Finite Horizon Predictive Control of Chemical Processes

APPROVED BY
DISSERTATION COMMITTEE:

Supervisor: ______________________________

This dissertation was prepared with the LaTeX document preparation system using additional macros written by John W. Eaton based on the thesis style written by Khe-Sing The for The University of Texas at Austin.

TeX is a trademark of the American Mathematical Society.

This dissertation was edited and composed on a VAXStation 3200 running the Ultrix operating system.

VAXStation and Ultrix are trademarks of Digital Equipment Corporation.

Printed in the United States of America.

10 9 8 7 6 5 4 3 2 1

To my parents, for their love and understanding,

and to Melissa, for convincing me that I would actually finish it.

Finite Horizon Predictive Control of Chemical Processes

by

JOHN WESLEY EATON, B.S.

DISSERTATION
Presented to the Faculty of the Graduate School of
The University of Texas at Austin
in Partial Fulfillment
of the Requirements
for the Degree of

DOCTOR OF PHILOSOPHY

THE UNIVERSITY OF TEXAS AT AUSTIN
May 1992

Acknowledgments

I would not have been able to finish this work without the help and encouragement of my thesis advisor, Professor James B. Rawlings. I am especially grateful that he allowed me the time to explore many interests only vaguely related to my research topic.

I would like to thank the members of my committee, Professors Thomas F. Edgar, David M. Himmelblau, Aristotle Arapostathis, and Robert H. Bishop, for providing guidance and many useful comments during the course of this work. Dr. Karl J. Åström provided many insights during the year he spent as a visiting professor at the University of Texas.

My fellow graduate students have also been very helpful. I would especially like to thank Andy Fordyce, Steve Miller, and Mike Liebman for helping with all sorts of problems related to research and other trivia, and for listening to me rave about my latest computing discoveries.

Thanks to Steve Swinnea for pointing out who wrote and recorded *Changed the Locks*, and for suggesting a place other than Texadelphia for dinner.

Thanks also to Glass Eye, Grains of Faith, and Zeitgeist for all the years of great music.

Finally, special thanks are reserved for Noel Bell for introducing me to UNIX.[1]

John Wesley Eaton

The University of Texas at Austin
May 1992

[1]UNIX is a registered trademark of AT&T.

Finite Horizon Predictive Control of Chemical Processes

Publication No. ____________

John Wesley Eaton, Ph.D.
The University of Texas at Austin, 1992

Supervisor: James B. Rawlings

This dissertation presents a strategy for model-predictive control of processes modelled by nonlinear differential-algebraic equations. This strategy makes use of a repeated optimization of an open-loop performance objective over a finite time horizon. The manipulated variable profile determined from the solution of the open-loop optimization is implemented until a plant measurement becomes available.

Two nonlinear programming approaches for solving the open-loop optimal control problem are examined in detail, and example problems are solved using both methods. The first approach eliminates the dynamic constraints by solving them as a subproblem. The second second approach uses orthogonal collocation on finite elements to convert the differential equations to algebraic constraints that are solved directly by standard nonlinear programming software.

Feedback is incorporated into the algorithm by using the measurement to update the optimization problem for the next time step. Several methods for updating the optimization problem are discussed, including resetting the model's states to measured or estimated values, estimating model parameters, or inferring output disturbances.

Table of Contents

List of Tables

List of Figures

Nomenclature

Some of the following symbols are used for a number of different purposes in the text. The local meaning should be clear from the context.

lower case symbols

a	—	Real RHP zero location.
b	—	Nucleation order in the crystallizer model.
c	—	Controller transfer function.
d	—	Estimated disturbance in the DMC algorithm.
d^*	—	Vector of projected disturbances over the prediction horizon in the DMC algorithm.
$\mathbf{d}$	—	Disturbance inputs.
e	—	Error, $r - y$ in Figure 1.3, error vector, $r - y_0 - d^*$ in Figure 3.1.
f	—	Crystal number density.
$\mathbf{f}$	—	Residual functions of the process model.
g	—	Growth order in the crystallizer model, plant transfer function.
$\mathbf{g}$	—	Output relationship for the process model.
$\mathbf{h}$	—	Physical constraints on the process model.
k_b	—	Crystal nucleation rate constant.
k_g	—	Crystal growth rate constant.
k_1	—	Constant in the substrate inhibition model.
k_m	—	Constant in the substrate inhibition model.
k_v	—	Volume constant in the crystallizer model.
m_{s0}	—	Initial mass of seeds in the crystallizer model.
p	—	Ratio of activation energies in the CSTR model.
$\mathbf{p}$	—	Model parameter vector.
q^{-1}	—	Backward shift operator.
r	—	Characteristic length of crystals in the crystallizer model, reference input.
s	—	Laplace transform variable, substrate concentration

t — Time.
u — Manipulated variable or input.
$\mathbf{u}$ — Manipulated variable vector.
w — Lagrange multipliers.
x — Cell mass concentration in the substrate inhibition model, process state.
$\mathbf{x}$ — Model state vector.
y — Process output.
$\tilde{y}$ — Model output in the DMC algorithm.
$\tilde{y}^*$ — Vector of model outputs over the prediction horizon (Figure 3.1).
y_0 — Vector of projected outputs over the prediction horizon (Figure 3.1).
$\mathbf{y}$ — Plant output vector.

UPPER CASE SYMBOLS

A — Dynamic matrix.
A_1 — First column of the dynamic matrix.
$\mathbf{A}$ — Matrix of first derivative collocation weights.
B — Crystal birth (nucleation) rate.
C — Control horizon, concentration.
$\mathbf{C}$ — Controller.
D — Dilution rate.
$\mathbf{E}$ — Estimator
E_b — Activation energy for crystal nucleation
E_g — Activation energy for crystal growth
F — Final time state penalty parameter, flow rate.
G — Crystal growth rate.
$\mathbf{G}$ — Plant.
H — Unit step function, Hessian matrix.
K — Feedback gain matrix.
L — Lagrangian function.
M — Plant model, mass of solvent in the crystallizer model.
P — Plant.

Q — State (or output) penalty parameter.
R — Input penalty parameter, gas constant.
S — Shift matrix in the DMC algorithm.
T — Temperature, prediction horizon.
$\mathbf{W}$ — Matrix of sensitivity derivatives for DAE model.

Greek Symbols

α — Ratio of preexponential factors in CSTR model.
ε — Extrapolation vector in the matrix S in the DMC algorithm.
θ — Time of setpoint change.
θ_p — Process time delay.
μ — Specific growth rate in the substrate inhibition model.
ρ — Density.
Γ — Input penalty parameter in the DMC algorithm.
Λ — Output penalty parameter in the DMC algorithm.
Φ — Controller performance objective function.

Superscripts

$*$ — Optimum.
$\hat{}$ — Model prediction.
T — Transpose.

Subscripts

0 — Initial time.
f — Final time.
i — Index of output.
k — Index of sampling time.
n — Nucleated crystals.
s — Seed crystals.

Chapter 1

Introduction

This dissertation presents a strategy for model-predictive control (MPC) of processes modelled by nonlinear differential-algebraic equations. MPC is distinguished from other control schemes by the use of a repeated optimization of an open-loop performance objective over a finite horizon. The manipulated variable profile determined from the solution of the open-loop optimization is implemented until a plant measurement becomes available. Feedback is incorporated by using the measurement to update the optimization problem for the next time step. Several of the more commonly used methods for updating the optimization problem include resetting the model's states to measured values, or estimating model parameters, states, or output disturbances.

This definition is emphasized because the distinction is important in what follows and other definitions have appeared in the literature. García *et al.* [25], for example, define model-predictive control as "that family of controllers in which there is a direct use of an explicit and separately identifiable model." This definition is much too general for our purposes as it includes many standard feedback controllers in which there is no horizon.

Because the MPC controller incorporates the solution of an open-loop optimal control calculation over a future horizon, it has an advantage over standard feedback controllers in that it is able to anticipate future setpoint changes (if available) and possible constraint violations. The remainder of this chapter provides an overview of this dissertation and motivates the discussion of the model-predictive controller.

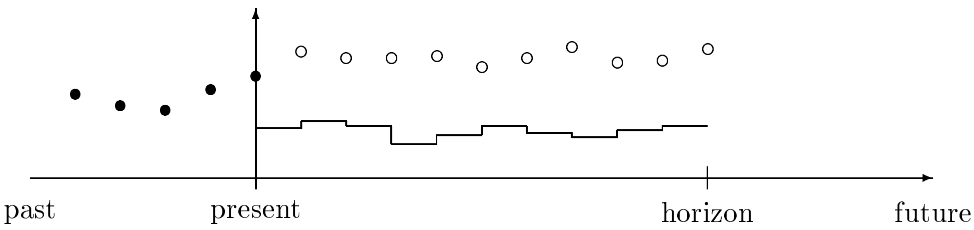

Figure 1.1: Prediction horizon.

1.1 Generic Predictive Controller Problem Statement

In its most general form, the MPC controller solves two separate problems, though each may be represented as a nonlinear program (NLP) with constraints in the form of a dynamic model. The first is an open-loop optimal control calculation over some future time horizon, based on any information available up to the current time. The form of the performance objective is not restricted. Any useful function may be used, allowing for the solution of batch or continuous time problems.

$$\min_{\text{inputs}} \quad \text{Performance Objective} \tag{NLP1}$$

subject to:

$$\text{Dynamic Process Model}$$

In this work, the prediction horizon is considered to extend from the current time to some finite time in the future, as illustrated by Figure 1.1. There are computational advantages to be gained by choosing a finite horizon, but there are a number of problems that can be avoided if the horizon is allowed to extend to infinity. These issues are discussed in Chapter 3.

The second subproblem that the MPC controller must solve is an update of the controller each time a new open-loop optimal control problem is to be solved. There are many possible methods that may be used to update the controller including disturbance, state, and parameter estimation. The update may be written as a nonlinear program that is very similar to the controller subproblem

$$\min_{\text{parameters}} \quad \text{Modelling Error} \tag{NLP2}$$

subject to:

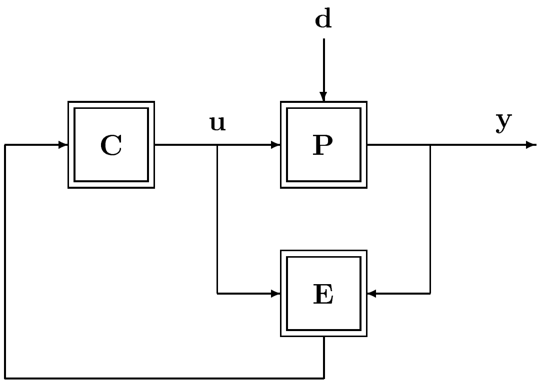

Figure 1.2: Feedback Structure.

Dynamic Process Model

The two separate subproblems are shown in Figure 1.2 where **P** is the process to be controlled, **C** is a controller that computes the solution to an open-loop optimal control problem over some future horizon, and **E** is estimator of some form, providing the controller with a feedback signal based on the inputs and outputs of the process. Each of the blocks may be nonlinear and the information provided by the feedback signal and exactly how it is used to update the open-loop optimization depends on the specific choice of the controller and estimator pair.

Given some estimate of the process model, the overall algorithm may be started by solving the open-loop optimal control calculation to determine the sequence of inputs that minimizes some performance index over some future time horizon. A portion of this control profile is implemented, and the process outputs are measured. The new information obtained from the process is then used to update the open-loop optimization. The update may be achieved by estimating model parameters, states, or disturbances.

It is common to return any remaining difference between the model prediction and the plant output to the controller as an additive output disturbance, even when other forms of estimation are used. This form of disturbance estimation is similar to the internal model control (IMC) approach of García and Morari [23]. In the simplest case, this is the only estimation that is performed.

In the most general case, the open-loop optimal control calculation and the method used to update it may both involve the solution of nonlinear optimization problems subject to differential-algebraic constraints. There are a number of techniques available for solving nonlinear programs subject to constraints of this form, though there appears to be no single method or implementation that can be considered clearly superior.

Two common methods based on nonlinear programming are examined in detail in Chapter 2. The first involves the elimination of the dynamic constraints by solving them as a subproblem and using the result to evaluate the objective function and other additional constraints (such as bounds on the process output). It should be emphasized that the modelling equations are constraints in the optimization and that eliminating these constraints in this manner can be the source of number of difficulties that are not present in the original problem statement. Constraint elimination is discussed in detail in Chapter 2.

The second method uses orthogonal collocation on finite elements to approximate the differential equation constraints as a set of algebraic equations that are solved simultaneously with the rest of the optimization. This technique requires the solution of a much larger nonlinear program, but for every additional decision variable introduced by the collocation technique, an additional equality constraint is also added. Even with the increased dimensionality, this technique can be significantly more efficient than constraint elimination.

In addition, by using an optimization approach coupled with a weighted residual solution technique, it is possible to obtain estimates of the sensitivities of the optimal solution to model parameters with only a small increase in computational effort. It is also possible to compute the sensitivity of the performance index to the manipulated variables using derivative information obtained from the solution of the nonlinear program.

1.2 Differences Between MPC and Standard Feedback Control

For linear systems, the feature that distinguishes MPC from standard feedback control is the explicit use of a prediction horizon in the control law formulation. In order to see this distinction clearly, consider the standard single input, single output block diagram in Figure 1.3. One of the characteristics of this structure is that the controller's response to zero error is zero control action,

$$e(t') = 0,\ t' \in [0,t] \implies u(t) = 0 \tag{1.1}$$

On the other hand, the prediction horizon allows the MPC controller to take control action at the current time in response to a forecast of a future error

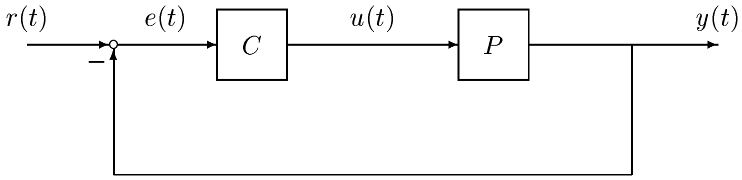

Figure 1.3: Standard feedback structure.

even though the error up to the current time is zero. In other words, Equation 1.1 does not hold for linear MPC controllers. The discussion in Section 3.3 demonstrates that it is precisely this property of the MPC controller that is beneficial for controlling (or, more precisely, scheduling) nonminimum phase plants and processes subject to input constraints.

This dissertation emphasizes the types of behavior that can be achieved with predictive control that are unattainable with standard feedback. This improved performance is only possible because the predictive controller is given information about future constraints and future inputs such as planned setpoint changes or forecasts of disturbances. Such information is often available: setpoint changes and throughput or raw material changes may be known in advance. Actuator saturation limits are always known in advance. The predictive control structure is nicely suited for using this kind of information, and will make the eventual goal of scheduling and optimization of the entire plant easier to attain.

