	Total Number of Iterations	Estimated State and Parameter Vector at t = 0	Estimated Costate Vector at t = 0.4	Estimated State and Parameter Vector at t = 0.4	Resultant Error Function
ZP	3800	45	45	-0.0126 0.0006 0.0 0.0027 0.1293 -6.5104 -44.400 -1.8008	-1.0298 -0.0059 0.2080 0.0001 0.1884 -0.0283 0.0174 -0.0970	-0.0202 -6.5025 -0.1422 -1.8067 0.1447 -6.5490 -44.375 -1.9898	0.1037
DFP	1321	45	45	-0.0071 0.0 0.0023 0.0 0.1372 -6.5098 -44.399 -1.8009	-0.4193 0.0032 -0.1815 0.0393 -0.2369 -0.0377 -0.0067 0.0677	-0.0209 -6.4782 -0.2191 -1.8330 0.1458 -6.4532 -44.413 -1.8022	0.464×10^{-2}

results shown in Tables 5.6 and 5.7 clearly indicate that the gradient dependent DFP algorithm is preferable to the nongradient ZP method.*

5.2.3 Timing Considerations

In Section 4.2, the execution time of the parallel (and serial) algorithms was estimated in terms of variables representing the number of additions, multiplications, function and gradient evaluations. In this section, these results are employed to estimate the time required to simultaneously estimate the state and parameters of the T-33 aircraft's equations of motion. Although this can be done using any of the timing equations given in Section 4.3, the timing results will be explicitly calculated assuming the PVM minimization method and the fourth-order PPC integration method are used in the direct SAP estimation algorithm given in Section 2.1.

For the T-33 aircraft, 10 additions and 13 multiplications must be performed when evaluating the RHS of the aircraft's equations of motion. Also, suppose the parallel predictor-corrector (PPC42) method requires 100 integration steps to integrate the appropriate differential equations over the integration interval [0, 1]. By substituting $A_{rhs} = 10$, $M_{rhs} = 13$ and $N = 100$ into eqn. (4.2.2-5), an estimate of the time required for one function evaluation can be obtained as follows:

$$t_{fe} = 1400\ A\ t_a + 1800\ M\ t_m \qquad (5.2.3\text{-}1)$$

If the direct state and parameter (SAP) estimation procedure is

* The parallel minimization algorithms were not considered here due to the enormous expense which would be incurred when simulating these methods on a serial computer.

utilized, then n = 8 and the execution time for one gradient evaluation is:

$$t_{ge} = n\ t_{fe} = 11{,}200\ A\ t_a + 14{,}400\ M\ t_m \qquad (5.2.3\text{-}2)$$

From Table 5.5, four function evaluations are performed during a line search on the average. Hence, let L = 4. By substituting eqns. (5.2.3-1) and (5.2.3-2) into eqn.(4.2.1-5), an estimate of the execution time for one iteration of Straeter's PVM algorithm may be obtained as follows:

$$t_{pi} = 23{,}638\ M\ t_m + 18{,}443\ A\ t_a \qquad (5.2.3\text{-}3)$$

To illustrate the speed achievable through parallelism, suppose the processor add and multiply times are $t_a = 200$ nsec and $t_m = 1000$ nsec , respectively. By substituting these values of t_a and t_m into eqn.(5.2.3-3), the execution time per iteration is only 0.0273 seconds. Note that this time may be reduced still further if the PPC42 integration method is used to integrate each state equation on separate processors. In particular, if 16 processors (two for each state equation) were available, the function and gradient evaluation time would be reduced to:

$$t_{fe} = 175\ A\ t_a + 225\ M\ t_m \qquad (5.2.3\text{-}4)$$

and

$$t_{ge} = 1400\ A\ t_a + 1800\ M\ t_m \qquad (5.2.3\text{-}5)$$

By substituting eqns. (5.2.3-4) and (5.2.3-5) into eqn.(4.2.1-5), the estimated execution time for one iteration of the PVM algorithm would be reduced to 0.0036 seconds.

By using the timing equations derived in Section 4.2, and the procedure described above, the execution time of many other algorithms may be estimated. To provide a basis for comparison, this was done for both direct and indirect SAP estimation procedures and the results are shown in Tables 5.8 and 5.9.

The results indicate:

- The execution time per iteration can be significantly reduced if a completely parallel (i.e., parallel minimization and parallel integration) algorithm is used.

- One iteration of the nongradient algorithms require less time to execute compared with the gradient-dependent methods.

- One iteration of the indirect method requires much more time and processors to execute compared with the direct method (see Tables 5.8 and 5.9).

Finally, the speed-up due to parallelism is illustrated in Figure 5.10 which illustrates the speed-up/iteration as a function of processors.

5.3 Evaluation of Controller Performance

The performance of the parallel nonlinear control algorithms presented in Section 2.2 was evaluated by designing a control system for controlling the longitudinal motion of an F-8 Crusader aircraft. The longitudinal equations of motion of the F-8 aircraft were obtained from the aircraft model shown in Figure 5.11. The aircraft model illustrates the forces which were considered and the coordinate system used by Garrard and Jordan in reference [43].

TABLE 5.8: T-33 AIRCRAFT

Direct SAP Estimation Timing

$n = 8$, $L = 4$, $N = 100$, $A_{rhs} = 10$, $M_{rhs} = 13$, $t_a = 200$ nsec, $t_m = 1000$ nsec

Integration Method / Minimization Algorithm	APC (1PE/RHS)	APC (1PE/STATE)	PPC42 (2PE/RHS)	PPC42 (2PE/STATE)
ZP	0.1747/1	0.0219/9	0.0854/3	0.0108/17
DFP	0.0558/1	0.0074/9	0.0275/3	0.0038/17
BFS	0.0559/1	0.0075/9	0.0276/3	0.0039/17
CM	0.0213/9	0.0026/65	0.0104/25	0.0013/137
PVM	0.0556/9	0.0072/65	0.0273/25	0.0036/137
PDFP	0.0558/9	0.0073/65	0.0275/25	0.0038/137
PBFS	0.05617/9	0.0077/65	0.0278/25	0.0042/137

KEY: t_{pi}/PE's

TABLE 5.9: T-33 AIRCRAFT

Indirect SAP Estimation Timing

$n = 8$, $L = 4$, $N = 100$, $\Lambda_{rhs} = 33$, $M_{rhs} = 42$, $t_a = 200$ nsec, $t_m = 1000$ nsec

Integration Method / Minimization Algorithm	APC (1PE/RHS)	APC (1PE/STATE)	PPC42 (2PE/RHS)	PPC42 (2PE/STATE)
ZP	0.4503/1	0.05638/17	0.2231/3	0.0279/33
DFP	0.1432/1	0.01830/17	0.07118/3	0.0093/33
BFS	0.1433/1	0.0184/17	0.07126/3	0.0094/33
CM	0.054916/9	0.006879/129	0.02721/25	0.003416/265
[illegible]	0.1430/9	0.01813/129	0.0710/25	0.00912/265
PDFP	0.1432/9	0.01830/129	0.07118/25	0.00929/265
PBFS	0.1435/9	0.01864/129	0.07152/25	0.009636/265

KEY: t_{pi}/PE's

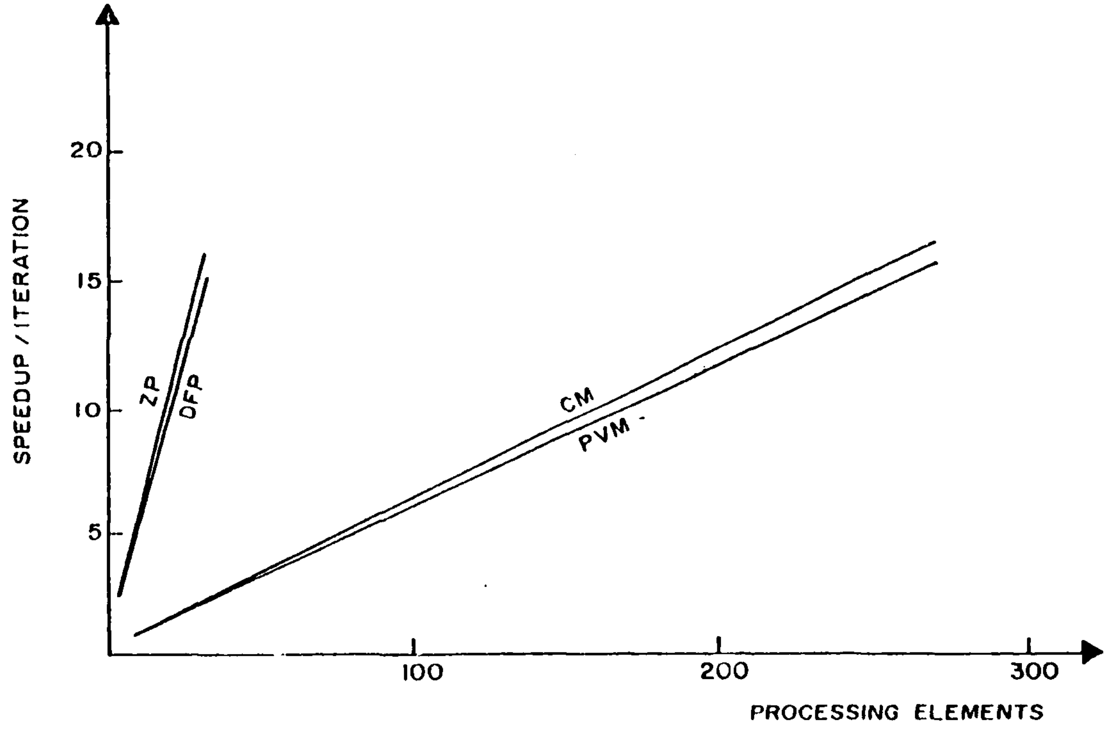


FIGURE 5.10: T-33 Aircraft: Indirect SAP Estimation Speed Up

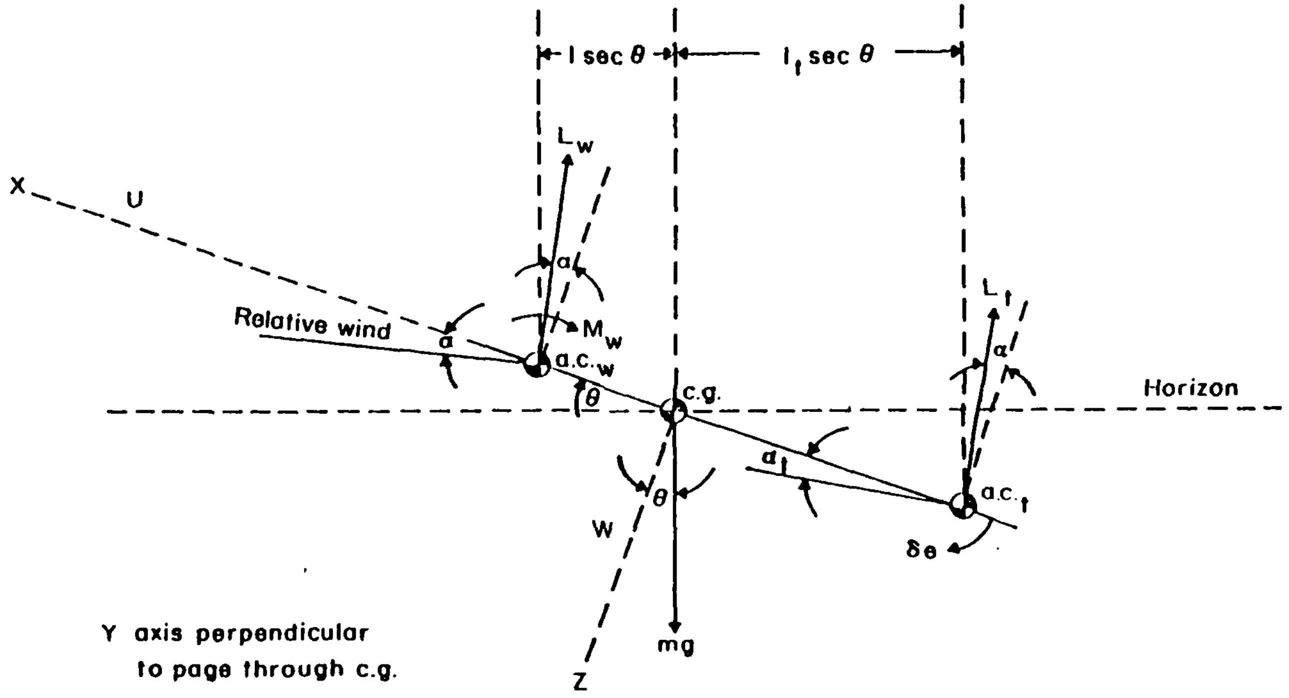


FIGURE 5.11: Aircraft Dynamical Model

It is assumed that the aerodynamic drag is negligible and that the lift may be separated into its wing and tail components. Under these conditions, it can be shown (cf. [43]) that the longitudinal equations of motion become:

$$m(\dot{u} + w\,\dot{\Theta}) = -mg \sin\Theta + L_w \sin\alpha + L_t \sin\alpha_t \qquad (5.3\text{-}1)$$

$$m(\dot{w} + u\,\dot{\Theta}) = mg \cos\Theta - L_w \cos\alpha - L_t \cos\alpha_t \qquad (5.3\text{-}2)$$

$$I_y\,\ddot{\Theta} = M_w + \ell\, L_w \cos\alpha - \ell_t\, L_t \cos\alpha_t - c\,\dot{\Theta} \qquad (5.3\text{-}3)$$

where

m = mass of aircraft

u = velocity of aircraft in X direction

w = velocity of aircraft in Z direction

Θ = angular displacement about Y axis, measured clockwise from the horizon as shown in Figure 5.11

I_y = moment of inertia of aircraft about Y axis

L_w = wing lift

L_t = tail lift

α = wing angle of attack

α_t = tail angle of attack

M_w = wing moment

ℓ = distance between wing aerodynamic center and aircraft center of gravity

ℓ_t = distance between tail aerodynamic center and aircraft center of gravity

$c\dot{\Theta}$ = damping moment.

In reference [43], Garrard and Jordan reduce eqns. (5.3-1) - (5.3-3) to a nonlinear dynamical model in which cubic and lower order terms are retained for the F-8 aircraft. This model of the F-8 aircraft which is given below can be used to study the effect disturbances have on the F-8 aircraft when it is perturbed from level, unaccelerated flight at Mach = 0.85 and an altitude of 30,000 feet.

For small angle of attack disturbances ($\alpha < 23.5^{o} = 0.41$ radians), the F-8 aircraft model is given by:

$$\begin{aligned}\dot{x}_1 = {} & (1 - x_1^2 - 0.088\ x_1)\ x_3 - 0.877\ x_1 + 0.47\ x_1^2 \\ & + 3.846\ x_1^3 - 0.215\ u + 0.28\ u\ x_1^2 + 0.47\ u^2\ x_1 \\ & + 0.63\ u^3 - 0.019\ x_2^2 \end{aligned} \qquad (5.3\text{-}4)$$

$$\dot{x}_2 = x_3 \qquad (5.3\text{-}5)$$

$$\begin{aligned}\dot{x}_3 = {} & -0.396\ x_3 - 4.208\ x_1 - 0.47\ x_1^2 - 3.564\ x_1^3 \\ & - 20.967\ u + 6.265\ u\ x_1^2 + 46\ u^2\ x_1 + 61.4\ u^3\end{aligned} \qquad (5.3\text{-}6)$$

while for large angle of attack disturbances ($\alpha \geq 23.5^{o} = 0.41$ radians) the F-8 aircraft model becomes:

$$\begin{aligned}\dot{x}_1 = {} & (1 - x_1^2 - 0.088\ x_1)\ x_3 - 0.019\ x_2^2 - 0.053\ x_1 \\ & + 0.006\ x_1^2 + 0.049\ x_1^3 - 0.215\ u + 0.28\ u\ x_1^2 \\ & + 0.47\ u^2\ x_1 + 0.63\ u^3\end{aligned} \qquad (5.3\text{-}7)$$

$$\dot{x}_2 = x_3 \qquad (5.3\text{-}8)$$

$$\begin{aligned}\dot{x}_3 = {} & -0.396\ x_3 - 5.116\ x_1 - 0.042\ x_1^2 - 0.32\ x_1^3 \\ & - 20.967\ u + 6.265\ u\ x_1^2 + 46\ u^2\ x_1 + 61.4\ u^3\end{aligned} \qquad (5.3\text{-}9)$$

The state variables, x_1, x_2, and x_3, represent the angle of attack, pitch angle, and pitch rate, respectively. The tail deflection angle, u, is the control variable which must be designed to reduce an angle of attack distrubance as rapidly as possible.

In the remainder of this section, an open loop and a closed loop controller will be designed and evaluated using the procedures discussed in Chapter Two. Also, the computation time required by the control synthesis algorithms will be estimated.

5.3.1 Open and Closed Loop Control Synthesis

In this section, an open loop controller is designed by solving a nonlinear two-point boundary value problem (NTPBVP) associated with the F-8 aircraft's longitudinal equations of motion. Also in this section, two closed loop controllers are designed. The first using linear quadratic regulator (LQR) theory and the second using the direct gain optimization algorithm described in Section 2.2.2. The controllers are designed assuming the angle of attack remains below 23.5° so that the low angle of attack model given by eqns. (5.3-4), (5.3-5) and (5.3-6) could be used.

Linear Controller Synthesis

To use LQR theory to design a controller for the F-8 aircraft, the equations of motion must be linearized. Linearizing eqns. (5.3-4), (5.3-5), and (5.3-6) results in the following linear model of the F-8 aircraft:

$$\dot{x}(t) = \begin{bmatrix} -0.877 & 0. & 1.0 \\ 0. & 0. & 1.0 \\ -4.208 & 0. & -0.396 \end{bmatrix} x(t) + \begin{bmatrix} -0.215 \\ 0. \\ -20.976 \end{bmatrix} u(t) \qquad (5.3.1\text{-}1)$$

which is of the form

$$\dot{x}(t) = A\ x(t) + b\ u(t) \qquad (5.3.1\text{-}2)$$

The optimal control must be determined to minimize the quadratic performance index

$$\int^{\infty} [x(t)^T Q\ x(t) + r\ u(t)^2]\ dt \qquad (5.3.1\text{-}3)$$

subject to the dynamic constraint given by eqn.(5.3.1-2). From LQR theory, it is well known that the optimal control is given by [44]

$$u(t) = -r^{-1} b^T P x(t) \qquad (5.3.1\text{-}4)$$

where P is the positive definite solution of the steady state matrix Riccati equation

$$A^T P + PA - Pbr^{-1} b^T P + Q = 0. \qquad (5.3.1\text{-}5)$$

The Q matrix and the scalar r were selected as

$$Q = \begin{bmatrix} 0.25 & 0. & 0. \\ 0. & 0.25 & 0. \\ 0. & 0. & 0.25 \end{bmatrix} \quad \text{and } r = 1.0$$

since this choice of Q and r gave good response without exceeding a maximum tail deflection of 25° (0.4363 radians) and a tail deflection rate of 60°/sec (1.0472 radians/sec). The optimal control problem above was solved using ORACLS - a collection of optimal regulator algorithms for the control of linear systems [45].

The resultant control law was determined to be:

$$u(t) = -0.053\, x_1(t) + 0.5\, x_2(t) + 0.521\, x_3(t) \qquad t \geq 0 \qquad (5.3.1\text{-}7)$$

To determine if a "better" controller could be designed by utilizing the nonlinear equations of motion of the F-8 aircraft directly, the direct gain optimization procedure discussed in Section 2.2.2 was employed. Since we want the controller to be linear and utilize feedback, the controller is constrained to be of the form:

$$u(t) = K_1\, x_1(t) + K_2\, x_2(t) + K_3\, x_3(t) \qquad t \geq 0 \qquad (5.3.1\text{-}8)$$

where K_1, K_2, and K_3 are constant gains which must be determined.

The optimization problem, in this case, is to find the control gains which minimize the performance index:

$$J = \tfrac{1}{2}\int_0^1 [x^T(t)\ Q\ x(t) + r\ u^2(t)]\ dt \qquad (5.3.1\text{-}9)$$

subject to the nonlinear equations of motion given by eqns. (5.3-4), (5.3-5), and (5.3-6). The matrix Q and the scalar r were selected to be the same as in the LQR design.

At this point, the direct gain optimization procedure could be used to find the optimal values of K_1, K_2, and K_3 needed to define the control. Initially, K_1, K_2, and K_3 were set to zero and updated by the Davidon-Fletcher-Powell (DFP) method until the performance index (5.3.1-9) was minimized. After twenty iterations of the direct gain optimization procedure, the gains had converged to their optimal values which when substituted into eqn (5.3.1-8) yields:

$$u(t) = 0.1368\ x_1(t) + 0.4331\ x_2(t) + 0.6797\ x_3(t) \quad t \geq 0 \qquad (5.3.1\text{-}10)$$

Nonlinear Controller Synthesis

To determine how well the linear feedback controllers defined by eqns (5.3.1-8) and (5.3.1-10) approximate the optimal control, the calculus of variations approach was considered. In this case, the problem is to find an open loop control which minimizes eqn. (5.3.1-9) subject to the satisfaction of eqns. (5.3-4) – (5.3-6). Recall that, when the calculus of variations is used to solve an optimal control problem, it is necessary to solve a nonlinear two-point boundary value problem (NTPBVP). In this case, the NTPBVP which must be solved is easily shown to be:

State Equations:

$$\dot{x}_1 = (1 - x_1^2 - 0.088\ x_1)\ x_3 - 0.877\ x_1 + 0.47\ x_1^2 + 3.846\ x_1^3 - 0.215\ u + 0.28\ u\ x_1^2 + 0.47\ u^2\ x_1 + 0.63\ u^3 - 0.019\ x_2^2 \qquad (5.3.1\text{-}11)$$

$$\dot{x}_2 = x_3 \qquad (5.3.1\text{-}12)$$

$$\dot{x}_3 = -0.396\ x_3 - 4.208\ x_1 - 0.47\ x_1^2 - 3.564\ x_1^3 - 20.967\ u + 6.265\ u\ x_1^2 + 46.\ u^2\ x_1 + 61.4\ u^3 \qquad (5.3.1\text{-}13)$$

Costate Equations:

$$\dot{\lambda}_1 = -0.25\ x_1 + \lambda_1\ (2.x_1\ x_3 + 0.088\ x_3 + 0.877 - 0.94\ x_1 - 11.538\ x_1^2 - 0.56\ u\ x_1 - 0.47\ u^2) + \lambda_3\ (4.208 + 0.56\ x_1 + 10.692\ x_1^2 - 12.53\ u\ x_1 + 46.\ u^2) \qquad (5.3.1\text{-}14)$$

$$\dot{\lambda}_2 = -0.25\ x_2 + 0.038\ \lambda_1\ x_2 \qquad (5.3.1\text{-}15)$$

$$\dot{\lambda}_3 = -0.25\ x_3 - \lambda_1\ (1 - x_1^2 - 0.088\ x_1) - \lambda_2 + 0.396\ \lambda_3 \qquad (5.3.1\text{-}16)$$

Boundary Conditions:

$$x(0) = \begin{bmatrix} 0.349 \\ 0 \\ 0 \end{bmatrix} \qquad \lambda\ (0) = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

Let the Hamiltonian be defined as:

$$H(x,\ u,\ \lambda,\ t) = \tfrac{1}{2}\left[0.25(x_1^2 + x_2^2 + x_3^2) + u^2\right] + \lambda_1\ \dot{x}_1 + \lambda_2\ \dot{x}_2 + \lambda_3\ \dot{x}_3 \qquad (5.3.1\text{-}17)$$

Then the optimal control must satisfy the necessary condition:

$$H_u = (1.89\ \lambda_1 + 184.2\ \lambda_3)\ u^2 + (1.0 + 0.94\ x_1\ \lambda_1$$
$$+ 92.\ x_1\ \lambda_3)\ u + \lambda_1\ (0.28\ x_1^2 - 0.215) + \lambda_3\ (6.265\ x_1^2$$
$$- 20.967) = 0 \qquad (5.3.1\text{-}18)$$

If we let

$$A = 1.89\ \lambda_1 + 184.2\ \lambda_3$$
$$B = 1.0 + 0.94\ x_1\ \lambda_1 + 92.\ x_1\ \lambda_3$$

and

$$C = \lambda_1\ (0.28\ x_1^2 - 0.215) + \lambda_3\ (6.265\ x_1^2 - 20.967),$$

then the necessary condition becomes:

$$A\ u^2 + B\ u + C = 0$$

which implies that the optimal control, u, is given by:

$$u = \frac{-B + \sqrt{B^2 - 4\ AC}}{2\ A} \qquad (5.3.1\text{-}19)$$

From the optimal control theory, it is well known that a sufficient condition for optimality is $H_{uu} > 0$. Thus,

$$H_{uu} = 2\ A\ u + B > 0 \qquad (5.3.1\text{-}20)$$

Substituting eqn.(5.3.1-19) into eqn. (5.3.1-20), we have

$$2\ A\ u + B = \pm \sqrt{B^2 - 4\ C}$$

which is positive only if the positive square root is used. Therefore, the optimal control is given by:

$$u = \frac{-B + \sqrt{B^2 - 4\ AC}}{2\ A}$$

Note that the optimal control is relatively complex to implement due to the square root operation required. Furthermore, to implement this controller, the solution to the NTPBVP must be known a priori in order to evaluate A, B and C. Observe that this is not an easy task because as the initial costate is adjusted from iteration to iteration, the term $B^2 - 4AC$ may become negative, in which case the optimal control is undefined. Unfortunately, this occurred when numerical methods were applied to this problem.

One way to overcome this difficulty is to use the approach taken by Garrard and Jordon [43] who approximated eqns.(5.3-4), (5.3-5), and (5.3-6) by eliminating the u^3, u^2 and cross product terms. In this case, the approximate equations of motion which are valid for low angles of attack are simply:

$$\dot{x}_1 = (1 - x_1^2 - 0.088\ x_1)\ x_3 - 0.877\ x_1 + 0.47\ x_1^2 + 3.846\ x_1^3 - 0.019\ x_2^2 - 0.215\ u \qquad (5.3.1\text{-}22)$$

$$\dot{x}_2 = x_3 \qquad (5.3.1\text{-}23)$$

$$\dot{x}_3 = -0.396\ x_3 - 4.208\ x_1 - 0.47\ x_1^2 - 3.564\ x_1^3 - 20.967\ u \qquad (5.3.1\text{-}24)$$

If this model of the F-8 aircraft is employed in the design of the open loop controller, the costate equations and control are given by:

<u>Costate Equations</u>:

$$\dot{\lambda}_1 = -0.25\ x_1 + \lambda_1\ (2 \cdot x_1\ x_3 + 0.088\ x_3 + 0.877 - 0.94\ x_1 - 11.538\ x_1^2) + \lambda_3\ (4.208 + 0.56\ x_1 + 10.692\ x_1^2 \qquad (5.3.1\text{-}25)$$

$$\dot{\lambda}_2 = -0.25\ x_2 + 0.038\ \lambda_1\ x_2 \qquad (5.3.1\text{-}26)$$

$$\dot{\lambda}_3 = -0.25\ x_3 - \lambda_1\ (1 - x_1^2 - 0.088\ x_1) - \lambda_2 + 0.396\ \lambda_3 \qquad (5.3.1\text{-}27)$$

Optimal Control:

$$u = 0.215\,\lambda_1 + 20.967\,\lambda_3 \qquad (5.3.1\text{-}28)$$

Note that in this case the optimal control (for the approximate F-8 model) is extremely simple to compute provided λ_1 and λ_3 are known. These quantities were obtained rather easily by solving the NTPBVP represented by eqns.(5.3.1-22) - (5.3.1-28). This was achieved by incorporating the serial DFP minimization algorithm and the APC integration method into the indirect control algorithm described in Section 2.2.2.1.* The resulting solution is, of course, optimal for the approximate F-8 model and is shown along with the LQR control and closed loop control designed using the direct gain optimization procedure in Figures 5.12 - 5.15. The trajectories displayed in Figures 5.12 - 5.15 indicate:

- The response of the F-8 aircraft due to the LQR control is significantly different from that due to the open and closed loop controls. This may be attributed to using a linearized model of the F-8 aircraft in the design process.