1.3 Previous Work

Model-predictive control, as defined above, has appeared in several branches of the control literature during the past thirty years. The concept of using an open-loop optimal control computation to synthesize a feedback controller is so natural that it probably occurred to many researchers in the optimal control field in the late 1950s and 1960s. In their textbook on optimal control, Lee and Markus describe the approach while pointing out that current (as of 1967) hardware and software make real time implementation of the controller difficult. In their review, García *et al.* [25] cite Propoi [55] as the first to introduce, explicitly, the finite moving horizon.

In the electrical engineering literature, the term receding (or moving) horizon control is used to refer to model-predictive control with the additional constraint that the state must be zero at the end of the prediction horizon.

In one of the earliest results appearing in the literature, Kleinman [35] uses the finite horizon concept to find a state feedback gain that stabilizes linear time-invariant systems. Thomas [67] formulates a quadratic objective function penalizing only the input with the explicit constraint that the state must be zero at the end of the horizon. He shows this formulation results in a state feedback that stabilizes linear time-invariant systems. Kwon and Pearson [37] generalize these results by considering the linear time-varying system,

$$\begin{aligned} \dot{x} &= A(t)x + B(t)u \\ y &= C(t)x \end{aligned}$$

and using the standard quadratic performance objective evaluated over a finite prediction horizon, $t_0 \leq t \leq t_0 + T$,

$$\Phi = \int_{t_0}^{t_0+T} \left(y^T Q(t) y + u^T R(t) u \right) dt \tag{1.2}$$

again including the constraint that $x(t_0+T) = 0$. They show that the receding horizon controller can stabilize linear time-varying systems. Kwon *et al.* [36] also consider linear time-varying systems with the quadratic objective

$$\Phi = \int_{t_0}^{t_0+T} \left(x^T Q(t) x + u^T R(t) u \right) dt + x^T F x \Big|_{t_0+T}$$

and show the solution to this problem is also a stabilizing state-feedback law. Mayne and Michalska [42] consider the quadratic objective function with final time constraint for the nonlinear system,

$$\dot{x} = f(x, u)$$

They show that under certain conditions, the receding horizon controller stabilizes the nonlinear system.

The use of MPC in the chemical engineering field started in the process industries in the 1970s under the names of "Model Predictive Heuristic Control" or "Model Algorithmic Control" (MAC) [62, 43] and "Dynamic Matrix Control" (DMC) [14, 53]. These algorithms make use of step response models for the efficient minimization of a quadratic performance objective. García and Morshedi [24] introduced "Quadratic Dynamic Matrix Control" (QDMC), an extended DMC algorithm that includes linear input and output constraints with the minimization solved as a quadratic program.

The recent review by García *et al.* [25] concludes that MPC has found wide acceptance in industrial practice, has flexible constraint handling capabilities, and can be adjusted rather easily for robustness. They also note that because of the finite horizon, DMC and other related algorithms are not guaranteed to produce stabilizing controllers and that they are generally tuned by trial and error methods.

Rawlings and Eaton [57] show through simulations that quadratic dynamic matrix control is effective in solving a linear multivariable control problem with unmeasured disturbances and uncertainty in the process gains. Using additional information provided about the disturbances that can be expected to affect the system, they are able to incorporate an estimation scheme to improve the performance of the controller.

Most of the algorithms discussed to this point are some variation of MPC given a linear process model. Linear models are widely used because they are often easier to obtain than nonlinear models[1] (the model used in DMC can be obtained directly from step response data) and there exist particularly efficient algorithms for solving the quadratic programs that result from the use of quadratic cost functions and linear models.

The use of full nonlinear models in a model-predictive controller is not particularly new, although only recently has it been reasonable to consider this as a practical alternative, primarily because of to the tremendous decrease in the cost of computer hardware in recent years. However, current algorithms cannot guarantee that a global optimum will be found, and it is usually more difficult to formulate and test nonlinear models.

Due to the high computational cost of such algorithms, a number of researchers have proposed using these nonlinear optimization methods in a supervisory role for plant-wide optimization. For example, Jang *et al.* [31], proposed using a "two-phase" approach on line as a feedback controller for chemical processes. Parameters for a nonlinear model of the process are determined from past input-output data. Using these parameters, the operating conditions of the plant are optimized and setpoints determined for lower level controllers. Even here it is assumed that the computational power required to control the plant operation directly is unavailable.

Eaton *et al.* [17] and Eaton and Rawlings [16] develop a somewhat different approach, in which the parameters obtained from the estimation step are used in a model-predictive controller at each sampling time to provide low-level control using nonlinear optimization techniques. Although these methods may require significant computing resources, the emphasis is on the performance of the algorithms as feedback controllers rather than the efficiency of a particular implementation. The assumption is that advances in computer technology will

[1] This is a commonly held religious belief, at least.

soon make these methods attractive.

In each of these approaches, the central idea is the combination of a model based controller with some sort of parameter and disturbance estimation scheme. In the most general formulation, the computation of the future control moves and the estimation of parameters involve the solution of a nonlinear optimization subject to differential-algebraic constraints. Though the problem to be solved is in general the same, it is usually approached differently depending upon the application. In the optimal control literature this class of problems is typically solved using Pontryagin's maximum principle, or by directly integrating the differential equations within a minimization procedure. In the parameter estimation literature, however, the direct approach is used almost exclusively. Either method may be used to solve this class of problems, and the specific advantages and disadvantages of both methods are explored in Chapter 2.

Recently, a number of algorithms based on QDMC have been proposed that attempt to handle nonlinear systems at what is assumed to be a reduced computational cost in comparison to nonlinear model-predictive control (NMPC). The central theme is the use of some linearization technique to obtain a step response model that can be used by the standard QDMC algorithm. Peterson *et al.* [50] and Zafiriou and Gattu [26] modify the QDMC algorithm by integrating a nonlinear model to predict the effect of past inputs over the prediction horizon while using a linearized model to compute the contribution of future inputs. A linear state-space model obtained at each time step is used to generate step-response coefficients.

Incorporating a nonlinear model in this fashion will only provide an advantage over the standard QDMC algorithm if the process behavior changes significantly in the normal operating region, and a reduced computational cost is the only possible advantage over NMPC. Because it is usually fairly difficult to obtain nonlinear process models, it seems that if one had a nonlinear model, it would be desirable to take full advantage of it.

The MPC framework is also used in aerospace engineering applications. Most publications concern aircraft trajectory optimization [7, 33]. Attention is presently being given to solving nonlinear, constrained optimal trajectories over finite horizons for real-time guidance [4], Jänsch and Paus [32], and Psiaki and Park [56].

1.4 Summary of this Dissertation

The remainder of this dissertation includes the development of techniques for solving the nonlinear optimal control problems described above. These methods are then applied to several examples that illustrate the properties of the algorithms.

Chapter 2 focuses on the formulation and solution of the optimal control problem required for the application of NMPC. Both sequential and simultaneous model solution and optimization strategies are described, and example problems are solved using the simultaneous optimization and solution methods. A number of problems with constraint elimination are discussed, and a method is developed for ensuring that evaluation of the model equations is always possible. In addition, sensitivity analysis of the nonlinear program is used to determine the sensitivity of the optimal solution of the dynamic optimization with respect to changes in model parameters, and an expression is developed for determining bounds on the manipulated variables corresponding to a given change in the performance index.

The properties of feedback systems based on these optimal control techniques are discussed in Chapter 3. A particular implementation of MPC for linear systems is examined, and a new, clear and concise block diagram is developed for the QDMC algorithm, from which it is possible to obtain the closed-loop transfer function for this algorithm in the form $y = \mathbf{G}(q)\mathbf{r}$, where $\mathbf{r}$ is the vector of future setpoints given to the model-predictive controller, $\mathbf{G}$(q) is a vector of transfer functions in terms of the shift operator, q, and y is the process output.

Also in Chapter 3, the full nonlinear solution strategy is applied to an open-loop unstable nonlinear system that exhibits multiple steady-state behavior. It is demonstrated that the method of feedback applied to update the controller calculation is critical to the performance of the controller.

Chapter 4 includes a final example that clearly illustrates some of the unresolved issues related to NMPC, including the difficulty of finding a global optimum during the calculation of the control moves and problems related to closed-loop stability.

When this work began, no reliable software was available to solve the open-loop optimal control problem using a simultaneous optimization and model solution approach. One of the goals of this work was to develop and evaluate such software. Appendix A describes a set of programs written in Fortran-77 to solve the nonlinear optimal control problem and simulate the response of the actual process in the closed-loop. The appendix also describes another pair of programs written in C to simulate the behavior of the QDMC controller for any user-defined process.

Must number four: You must get started.

— James R. Cook, *The Start-up Entrepreneur* (1986)

Chapter 2

Solution of the Open-Loop Control Problem

As stated in the introduction, the distinguishing feature of nonlinear model-predictive control is the repeated solution of a nonlinear open-loop optimal control calculation over some future time horizon. This chapter examines two nonlinear programming approaches to the solution of the open-loop control problem and discusses the advantages and disadvantages of both methods. Many of the problems that arise in solving the controller subproblem, particularly those that are related to the accuracy of the solution, may not be terribly important in the closed-loop sense—the controller and estimator together must be able to handle some amount of modelling error to be useful, and suboptimal solutions may be considered to be another form of modelling error.

The purpose of describing these problems is simply to acknowledge the fact that these methods are not foolproof, to point out some of the difficulties that are commonly encountered, and to suggest possible solutions.

2.1 The Controller Subproblem

The goal of the model-predictive controller is to determine the best set of inputs over some future horizon given the best estimate of current conditions. In this chapter it is assumed that the process model is the most accurate available, and that future setpoints and constraints are well known. Given this information, the the problem to be solved by the controller may be stated as

$$\min_{\mathbf{u}(t)} \quad \Phi(\mathbf{x}(t), \mathbf{y}(t), \mathbf{u}(t)) \tag{NLP3}$$

subject to:

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}(t), \mathbf{y}(t), \mathbf{u}(t))$$
$$0 = \mathbf{g}(\mathbf{x}(t), \mathbf{y}(t), \mathbf{u}(t))$$

$$\mathbf{h}_l(t) \leq \mathbf{h}(\mathbf{x}(t), \mathbf{y}(t), \mathbf{u}(t)) \leq \mathbf{h}_u(t)$$

with

$$t_0 \leq t \leq t_0 + T$$

and

$$\mathbf{x}(t_0) = \mathbf{x}_0 \qquad \mathbf{y}(t_0) = \mathbf{y}_0$$

where $\mathbf{x}$ and $\mathbf{y}$ are the states of the process model, but only the time derivatives of $\mathbf{x}$ appear explicitly in the model, and $\mathbf{u}$ is a vector of manipulated variables. A parameter vector, $\mathbf{p}$, could also be added but we will assume that any additional model parameters are *internal* to $\mathbf{f}$ and $\mathbf{g}$ as part of the function definitions, unless it is important for them to be listed explicitly, *e.g.*, for the computation of parametric sensitivities. The vector-valued functions $\mathbf{f}$ and $\mathbf{g}$ define the process model and the vector-valued function $\mathbf{h}$ defines the physical constraints of the process. These functions are written as path constraints, though they may include constraints at specific points as well. The prediction horizon spans the interval from the current time, t_0, to some future time t_0+T.

The modelling equations have been written in the form of a set of semi-implicit differential-algebraic equations (DAE) rather than in the fully implicit form

$$\mathbf{f}(\dot{\mathbf{x}}(t), \mathbf{x}(t), \mathbf{u}(t)) = 0 \tag{2.1}$$

because the semi-implicit form is sufficiently general for the purpose of exploring the properties of the feedback, and because it greatly simplifies the implementation of the simultaneous solution technique described below.

For most problems, the manipulated variables, $\mathbf{u}(t)$, are the only degrees of freedom in the optimization, though one may imagine additional decision variables for some specialized problems, such as the determination of initial conditions or a minimum final time. These additional degrees of freedom often arise in the solution of batch problems.

The physical constraints on the process may include simple bounds or other constraints on the process inputs and outputs. Simple bounds on the inputs often arise from actuator limits and the model-predictive controller is generally capable of handling them given an open-loop stable process. For open-loop unstable processes, it is clearly possible to add constraints that will make the system uncontrollable.

The desire to include output constraints usually stems from the need to assert some sort of quality control specification. While hard output constraints may be able to help meet such goals, they should be avoided unless there is some method included in the algorithm that allows the controller to detect such infeasibilities and remove constraints, or convert the hard constraints to penalties, until a feasible solution can be found. The removal or relaxation of some constraints is not easy to do in a general fashion because it is not possible to determine which constraints should be relaxed without some knowledge of the specific problem being solved. In addition, conversion of hard constraints to penalties introduces an entirely new set of difficulties for the optimization algorithm, because it becomes necessary to determine the relative costs of constraint violations. Without such safeguards, however, it is possible to define problems that have no feasible solutions or that will produce closed-loop instability even though the controller is able to find a solution at each sampling time. These problems are clearly demonstrated with processes that include time delays or other nonminimum phase elements [73].

Rawlings and Muske [58] avoid the stability problems generally associated with model-predictive controllers, for both unconstrained and constrained linear systems by adopting an infinite horizon. The resulting problem either has a stable closed-loop solution for the nominal plant, or has no feasible solution for the optimization. In most cases, problems with no feasible solutions are easily detected by the optimization algorithm. The problem then becomes one of determining exactly what measures to take once such an error has been detected but the process is not yet out of control. The simplest solution of shutting the process down may be unacceptable in many applications and it is desirable to find alternative approaches for error recovery.

The task of determining whether any feasible solutions exist is straightforward for optimization subject to linear constraints. Most algorithms for this class of problems are feasible-point methods, which means that if the initial point is feasible, all subsequent iterations are feasible. Therefore, use of these algorithms requires an initial feasible point to be found. One method of finding an initial feasible point is to solve a linear program (LP) involving the constraints only. If this LP fails to have a feasible point, there is no feasible point for the linear constraints.

For nonlinear constraints, determining whether there are any feasible solutions depends on the algorithm. Generalized reduced gradient (GRG) methods choose a working set of constraints (violated or currently active) and solve a subproblem to make the working set feasible. All the constraints must then be checked at the new point to ensure that there are no new infeasibilities. If there are, the algorithm must update the working set.

Sequential quadratic programming (SQP) methods do not require the nonlinear constraints to be satisfied at intermediate points, but some method must be used to solve the QP subproblem. If that method is a feasible-point method, then the linearized constraints will be required to be feasible. `NPSOL`, the SQP implementation of Gill *et al.* [28], for example, terminates if no feasible point can be found for the linearized subproblem. These ideas are discussed in detail in most optimization texts.

The model-predictive controller attempts to determine the set of inputs that will provide the best performance over some future time horizon given current information. The exact meaning of "best performance" is determined by the objective function. Clearly, it is possible to have undesirable performance that is also optimal if one chooses an inappropriate performance objective, or if one has incorrect or inaccurate information about the behavior of the system. The quality of the solution provided by the controller depends vitally on the quality of the process model.

Given a process model and a suitable performance objective, there are many techniques available for solving this class of problems. The correct choice depends on a number of factors, including the specifics of the problem to be solved and whether information in addition to the optimal solution is desired.