- The response of the F-8 aircraft due to the closed loop control designed using the low angle of attack model compares very favorably with that due to the open loop control which is optimal for the approximate F-8 model.

The second result, along with the fact that the closed loop control utilizes feedback while the optimal control does not, indicates

* Parallel algorithms were not considered here because only the effectiveness of the resultant control was being studied not the method used to obtain it.

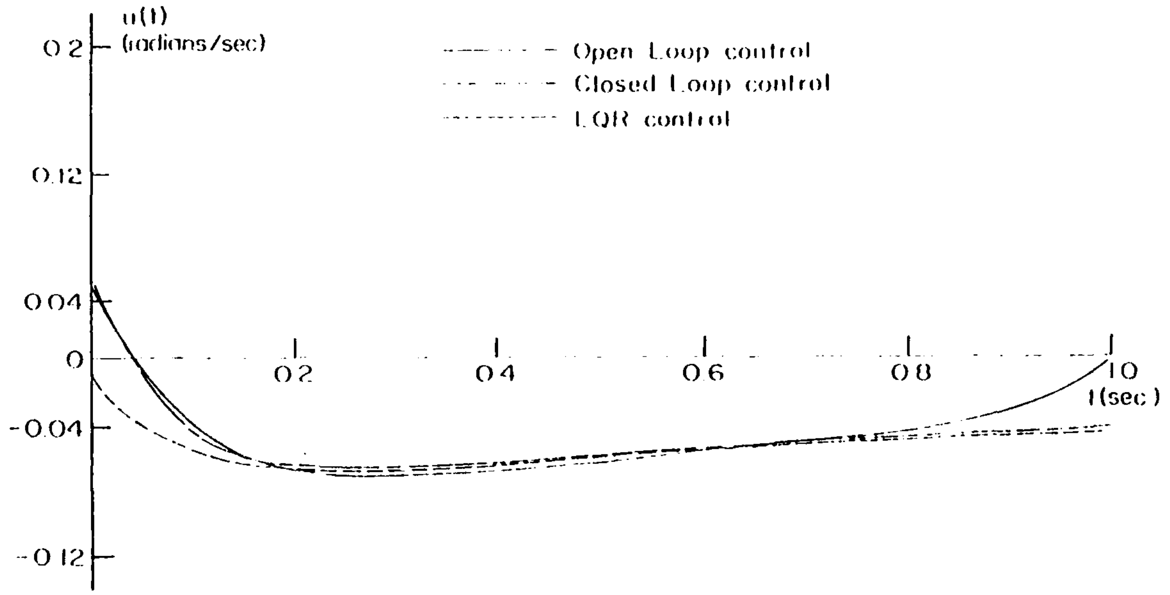


FIGURE 5.12: F-8 Aircraft: Control Trajectories

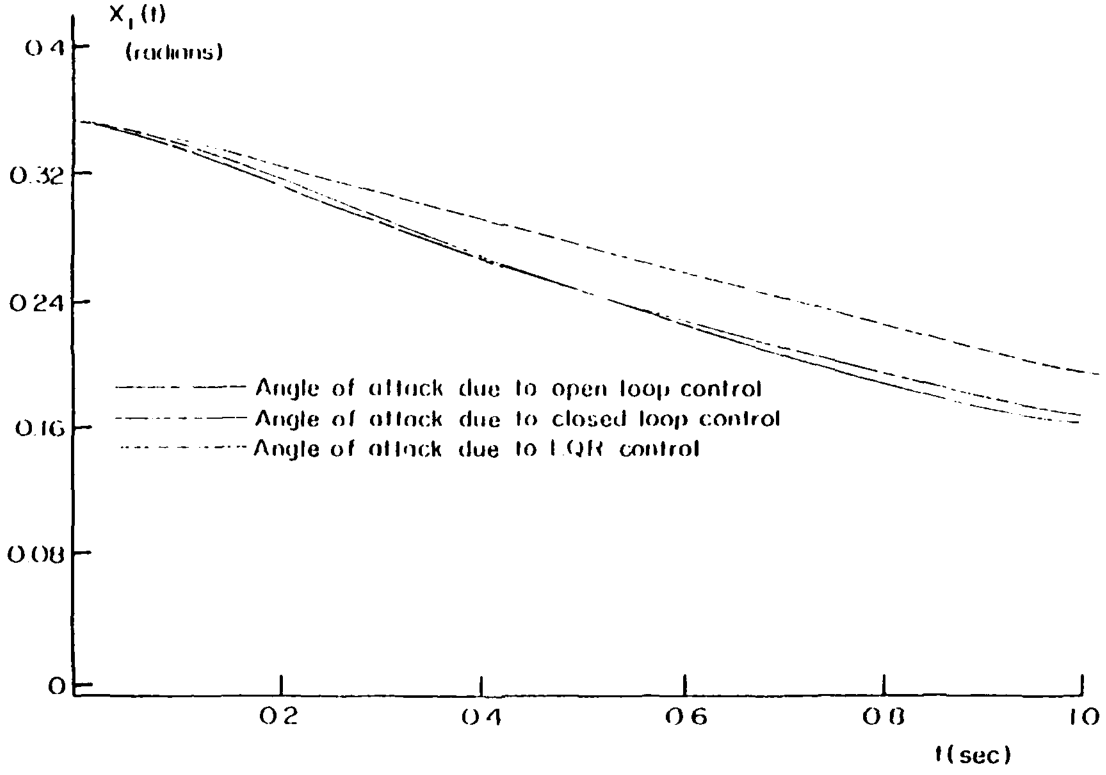


FIGURE 5.13: F-8 Aircraft: Angle of Attack Trajectories

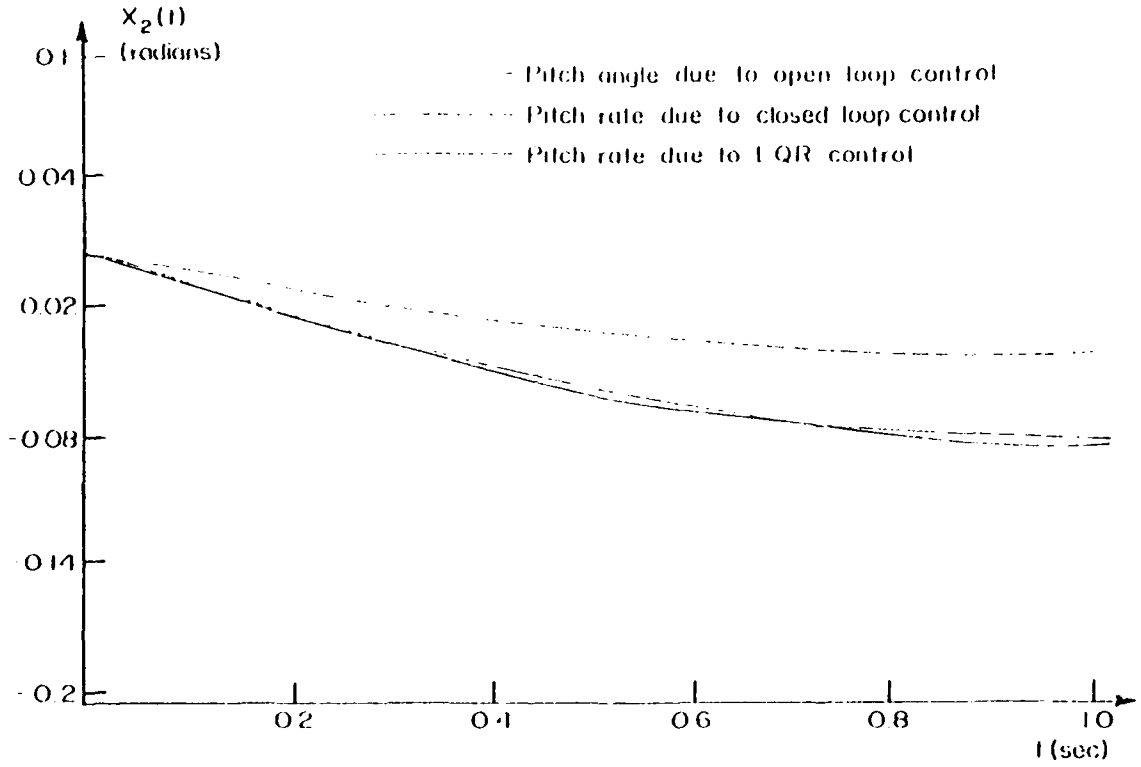


FIGURE 5.14: F-8 Aircraft: Pitch Angle Trajectories

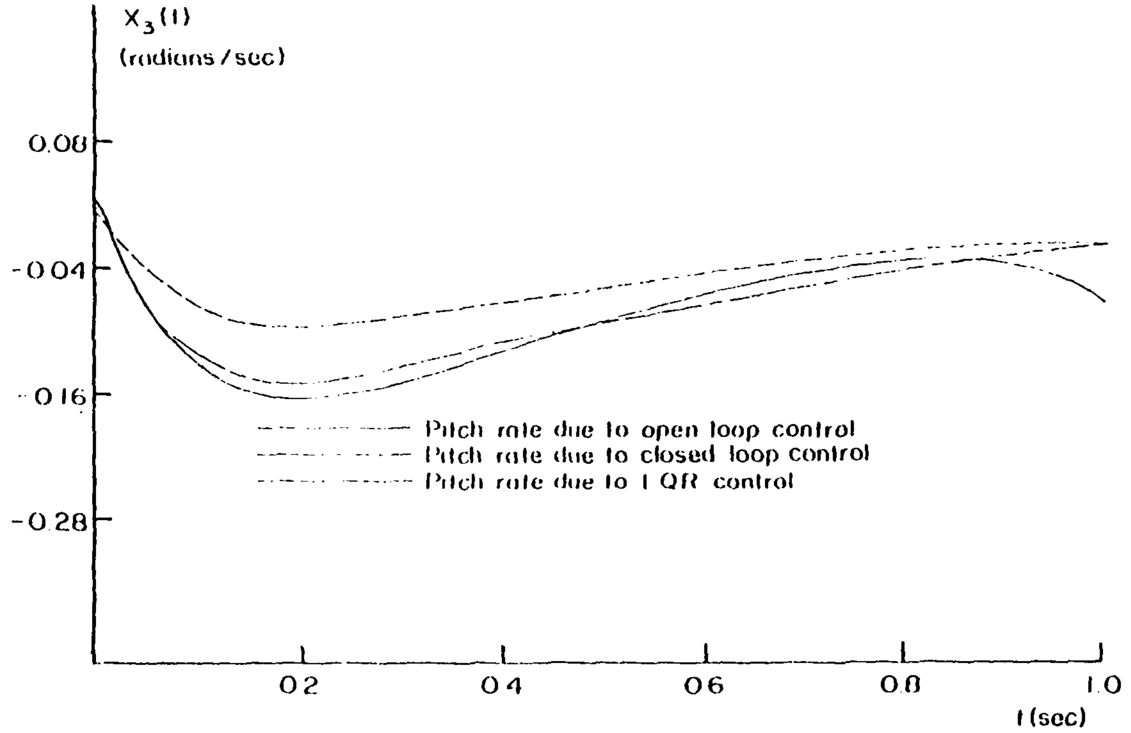


FIGURE 5.15: F-8 Aircraft: Pitch Rate Trajectories

that the closed loop control may be preferable in this case. Finally, it should be emphasized that the closed loop control was designed without approximating the nonlinear equations of motion of the F-8 aircraft. This attribute of the direct gain optimization procedure is studied further in the next section in which a number of feedback controllers are compared.

5.3.2 Feedback Control Laws

In this section, several feedback controllers are designed and compared at high angles of attack. The problem of interest this time is to determine the feedback control law $u(t) = g\,(x(t))$ which minimizes

$$J = \tfrac{1}{2}\int_0^5 [x^T Q\, x + r\, u^2]\ dt \qquad (5.3.2\text{-}1)$$

subject to the F-8 aircraft equations of motion given by eqns. (5.3-4) - (5.3-9). In this example, the initial state was $x_o = (0.575 \quad 0 \quad 0)$ and the matrix Q and the scalar r were selected as:

$$Q = \begin{bmatrix} 0.25 & 0. & 0. \\ 0. & 0.25 & 0. \\ 0. & 0. & 0.25 \end{bmatrix} \quad \text{and} \quad r = 1.0.$$

Note that both high and low angle of attack models are used in this case.

The control problem above was originally considered by Garrard and Jordan who used LQR and perturbation theory to design the following flight controllers [43]:

- Linear Control

$$u = -0.053\, x_1 + 0.5\, x_2 + 0.521\, x_3 \qquad (5.3.2\text{-}2)$$

- Quadratic Control

$$u = -0.053\ x_1 + 0.5\ x_2 + 0.521\ x_3 + 0.04\ x_1^2 - 0.048\ x_1\ x_2 \qquad (5.3.2\text{-}3)$$

- Cubic Control

$$u = -0.053\ x_1 + 0.5\ x_2 + 0.521\ x_3 + 0.04\ x_1^2 - 0.048\ x_1\ x_2 + 0.374\ x_1^3 - 0.31\ x_1^2\ x_2 \qquad (5.3.2\text{-}4)$$

Because these designs do not account for the quadratic and cubic control terms, as well as the cross terms involving x and u appearing in eqns.(5.3-4), (5.3-6), (5.3-7), and (5.3-9), it seems plausible that a better controller might be designed. To show this, suppose the controller is constrained to be of the form:

$$u(t) = K_1\ x_1(t) + K_2\ x_2(t) + K_3\ x_3(t) \qquad t \geq 0 \qquad (5.3.2\text{-}5)$$

If the direct gain optimization procedure described in Section 2.2.3 is employed, then the optimal gains (K_1, K_2, and K_3) could be found by initially setting them to zero and updating the values of K_1, K_2, and K_3 using an iterative scheme until the performance index (eqn. 5.3.2-1) is minimized. To speed computations, the parallel integration methods discussed in Section 3.2 may be used to integrate eqns.(5.3-4) - (5.3-9), while the selection of the next value of K_1, K_2, and K_3 may be made using one of the parallel minimization methods described in Section 3.1. Because the parallel numerical procedures can account for all the nonlinearities in eqns. (5.3-4), (5.3-6), (5.3-7), and (5.3-9), the optimized control law:

$$u(t) = 0.138\ x_1(t) + 0.385\ x_2(t) + 0.243\ x_3(t) \qquad t \geq 0 \qquad (5.3.2\text{-}6)$$

should be superior to the controllers defined by eqns. (5.3.2-2), (5.3.2-3), and (5.3.2-4).