Only direct nonlinear programming approaches—those that do not require the user to derive the necessary conditions for the optimum—will be considered here.

2.2 Sequential Solution Techniques

Sequential solution techniques are distinguished by the elimination of the differential-algebraic constraints by solving the modelling equations explicitly. The resulting solution is then used to evaluate the objective function. Then NLP3 becomes

$$\min_{\mathbf{u}(t)} \quad \Phi(\mathbf{x}(t), \mathbf{y}(t), \mathbf{u}(t)) \tag{NLP4}$$

subject to:

$$\mathbf{h}_l(t) \leq \mathbf{h}(\mathbf{x}(t), \mathbf{y}(t), \mathbf{u}(t)) \leq \mathbf{h}_u(t)$$

with

$$t_0 \leq t \leq T$$

and $\mathbf{y}(t)$ and $\mathbf{y}(t)$ are given by the explicit solution of the differential-algebraic equation constraints

$$\begin{aligned}\dot{\mathbf{x}} &= \mathbf{f}(\mathbf{x}(t), \mathbf{y}(t), \mathbf{u}(t)) \\ 0 &= \mathbf{g}(\mathbf{x}(t), \mathbf{y}(t), \mathbf{u}(t))\end{aligned}$$

with

$$\mathbf{x}(t_0) = \mathbf{x}_0 \qquad \mathbf{y}(t_0) = \mathbf{y}_0$$

Note that the differential-algebraic equations must be solved at least once each time an objective function evaluation is required. Clearly, for most nonlinear process models of interest, analytical solutions do not exist and a numerical solution is required, which presents a small difficulty if the optimization algorithm requires gradient information because standard DAE solvers are not usually written to provide parametric sensitivities of the solution.

These methods offer the advantage that any intermediate solution is acceptable, in the sense that it is at least feasible. If no feasible solution exists, this is an indication that something is drastically wrong with the problem statement. The controller may reach a state where no feasible solution exists if a disturbance enters the system and moves an output outside a hard constraint and it is impossible to move the system back inside the feasible region by the next sampling time.

Assuming that *a* feasible solution exists, the sequential solution technique is useful if there is only a fixed amount of time available to compute the optimal solution but a solution is required at the end of that time even if it is not optimal.

2.2.1 Problems with constraint elimination

There are a number of problems with the elimination of constraints that can easily be avoided with the simultaneous solution methods described in Section 2.3, but are much more difficult to handle with the sequential methods discussed here. The following example from Fletcher [22] (Problem 7.4) clearly illustrates some of these problems.

Solve the NLP

$$\min \quad x^2 + y^2$$

subject to:

$$x^2 = (y-1)^3$$

(a) Graphically.

(b) By eliminating x.

By solving the problem graphically, the implicit constraint $y \geq 1$ is retained[1] and the solution $x = 0$, $y = 1$ is obtained. Standard constrained NLP codes should also find this solution because they will not look for complex values of x and y.

Substituting the constraint into the objective function gives

$$\min \quad (y-1)^3 + y^2$$

This function does not have any (real-valued) stationary points because

$$\frac{d}{dy}\left[(y-1)^3 + y^2\right] = 3y^2 - 4y + 3$$

does not have any real roots.

In this case, the implicit constraint $y \geq 1$ has been removed by eliminating x. It is important to carefully consider the consequences of removing constraints because it is possible to inadvertently change the character of a problem.

One might argue that this sort of problem will not happen if the constraint equations are solved as a subproblem (*i.e.* for a given value of y, the constraint equation is used to solve for x and the result is used to evaluate the objective function) because this is similar to solving the problem graphically.

Unfortunately, the unconstrained NLP code does not know about the implicit constraint $y \geq 1$ either, and it is still possible to encounter problems if the additional implicit constraint $y \geq 1$ is ignored.

Note that if this problem is solved with a constrained NLP code, it is not necessary to worry about starting out feasible with respect to either the explicit or the implicit constraint because no matter what x and y the optimization code chooses, it is always possible to evaluate the constraint function.

Similar problems can arise for optimization subject to differential-algebraic constraints. It is often possible, however, to avoid these problems by introducing new variables with corresponding simple bounds. This approach does not work for sequential solution techniques because current DAE solvers do not handle inequality constraints, nor is it easy to see how to add this capability since it involves adding and removing constraints from a DAE system as it is being solved, and the addition of constraints may result in no solution for the system.

For example, consider the minimization

$$\min_{u(t)} \quad \int_0^T x^2 dt$$

[1] We are typically interested in real-valued solutions. If we are interested in any solution, many problems with constraint elimination disappear.

subject to:

$$\dot{x} = \sqrt{x - u} - u$$

Clearly, $x \geq u$ must be satisfied over the interval $0 \leq t \leq T$, and as long as this implicit constraint is satisfied, the problem may be solved using the sequential methods described below. If the constraint is violated, these methods fail because the model equations do not have a solution. It is possible to explicitly add the constraint so that the problem becomes

$$\min_{u(t)} \quad \int_0^T x_1^2 dt$$

subject to:

$$\begin{aligned}
\dot{x}_1 &= \sqrt{x_2} - u \\
x_2 &= x_1 - u \\
x_2 &\geq 0
\end{aligned}$$

but this set of equations cannot be solved explicitly to evaluate the objective function because there are inputs for which there is no solution (for example, if u is constant and greater than zero, with $x_1(t = 0) = u$). This problem cannot be avoided by using the constraint conversion technique given by Sargent and Sullivan [64] (Equation 2.3 described in Section 2.2.3 below), because it does not enforce the inequality constraint for each iteration of the NLP.

Using the simultaneous methods discussed in Section 2.3, it is possible to evaluate the constraint equations for any input, even if it is not possible satisfy them, because all reasonable optimization algorithms enforce simple bounds so that they are never violated. For this particular example, the optimization algorithm will ensure that x_2 is always positive, which implies that it will always be possible to evaluate $\dot{x}_1$, and the constraint $x_2 = x_1 - u$

Even with these problems, sequential methods do have some important advantages over simultaneous methods. They are more easily able to accommodate stiff systems of equations and discontinuities in the inputs and state profiles, primarily because current generation DAE solvers are able to handle these problems without modification. It is also much easier to place bounds on the accuracy of the model solution. All current DAE solvers will attempt to satisfy a user specified local error tolerance by adapting the step size or approximation order. It would be difficult to provide these capabilities using the collocation approach discussed in Section 2.3, as it would require the ability to adapt the finite element grid and change the dimension of the optimization problem on the fly.

Biegler [3] and Renfro *et al.* [61] have argued that sequential solution methods are generally less efficient than simultaneous solution methods, because they require that the model equations be solved accurately at each iteration within the optimization, even when the iterates are far from the final optimal solution. This statement is impossible to prove, because the observed efficiency of an algorithm depends on the specific implementation, and the particular problem being solved. In addition, Biegler used a relatively inefficient direct search method[2] for implementing the simultaneous approach for his comparison.

One serious disadvantage of the sequential approach is that variations in the accuracy of the solution of the differential equations from one iteration to the next will cause difficulties for most nonlinear programming algorithms, particularly if simple finite difference methods are used to compute gradients. Note that although Biegler's choice of a direct search technique avoided this problem, it resulted in a significant increase in the number of function evaluations, one of the primary reasons for the method's observed inefficiency.

Although in some cases it may be possible to avoid the numerical problems introduced by embedding an iterative DAE solution technique within the nonlinear program (even when using finite difference methods to obtain gradients) by decreasing the value of the requested accuracy for the DAE solver, such action may cause the DAE solver to fail, or cause an unacceptable increase in the time required to solve the system of equations.

2.2.2 Obtaining gradient information

Most efficient methods for solving nonlinear programs require the evaluation of partial derivatives of the objective and constraint functions with respect to the decision variables. In order to take advantage of the efficiency of these methods, it is important to be able to obtain gradient information efficiently and accurately.

The objective function and constraints for NLP3 are written in terms of continuous functions $\mathbf{y}(t)$, $\mathbf{x}(t)$, and $\mathbf{u}(t)$. Before one can evaluate partial derivatives for the optimization algorithm, it is necessary to choose a finite dimensional representation of $\mathbf{u}(t)$ and also to write the objective function Φ in terms of $\mathbf{y}(t)$ and $\mathbf{x}(t)$ evaluated at a finite number of points. For the purposes of illustration, consider a system with one input and one output (represented

[2] Even though such comparisons of efficiency depend on many factors, it is generally accepted that direct search techniques are much less efficient than gradient-based methods, particularly for Biegler's small example problem for which it is easy to obtain accurate gradient information.

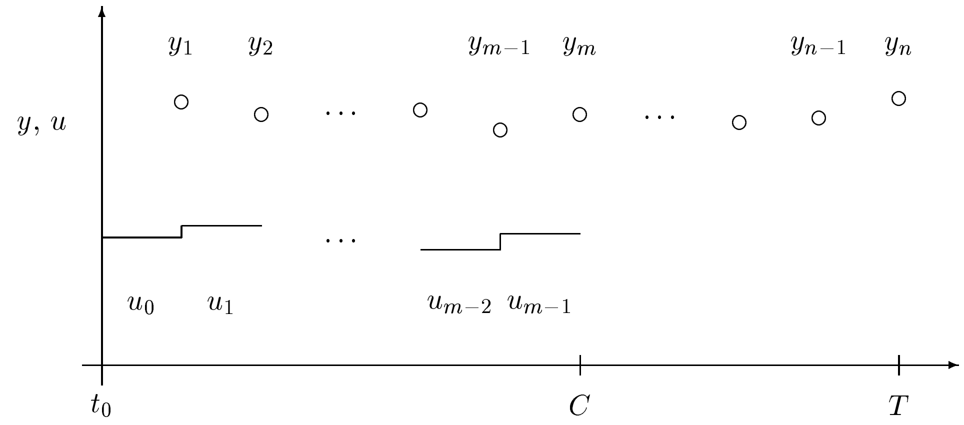

Figure 2.1: Prediction horizon for a system with one input and one output.

here by $y(t)$). Future manipulated variable moves are parameterized as a piecewise constant function. Suppose that the objective function depends on the output at the end of each sampling period and the value of the manipulated variable during the sampling period as shown in Figure 2.1.

The derivatives of the objective function that are required are

$$\begin{bmatrix} \frac{d\Phi}{du_0} \\ \frac{d\Phi}{du_1} \\ \vdots \\ \frac{d\Phi}{du_{m-1}} \end{bmatrix} = \begin{bmatrix} \frac{\partial y_1}{\partial u_0} & \frac{\partial y_2}{\partial u_0} & & \cdots & & \frac{\partial y_n}{\partial u_0} \\ 0 & \frac{\partial y_2}{\partial u_1} & & & & \frac{\partial y_n}{\partial u_1} \\ \vdots & & \ddots & & & \vdots \\ 0 & \cdots & 0 & \frac{\partial y_{m-1}}{\partial u_{m-1}} & \cdots & \frac{\partial y_n}{\partial u_{m-1}} \end{bmatrix} \begin{bmatrix} \frac{\partial \Phi}{\partial y_1} \\ \frac{\partial \Phi}{\partial y_2} \\ \vdots \\ \frac{\partial \Phi}{\partial y_n} \end{bmatrix} + \begin{bmatrix} \frac{\partial \Phi}{\partial u_0} \\ \frac{\partial \Phi}{\partial u_1} \\ \vdots \\ \frac{\partial \Phi}{\partial u_{m-1}} \end{bmatrix} \tag{2.2}$$

A similar equation results for each constraint function, provided that the constraints can be written in terms of the value of the output at each sampling point. If the objective or constraint functions require $y(t)$ to be evaluated at a different number of points, these equations must simply be modified to reflect the number of evaluations of $y(t)$.

The matrix of partial derivatives $\partial y_i / \partial u_j$ is upper trapezoidal because the system is causal—future inputs do not affect previous values of the output. Integration of sensitivity equations along with the differential equations provides a reasonably efficient method for evaluating these derivatives.

Caracotsios and Stewart's Fortran subroutine `DASAC` [8], based on Petzold's implicit integrator `DASSL` [51], may be used to compute the states and the first-order sensitivity coefficients required to evaluate the gradient of the objective function via Equation 2.2. `DASAC` solves DAE systems of the form

$$\mathbf{f}(\dot{\mathbf{y}}(t), \mathbf{y}(t), t; \mathbf{p}) = 0$$

with

$$\mathbf{y}(t_0) = \mathbf{y}_0(\mathbf{p})$$

where $\mathbf{p}$ is a vector of constant parameters, and the system of differential-algebraic equations is allowed to take the fully implicit form of Equation 2.1.

The parametric sensitivities

$$\mathbf{W}(t) = \frac{\partial \mathbf{x}(t, \mathbf{p})}{\partial \mathbf{p}^T} = \left[\frac{\partial x_i}{\partial p_j}\right]$$

are computed by `DASAC` simultaneously with the solution of the DAE system.

Although `DASAC` is not capable of computing the required sensitivities if the manipulated variable is treated as a single time-varying parameter, it can be made to work for the specific problem discussed here if a sequence of piecewise constant inputs is assumed. This limitation arises because `DASAC` is only capable of computing sensitivities of the states of the differential equations for parameters that are constant over the entire time of integration.

By treating the sequence of inputs as a vector of parameters, `DASAC` may be used to solve for the required sensitivities, allowing the objective function gradient information to be computed with a single integration of the DAEs. This technique requires the input to be piecewise constant, and works as long as `DASAC` is not reinitialized at an intermediate point. Avoiding reinitialization is necessary so that the initial conditions of the sensitivity equations are preserved across the boundaries between sampling intervals because when `DASAC` initializes itself, it sets the values of the sensitivities to zero. This problem is a limitation of `DASAC`, *not* of the method itself.

The algorithm outlined above is similar to the approach of Morshedi [44] except that he used a less accurate method for computing the sensitivity derivatives.

2.2.3 Algebraic constraints

Constraints that involve the state variables at specified times are easily included by simply evaluating the solution of the differential equations at the specified times. However, constraints that must be satisfied at all points in the interval $t_0 \leq t \leq t_0 + T$ complicate the solution because there is no simple way to guarantee that they are always satisfied. Standard DAE solvers may be used to integrate the set of differential equations and any equality constraints involving the states together, but there is no general solution for inequality constrained DAEs that does not also involve the solution of an optimization problem. Even if one knows the parameters that may be manipulated to make the solution feasible, it is impossible to know which solution is best unless some sort of cost is associated with each manipulated parameter.

Given that we are solving an optimization problem, there are several methods available for incorporating general inequality constraints involving the states. The simplest approach is to approximate the path constraints with a fixed number of constraints over the interval of interest. The obvious difficulty with this is that the path constraints are no longer required to be satisfied everywhere and may in fact be violated between sampling points. Even if one is only concerned with finding a reasonable compromise, this approach introduces another tuning parameter and a potentially large number of constraint functions. In addition, unless one chooses a very large number of sampling points, a typical DAE solver will evaluate the residual functions a number of times between sampling points, suggesting that one should add constraint handling to the DAE solver.