To determine if this was indeed true, a simulation of the response of the F-8 aircraft to each feedback control law was conducted and the resulting trajectories were plotted (see Figures 5.16 - 5.19). Since the objective was to reduce an angle of attack disturbance to the origin as rapidly as possible, it is clear that the controller given by eqn.(5.3.2-6) is indeed superior to the others.

Also of interest was a comparison of the performance of the parallel and serial minimization methods discussed in Chapter Three. In Table 5.10, the parallel and serial methods are compared by counting the total number of iterations required for convergence to a set of control gains for the F-8 aircraft.* This is appropriate because the parallel algorithms require n + 1 gradients to be evaluated simultaneously using n + 1 processors and a univariate search per iteration while the serial algorithms require one gradient evaluation and a univariate search per iteration. Hence, the computer time needed per iteration by each method will be nearly the same provided the parallel operations are done simultaneously.

If we assume the parallel algorithms are executed on the parallel computer discussed in Section 4.1, and that the criterion above is used to compare the methods, then the results in Table 5.10 indicate that the parallel algorithms would require significantly less time to

* One cycle of either method consists of evaluating the gradient of eqn. (5.3.2-1), using this information to construct a direction of search, and performing a univariate search in this direction.

AD-A107 914 RENSSELAER POLYTECHNIC INST TROY NY DEPT OF ELECTRIC--ETC F/G 9/2
PARALLEL COMPUTATION FOR DEVELOPING NONLINEAR CONTROL PROCEDURE--ETC(U)
JUL 81 H KAUFMAN, R TRAVASSOS AFOSR-77-3418

UNCLASSIFIED AFWAL-TR-81-3016 NL

3 OF 3
AD
A107914

END
DATE
FILMED
1-82
DTIC

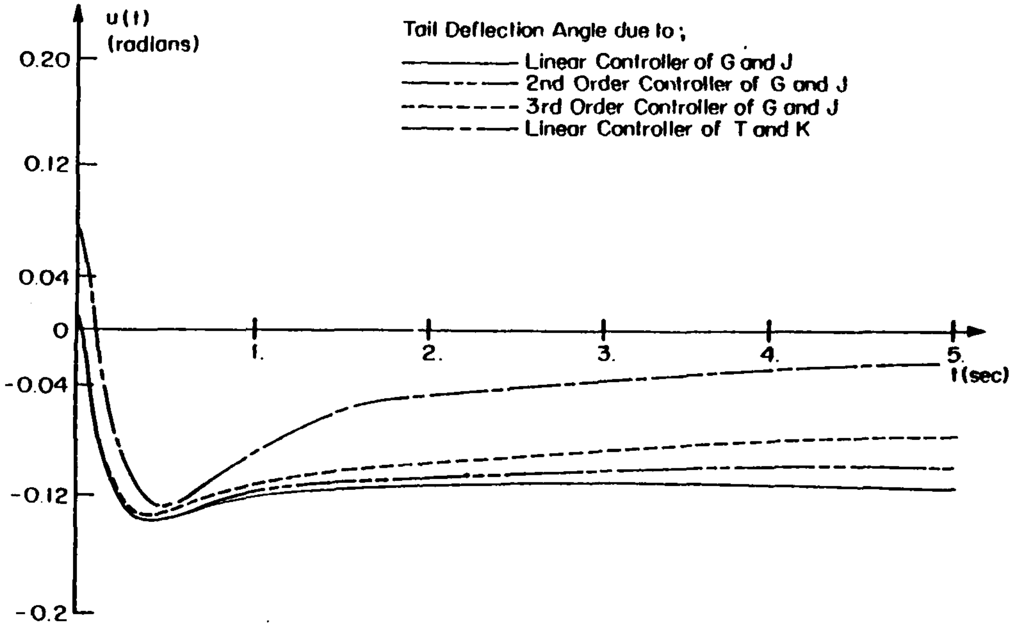


FIGURE 5.16: Tail Deflection Angle Trajectories

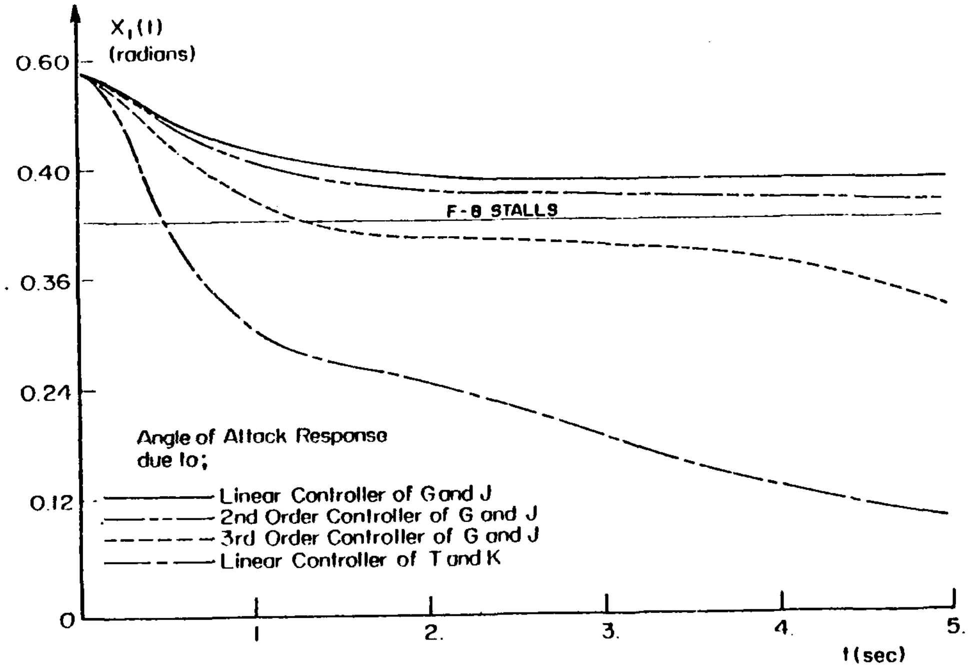


FIGURE 5.17: Angle of Attack Trajectories

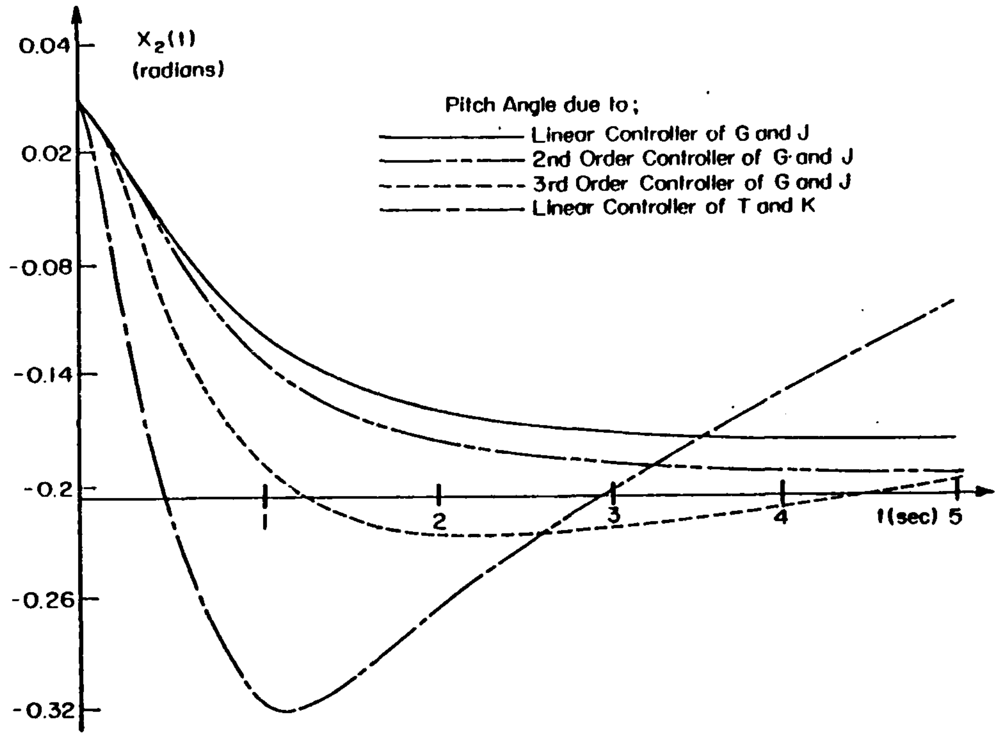


FIGURE 5.18: Pitch Angle Trajectories

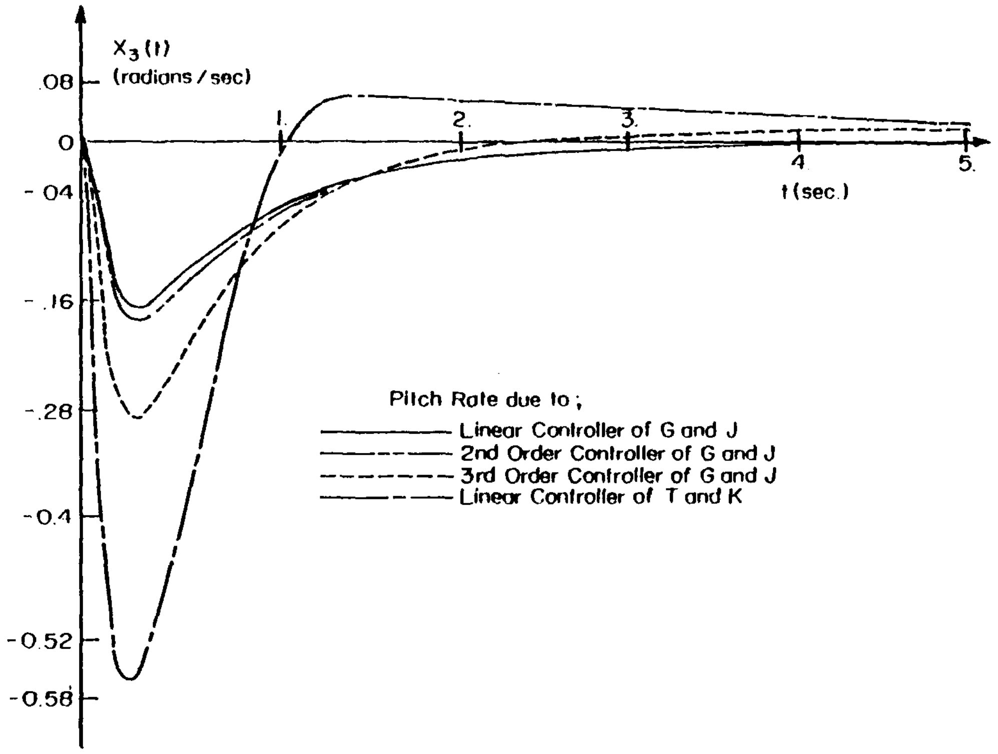


FIGURE 5.19: Pitch Rate Trajectories

TABLE 5.10: F-8 AIRCRAFT

Direct Gain Optimization at a High Angle of Attack

Minimization Algorithm	Total Number of Function Evaluations	Total Number of Gradient Evaluations	Total Number of Iterations	Resultant Optimal Gains	Optimal Cost J^*
ZP	155	—	8	0.13809 0.38485 0.24304	0.1013542
DFP	125	14	14	0.13808 0.38484 0.24301	0.1013542
BFS	95	11	11	0.13811 0.38487 0.24304	0.1013542
CM $c = 10^{-1}$	116	—	12	0.13809 0.38483 0.24302	0.1013542
PDFP $c = 10^{-3}$	77	10	9	0.13800 0.38475 0.24297	0.1013542
PBFS $c = 10^{-3}$	96	11	10	0.13791 0.38465 0.24288	0.1013542
FVM $c = 10^{-3}$	71	8	8	0.138732 0.385651 0.243310	0.1013542

Initial Performance Index: $J = 0.599863$

Initial State: $x(0) = (0.5759586 \quad 0 \quad 0)$

Initial Choice of Gains: $K = (0 \quad 0 \quad 0)$

execute compared with the serial algorithms. Also, the results in Table 5.10 show that the parallel methods used fewer function evaluations to achieve convergence.* Thus, it can be concluded that the parallel algorithms can be very effective in determining the control gains needed by flight control systems.

5.3.3 Timing Consideration

In this section, the execution time required for convergence to a set of optimal control gains will be estimated for the F-8 aircraft. First this will be done assuming a completely sequential algorithm is executed on a serial computer. Secondly, the execution time will be estimated for a completely parallel algorithm which is assumed to be executed on the parallel computer described in Section 4.1. Finally, these two estimates will be used to estimate the speed-up due to parallelism.

From the high angle of attack model of the F-8 aircraft, it is easily verified that $n = 6$, $A_{rhs} = 21$, and $M_{rhs} = 32$. If we let $N = 100$, $t_a = 200$ nsec, $t_m = 1000$ nsec, and $L = 8$, then for the DFP method with APC integration the execution time per iteration is 0.1277 seconds using eqn. (4.2.2-12). On the other hand, for the PVM algorithm with PPC42 integration, the execution time per iteration is 0.06316 seconds using eqn. (4.2.2-14). In this case, the speed-up due to parallelism is simply 0.1277/0.06316 = 2.02 which shows that one iteration of the parallel algorithms will execute about twice as fast as the serial algorithms. Note that a further reduction in computation time can be achieved if the RHS evaluations were performed by separate

* One function evaluation includes all arithmetic operations needed to evaluate eqn. (5.3.2-1).

processors. i.e., one processor for each state variable.