Sargent and Sullivan [64] proposed a technique that does essentially this, by including an additional state variable defined by

$$\dot{x}_{n+1} = \sum_{i=1}^{N} \{\max[0, h_{l,i} - h_i(\mathbf{y}, \mathbf{u})]\}^2 + \sum_{i=1}^{N} \{\max[0, h_i(\mathbf{y}, \mathbf{u}) - h_{u,i}]\}^2 \quad (2.3)$$

with

$$x_{n+1}(t_0) = 0$$

Where h_l and h_u are elements of the bounds vectors $\mathbf{h}_l$ and $\mathbf{h}_u$, and for simplicity, $\mathbf{y}$ includes both of the variables $\mathbf{y}$ and $\mathbf{x}$ from NLP3.

At each step in the solution of the differential equations, the right hand side of Equation 2.3 is evaluated and any violations of the path constraints force x_{n+1} to become nonzero. By adding the terminal constraint $x_{n+1}(t_f) = 0$ the optimizer will be forced to eliminat the violations of the path constraints. This approach avoids the computational difficulties of including a penalty term in the objective function and does not require the addition of a possibly large number of inequality constraints to the problem. It also has a somewhat better chance of finding constraint violations that might otherwise be missed due to infrequent sampling because the constraints are evaluated each time the DAE solver requests a residual function evaluation.

2.3 Simultaneous Solution Techniques

This class of solution techniques is characterized by the conversion of the differential-algebraic constraints to algebraic constraints, usually by the application of the method of weighted residuals. The resulting nonlinear program may then be solved using any algorithm suitable for the constrained minimization of a nonlinear function. If one uses a Lagrange-Newton method to solve the minimization, this method is essentially the same as solving the necessary conditions with a weighted residual technique.

With these methods, the constraints are approximated rather than eliminated, which in some cases provides an advantage in terms of being able to evaluate the constraint equations, but has the distinct disadvantage in that the number of decision variables and algebraic constraints increase dramatically, leading to very large nonlinear programs. Although the resulting NLPs are typically sparse, as of this writing there are few, if any, quality codes for the solution of sparse NLPs.

Other methods for solving nonlinearly constrained optimization problems may be used, but the advantage of using an infeasible path method is that the model equations are solved simultaneously with the other constraints. Using this approach, the differential equations are treated in the same manner as the other constraints and are not solved accurately until the final iterations of the optimization. This could provide an advantage over methods that require repeated accurate solutions of the modelling equations.

It is worth noting that the simultaneous solution methods do not generally require that the initial estimate of the optimal solution be feasible, or even that the state profiles represent a solution of the modelling equations for the given sequence of inputs. For small problems, or problems that are nearly linear, this does not present a significant difficulty, though experience has shown that as the dimension of the problem increases, it is important to begin with a solution that is nearly feasible.

Tabak and Kuo [66] were among the first to approximate the differential equations as algebraic equations using the method of weighted residuals and include them as constraints in the nonlinear program. Using this approach, they solved a problem in which a missile was required to land within a specified region. Finite differences were used to approximate derivatives and the width of each difference interval was also included in the optimization.

Pollard and Sargent [52] determined the optimal control policy for a continuous distillation column by integrating the differential equations within a steepest descent algorithm. Pontryagin's maximum principle was used to solve the problem initially, but was abandoned after problems with the numerical methods for solving two point boundary value problems were encountered (they mention 'extreme sensitivity of the final [state] values to the estimates of the initial adjoint variables' as a reason for the failure). They note that a second-order method for solving the optimization was not attempted because it would have required too many function evaluations and too much memory for storing the Hessian. Inadequate storage and computational speed are mentioned frequently in the papers published more than five to ten years ago. It appears that the major difficulty was often not a lack of numerical solution techniques, but that the necessary hardware or software was simply not available.

In two papers, Neuman and Sen [45, 46] used cubic spline collocation on a uniform mesh to solve several optimal control problems with quadratic objective functions. The second paper demonstrates the optimal control of a distributed parameter system using this technique. Galerkin's method was used to approximate the solution in the spatial dimension and cubic splines were again used to approximate the solution in time. Rather than solve the problems as quadratic programs (all the problems considered have linear models and quadratic objective functions), they solved them in a sequential manner by eliminating the constraints and using an unconstrained optimization code.

Tsang *et al.* [68] solved the same example as Neuman and Sen [47] but used collocation and constrained nonlinear programming code to solve the model along with the optimization. The states and manipulated variables were approximated as power function expansions of the independent variable (time) so that the decision variables in the optimization are the coefficients in the series expansion. This approach makes it difficult to include constraints on the states and manipulated variables because bounds on these variables do not translate directly into bounds on the coefficients in the power series expansion. To simplify the addition of bounds and the selection of initial values, it is better to use a Lagrange interpolation approach and solve for the values of the states and manipulated variables directly.

Gill and Murray [27] examined the solution of optimal control problems involving linear models and quadratic objective functions using basis function expansions for the state and manipulated variables. They considered the sparsity of the Hessian to be an important feature of the problem and presented several ways of taking advantage of this within the specific optimization methods considered.

Biegler [2] noted that the use of an infeasible path optimization algorithm such as SQP, together with orthogonal collocation techniques to convert the differential equation constraints to algebraic constraints, resulted in an efficient solution of optimal control problems. The noted efficiency of the technique was attributed to the fact that the SQP algorithm converges to the solution of the constraints (the differential equations) simultaneously as it converges to the optimum.

Cuthrell, in his dissertation [10], and in several papers with Biegler [11, 12, 13] applied the method to a number of chemical engineering problems, including several reactor models, flowsheet optimization, and a biological system. The method was also extended to collocation on finite elements, which allows the solution of problems with discontinuous state and control profiles. In order to find the discontinuities, a subset of the finite element boundaries are included as variables in the optimization. Although this method is shown to be effective in providing an accurate solution, it adds a significant computational burden to the algorithm and it is not clear that it is the best method to use, particularly if the solution will be updated in a feedback scheme.

Renfro [60] and Renfro *et al.* [61] used SQP with collocation on finite elements to solve open loop optimal control problems.

Li and Biegler [41] used a moving time horizon approach and a Newton-type control law like that developed in Economou [19] and Economou and Morari [18] as a means of nonlinear feedback control. This approach is significantly different from the method proposed in the present work, as it makes use of a linearized model at each step. It is not obvious that this method provides any advantage over the direct solution of nonlinear optimal control problem followed by parameter and disturbance estimation. It is probably true that the solution of the nonlinear optimization is more likely to fail numerically than the solution of the linear subproblem. However, there is no guarantee that repeated linearization will result in a reliable solution of the original nonlinear problem.

Eaton *et al.* [17] used the simultaneous optimization and solution approach to solve the open loop optimal control problem and discuss the use of parameter estimation as a means of incorporating feedback. A method for computing the sensitivity of the solution of the optimal control problem with respect to model parameters and the manipulated variables is also presented.

Finally, Patwardhan *et al.* [48] compare the simultaneous solution and optimization approach with linearization techniques for the continuous stirred tank reactor model of Uppal *et al.* [69]. A modified output disturbance estimation scheme is used to keep the plant at an open-loop unstable steady state even in the presence of significant modelling error.

2.3.1 Approximate solutions using orthogonal collocation and successive quadratic programming

Orthogonal collocation on finite elements may be used in connection with successive quadratic programming to solve NLP3. The original problem is restated as

$$\min_{\mathbf{x},\mathbf{u}} \quad \Phi(\mathbf{y},\mathbf{u}) \tag{NLP5}$$

subject to:

$$\begin{aligned} \mathbf{A}\mathbf{x} &= \mathbf{f}(\mathbf{x},\mathbf{y},\mathbf{u}) \\ 0 &= \mathbf{g}(\mathbf{x},\mathbf{y},\mathbf{u}) \end{aligned}$$

$$\mathbf{h}_l \leq \mathbf{h}(\mathbf{x},\mathbf{y},\mathbf{u}) \leq \mathbf{h}_u$$

where $\mathbf{x}$ and $\mathbf{u}$ are now used to represent vectors of values of the states and manipulated variables at discrete points over he prediction horizon, and $\dot{\mathbf{x}}$ has been approximated by $\mathbf{A}\mathbf{x}$, and $\mathbf{A}$ is a matrix of collocation weights [71]. Any derivatives with respect to spatial coordinates may be handled in a similar manner, and integral terms may be included efficiently by using appropriate quadrature formulae [70, 15]. In addition, this reformulation offers several advantages over solving the original problem by traditional methods. A wide range of constraints may be easily incorporated, the manipulated variable profile may be either continuous or discrete, and the solution of the optimization problem provides useful sensitivity information at little additional cost.

State and manipulated variable constraints may be included by simply defining them as functions of the state and manipulated variables at the collocation points. Although there are methods for handling these types of constraints within sequential solution methods, none are as straightforward to apply.

The sensitivity of the solution of NLP5 yields valuable information concerning those parameters and manipulated variables that are most important, and the time periods over which they matter most. The sensitivities may be used off-line to diagnose controller difficulties and may indicate the need for a better model of the process, additional or different measurements, or design changes.

On line, the parametric sensitivities might be used to determine the parameters that are most important and need to be estimated most accurately. The manipulated variable sensitivities will determine the inputs that have the greatest effect on the solution of the controller subproblem. This information could be used to diagnose possible conditioning problems that may arise in solving NLP5.

The primary disadvantage of this approach is that it is difficult to handle discontinuities in the state variable profiles. The use of finite elements provides a possible solution to this problem, though one must either know the locations of the discontinuities in advance, or be able to adapt the grid to handle them. It is rarely the case that the locations of the discontinuities are known prior to obtaining a solution, and the addition of a moving grid adds a significant computational burden [12].

Even though there are a number of difficulties with these solution techniques, they can be effective in practice, and several examples of are given after the following section.

2.3.2 Sensitivity analysis

By solving the model equations as additional algebraic constraints in the nonlinear program as discussed above, it is a straightforward matter to compute the sensitivity of the optimal solution to changes in parameter values and inputs in addition to simply determining the optimal set of future inputs. The parametric sensitivities computed for a given optimization step determine the parameters that have the greatest effect on the location of the optimum and must be known most accurately. The sensitivity of the optimal solution with respect to the inputs shows the extent that the objective function will change if there are errors in the inputs.

Large parametric sensitivities indicate that errors in the model parameters will lead to a significant error in the optimum solution (*i.e.* the solution obtained will be far from optimal for the true plant). Similarly, if the input sensitivities are large, errors in implementing the computed input profile will lead to an unacceptable increase the value of the objective function.

Parametric sensitivity. Given the nonlinear program NLP5, it is desired to find the sensitivity of the optimal solution with respect to changes in parameter values $\mathbf{p}$. These sensitivities will prove useful in determining the model parameters that have the greatest effect on the solution of the dynamic optimization and must be known most accurately. Fiacco [21] provides a thorough development of sensitivity results for this class of problems. His results are summarized below.

Considering only those constraints that are active at the optimum, the following necessary conditions hold. Note that all decision variables, including the variables $\mathbf{x}$, $\mathbf{y}$, and $\mathbf{u}$ are now represented by $\mathbf{x}$.

$$\begin{aligned} \nabla_x L(\mathbf{x}, \mathbf{p}) &= 0 \\ h_i(\mathbf{x}, \mathbf{p}) &= 0 \qquad i = 1, n \end{aligned}$$

where L is the Lagrangian function

$$L(\mathbf{x}, \mathbf{w}, \mathbf{p}) \equiv \Phi(\mathbf{x}, \mathbf{p}) - \sum_{i=1}^{n} w_i h_i(\mathbf{x}, \mathbf{p})$$

By differentiating the necessary conditions and noting that at the optimum the constraints must remain satisfied, the sensitivity equations may be written as

$$\begin{bmatrix} \nabla_{xx} L & \nabla_x \mathbf{h}^T \\ \nabla_x \mathbf{h} & 0 \end{bmatrix} \begin{bmatrix} \nabla_p \mathbf{x} \\ \nabla_p \mathbf{w} \end{bmatrix} = - \begin{bmatrix} \nabla_{xp} L \\ \nabla_p \mathbf{h} \end{bmatrix}$$

The solution of these equations at the optimum yields the sensitivity of the solution of the approximate nonlinear program to changes in parameter values.

Sensitivity of the optimal solution to independent variables. It is desired to find the sensitivity of the objective function to changes in the independent variables while requiring that the constraints remain satisfied. Physically this corresponds to changing one of the independent variables, solving the model equations (the constraints) and observing the change in the objective function. What follows is a derivation of the second order constrained optimality conditions with an extension applicable to cases in which the decision variables may be partitioned into independent and dependent sets.

Let $\delta\mathbf{u}$, $\delta\mathbf{x}$, $\delta\mathbf{h}$, and δf, be variations in the independent and dependent variables and the constraint and objective functions respectively. In solving control problems, the independent variables are the manipulated variables, and the dependent variables are the states. The result given below requires that the number of dependent variables be the same as the number of active constraints. Because of this, one must choose additional manipulated variables to consider as dependent for cases involving auxiliary constraints (for example, problems that have specified values at a fixed final time, or problems with active state constraints).