If this is considered at the expense of additional processors, then the execution time for one iteration of the PVM algorithm would be only 0.0106 seconds. This time the speed-up due to parallelism is 0.1277/0.0106 = 12.05, which is rather significant. Other possibilities, along with the processors required, are shown in Table 5.11.

To estimate the time required for convergence to a set of optimal gains, the results shown in Table 5.10 and Table 5.11 can be used. For example, the execution time of the DFP/APC and PVM/PPC42 algorithms could be estimated as follows. From Table 5.10, the serial DFP algorithm required 14 iterations to converge. Using this fact, and the fact that the execution time for one iteration of the DFP/APC algorithm requires 0.1277 seconds (see Table 5.11) when one processor is available, the execution time required for convergence is simply 14 x 0.1277 = 1.7878 seconds. On the other hand, if the PVM/PPC42 algorithm is executed using 8 processors, then the time required for convergence is only 8 x 0.0106 = 0.0848 seconds from the results shown in Table 5.10 and Table 5.11. Thus, if a completely parallel algorithm is executed on the parallel computer described in Section 4.1, the timing required for control computations might be rapid enough to permit adjustment of the control gains in real time. Finally, the advantage of using a completely parallel algorithm is further enforced by computing the speed-up due to parallelism based on the total time required for convergence. From the calculations above, the speed-up is 1.7878/0.0848 = 21.08 which indicates that the parallel algorithm converged more than 20 times faster than the serial algorithm.

TABLE 5.11: F-8 AIRCRAFT

Direct Gain Optimization Timing

$n = 6$, $L = 8$, $N = 100$, $A_{rhs} = 21$, $M_{rhs} = 32$, $t_a = 200$ nsec, $t_m = 1000$ nsec

Integration Method / Minimization Algorithm	APC (1PE/RHS)	APC (1PE/STATE)	PPC42 (2PE/RHS)	PPC42 (2PE/STATE)
ZP	0.5525/1	0.09215/7	0.2731/3	0.0456/15
DFP	0.1277/1	0.0215/7	0.06326/3	0.01076/15
BFS	0.1278/1	0.02156/7	0.06331/3	0.01081/15
CM	0.07651/7	0.01276/50	0.03781/22	0.00631/106
PVM	0.1276/7	0.0214/50	0.06316/22	0.0106/106
PDFP	0.1277/7	0.0215/50	0.06325/22	0.0107/106
PBFS	0.1279/7	0.0217/50	0.06345/22	0.0109/106

KEY: t_{pi}/PE's

5.4 Evaluation of Adaptive Controller Performance

The purpose of this section is to evaluate the performance of the explicit adaptive control scheme discussed in Section 2.3. This is accomplished by initially designing a feedback controller in Section 5.4.1 which will cause the F-8 aircraft to follow a nominal pitch rate command of 5°/sec based upon a nominal set of aerodynamic parameters. In Section 5.4.2, the effectiveness of the parallel algorithm is demonstrated by adjusting the feedback control gains on-line in response to a 10°/sec pitch rate command. Finally, in Section 5.4.3, the feedback control gains will be adapted in response to variations in the aerodynamic parameters of the F-8 aircraft using a moving window, explicit adaptive control scheme.

5.4.1 Gain Optimization

In this section, the problem is to find a feedback control law which causes the F-8 aircraft to follow a pitch rate command. The pitch rate command considered in this example was:

$$\dot{\Theta}_c = \begin{cases} 5^\circ/\text{sec} & t \, \varepsilon \, [0,\ 2] \\ 0 & \text{otherwise} \end{cases} \qquad (5.4.1\text{-}1)$$

and the state model (valid for low angles of attack) considered was:

$$\begin{aligned} \dot{x}_1 = {} & (1 - x_1^2 - 0.088\ x_1)\ x_3 - 0.877\ x_1 + 0.47\ x_1^2 \\ & + 3.846\ x_1^3 - 0.215\ u + 0.28\ u\ x_1^2 + 0.47\ u^2\ x_1 \\ & + 0.63\ u^3 - 0.019\ x_2^2 \end{aligned} \qquad (5.4.1\text{-}2)$$

$$\dot{x}_2 = x_3 \qquad (5.4.1\text{-}3)$$

$$\begin{aligned} \dot{x}_3 = {} & -0.396\ x_3 - 4.208\ x_1 - 0.47\ x_1^2 - 3.564\ x_1^3 \\ & - 20.967\ u + 6.265\ u\ x_1^2 + 46\ u^2\ x_1 + 61.4\ u^3 \end{aligned} \qquad (5.4.1\text{-}4)$$

$$\dot{x}_4 = 0.$$

where $x_1(t)$, $x_2(t)$, $x_3(t)$, and $x_4(t)$ represent the angle of attack, pitch angle, pitch rate, and pitch rate command, respectively.

The structure of the controller is shown in Figure 5.20 and the control law to be optimized is:

$$u(t) = K_1\, x_1(t) + K_2\, x_2(t) + K_3\, x_3(t) + G(x_4(t) - x_3(t)) \qquad (5.4.1\text{-}5)$$

Since the objective is to find the control gains K_1, K_2, K_3, and G which cause the pitch rate of the F-8 aircraft to track the pitch rate command, the following performance index was specified:

$$J = \tfrac{1}{2}\int_0^5 [x^T\, Q\, x + r\, u^2]\, dt \qquad (5.4.1\text{-}6)$$

where

$$Q = \begin{bmatrix} 0.25 & 0. & 0. & 0. \\ 0. & 0.25 & 0. & 0. \\ 0. & 0. & 1000.0 & -1000.0 \\ 0. & 0. & -1000.0 & 1000.0 \end{bmatrix} \quad \text{and } r = 1.0.$$

Initially, the unknown control gains were selected as $K_1 = -0.1$, $K_2 = -0.001$, $K_3 = -0.04$ and $G = -0.4$ since these values caused the F-8 aircraft to remain stable over the entire mission time interval [0, 5].

At this point, the direct gain optimization algorithm discussed in Section 2.2.3 was used to optimize the control gains assuming the initial state of the F-8 aircraft was $x_0 = (0\ 0\ 0\ 0.087)^T$. The performance of the different methods considered are shown in Table 5.12.

The results indicate:

- Each of the minimization procedures converged to a set of gains which cause the tracking error (performance index) to be reduced from $J = 0.635004 \times 10^3$ initially to $J = 0.559612 \times 10^3$ upon convergence.

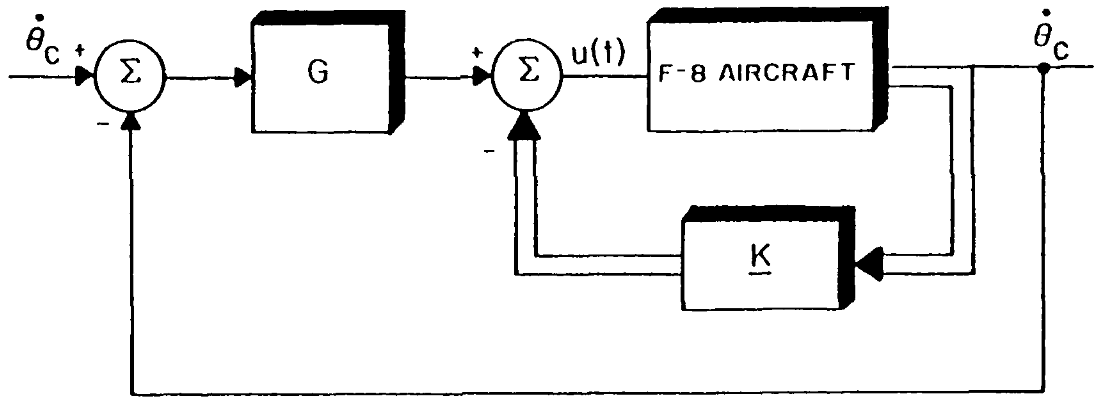


FIGURE 5.20: F-8 Aircraft Controller

- The PVM algorithm required nearly half the function and gradient evaluations to converge compared with the serial DFP algorithm.

By substituting the optimized control gains shown in Table 5.12 into eqn. (5.4.1-5), the optimized control law can be obtained as follows:

$$u = -0.128\ x_1 - 0.009\ x_2 - 0.046\ x_3 - 0.427\ (x_4 - x_3) \qquad (5.4.1\text{-}7)$$

To determine how well this controller would cause the F-8 aircraft to track a 5°/sec pitch rate command, a simulation was performed. The pitch rate command and the pitch rate response which resulted from applying the optimized control law (eqn. 5.4.1-5) are shown in Figure 5.21. Looking at Figure 5.21, it is clear that the pitch rate response of the F-8 aircraft tracked the pitch rate command relatively well. Thus, our nominal design is complete.

5.4.2 Adaptive Gain Optimization

In the previous section, the controller given by eqn. (5.4.1-5) is optimal only if the initial state of the F-8 aircraft is at the origin and a 5°/sec (0.087 radian/sec) pitch rate command is considered. If, however, the magnitude of the pitch rate command is different from 5°/sec, the gains will have to be reoptimized. Thus, it is of interest to determine how rapidly the parallel algorithms can adjust the feedback gains in response to a different pitch rate command.

To illustrate this, suppose the pitch rate command is 10°/sec (0.174 radians/sec), i.e., twice the magnitude of the nominal pitch rate, and the initial state of the F-8 aircraft is (0 0 0 0.174). The problem is then to reoptimize the control gains to account for the new pitch rate command.

TABLE 5.12: $\dot{\Theta}$ COMMAND FOLLOWING

$$\dot{\Theta}_c = \begin{cases} 5^o/\text{sec} & t \in [0, 2] \\ 0 & \text{otherwise} \end{cases}$$

$\hat{K}_0 = (-0.1 \quad -0.001 \quad -0.04 \quad -0.4)$ and $J^o = 0.635004 \times 10^3$

Minimization Algorithm	Total Number of Function Evaluations	Total Number of Gradient Evaluations	Total Number of Iterations	Resultant Optimal Gains	Resultant Cost
DFP	55	7	7	-0.128588 -0.00929 -0.04632 -0.42714	0.5596122×10^3
PVM	29	4	4	-0.12856 -0.00929 -0.04633 -0.42713	0.5596123×10^3

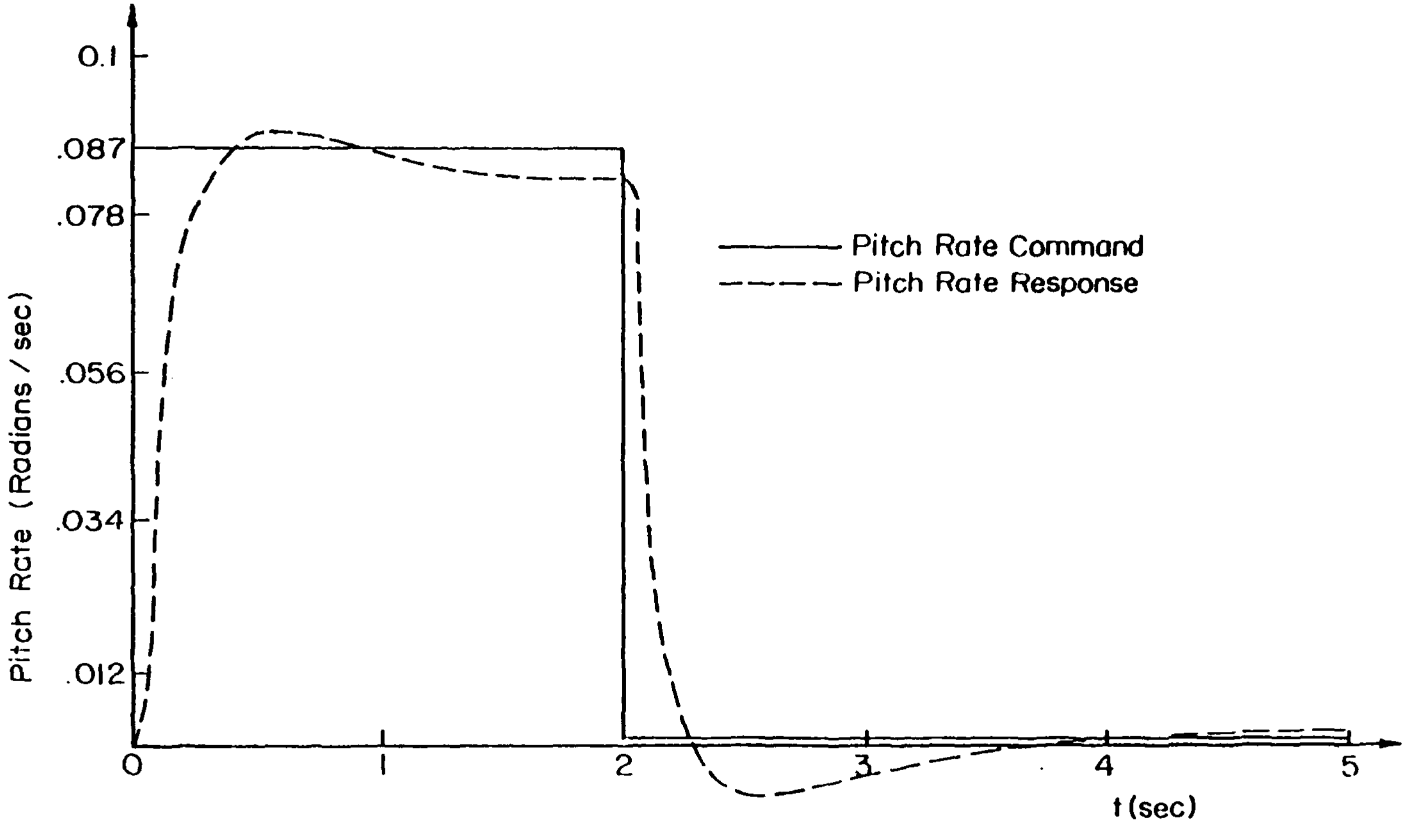


FIGURE 5.21: Pitch Rate Tracking

This was accomplished by performing only two iterations of the direct gain optimization algorithm which was initialized with the optimal gains for the nominal 5°/sec pitch rate command case. The results are shown in Table 5.13.

The results indicate that after only two iterations of the PVM algorithm, convergence to the optimal set of control gains is possible. However, if the DFP algorithm is utilized to adapt the control gains, the gains obtained after adaptation were still relatively far from optimal.

In view of these results, and the fact that the execution time required for one iteration of the parallel algorithms has been shown to be much shorter than the sequential methods, it appears that the parallel algorithms may be applicable to update the control gains in real time. This concept is pursued further in the next section.

5.4.3 Moving Window Adaptive Gain Optimization

As a final topic in this chapter, an explicit adaptive controller will be designed to stabilize the F-8 aircraft as the aircraft's center of gravity is moved aft during flight.

To determine the point at which the F-8 aircraft becomes unstable, the differential pitch rate was computed analytically using the expression:

$$d\,\dot{x}_3 = \frac{\partial x_3}{\partial \ell}\,d\ell \qquad (5.4.3\text{-}1)$$

where $d\ell$ is an incremental change in the distance between the wing aerodynamic center and the aircraft's center of gravity (see Figure 5.11).

TABLE 5.13: F-8 AIRCRAFT

Adaptive $\dot{\Theta}$ Command Following:

$$\dot{\Theta}_c = \begin{cases} 10^{\circ}/\text{sec} = 0.174 \text{ rad/sec} & t \,\varepsilon\, [0, 2] \\ 0 & \text{otherwise} \end{cases}$$

$\hat{K}_0 = (-0.1285 \quad -0.00929 \quad -0.04632 \quad -0.42714)$ and $x_0 = (0 \quad 0 \quad 0 \quad 0.174)$

Minimization Algorithm	Total Number of Function Evaluations	Total Number of Gradient Evaluations	Resultant Control Gains	Resultant Cost
DFP	18	2	-0.1346448 -0.00999048 -0.04676074 -0.4279358	2235.079
PVM	18	2	-0.1398054 -0.00908506 -0.0427181 -0.4292276	2234.032

$J^{o} = 2242.238$ $\qquad J^{*} = 2234.032$

$K^{*} = (-0.1397815 \quad -0.00909304 \quad -0.04272491 \quad -0.4292372)$

To evaluate eqn. (5.4.3-1), the pitch rate equation below was utilized.

$$\begin{aligned}\dot{x}_3 = {} & M_w/I_y + (C^0_{L_w} + C^1_{L_w} x_1 - C^2_{L_w} x_1^3 - C^0_{L_w} x_1^2/2 \\ & - C^2_{L_w} x_1^3/2)\ \bar{q}\ S/I_y - \{C^0_{L_t} + C^1_{L_t}(x_1 - \varepsilon_0 - a_e\ x_1 + u) \\ & - C^2_{L_t}\ (x_1 - \varepsilon_0 - a_e + u)^3 + a_e\ u\} \\ & \cdot (1 - (x_1 - \varepsilon_0 - a_e\ x_1 + u)^2/2)\ \bar{q}\ S_t\ \ell_t/I_y \\ & - c\ x_3/I_y \qquad (5.4.3\text{-}2)\end{aligned}$$

By substituting the aerodynamic data shown in Table 5.14 into eqn. (5.4.3-2) and using the fact that $\ell + \ell_t = 16.889$, it can be shown with some effort that

$$\begin{aligned}\frac{\partial \dot{x}_3}{\partial \ell} = {} & 5.27\ x_1 - 22.5\ x_1^3 + 1.23\ u - 0.614\ u^3 - 0.7748\ u\ x_1^2 \\ & - 3.24\ x_1\ u^2. \qquad (5.4.3\text{-}3)\end{aligned}$$

Substituting eqn. (5.4.3-1) into eqn. (5.3-6), the modified equations of motion of the F-8 aircraft become:

$$\begin{aligned}\dot{x}_1 = {} & (1 - x_1^2 - 0.088\ x_1)\ x_3 - 0.877\ x_1 + 0.47\ x_1^2 \\ & + 3.846\ x_1^3 - 0.215\ u - 0.019\ x_2^2 + 0.28\ u\ x_1^2 \\ & + 0.47\ u^2\ x_1 + 0.63\ u^3 \qquad (5.4.3\text{-}4)\end{aligned}$$

$$\dot{x}_2 = x_3 \qquad (5.4.3\text{-}5)$$

TABLE 5.14: F-8 AIRCRAFT DATA

Mach = 0.85   Altitude = 30,000 ft.

$C^0_{L_w}$	$= C^0_{L_1} = 0$
$C^1_{L_w}$	$= C^1_{L_1} = 4.0$
$C^2_{L_w}$	$= C^2_{L_1} = 12.0$
a_e	$= 0.1$
S	= 375 ft^2 (33.75 m^2)
S_t	= 93.4 ft^2 (8.41 m^2)
m	= 667.7 slugs (9773 kg)
a_e	$= 0.75$
ε_0	$= 0$
$C_{m_{a.c.}}$	$= 0$
$\bar{c}$	= 11.78 ft (3.53 m)
I_y	= 96,800 slug ft^2 (127,512 kg-m^2)
ℓ	= 0.189 ft (0.06 m)
ℓ_t	= 16.7 ft (5.01 m)
$\bar{q}$	= 318.0116

$$\begin{aligned}\dot{x}_3 = {} & -0.396\ x_3 + (5.27\ d\ell - 4.208)\ x_1 - 0.47\ x_1^2 \\ & - 3.564\ x_1^3 - 20.967\ u + 6.265\ u\ x_1^2 + 46.\ u^2\ x_1 \\ & + 61.4\ u^3 + (1.23\ u - 22.25\ x_1^3 - 0.614\ u^3 \\ & - 0.7748\ u\ x_2^2 - 3.24\ u^2\ x_1)\ d\ell \qquad (5.4.3\text{-}6)\end{aligned}$$

Note that if $d\ell = 0$, then eqns. (5.4.3-4) - (5.4.3-6) reduce to the low angle of attack model of the F-8 aircraft given by eqns. (5.3-4) - (5.3-6).

By increasing $d\ell$ incrementally from $d\ell = 0$ to $d\ell = 1.5$ in eqns. (5.4.3-4) - (5.4.3-6), the point at which the F-8 aircraft becomes unstable can be determined by monitoring the open loop response of the aircraft and determining when the response doubles in amplitude. From the open loop response, it was concluded that under nominal conditions ($d\ell = 0$), the aircraft is stable. However, as $d\ell$ is increased the aircraft became unstable for $d\ell \geq 1$.

In view of these results, the remainder of this section is concerned with the design of an explicit adaptive controller which will stabilize the F-8 aircraft as $d\ell$ is increased from $d\ell = 0$ to $d\ell = 1.5$. Since the direct adaptive control algorithm discussed in Section 2.2.3 must be initialized with a set of stabilizing gains, such a set of gains must be determined a priori based upon a set of nominal conditions.

As indicated earlier, the nominal conditions for the example under consideration are $d\ell = 0$ and a nominal initial state of $x_0 = (0.349 \quad 0 \quad 0)$. If we restrict the control to be linear of the form:

$$u(t) = K_1\ x_1(t) + K_2\ x_2(t) + K_3\ x_3(t) \qquad t \geq 0 \qquad (5.4.3\text{-}7)$$

then the problem is simply to find the control gains K_1, K_2 and K_3 which minimize a suitably defined performance index such as:

$$J = \tfrac{1}{2} \int_0^5 (x^T(t)\ Q\ x(t) + r\ u^2(t))\ dt \qquad (5.4.3\text{-}8)$$

subject to the F-8 aircraft's equations of motion described by eqns. (5.4.3-4) - (5.4.3-6). The Q and r matrices were selected as:

$$Q = \begin{bmatrix} 0.25 & 0. & 0. \\ 0. & 0.25 & 0. \\ 0. & 0. & 0.25 \end{bmatrix} \qquad \text{and } r = 1.0$$

since this choice of Q and r gave good response in previous examples when $d\ell = 0$. Because the open loop response of the F-8 aircraft is stable over the entire mission time inverval [0, 5], the control gains were initially set to zero.

At this point, the direct gain optimization procedure was employed to optimize the control gains. The resulting control law was determined to be:

$$u(t) = 0.1416\ x_1(t) + 0.8036\ x_2(t) + 0.6488\ x_3(t) \qquad t \geq 0 \qquad (5.4.3\text{-}9)$$

Now that the nominal design is complete, the optimized control gains $K_1 = 0.1416$, $K_2 = 0.8036$, and $K_3 = 0.6488$ can be used to initialize the adaptive control algorithm.

Before the direct adaptive control algorithm described in Section 2.3.2 can be utilized, the adaptation times $t_1, t_2, \ldots, t_N$ must be specified a priori. However, since the adaptation times are, in general, chosen somewhat arbitrarily, uniform adaptation intervals were considered. In particular, since the duration of the mission time is only five seconds, the adaptation times were selected as $t_1 = 1$, $t_2 = 2$, $t_3 = 3$, and $t_4 = 4$. Also, in the simulations it was assumed that $d\ell$ varies linearly from the stable condition ($d\ell = 0$), to the unstable condition ($d\ell = 1.5$) as follows:

$$d\ell(t) = 3/10\ t \qquad \forall\ t \in [0, 5] \tag{5.4.3-10}$$

Because only the effectiveness of the control update algorithm was being studied in this example, it was assumed that perfect estimates of $d\ell$ were available as needed. The adaptive control scheme was evaluated by performing one iteration of the direct gain optimization procedure assuming the actual values of $d\ell$ were available at the adaptation times.

To determine if the parallel algorithm could indeed optimize the control gains more rapidly than serial methods, the PVM and DFP algorithms were considered. The resul:s obtained are shown in Tables 5.15 and 5.16.

The results indicate that PVM algorithm could indeed reduce the performance index more rapidly than the sequential DFP method. This is more clearly revealed by summing the performance index values after adaptation for each method. For the PVM algorithm, this amounts to:

$$\sum_{i=0}^{4} J_i = 0.042$$

TABLE 5.15

Adaptive Controller Performance: DFP Algorithm

Time of Adaptation (sec)	Gains Before Adaptation	Gains After Adaptation	Performance Index Before Adaptation	Performance Index After Adaptation
0.0	0.1416 0.8036 0.6488	------	0.1646628×10^{-1}	------
1.0	0.1416 0.8036 0.6488	0.0755 0.8351 0.4643	0.1632203×10^{-1}	0.1418935×10^{-1}
2.0	0.0755 0.8351 0.4643	0.1904 0.7396 0.6578	0.1091726×10^{-1}	0.9644475×10^{-2}
3.0	0.1904 0.7396 0.6578	0.2844 0.61106 0.7839	0.4506108×10^{-1}	0.4247605×10^{-2}
4.0	0.2844 0.61106 0.7839	0.3682 0.3521 0.9175	0.1586995×10^{-2}	0.1446075×10^{-2}

TABLE 5.16

Adaptive Controller Performance: PVM Algorithm

Time of Adaptation (sec)	Gains Before Adaptation	Gains After Adaptation	Performance Index Before Adaptation	Performance Index After Adaptation
0.0	0.1416 0.8036 0.6488	------	0.1646628×10^{-1}	------
1.0	0.1416 0.8036 0.6488	-0.0285 0.2691 0.5537	0.1632203×10^{-1}	0.1517128×10^{-1}
2.0	-0.0285 0.2691 0.5537	-0.0295 0.2478 0.5053	0.7664331×10^{-2}	0.7661227×10^{-2}
3.0	-0.0295 0.2478 0.5053	0.4517 0.7492 0.5567	0.2256340×10^{-2}	0.2237675×10^{-2}
4.0	0.4517 0.7492 0.5567	0.9352 1.6978 1.1703	0.5040772×10^{-3}	0.5034828×10^{-3}

while for the DFP method the

$$\sum_{i=0}^{4} J_i = 0.046.$$

Note that a reduction in cost of approximately 10% may be realized if the parallel method is used in this case.

CHAPTER SIX

CONCLUDING REMARKS

In Section 1.2, a survey of existing parallel identification, estimation and control algorithms and an evaluation of their usefulness was made in terms of accuracy, speed, processor requirements, and numerical efficiency. From this survey, it was clear that the major problems with existing methods were the lack of accuracy and excessive computation time. Also, it was revealed that parallelism can be employed to alleviate such problems. Thus, the need for developing more efficient parallel procedures based upon modern nonlinear estimation and control theory was established. This fact led to the development of several identification, estimation and control algorithms which employ a high degree of parallelism but at the same time were not extravagant in the utilization of processing elements. Whereas most existing estimation and control algorithms had been designed using approximate linearized equations of motion, the parallel procedures developed in this thesis utilize the nonlinear process equations directly.

The nonlinear estimation and control algorithms developed in this thesis employ parallel minimization methods to accelerated convergence, parallel methods for integrating ordinary differential equations to facilitate computations, and a procedure based upon partitioning the integration interval to improve accuracy and reduce the sensitivity of the overall algorithm.

The major contributions which resulted from investigating each phase of the nonlinear estimation and control algorithms consisted of:

- Developing a class of parallel rank-two quasi-Newton methods for unconstrained minimization.
- Establishing a strategy for optimally selecting the number of subintervals and mesh points associated with the parallel shooting approach to solving nonlinear two-point boundary value problems.
- Developing a procedure which automatically adjusts the step size of a parallel predictor-corrector integration scheme to maintain a desired level of accuracy.
- Demonstrating with representative examples that the newly developed parallel algorithms do indeed perform better than existing sequential methods in terms of speed, accuracy, and reliability.
- Applying the PQN method, PVM algorithm and the CM method to solving dynamic optimization problems (such as nonlinear estimation and control problems) rather than static optimization problems involving algebraic functions.