Given the following problem, in which all active constraints are included in the vector $\mathbf{h}$,

$$\min \qquad \Phi(\mathbf{x}, \mathbf{u})$$

subject to:

$$\mathbf{h}(\mathbf{x}, \mathbf{u}) = 0$$

the Lagrangian is defined as

$$L(\mathbf{x}, \mathbf{u}) \equiv \Phi(\mathbf{x}, \mathbf{u}) - \sum_{i=1}^{n} w_i h_i(\mathbf{x}, \mathbf{u})$$

and the first order necessary conditions for an optimum are

$$\nabla L(\mathbf{x}, \mathbf{u}) = 0$$

Let $(\mathbf{x}^*, \mathbf{u}^*)$ satisfy these conditions. Then a Taylor series expansion about this point gives

$$\delta L = \left[\begin{array}{c|c} \nabla_x L & \nabla_u L \end{array} \right] \left[\begin{array}{c} \delta\mathbf{x} \\ \delta\mathbf{u} \end{array} \right] + \frac{1}{2} \left[\begin{array}{c|c} \delta\mathbf{x}^T & \delta\mathbf{u}^T \end{array} \right] \left[\begin{array}{cc} \nabla_{xx} L & \nabla_{xu} L \\ \nabla_{ux} L & \nabla_{uu} L \end{array} \right] \left[\begin{array}{c} \delta\mathbf{x} \\ \delta\mathbf{u} \end{array} \right] + \cdots$$

Substitution of the first order necessary conditions gives

$$\delta L(\mathbf{x}^*, \mathbf{u}^*) = \frac{1}{2} \left[\begin{array}{c|c} \delta\mathbf{x}^T & \delta\mathbf{u}^T \end{array} \right] \left[\begin{array}{cc} \nabla_{xx} L & \nabla_{xu} L \\ \nabla_{ux} L & \nabla_{uu} L \end{array} \right] \left[\begin{array}{c} \delta\mathbf{x} \\ \delta\mathbf{u} \end{array} \right] + \cdots \tag{2.4}$$

Expanding the constraint functions in a Taylor series gives

$$\delta\mathbf{h}(\mathbf{x}^*, \mathbf{u}^*) \approx \left[\begin{array}{c|c} \nabla_x \mathbf{h} & \nabla_u \mathbf{h} \end{array} \right] \left[\begin{array}{c} \delta\mathbf{x} \\ \delta\mathbf{u} \end{array} \right] \tag{2.5}$$

For any point $(\mathbf{x}^* + \delta\mathbf{x}, \mathbf{u}^* + \delta\mathbf{u})$ to remain feasible, $\delta\mathbf{h}$ must be zero. Ignoring higher order terms, Equation 2.5 may be solved to find the required change in the dependent variables with respect to a given change in the independent variables. Substitution of this result into Equation 2.4 followed by rearrangement yields

$$\delta\Phi(\mathbf{x}^*, \mathbf{u}^*) \approx \frac{1}{2}\delta\mathbf{u}^T \mathbf{M} \delta\mathbf{u} \tag{2.6}$$

With

$$\begin{aligned} \mathbf{M} \;=\; & \nabla_{uu} L - \nabla_u \mathbf{h}^T \nabla_x \mathbf{h}^{-1^T} \nabla_{xu} L - \nabla_{ux} L \nabla_x \mathbf{h}^{-1} \nabla_u \mathbf{h} + \\ & \nabla_u \mathbf{h}^T \nabla_x \mathbf{h}^{-1^T} \nabla_{xx} L \nabla_x \mathbf{h}^{-1} \nabla_u \mathbf{h} \end{aligned}$$

Used directly, this result provides an estimate of the change in the objective function for a given deviation from the optimal path. If one assumes independent variations in the elements of $\mathbf{u}$, Equation 2.6 may be solved to determine the magnitude of the change in any one manipulated variable that will give rise to a specified change in the objective function.

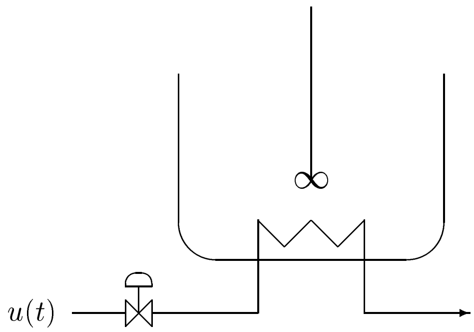

Figure 2.2: Batch reactor.

2.3.3 Example—optimal temperature profile for a batch reactor

To illustrate the computational techniques described above, including the computation of sensitivities of the optimal solution of a nonlinear optimal control problem, consider the problem given by Ray [59], in which the reactions

$$\begin{array}{ccc} A & \xrightarrow{k_1} & B \\ & \searrow_{k_2} & \\ & & C \end{array}$$

are carried out in the batch reactor shown in Figure 2.2. The objective is to maximize the concentration of B at the end of a specified reaction time, t_f, given the following dimensionless process model describing the dynamics of the reactant and product concentrations.

$$\begin{aligned} \frac{dx_1}{dt} &= -(u + \alpha u^p)x_1 \qquad & x_1(0) = 1 \\ \frac{dx_2}{dt} &= ux_1 \qquad & x_2(0) = 0 \end{aligned}$$

$$k_i = k_{i0} e^{-E_i/RT}$$

where $x_1 = C_A/C_{A_0}$ and $x_2 = C_B/C_{A_0}$. The model parameters p and α are defined as follows

$$p = E_2/E_1 \qquad \text{ratio of activation energies}$$

$$\alpha = k_{20}/k_{10}^p \qquad \text{ratio of preexponential factors}$$

For the example shown below, $p = 2.0$, $\alpha = 0.5$, and u is required to be between 0.0 and 5.0. The objective is simply to maximize the yield of B at the final time,

$$\max \quad \Phi = x_2(t_f)$$

Figure 2.3 shows the optimal temperature profile for the given one hour reaction time. This result is obtained by solving for the optimal inputs over the entire reaction time. To include feedback for batch problems of this type, the algorithms discussed in Chapter 3 for continuous problems may be applied provided the method used to update the time horizon is modified to account for the fixed final time.

Note that the upper bound on the input is active at the final time step. This constraint is easily handled using piecewise constant input moves and does not require special treatment to insure that interpolated values also remain within the bound (as is needed if the input moves are polynomials of degree greater than one).

The sensitivity of the objective function to changes in the manipulated variable profile are shown in Figure 2.4. The center line is the optimal profile, and the upper and lower lines indicate the input, if used for any *one* of the steps, would result in (to the second order approximation given by Equation 2.6) a one percent change the objective function. If any two steps are at the upper or lower lines (positive or negative deviations or both) the objective function will change by approximately two percent and so on. This result demonstrates several important features of this problem. First, the objective function is relatively insensitive to errors in the input. Assuming that the model is perfect (see below), one would have to miss the optimal profile by a significant amount for a relatively long period of time to see a reduction in the yield of even five percent. Even though the magnitude of the change is small, it is more important to stay near the optimal profile at the beginning of the run. In fact, over the last several time steps, it is virtually impossible to affect the value of the objective function without violating input constraints. The sensitivities give quantitative support to the physically intuitive strategy of maintaining low temperatures near the beginning to favor selectivity of product B and high temperatures near the final time to achieve complete conversion of A. Notice that although raising the temperature late in the batch is optimal, the sensitivity is so small that the large final temperature values are irrelevant. The optimal state profiles sensitivities to the model parameters are shown in Figure 2.5, where

$$s[i,j] = p_j \frac{\partial x_i}{\partial p_j}$$

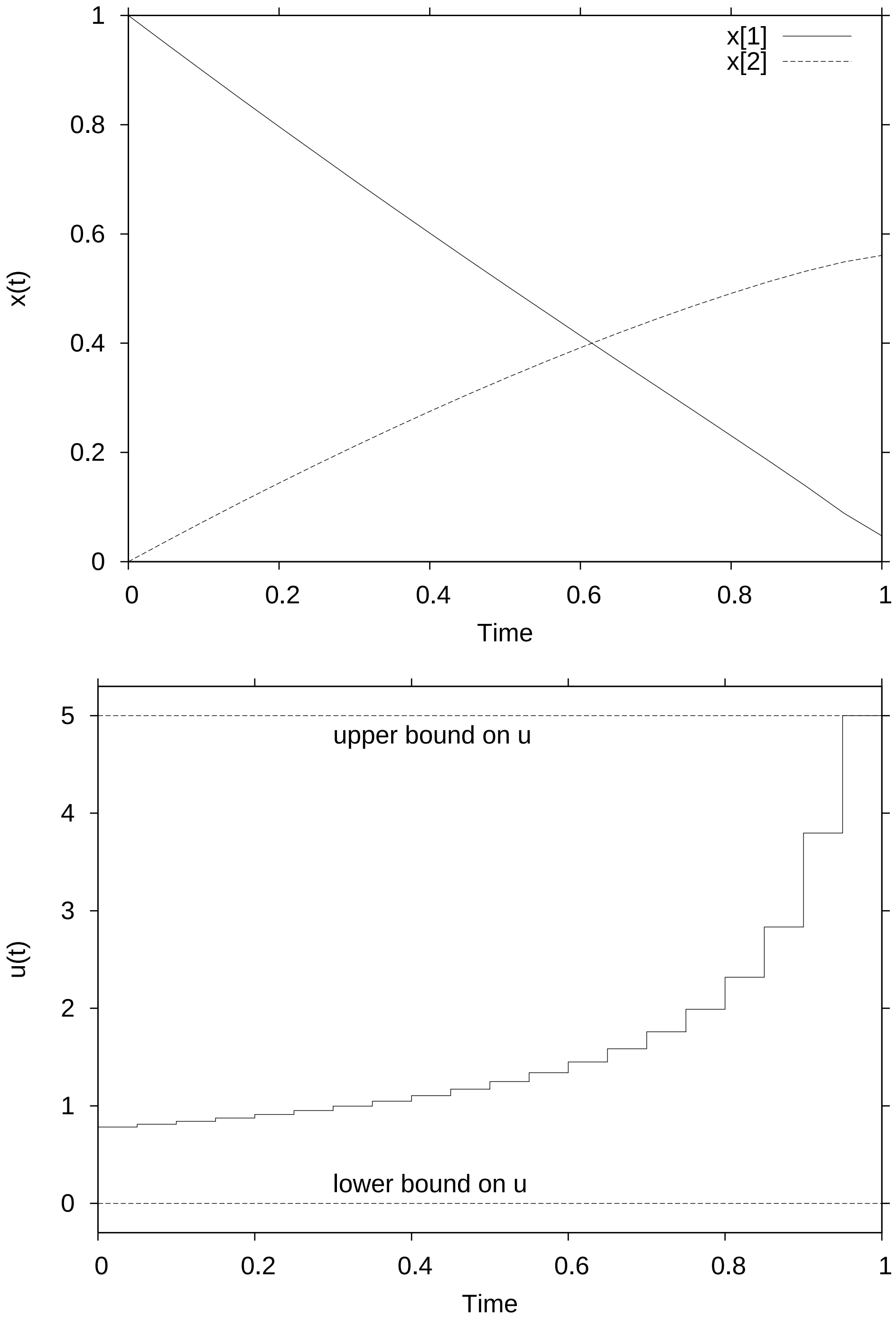

Figure 2.3: Optimal control profile and state profiles for the batch reactor of for the batch reactor of Section 2.3.3.

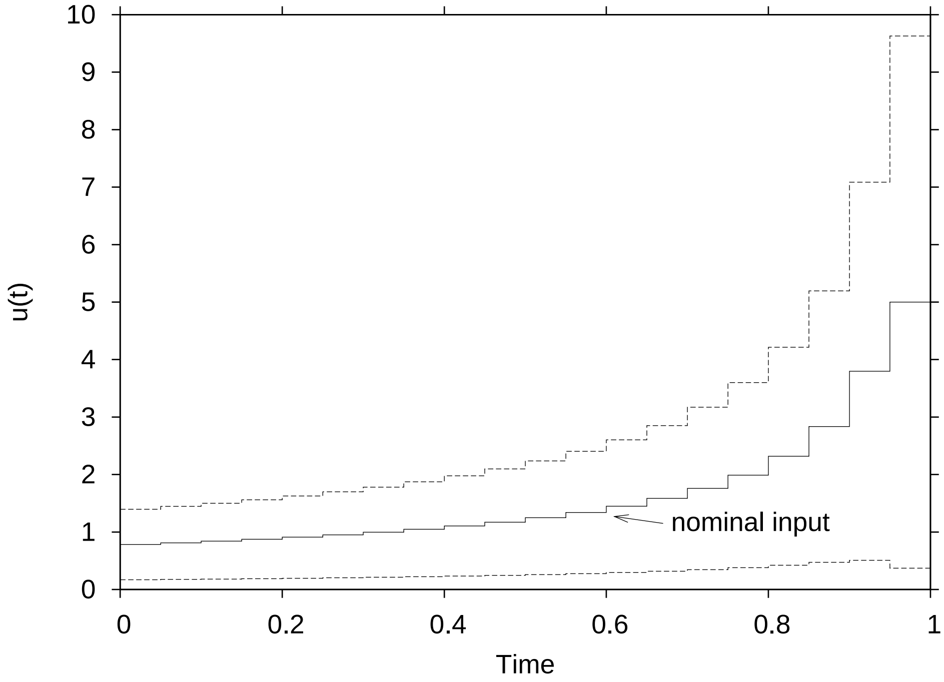

Figure 2.4: Bounds on the manipulated variable profile for a 1% change in the objective function. A deviation from the nominal input to either the upper or lower dashed line for a single time step results in a 1% change in Φ.

with $p_1 = \alpha$ and $p_2 = p$ in the model above. Notice that for positive changes in either parameter, less B will be produced, though the amount of A that remains at the final time will not be greatly affected. This is due to the fact that the reaction is being carried out to near completion and the production of B will be traded for C.

The magnitudes of the manipulated variable sensitivities are given in Figure 2.6 where

$$s[i] = \left| p_i \frac{\partial u}{\partial p_i} \right|$$

with $p_1 = \alpha$ and $p_2 = p$ in the model above. This figure indicates that the optimal manipulated variable profile is much more sensitive to changes in the model parameters near the final time, although it is zero for the final time step due to the presence of a constraint.

It is interesting to note that during the time when parameter values are most important, the manipulated variable profile is least important, and that the converse is also true (see Figure 2.4). This indicates that, for this prob-

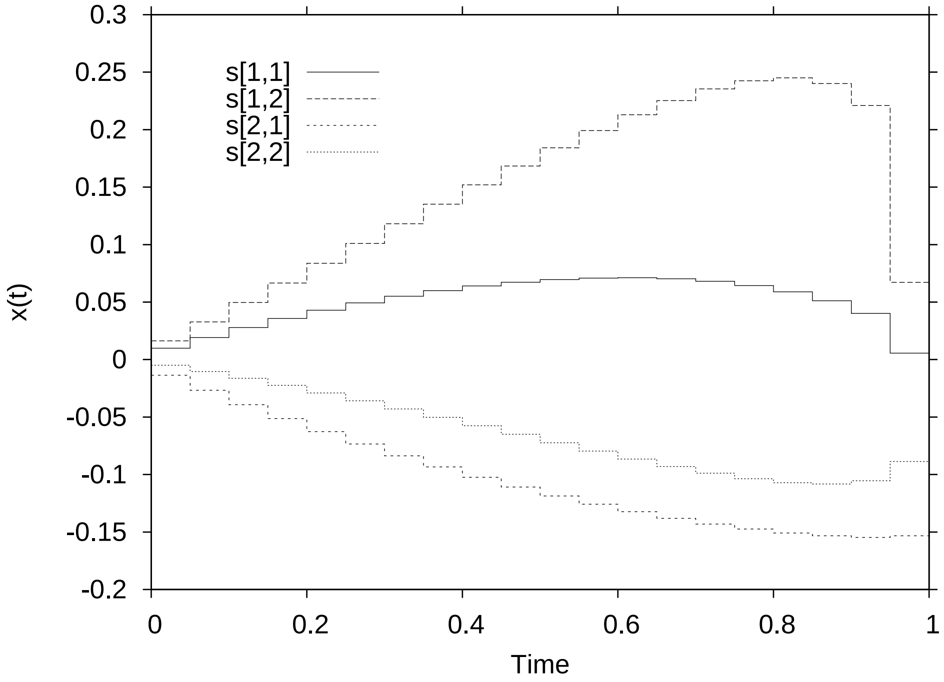

Figure 2.5: State variable sensitivity. $s[i,j] = p_j \frac{\partial x_i}{\partial p_j}$ with $p_1 = \alpha$ and $p_2 = p$ in the model above.

lem, if one begins with reasonable estimates for the plant parameters and implements the optimal profile for approximately the first three-fourths of the reaction time, it makes almost no difference what happens during the remaining time as the effect of errors in the parameters late in the reaction will go essentially unnoticed due to the insensitivity of the objective function to the manipulated variable.