The remainder of this chapter is divided into three sections. In Section 6.1, some conclusions are drawn based upon the results obtained as a consequence of conducting this research. In Section 6.2, some recommendations are made, and areas of future research are suggested in Section 6.3.

6.1 Conclusions

In this section, some conclusions are drawn based upon the analytical and empirical results obtained in Chapters Three, Four and Five.

From the results in Chapter Three, it can be concluded that without question the parallel minimization algorithms do indeed require significantly fewer iterations for convergence compared with serial methods (see Tables 3.1-3.6). In fact, it was shown analytically that if the PQN algorithm is utilized to minimize a quadratic function in n variables, then convergence to the location of the minimum is guaranteed in only one iteration provided n + 1 degrees of parallelism are employed (see Theorem 3.2 and Table 3.1). Since the PQN method was generally more robust than the PVM algorithm, this result suggests that parallel double-rank methods might be more robust than parallel rank-one methods (see Figures 3.1-3.6).

From the timing equations derived in Chapter Four and the timing results in Chapter Five, it was revealed that one iteration of the (parallel or serial) nongradient algorithms required much less time to execute than did the (parallel or serial) gradient-dependent methods, although more iterations of the nongradient methods were usually required for convergence (see eqns. (4.2.2-11) - (4.2.2-14) and Tables 5.8, 5.9, and 5.11). Also along these lines, it was shown that one iteration of the indirect control algorithm required significantly more time and processors to execute compared with the direct gain optimization procedure (see Tables 5.8 and 5.9). This observation was also valid for the indirect and direct SAP estimation algorithms as well.

From the simulations performed in Chapter Five, it can be concluded that although the gradient of a highly nonlinear function may be difficult at best to compute numerically, the convergence properties of the gradient-dependent algorithms were clearly preferable to the nongradient methods (see Tables 5.1-5.7). From the results shown in Figure 5.2, it was revealed that the robustness of the PVM algorithm was enhanced the most when parallel methods rather than serial methods were employed to integrate the state and costate equations associated with the Van der Pol system. This result was obtained using ordinary shooting. However, when parallel shooting was considered, the number of unknown boundary conditions which must be found was artificially increased from 2n to n(2N-1) where n is the order of the system and N is the number of subintervals. Despite this fact, as the integration interval is partitioned into many subintervals, the sensitivity of the solution will be reduced, and in general, the solution obtained will be more accurate. Unfortunately, since a high order optimization problem must be solved (i.e., n(2N-1) unknowns must be found), the number of iterations required for convergence increases as well (see Tables 5.6 and 5.7).

When the AMS algorithm was used to optimally select the mesh points required by the parallel shooting algorithm, it was revealed that the local truncation error could indeed be minimized, although many iterations were required here also. In fact, it was shown that a 20% improvement in accuracy was possible by employing the AMS algorithm (see Section 5.1.3).

From the SAP estimation results obtained in Section 5.2, it can be concluded that even if poor estimates of the unknown initial state and parameters of the T-33 aircraft were available initially, convergence to the true initial state and parameters was possible even when the measurement data was extremely noisy (see Table 5.5).

From the results obtained in Section 5.3, it can be concluded that the response of the F-8 aircraft could be improved significantly if the control was designed by employing the nonlinear control algorithms developed in Section 2.2. In particular, it was revealed that it was better to design a simple feedback control law using the F-8 aircraft's nonlinear equations of motion directly rather than to approximate the equations of motion and employ linear quadratic regulator (LQR) theory or utilize a more complex control law.

Finally, it can be concluded from the adaptive control results obtained in Section 5.3, that the direct gain optimization procedure might be implemented in an on-line adaptive type fashion. This follows from the fact that after only two iterations, the PVM algorithm converged to an optimal set of control gains while after two iterations of the serial DFP method, the control gains remained relatively far from optimal (see Table 5.13).

6.2 Recommendations

In view of the results obtained in this thesis, the following recommendations are in order.

- The weighting parameter, c, which defines a set of basis vectors for the PVM, PBFS, and PDFP methods, should be set to $c = 10^{-6}$ since this choice of c gave the best overall performance (see Figures 3.1-3.6).

- In view of the superior robustness characteristics of the PBFS method, it might be considered rather than the PDFP and PVM methods, although the PVM method did converge faster than the PBFS method in many test cases (see Tables 3.1-3.6).

- If convergence problems are encountered when using the PQN method to solve nonlinear estimation and control problems, the following modifications of the basic procedure are recommended:

1. Update the inverse Hessian, $H_j^{(\ell+1)}$, in Step 4 of the PQN algorithm only if $d_j^T y_j > 0 \quad \forall\ j = 1, 2, \ldots, n$. As indicated by Proposition 3.4, this modification will guarantee that the inverse update will be positive definite.

2. Replace Step 3 of the PQN method with the following:

   a. Compute n + 1 gradients of f(x) at n + 1 distinct points in parallel:

   $g(x^{(\ell)})$ and $g_j = g(x^{(\ell)} + c\ d_j) \qquad j = 1, 2, \ldots, n$

   b. Compute the gradient difference in parallel:

   $$y_j = (g_j - g(x^{(\ell)}))/c$$

In the above modification, c is the same weighting parameter used to define the basis set

$$\Sigma = (\sigma_1, \sigma_2, \ldots, \sigma_n) = c\ I_n; \qquad c > 0.$$

By performing computations in this manner, the gradients required can be computed more reliably because the forward integration of the state and costate equations will remain stable. Note that when f(x) is quadratic, $y_j = (c\ A\ d_j)/c = A\ d_j$ for the modified version of Step 3.

But since $y_j = A\ d_j$ in Step 3 originally, the remaining steps of the PQN method are unaffected by the modifications cited above.

In view of the results presented in Section 3.2.3, the PPC42V integration scheme is recommended for solving the required initial-value problems (IVP's) since the accuracy of the solution can be specified a priori. Also, because the PPC42V method has been designed to execute on separate processors, the solution to an IVP can be obtained extremely rapidly.

With regard to the nonlinear state and parameter (SAP) estimation algorithms, the direct SAP estimation algorithm should be used only if process noise is omitted from the state model. On the other hand, if process noise is included in the state model, then the indirect method should be considered. If sensitivity problems are encountered, the parallel shooting method with adaptive mesh selection has proven to be very effective in alleviating such problems.

With regard to the control algorithms, the direct gain optimization procedure is highly recommended in view of the fact that near optimal response was obtained without an excessive amount of computation (see Figures 5.6-5.10). Also, this method should be seriously considered because the equations of motion of a highly nonlinear system can be utilized directly in the control system design process.

6.3 Areas f Future Research

In this section, some areas of future research are suggested.

One aspect of the PQN method which could benefit from additional research is the generation of a set of mutually conjugate directions. In particular, alternate parallel methods should be considered for solving the linear system of equations required to generate the direction vectors. Since each row of the C_{m-1} matrix defined in Proposition 3.2 is known once m has been specified, the Gaussian Elimination procedure [22] might be modified to solve the resultant linear system in a row-wise fashion. Of course, this modification should be amenable to parallel computation.

Another area of future research might be the extension of the parallel variable step size integration method derived in Section 3.2.2 such that the order of the method, as well as the step size, can be automatically adjusted to maintain a desired level of accuracy. This concept was initially investigated by C. W. Gear in reference [46] although Gear's work was concerned with purely sequential methods at that time.

With regard to the parallel computer described in Section 4.1, future research should be conducted in the following areas:

- Specifying processor add, multiply and transfer times to permit real time estimation and control.
- Estimating memory size and peripheral requirements.
- Studying the effects of wordsize.
- Analytically modeling the reliability of the proposed design and studying the effects of component failures (such as one of the processing elements).
- Determining the feasibility of implementation and cost.

Another possibility might be to develop a parallel nonlinear estimation and control algorithm based upon Hamilton-Jacobi-Bellman (HJB) theory [47]. To illustrate how the control algorithm might be arranged, consider the optimal control problem:

$$\min_u \quad V(x_o, t_o) = \phi(x(t_f), t_f) + \int_{t_o}^{t_f} L(x, u, t) \quad dt \tag{6.3-1}$$

subject to

$$\dot{x} = f(x, u, t) \qquad\qquad t \in [t_o, t_f] \tag{6.3-2}$$

If we assume that $x(t_o) = x_o$ is known, t_f is specified and $x(t_f)$ is unspecified, then the HJB equation which must be satisfied is:

$$\frac{\partial V(x, t)}{\partial t} + L(x, u, t) + \left[\frac{\partial V(x, t)}{\partial x}\right]^T f(x, u, t) = 0 \tag{6.3-3}$$

The boundary condition associated with eqn. (6.3-2) is simply:

$$V(x(t_f), t_f) = \phi(x(t_f), t_f) \tag{6.3-4}$$

By defining the Hamiltonian as:

$$H(x, u, \lambda, t) = L(x, u, t) + \lambda^T(t) f(x, u, t) \tag{6.3-5}$$

then it can be shown that the adjoint variable, $\lambda(t)$, is given by $\lambda(t) = \partial V(x, t)/\partial x$. From the maximum principle, it is well known that the optimal controls must satisfy the necessary condition $\partial H/\partial u = 0$. If this condition can be solved explicitly for $u(t)$, the control will be of the form:

$$u(t) = h[x(t), \lambda(t), t] \tag{6.3-6}$$

But since $\lambda(t) = \partial V(x, t)/\partial x$, the optimal control is:

$$u(t) = h[x(t), \partial V(x, t)/\partial x, t] \tag{6.3-7}$$

By substituting eqn. (6.3-7) into eqn. (6.3-3), the result is:

$$\frac{\partial V(x, t)}{\partial t} + L(x, h[x, \frac{\partial V}{\partial x}, t], t)$$

$$+ \left[\frac{\partial V(x, t)}{\partial x}\right]^T f(x, \frac{\partial V}{\partial x}, t) = 0 \qquad (6.3\text{-}8)$$

The problem then is to find the continuous function V(x, t) which satisfies eqn. (6.3-8) and the boundary condition (6.3-4) subject to the dynamic constraint (6.3-2).

In view of the above problem formulation, an appropriate error function might be:

$$E = \tfrac{1}{2}\int_{t_o}^{t_f} e^2(t) \quad dt \qquad (6.3\text{-}9)$$

where

$$e(t) = L(x, h[x, \frac{\partial V}{\partial x}, t], t)$$

$$+ \left[\frac{\partial V}{\partial x}\right]^T f(x, h(x, \frac{\partial V}{\partial x}, t), t) + \frac{\partial V}{\partial t}$$

Note that if the time functions V(x, t) can be found such that eqn. (6.3-9) is identically equal to zero and the constraints given by eqn. (6.3-2) and eqn. (6.3-4) are satisfied, then we would have a solution to the original optimal control problem For computation reasons, however, V(x, t) is usually approximated by a power series of the form:

$$V(x, t) = \sum_{j=1}^{n} c_j x_j + \frac{1}{2!} \sum_{j=1}^{n} \sum_{k=1}^{n} c_{jk} x_j x_k$$

$$+ \frac{1}{3!} \sum_{i=1}^{n} \sum_{j=1}^{n} \sum_{k=1}^{n} c_{ijk} x_i x_j x_k + \ldots.. \qquad (6.3\text{-}10)$$

where the c's are time functions which must be determined. However, because the c's are functions of time, the problem at hand is more difficult that it appears. One way to overcome this difficulty is to approximate the c's using a Taylor series as follows:

$$c(t) = d_0 + d_1(t - t_o) + \tfrac{1}{2} d_2(t - t_o)^2 + O((t - t_o)^3) \qquad (6.3\text{-}11)$$

where the d's are constants which must be determined.

Thus, the problem has been converted to one of finding a set of constants rather than time varying unknowns. Since we are now confronted with solving a finite-dimensional minimization problem, the parallel minimization algorithms discussed in Section 3.1 can be used to optimize the d's in eqn. (6.3-11). Also, the parallel integration methods described in Section 3.2 may be used to integrate eqn. (6.3-2) which is necessary to evaluate the error function (6.3-9).

On the basis of the results obtained in this thesis, it is felt that the parallel Hamilton-Jacobi-Bellman (PHJB) method outlined above should also benefit a great deal from the use of parallelism. Since the control gains obtained by this method are, in general, time varying, the PHJB method should provide better control than the direct gain optimization procedure presented in Section 2.2.3. Of interest then, would be a comparison of the response of a given system due to the control laws designed by each method along with the number of processors required to implement each procedure. Using this information, a trade-off could then be made between the number of processors and the response of a given system.

Finally, it is hoped that these remarks and the encouraging results obtained in this thesis motivate future research in this area.

LITERATURE CITED

1. Nyquist, H., "Regeneration Theory," Bell Systems Journal, Vol. 11, 1932, pp. 126-147.

2. Bode, M. W., Network Analysis and Feedback Amplifier Design, Van Nostrand, Reinhold, NY, 1945.

3. Fisher, R. A., "On An Absolute Criterion for Fitting Frequency Curves," Messenger of Math, Vol. 41, 1912, p. 155.

4. Kalman, R. E., "A New Approach to Linear Filtering and Prediction Problems," J. Basic Eng., March 1960, pp. 35-46.

5. Gauss, K. F., Theoria Motus, 1809; also in Theory of the Motion of the Heavenly Bodies About the Sun in Conic Sections, Dover, New York, 1963.

6. Gelb, A., et al., Applied Optimal Estimation, MIT Press, Cambridge, MA, 1974, pp. 335-348.

7. Roberts, S. M., and J. S. Shipman, Two-Point Boundary Value Problems: Shooting Methods, American Elsevier, New York, 1972.

8. Miele, A., and R. R. Iyer, "Modified Quasilinearization Method for Solving Nonlinear Two-Point Boundary Value Problems," Math Anal. and Appl., Vol. 36, No. 3, 1971, pp. 674-692.

9. Guderley, K. G., "A Unified View of Some Methods for Stiff Two-Point Boundary Value Problems," SIAM Review, Vol. 12, No. 4, 1975, pp. 416-442.