2.4 Distributed Parameter Systems

Optimal control problems for systems modelled by partial differential equations may be solved in a straightforward manner using the techniques described above. By reducing the PDE to a set of DAEs using weighted residual techniques, an approximate model is obtained that fits the existing framework. The only additional difficulty is related to the size of the resulting optimization.

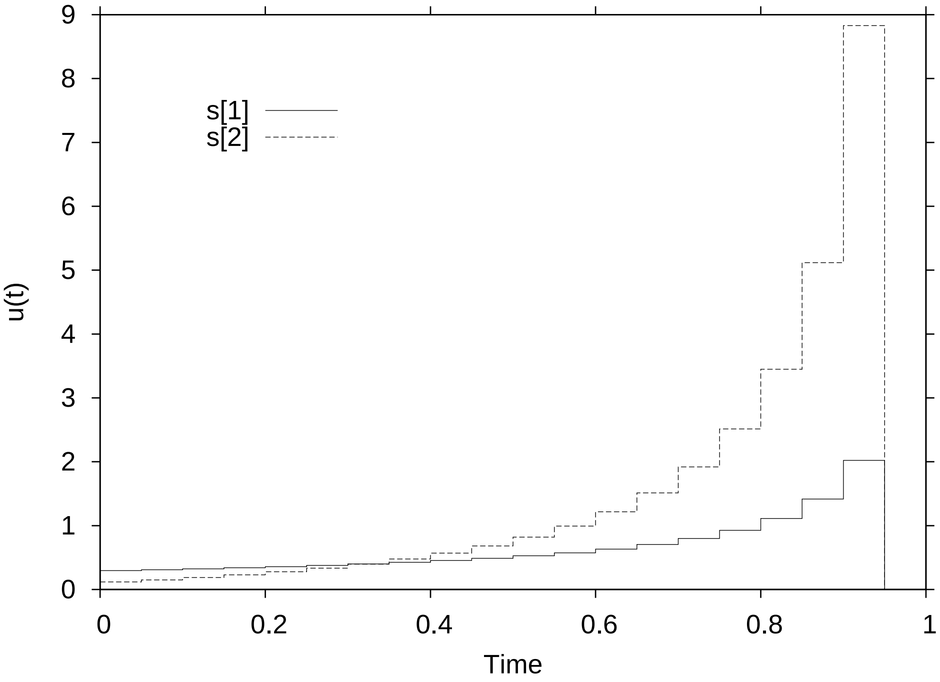

Figure 2.6: Manipulated variable sensitivity. $s[i] = |p_i(\partial u/\partial p_i)|$ with $p_1 = \alpha$ and $p_2 = p$ in the model above.

2.4.1 Optimal cooling curve for a batch crystallizer

In order to demonstrate these solution methods applied to nonlinear distributed parameter systems, consider a batch crystallization carried out in the crystallizer shown in Figure 2.7. Under the assumptions that the crystallizer is well mixed, the crystallizer temperature is directly related to the flow rate of a cooling medium (shown here as $u(t)$) and may be considered as the manipulated variable.

Applying the population balance approach to model the crystal size distribution (CSD), the following dynamic model is obtained,

$$\frac{\partial f(r,t)}{\partial t} = -G\frac{\partial f(r,t)}{\partial r}$$

$$\frac{dc}{dt} = -3k_v\rho G\int_0^\infty f(r,t)r^2 dr$$

where $f(r,t)dr$ is the concentration of crystals of size r to $r+dr$ at time t. The variable c is the solute concentration, and G is the crystal growth rate.

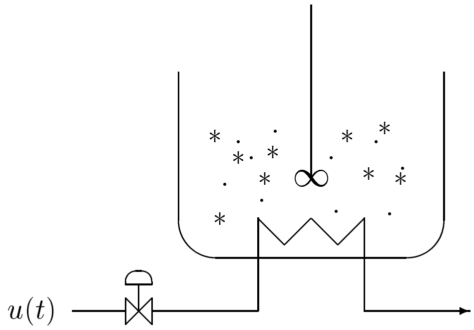

Figure 2.7: Batch crystallizer.

It is convenient to split the CSD into seed and nucleated subpopulations,

$$f = f_s + f_n$$

The following initial and boundary conditions are given,

$$\begin{aligned} c(t = 0) &= c_0 \\ f_s(r, t = 0) &= f_{s_0}(r) \\ f_n(r, t = 0) &= 0 \\ f_n(r = 0, t) &= B/G \end{aligned}$$

where B is the crystal nucleation rate. Power law constituitive relations for the nucleation and growth rates are assumed

$$\begin{aligned} G &= k_g(\Delta c/c_{sat})^g e^{-E_g/RT} \\ B &= k_b(\Delta c/c_{sat})^b e^{-E_b/RT} \\ \Delta c &= c - c_{sat} \end{aligned}$$

where Δc is the supersaturation driving force for nucleation and growth. Note that the model is nonlinear in temperature due to the activation energy terms and the nonlinear dependence of c_{sat} on T. Witkowski and Rawlings [72] discuss additional modelling and stability issues for this system.

The following objective function is used

$$\Phi = \frac{\displaystyle\int_0^\infty f_n(r) r^3 dr}{\displaystyle\int_0^\infty f_s(r) r^3 dr}$$

parameter	value	units
g	1.5	dimensionless
E_g/R	4.859×10^3	K
k_g	8.64×10^9	$\mu s^{-1}(\mathrm{cm}^3/\mathrm{g})^g$
b	2.25	dimensionless
E_b/R	7.517×10^3	K
k_b	2.14×10^{15}	$\mu s^{-1}(\mathrm{cm}^3/\mathrm{g})^b$
c_0	174.2×10^{-3}	g solute/g solvent
M	3.0×10^3	g
ρ	2.66	$\mathrm{g/cm}^3$
k_v	1.5	dimensionless
m_{s0}	16.6	g

Table 2.1: Model parameters for the crystallizer example.

This represents a desire to maximize the final mass of seed crystals while keeping the mass of nucleated crystals small. A constraint that specifies a minimum acceptable yield must be added in order for nucleation and growth to occur at all. We have chosen the specification

$$c(t_f) \leq c_f$$

with $c_f = 0.13$.

The initial seed distribution is assumed to be parabolic between 250 and 300 microns, with $f_s(275) = 2.0$ The initial mass of seeds is given by

$$m_{s0} = M\rho k_v \int_0^\infty f_s(r) r^3 dr$$

where M is the mass of solvent, ρ is the crystal density, and k_v is a volume constant.

Other parameter values used in the simulations are given in Table 2.1. The saturation concentration as a function of temperature was computed using the relationship

$$c_{sat} = -76.3 \times 10^{-3} - 1.44 \times 10^{-3} T(\mathrm{K}) + 7.14 \times 10^{-6} T(\mathrm{K})^2$$

In order to minimize the objective function and satisfy the additional constraint, the crystallizer temperature is chosen as a manipulated variable. This

Figure 2.8: Case 1. Crystal size distribution due to constant temperature profile.

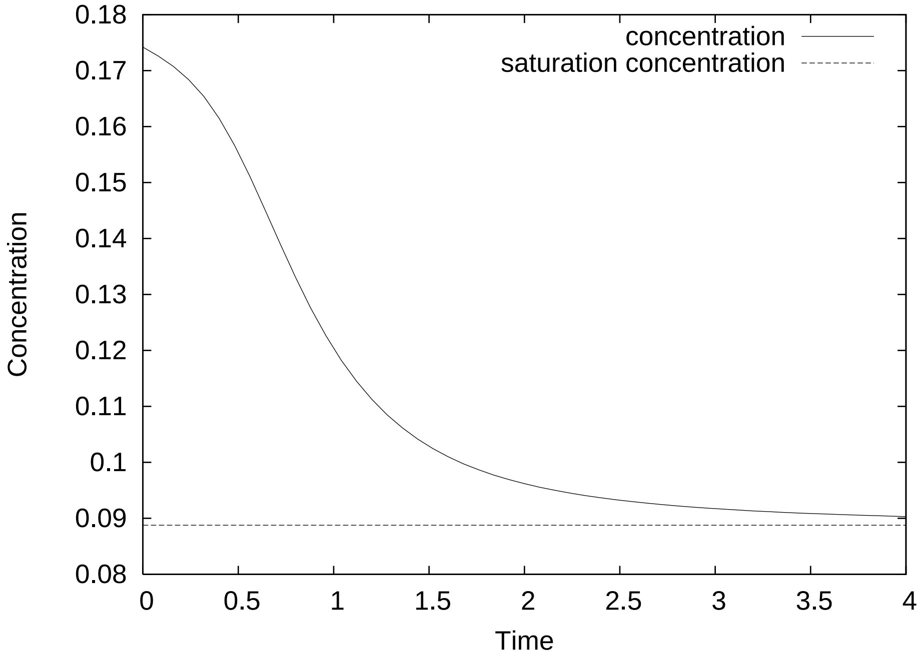

Figure 2.9: Case 1. Concentration profile due to constant temperature profile.

Figure 2.10: Case 2. Optimal crystal size distribution.

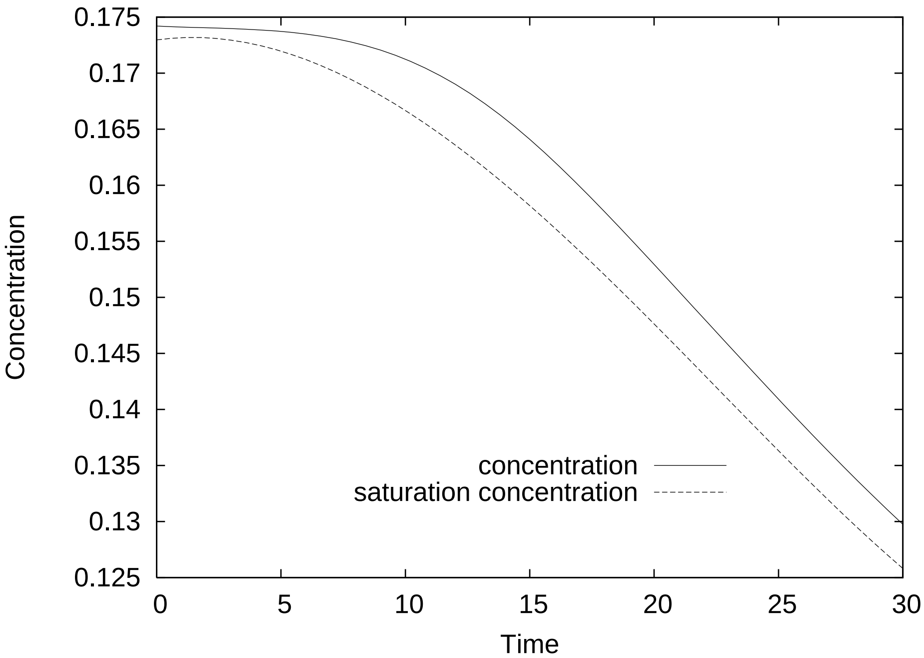

Figure 2.11: Case 2. Optimal concentration profile.

allows one to vary Δc, the driving force for nucleation and growth, making it possible to achieve much better performance than is possible with constant temperature operation which is commonly used industrially.

Figure 2.9 shows the concentration profile in the reactor as a function of time for an *isothermal* run. As shown in Figure 2.8, the large driving force leads to a high nucleation rate in the beginning of the run and that these crystals grow to sizes that make up a significant portion of the final mass of crystals. The optimal concentration profile, which holds the nucleation rate at small values until near the final time, is shown in Figure 2.11. This agrees qualitatively with previous theoretical and experimental results that indicate that a constant nucleation rate leads to increased growth of seed crystals while keeping nucleation at a minimum [34].

It is reasonable to assume that the optimum in the above problem will lie in a region where the supersaturation is always positive, and for the model given above, this is required in order to evaluate of the nucleation and growth rates, B and G. Because we do not have a valid model for dissolution, and because the optimizer can (and will) choose temperature profiles that lead to negative supersaturation, it is necessary to add constraints to prevent this.[3] Simply adding the constraint $c \geq c_{sat}$ is not sufficient because infeasible path optimization methods may violate it on the way to determining the optimal solution. To avoid this problem one may use the approach discussed in Section 2.2.1 and introduce an additional variable, Δc, with the simple bound $\Delta c \geq 0$ and corresponding constraint $\Delta c = c - c_{sat}$. This allows the nucleation and growth rates to be evaluated for all inputs (using the new variable Δc) because the simple bound on Δc is never violated.

Figure 2.10 shows the CSD for the optimal temperature profile as a function of time. For both cases, note that the seed crystals (whose size distribution is represented by the peak centered at 275 microns at $t = 0$) grow to approximately the same size at the final time. The CSD obtained for the optimal temperature profile is clearly superior to the isothermal case in terms of minimizing the mass of nucleated crystals compared to the mass of seed crystals, and represents a reduction in the objective function by a factor of approximately six.

It is particularly important to choose the right numerical formulation

[3]For this particular problem it may possible to develop a model for dissolving crystals, but doing so is a wasted effort if there are methods for avoiding the problem of evaluating B and G and the optimum *does* lie in a region where the supersaturation is always positive.

when the dimensionality of the problem increases.

— John Villadsen and Michael L. Michelsen,
Chemical Reaction and Reactor Engineering (1987)

Chapter 3

Closed-loop Properties of the Feedback

This chapter examines the closed-loop behavior of model-predictive control algorithms. The first part of this chapter focuses on Shell's DMC algorithm, a particular implementation of MPC for linear process models. The DMC algorithm is used because it provides an adequate means of examining a number of the key features of many MPC algorithms. In addition, the details of the DMC algorithm present a number of problems that are not shared by other MPC algorithms and do not appear to be well understood.

The second part of the chapter includes a discussion of nonlinearities arising from constraints and nonlinear process models, including a model of a continuous fed-batch bioreactor that exhibits open-loop unstable behavior and has multiple steady state solutions.

Several examples from classes of processes that lead to closed loop stability problems clearly demonstrate some of the difficulties with standard MPC implementations. A simple but effective method of estimation that can be used to solve the stability problem is also examined.

3.1 DMC/IMC Output Disturbance Estimation

A convenient and widely used means of incorporating feedback into the MPC algorithm is to estimate a disturbance in each output based on its measurement. This is the internal model approach of García and Morari [23] and the method used in the DMC algorithm of Cutler and Ramaker [14] and Prett and Gillette [53].

This approach is attractive because it is simple to implement. For stable plants and step disturbances, it results in zero steady state offset, even with modelling error. The implicit assumption of this approach is that the difference between the predicted and observed outputs are due only to step disturbances in the output of the process. If the error is due to step output disturbances, this method will work well. If not, performance may still be acceptable, but there are very simple cases for which performance will be poor. For example, if the output disturbance is a ramp rather than a step, the IMC feedback will not be able to remove steady-state offset.

For unstable plants, however, this control structure is not even internally stable. Several modifications that allow one to control unstable plants are possible and may be justified from a practical standpoint. Patwardhan *et al.* [48] were able to simulate the control of an unstable continuous stirred tank reactor using a simultaneous optimization and solution method by resetting the initial conditions of the modelling equations to the plant output at each time step *and* estimating the output disturbance. For this approach to work, one must be able to either measure or estimate all of the states.

3.1.1 The DMC algorithm

Because the algorithms for MPC of linear systems have similar features, we will focus attention on one of them, quadratic dynamic matrix control (QDMC), since it has been most fully described in the literature [24, 63]. Satisfaction of hard input constraints, one of the key reasons for the success of QDMC in practice, must be discarded at this point to keep the discussion focused on linear systems. The behavior of MPC for problems with hard input constraints is discussed in section 3.2. The remainder of this section describes the standard SISO DMC algorithm.[1]

The inputs in the feedback structure shown in Figure 3.1 are a sequence of future setpoints, r, and standard plant input and output disturbances, u_1 and u_2. It is important to note that if the controller is given future setpoint information, it will be able to improve performance by acting on estimates of future error. Ricker [63] provides an example of this feature of the algorithm applied to the control of a continuous evaporator. The future setpoint information is also the natural way to define the batch control problem. The ability of the the model-predictive controller to take action at the present time based

[1]The discussion focuses on SISO systems for simplicity of presentation. The features of the algorithm extend naturally to multiple-input multiple-output systems.

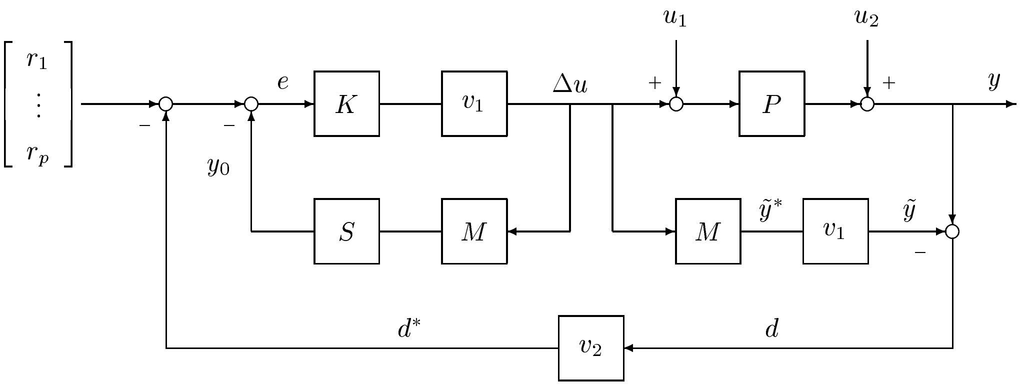

Figure 3.1: Block diagram representation of the DMC algorithm.

on an estimation of future conditions is also examined in Section 3.2. The future horizon is the part of MPC that allows the same framework to address both batch and continuous problems. In batch problems, however, one often needs to use a nonlinear model because the states may move across a large portion of the state space during the batch run.

The DMC model is the vector of transfer functions

$$M = (qI - S)^{-1} A_1$$

that produce a vector of future outputs due to the sequence of all inputs up to the current Δu. Of these, only the first is normally used as the output of the model. The column vector A_1 is the first column of the "dynamic matrix" of step response coefficients

$$A \equiv \begin{bmatrix} a_1 & & 0 \\ \vdots & \ddots & \\ a_n & & a_1 \\ \vdots & & \vdots \\ a_p & & a_{p-n+1} \end{bmatrix}$$

that determine the dynamics of the (stable) process model. The row vector $v_1 = [\, 1 \ 0 \ \cdots \ 0 \,]$ selects the first element of the predicted outputs, $\tilde{y}^*$ for comparison with the plant output, y, and the column vector v_2 projects the current estimate of the disturbance into the future. In the standard DMC

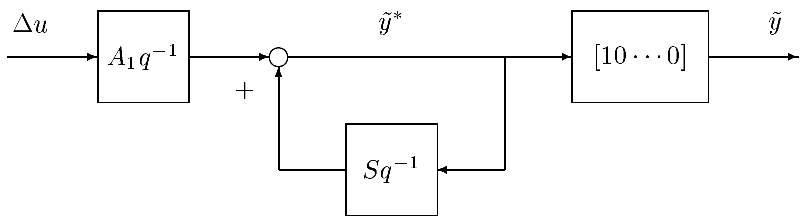

Figure 3.2: Block diagram for the DMC model, M.

algorithm it is assumed that the estimated disturbance will remain constant over the entire prediction horizon and $v_2 = [\,1 \ \cdots \ 1\,]^T$. This assumption is not necessary, and v_2 may be modified to reflect projected changes in the disturbance if that information is available.

The matrix S has the form

$$S \equiv \left[\begin{array}{c|ccc} 0 & & & \\ \vdots & & I & \\ 0 & & & \\ \hline \varepsilon_1 & \varepsilon_2 & \cdots & \varepsilon_p \end{array}\right]$$

performs a shift and extrapolation of the vector of future output predictions $\tilde{y}^*$. In the standard DMC algorithm, it is assumed that the system will reach steady state by the end of the prediction horizon and the extrapolation vector, ε, is given by $\varepsilon = [\,0 \ \cdots \ 0 \ 1\,]$. Clearly, from the definition of M, the poles of the plant are the eigenvalues of S. The characteristic equation for the eigenvalues can be found from the Laplace expansion of the determinant of $qI - S$

$$\det(qI - S) = q^p - \varepsilon_p q^{p-1} - \cdots - \varepsilon_2 q - \varepsilon_1 = 0$$

The poles of the standard DMC model are therefore the roots of $q^{p-1}(q-1) = 0$, or $p-1$ poles at the origin and one at $q = 1$ (an integrator). The integrator is simply a result of formulating the DMC model in Δu rather than u.

To see how the transfer function is related to the dynamic matrix, consider Figure 3.2. The model output, $\tilde{y}$, is obtained from the sum of $A_1 \Delta u$ and the previous estimate of the output over the entire prediction horizon, which is shifted and extrapolated to account for the horizon moving forward one sample period each time the output is computed.

The gain of the standard QDMC controller, K, is obtained from the solution of the following quadratic program:

$$\min_{\Delta u^*} \quad \Phi, \qquad \begin{aligned} \Phi &= (r - y_m)^T \Lambda^T \Lambda (r - y_m) + \Delta u^{*T} \Gamma^T \Gamma \Delta u^* \\ &= \tfrac{1}{2} \Delta u^{*T} H \Delta u^* + c^T \Delta u^* \end{aligned} \tag{QP1}$$

subject to:

$$\begin{bmatrix} \Delta u_l \\ u_l - u_{curr} \\ \tilde{y}_l \end{bmatrix} \leq \begin{bmatrix} I \\ L \\ A \end{bmatrix} \Delta u^* \leq \begin{bmatrix} \Delta u_u \\ u_u - u_{curr} \\ \tilde{y}_u \end{bmatrix}$$

where

$$\begin{aligned} H &= A^T \Lambda^T \Lambda \, A + \Gamma^T \Gamma \\ c^T &= e^T A \\ e &= r - y_0 - d^* \\ L &= \begin{bmatrix} 1 & & 0 \\ \vdots & \ddots & \\ 1 & \dots & 1 \end{bmatrix} \end{aligned}$$

and y_l and y_u are p-vectors of bounds on the output, u_l and u_u are n-vectors of bounds on the actual value of the input, Δu_l and Δu_u are n-vectors of bounds on the change in the input from one sampling time to the next, and u_{curr} is the current value of the input. The vector y_m represents the best estimate of the plant output over the prediction horizon including disturbance information and the vector y_0 is the model output over the future horizon if all future inputs are zero. The vector d^* is a disturbance estimate over the prediction horizon.

For the strictly linear DMC algorithm, where there are no constraints, the solution of QP1 is given by

$$\begin{aligned} \Delta u^* &= -H^{-1} c \\ &= -H^{-1} A^T e \\ &= K e \end{aligned}$$

and $-H^{-1}A^T$ need be evaluated only once and the error, e is simply updated in order to compute the controller output.

With these definitions, the steps of the algorithm are:

0. **Initialize** the dynamic matrix, A, the initial estimate of future outputs, $\tilde{y}^*$, and the initial estimate of the output disturbance, d. For simulations, $\tilde{y}^*$ and d are normally initialized to zero.

   Compute the Hessian for the QP. This will not need to be modified unless either the input or output penalties or the dynamic matrix changes.

   Initialize the constraint matrix. For the purposes of the current discussion, we will skip this step and focus on the standard unconstrained DMC algorithm.

1. **Update** the vector of linear coefficients for the QP objective function. The computation of $A^T(r - y_0 - d^*)$ must be done at each sampling time.

2. **Solve** QP1 for the changes in the manipulated variables. If the problem is strictly linear and unconstrained, this is simply a multiplication of the controller matrix, $K = -H^{-1}A^T$, and the error vector, e.

3. **Implement** some number of the computed inputs (normally assumed to be just one).

4. **Measure** the plant output, y.

5. **Compute** the predicted output for the plant over the current prediction horizon based on the inputs used in step 3.

$$\begin{bmatrix} \tilde{y}_1^* \\ \vdots \\ \tilde{y}_p^* \end{bmatrix} \leftarrow A\Delta u + \begin{bmatrix} \tilde{y}_1^* \\ \vdots \\ \tilde{y}_p^* \end{bmatrix}$$

6. **Estimate** future values of the disturbance, d^* from $d = y - \tilde{y}$, and the particular choice of v_2.

7. **Update** the predicted output over the future time horizon by setting $y_{0,i} = \tilde{y}_{i+1}^*$ for $i = 1$ to p. Note that a value for $\tilde{y}_{p+1}^*$ is required and that this point lies one sampling time beyond the current prediction horizon. If the plant is stable and the prediction horizon is greater than the control horizon by at least the settling time of the plant then it is reasonable to choose $\tilde{y}_{p+1}^* = \tilde{y}_p^*$. This is what is done in the standard DMC algorithm. In the following sections we will see that this approach does not work well if the prediction horizon is relatively short or the plant is unstable.

8. **Step** forward one sampling time and return to step 1.

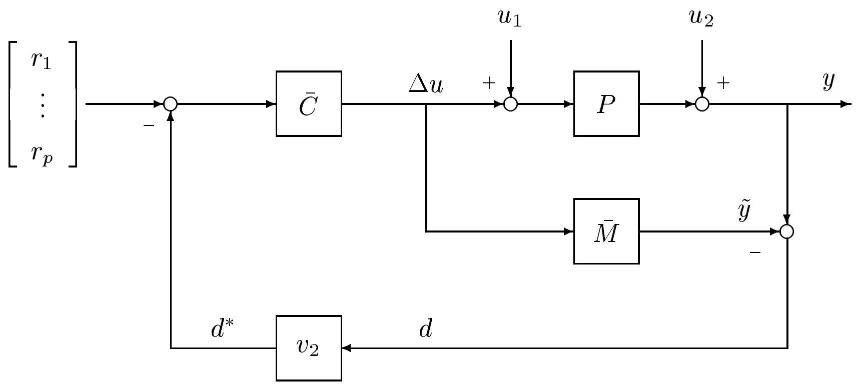

Figure 3.3: Simplified block diagram representation of the DMC algorithm.

Given this block diagram it is possible to construct the closed-loop transfer functions for the DMC algorithm. Experience has shown, however, that while the procedure is straightforward, it is rather tedious and time consuming, even using the latest tools for symbolic computation such as Maple or *Mathematica*. The primary difficulty lies in the matrix inverses that must be computed symbolically to obtain the closed-loop transfer functions in terms of q.

The block diagram shown in Figure 3.1 can be simplified to the diagram shown in Figure 3.3 where the controller, $\bar{C}$ and the model, $\bar{M}$ are given by the following relationships.

$$\begin{aligned}\bar{C} &= (1 + v_1 KSM)^{-1} v_1 K \\ \bar{M} &= v_1 M\end{aligned}$$

Given these definitions, the input-output relationships for the controller are

$$\begin{bmatrix} 1 & 0 & -P \\ 0 & 1 & -\bar{M} \\ \bar{C}v_2 & -\bar{C}v_2 & 1 \end{bmatrix} \begin{bmatrix} y \\ \tilde{y} \\ \Delta u \end{bmatrix} = \begin{bmatrix} P & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & \bar{C} \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ r \end{bmatrix}$$

For the nominal case, with $\bar{M} = P$, the closed-loop transfer functions are given by

$$\begin{bmatrix} y \\ \tilde{y} \\ \Delta u \end{bmatrix} = \begin{bmatrix} P(1 - \bar{C}v_2 P) & 1 - P\bar{C}v_2 & P\bar{C} \\ -P\bar{C}v_2 P & -P\bar{C}v_2 & P\bar{C} \\ -\bar{C}v_2 P & -\bar{C}v_2 & -\bar{C} \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ r \end{bmatrix}$$

and are clearly internally unstable because of the transfer function between u_1 and y.

Other researchers have independently obtained state-space representations of similar MPC algorithms. Li *et al.* [40] develop a state-space representation for their multivariable optimal constrained control algorithm (MOCCA), which is closely related to DMC, and Lee *et al.* [39] describe a state-space form of DMC in which each element of the predicted output (the vector $\tilde{y}^*$ above) becomes a state in the controller. Lee *et al.* propose using a Kalman filter to estimate all of these predicted outputs given process measurements. In addition, they suggest that one should choose the extrapolation formula (given by ε above) so that the long term dynamics are represented by a first order system. The justification for these modifications is purely heuristic. Finally, the treatment of each of the predicted outputs as a state in the controller equations is apparently an attempt to retain the step response model in the controller formulation, even though algorithm could be greatly simplified by abandoning it in favor of a low-order transfer function model.

3.2 Input Constraints

In this section, the QDMC algorithm described in Section 1.2 is applied to a simple first order linear process with input constraints. The addition of constraints requires that the simple solution method given in step 2 of the algorithm be replaced by a solution of the full quadratic program QP1.

The following example demonstrates the algorithm's ability to handle input constraints in a way that is different from standard feedback control with anti windup. The predictive controller is able to forecast the violation of constraints in the future. This feature allows current action to be taken to minimize the errors caused by constraints that are predicted to become active in the future. There are some examples in the literature that demonstrate the use of the prediction horizon to anticipate future setpoint changes, but in most discussions of the algorithm it is assumed that the setpoint is constant over the entire horizon.