10. Larson, R. E., and E. Tse, "Parallel Processing Algorithms for the Optimal Control of Nonlinear Dynamic Systems," IEEE Trans. Comp., Vol. C-22, No. 8, August 1973, pp. 777-786.

11. Larson, R. E., and E. Tse, "Parallel Processing Algorithms for Modal Trajectory Estimation," Proc. JACC, Stanford, CA, 1972, pp. 758-767.

12. Falk, Howard, "Reaching for a Gigaflop - What Went Wrong," IEEE Spectrum, October 1976, pp. 64-73.

13. Reid, J. G., "A New Parallel Identification Algorithm for Linear Time-Invariant Systems," Second Annual Workshop on the Information Linkage Between Applied Mathematics and Industry, Monterey, CA, February 1979.

14. Heller, D., "A Survey of Parallel Algorithms in Numerical Linear Algebra," SIAM Review, Vol. 20, No. 4, October 1978, pp. 740-777.

15. Miranker, W. L., "A Survey of Parallelism in Numerical Analysis," SIAM Review, Vol. 13, No. 4, October 1971, pp. 524-547.

16. Reid, J. G., "Structural Identifiability in Linear Time Invariant Systems," IEEE Trans. on Auto. Control, Vol. AC-22, April 1977, pp. 242-246.

17. Kailath, T., et al., "Square-Root Algorithms for Parallel Processing in Optimal Estimation," IFAC Journal, 1979, pp. 299-306.

18. Fargeon, C., et al., "Performance of Parallel Algorithms for Parameter and State Estimation," IFAC Conference Proceedings, 1978, pp. 2139-2145.

19. Sage, A. P., and J. L. Melsa, System Identification, Academic Press, New York, 1971, pp. 45-51.

20. Sage, A. P., and J. L. Melsa, Estimation Theory with Applications to Communications and Control, McGraw Hill, New York, 1971, pp. 441-445.

21. Keller, H. B.,"Numerical Methods of Two-Point Boundary Value Problems,"SIAM Monograph, 1976, pp. 13-15.

22. Dahlquist, G., and A. Björck, Numerical Methods, Prentice-Hall, Englewood Cliffs, New Jersey, 1974.

23. Pontryagin, L. S., et al., The Mathematical Theory of Optimal Processes, Interscience Publishers, Inc., New York, 1962.

24. Isaacs, D., et al., "On a Sequential Optimization Approach in Nonlinear Control," Proc. JACC, Seattle, WA, June 1966, p. 158.

25. Chazan, D., and W. L. Miranker,"A Nongradient and Parallel Algorithm for Unconstrained Minimization," SIAM J. Control, Vol. 8, No. 2, May 1970, pp. 207-217.

26. Straeter, T. A., "A Parallel Variable Metric Optimization Algorithm," NASA TN D-7329, December 1973.

27. Straeter, T. A., and A. T. Markos, "A Parallel Jacobson-Oksman Optimization Algorithm," NASA TN D-8020, September 1975.

28. Zangwill, W. I., "Minimizing a Function Without Calculating Derivatives," Comp. J., Vol. 10, 1967, pp. 293-296.

29. Broyden, C. G., "The Convergence of a Class of Double-Rank Minimization Algorithms - Parts I and II," J. Inst. Maths. Applics., Vol. 6, pp. 76-91, and pp. 222-232.

30. Jacobson, D. H., and W. Oksman, "An Algorithm that Minimizes Homogeneous Functions in N Variables in N+2 Iterations and Rapidly Minimizes General Functions," Tech. Rep. No. 618, Div. Eng. and Appl. Phys., Harvard Univ., October 1970.

31. Fletcher, R., and M. J. D. Powell,"A Rapidly Convergent Descent Method for Minimization, Comp. J., Vol. 6, No. 2, July 1963, pp. 163-168.

32. Fletcher, R., "A New Approach to Variable Metric Algorithms," Comp. J., Vol. 13, No. 3, August 1970, pp. 317-322.

33. Dixon, L. C. W., "Conjugate Directions Without Linear Searches," J. Inst. Maths. Applics., Vol. 11, 1973, pp. 317-328.

34. Dixon, L. C. W., "The Choice of Step Length, a Crucial Factor in the Performance of Variable Metric Algorithms," in: Numerical Methods for Nonlinear Optimization, F. A. Lootsma (ed.), Academic Press, New York, 1972, pp. 149-170.

35. Himmelblau, D. M., "A Uniform Evaluation of Unconstrained Optimization Techniques," in: Numerical Methods for Nonlinear Optimization, F. A. Lootsma (ed.), Academic Press, New York, 1972, pp. 69-97.

36. Rudin, W., Principles of Mathematical Analysis, McGraw Hill, New York, 1976, pp. 14-15.

37. Downs, H. R., "Parallel Computation of the Solution of a Differential Equation," Hawaii Systems Conference, 1970.

38. Miranker, W. L., and W. M. Liniger, "Parallel Methods for the Numerical Integration of Ordinary Differential Equations," Math Comp., Vol. 21, 1967, pp. 303-320.

39. Burden, et al., Numerical Analysis, Prindle, Weber, and Schmidt Publishers, Boston, 1979, pp. 254-255.

40. Hale, J. K., Ordinary Differential Equations, Wiley-Interscience, New York, 1969, pp. 193-195.

41. Pease, M. C., "Matrix Inversion Using Parallel Processing," J. Assoc. Comp. Mach., Vol. 14, 1967, pp. 757-764.

42. Kaufman, H., "Aircraft Parameter Identification Using Kalman Filtering, Nonlinear Estimation and Control Conference, Chicago, December 1969, pp. 85-89.

43. Garrard, W. L., and J. M. Jordan, "Design of Nonlinear Automatic Flight Control Systems," Automatica, Vol. 13, 1977, pp. 497-505.

44. Kirk, D., Optimal Control Theory, Prentice Hall, Englewood Cliffs, NJ, 1970.

45. Armstrong, E. A., "ORACLS - A System for Linear-Quadratic-Gaussian Control Law Design," NASA Technical Paper 1106, April 1978.

46. Gear, C. W., Numerical Initial Value Problems in Ordinary Differential Equations, Prentice Hall, Englewood Cliffs, NJ, 1971.

47. Sage, A. P., Optimum Systems Control, Prentice Hall, Englewood Cliffs, NJ, 1968, pp. 75-83.

APPENDIX

PARALLEL MINIMIZATION PROCEDURES

In this appendix, the Chazan-Miranker method, the parallel variable metric (PVM) algorithm due to Straeter, and the parallel Jacobson-Oksman (PJO) procedure reported by Straeter and Markos are given for reference. These methods are useful in minimizing a function f: $R^n \to R^1$ which is assumed to be continuous and differentiable in each variable. The gradient of f will be denoted as the function g: $R^n \to R^n$. With these preliminary remarks, the parallel minimization methods can be presented formally as follows:

Chazan-Miranker Procedure

Let ℓ represent the iteration number and define the following quantities:

- UP $\triangleq \{up_1, up_2, \ldots, up_n\}$ = a set of n linearly independent unit vectors.
- β_ℓ, $\ell = 1, 2, \ldots, \triangleq$ a sequence of positive scalars tending to zero.
- $WP_\ell^n \triangleq up_i$ if $\ell \triangleq i \bmod n$ where $i = 1, 2, \ldots, n$
- PT_ℓ^1, $\ell = 1, 2, \ldots, \triangleq$ a sequence of n vectors
- $V_{\ell+j}^j$, $j = 1, 2, \ldots, n$, $\ell = 1, 2, \ldots, \triangleq$ a sequence of n vectors called search direction vectors.

Then the value of x which minimizes f(x) may be obtained by performing the following steps:

Step 1:

Determine the scalars, $\alpha^j_{\ell+1}$, by performing simultaneously the univariate minimizations

$$\min_{\alpha^j_{\ell+1}} \quad f(w_\ell + \alpha^j_{\ell+1}\, s_{\ell+1}) \qquad j = 1, 2, \ldots, n$$

where

$$w_\ell \triangleq PT^1_\ell + \sum_{i \le j} v^i_{\ell+i}$$

$$s_{\ell+1} \triangleq v^1_{\ell+1}/||v^1_{\ell+1}||$$

Step 2:

Update the n vector, PT, such that:

$$PT^1_{\ell+1} = PT^1_\ell + (1 + \alpha^1_{\ell+1})\, v^1_{\ell+1}/||v^1_{\ell+1}||$$

Step 3:

Compute $f(PT^1_{\ell+1})$ and terminate the algorithm if:

$$|f(PT^1_{\ell+1}) - f(PT^1_\ell)|$$

is sufficiently small; otherwise, continue to Step 4.

Step 4:

Update the search direction n vectors, such that:

$$v^j_{\ell+j+1} = (\alpha^{j+1}_{\ell+1} - \alpha^j_{\ell+1})\, v^1_{\ell+1}/||v^1_{\ell+1}|| + v^{j+1}_{\ell+j}$$

$j = 1, 2, \ldots, (n-1)$

Step 5:

Update the n^{th} search direction vector by selecting one of the linearly independent unit vectors from UP as follows:

$$v^n_{\ell+n+1} = \beta_{\ell+n+1}\, WP^n_{\ell+1}$$

where

$WP^n_{\ell+1}$ is chosen cyclically from the set UP.

Set $\ell \leftarrow \ell + 1$ and return to Step 1.

Parallel Variable Metric Algorithm

Let ℓ denote the iteration number and define:

- $\Sigma \triangleq \{\sigma_1, \sigma_2, \ldots, \sigma_n\}$ as a set of n linearly independent vectors.
- V_0 as any positive definite nxn matrix; typically $V_0 = I_n$.

Then the value of x which minimizes f(x) may be obtained by performing the following steps:

Step 1:

a. Evaluate the function and its gradient at n distinct points simultaneously in parallel.

$$f(x_\ell + \sigma_j) \quad \text{and} \quad g_j = g(x_\ell + \sigma_j)$$

$$\forall\ j = 1, 2, \ldots, n$$

b. Compute $g_j^T g_j$ and terminate the algorithm if

$$g_j^T g_j \qquad j = 1, 2, \ldots, n$$

is sufficiently small; otherwise, continue to Step 2.

Step 2:

a. Compute $y_j \triangleq g_j - g(x_\ell) \qquad j = 1, 2, \ldots, n$

b. Compute the residual vectors*

$$r_j \triangleq V_{\ell-1}\, y_j - \sigma_j - \sum_{k=1}^{j-1} \frac{r_k^T y_j}{r_k^T y_k}\, r_k \qquad j = 2, \ldots, n$$

* If the denominator in step 2b is zero for any term, that term is deleted from the sum.

where

$$r_1 = V_{\ell-1}\, y_1 - \sigma_1$$

c. Compute the n scalars, τ_j, and modify the metric:

$$\tau_j \triangleq \begin{cases} -(y_j^T\, r_j)^{-1} & \text{for } y_j^T\, r_j \neq 0 \\ 0 & \text{otherwise} \end{cases}$$

$$V_\ell = V_{\ell-1} + \sum_{j=1}^{\ell\ n} \tau_j\, r_j\, r_j^T \qquad j = 1, 2, \ldots, n$$

Step 3:

a. Determine the scalar α_ℓ by performing a single univariate search.

$$\min_{\alpha_\ell} f(x_\ell + \alpha_\ell\, s_\ell)$$

where

$$s_\ell \triangleq -\, V_\ell\, g(x_\ell)$$

b. Update the n vector x, such that

$$x_{\ell+1} = x_\ell + \alpha_\ell\, s_\ell$$

c. Compute $f(x_{\ell+1})$ and $g(x_{\ell+1})$ simultaneously in parallel and terminate the algorithm if

$$||g(x_{\ell+1})||^2$$

is sufficiently small; otherwise, set $\ell \leftarrow \ell + 1$ and return to Step 1.

Parallel Jacobson-Oksman Procedure

Let ℓ denote the iteration number and define:

$\Sigma \triangleq \{\sigma_1, \sigma_2, \ldots, \sigma_{n+1}\}$ as a set of n+1 linearly independent vectors

Then the value of x which minimizes f(x) may be obtained by performing the following steps.

Step 0:

Let x_0 be the initial estimate of the minimum of f(x) and compute $f(x_0)$ and $g(x_0)$. Set $\ell = 0$.

Step 1:

Define:

$$\tilde{x}_j = x_\ell + \sigma_j \qquad j = 1, 2, \ldots, n+1$$

and evaluate $f(x_j)$ and $g(x_j)$ in parallel.

Step 2:

Set $\tilde{x}_{n+2} = x_\ell$ and solve the linear system:

$$C\,\alpha = v$$

where

$$C_{ij} \triangleq \begin{cases} \dfrac{\partial f(\tilde{x}_i)}{\partial x_j} & i = 1, 2, \ldots, n+2 \\ & j = 1, 2, \ldots, n \\ f(\tilde{x}_i) & i = 1, 2, \ldots, n+2 \\ & j = n+1 \\ -1 & i = 1, 2, \ldots, n+2 \\ & j = n+2 \end{cases}$$

$$\alpha = [\tilde{\beta}^T, \tilde{\gamma}, \tilde{\omega}]^T$$

and

$$v_j = \tilde{x}_j^T g(\tilde{x}_j) \qquad j = 1, 2, \ldots, n+2$$

Step 3:

Compute the search direction vector:

$$s_\ell = \tilde{\beta} - x_\ell$$

and evaluate $f(\tilde{\beta})$ and $g(\tilde{\beta})$. If $||g(\tilde{\beta})||$ is sufficiently small, stop. If not, and if $f(\tilde{\beta}) < f(x_\ell)$, then set $x_{\ell+1} = \tilde{\beta}$, $\ell \leftarrow \ell+1$, and return to Step 1. Otherwise, perform a line search:

$$\min_{\lambda} f(x_\ell + \lambda\, s_\ell)$$

and set $x_{\ell+1} = x_\ell + \lambda\, s_\ell$, $\ell \leftarrow \ell+1$ and go to Step 1.

FILMED
8