To demonstrate this feature, the performance of the QDMC algorithm on the simple first order system

$$\frac{dx}{dt} = -x + u$$

with

$$x(t = 0) = 0$$

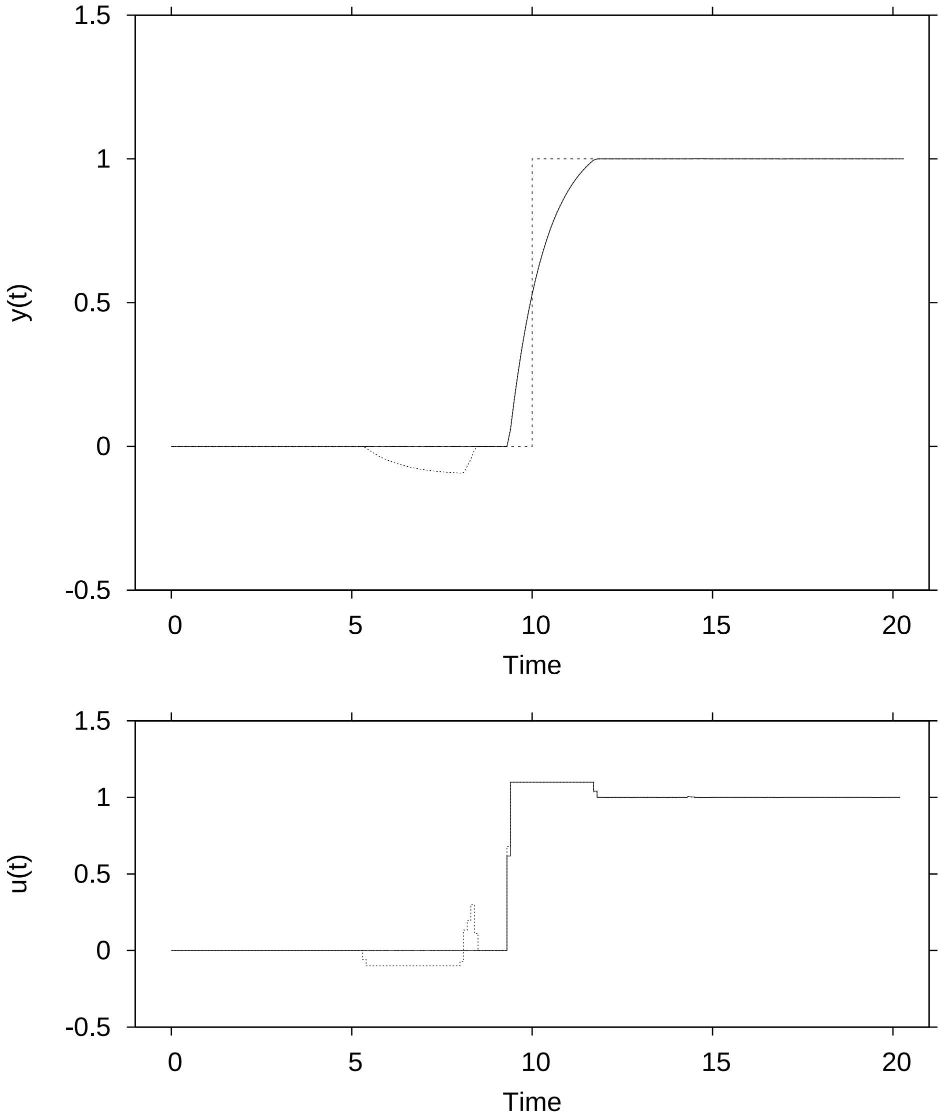

Figure 3.4: Performance of QDMC for $G(s) = 1/(s+1)$ for a sampling time of 0.1 and input constraint $-0.1 \leq u(t) \leq 1.1$ for two choices of prediction and control horizons (solid, $T = 5.0$, $C = 5.0$; dotted, $T = 5.0$, $C = 1.0$).

is examined, with a nonlinearity in the closed loop introduced by the additional requirement that $-0.1 \leq u(t) \leq 1.1$. The sample time used in the simulations is 0.1. In both cases shown in Figure 3.4 the prediction horizon is 5.0. The dashed line shows the controller action and the system response for a control horizon of 1.0.

The negative control action that begins near $t = 5.0$ is caused by the difference in the control and prediction horizons. Once again, although the control action is optimal at each sampling time for the chosen horizons, the overall sequence of inputs that is implemented is not particularly desirable for some choices of C and T. This is again due to the fact that the controller is asked to predict the required control action for a changing profile while also being forced to predict one relatively long constant control move over the final portion of the prediction horizon.

The solid line shows the system response for a control horizon equal to the prediction horizon. Note that the performance is again improved by choosing $C = T$ because the controller is always able to predict the proper control action.

For comparison, consider the optimal bang-bang profile for this problem,

$$u(t) = \begin{cases} 0, & t \leq t_1 \\ u_{\max}, & t_1 < t < t_2 \\ 1, & t \geq t_2 \end{cases}$$

with

$$t_1 = \theta - \log\left(\frac{2u_{\max}}{2u_{\max} - 1}\right), \qquad t_2 = t_1 + \log\left(\frac{u_{\max}}{u_{\max} - 1}\right)$$

where θ is the time of the step change in the setpoint, $u_{\max}$ is the upper bound on the input, and t_1 and t_2 are the switching times. For this particular example, $t_1 = 9.39$ and $t_2 = 11.79$. This is close to the solution that the QDMC controller finds, provided that the control and prediction horizons are sufficiently long.

A standard feedback controller (with a perfect anti windup strategy) would find a solution similar to that of the MPC controller, except that no control action would take place prior to the setpoint change (the curves for the optimal bang-bang controller described above would be shifted $\theta - t_1$ time units to the right). The ISE for MPC and the standard feedback controller are shown as a function of $u_{\max}$ in Figure 3.5. Notice that MPC controller is able to take action ahead of the desired setpoint change and achieves performance superior to that of the standard feedback controller regardless of the value of $u_{\max}$ and that the ratio of the ISEs approaches 0.25 in the limit of large $u_{\max}$.

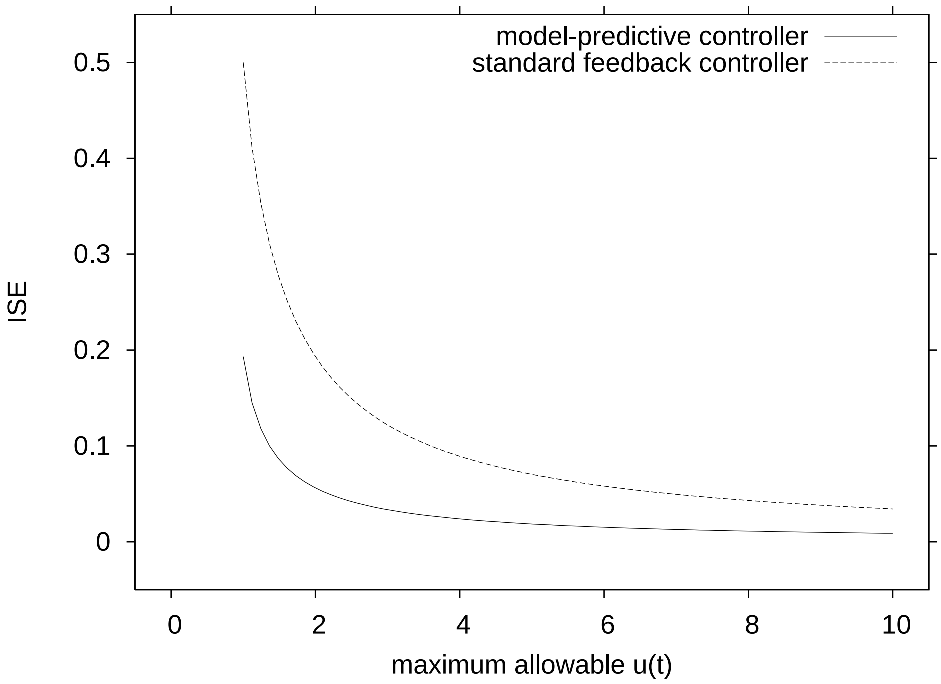

Figure 3.5: Integral square error for predictive control (solid) and optimal feedback control (dotted) for the first order system with input constraints.

3.3 Nonminimum Phase Plants

In Section 3.2 we saw that the performance of the MPC controller is superior to conventional feedback control for minimum phase, linear plants with input constraints. The MPC controller also has an advantage over conventional feedback controllers for processes with nonminimum phase elements because it has the ability to take action based on the prediction of future error.

3.3.1 Time delay processes

The simplest nonminimum phase process is the pure time delay, $g(s) = e^{-\theta_p s}$. Perfect setpoint tracking is achieved by the MPC controller provided that any changes in the reference are known at least θ_p time units in advance.

The optimal input for this case is

$$u(t) = r(t + \theta_p) \tag{3.1}$$

and all reasonable MPC controllers compute the input, $u(t)$, in Equation 3.1

without any special tuning. Indeed, one of the strengths of MPC is its inherent time delay compensation.

Note that the prediction horizon *must* be at least as long as the process time delay, otherwise the controller will not be able to predict any process response. This is easily seen for the DMC controller, because all of the step response coefficients will be zero.

3.3.2 Processes with right half plane zeros

Right half plane zeros also place limits on achievable closed-loop performance. Given setpoint information in advance, the model-predictive controller is able to minimize these limitations.

As an example, consider an allpass plant with a real right half plane zero at $s = a,\ a > 0$,

$$g(s) = \frac{-s+a}{s+a}$$

with the realization given by

$$\begin{aligned} \frac{dx}{dt} &= -ax + u \\ y &= 2ax - u \end{aligned} \tag{3.2}$$

Assume that one wants to make a step change in the setpoint for this plant at $t = \theta$,

$$r(t) = H(t - \theta)$$

The open-loop control action that achieves zero tracking error for this plant on any bounded interval, $t \in [0, B],\ B > \theta$, is

$$u(t) = \begin{cases} 0, & t \in [0, \theta) \\ 1 - 2e^{a(t-\theta)}, & t \in [\theta, B] \end{cases} \tag{3.3}$$

This control action is obviously undesirable because of its exponential growth after $t = \theta$. A feedback controller that produces this control action is easily avoided since such a controller produces a feedback loop that is internally unstable. Such a controller inverts the RHP zero in g and tries to use unstable pole zero cancellation with the plant. If one finds a feedback controller to minimize the integral square error (ISE) of e, and stipulates that the feedback system is internally stable (preventing the RHP zero inversion), the controller is

$$c(s) = \frac{s+a}{2s}$$

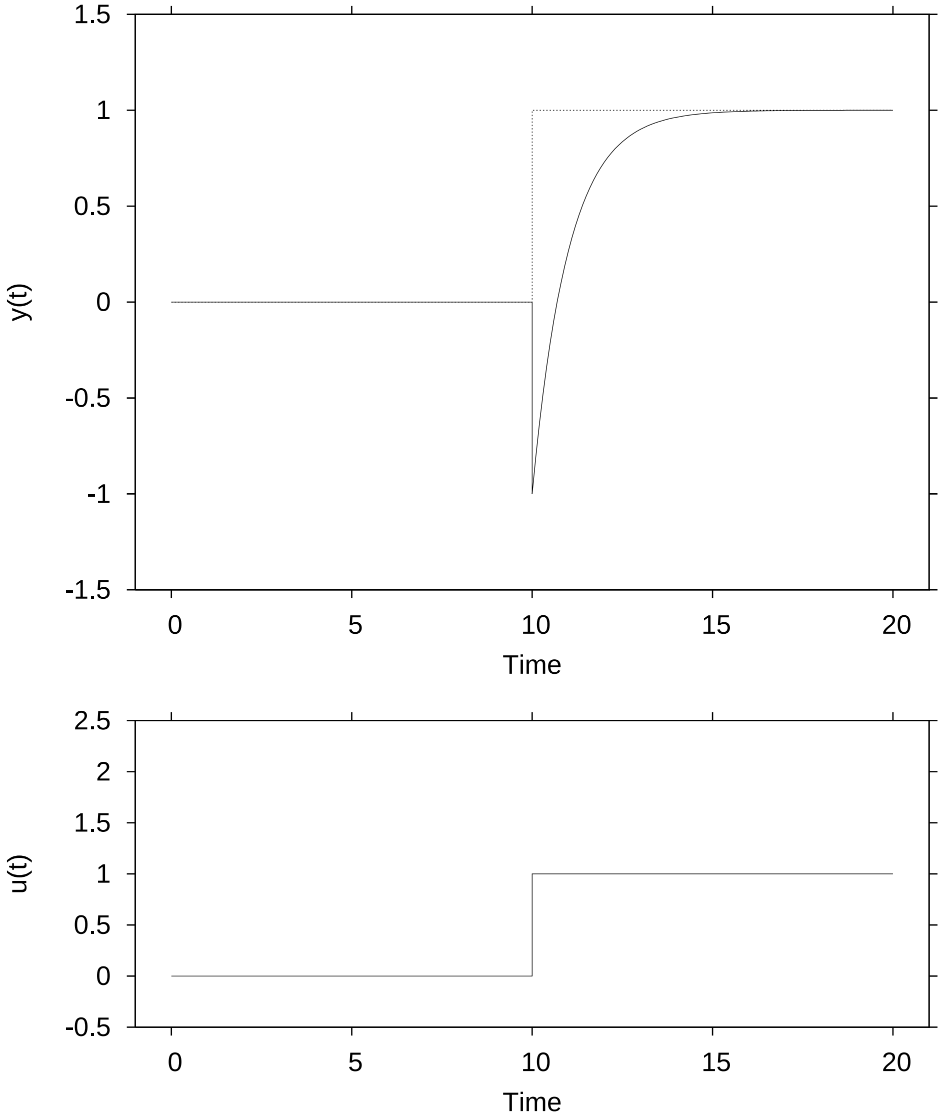

Figure 3.6: Input and response for the system $(-s+a)/(s+a)$ using an internally stable feedback.

This optimal feedback controller produces the control action and output shown in Figure 3.6. The ISE for the controller is $2/a$. In this sense, the RHP zero has limited the achievable performance of any stabilizing feedback controller.

Consider now the MPC controller. Since it does not satisfy Equation 1.1, one is led to consider what anticipatory control action, $u(t),\ t \in [0, \theta]$, should be taken if one knows a step change in setpoint is planned at $t = \theta$. This question becomes clearer if one instead considers the doubly infinite time horizon, $t \in (-\infty, \infty)$, and solves the following optimization problem,

$$\min_{u(t)} \Phi = \int_{-\infty}^{\infty} e^2(t)dt \tag{3.4}$$

The control action that achieves zero tracking error on the infinite interval is,

$$u(t) = \begin{cases} 2e^{a(t-\theta)}, & t \in (-\infty, \theta) \\ 1, & t \in [\theta, \infty) \end{cases} \tag{3.5}$$

This result can be checked using Equation 3.2 and $x(-\infty) = 0$ as the initial condition. Notice that this control action, shown as the dashed lines in Figures 3.7 and 3.8, is small (less than two) for all t, in contrast to Equation 3.3.

In any practical situation, one has only a finite prediction horizon. In order to examine the performance loss due to the finite horizon, $T,\ T < \theta$, consider the following approximation to Equation 3.5.[2]

$$u(t) = \begin{cases} 0, & t \in [0, \theta - T) \\ 2e^{a(t-\theta)}, & t \in [\theta - T, \theta) \\ 1, & t \in [\theta, \infty) \end{cases} \tag{3.6}$$

Substituting this input into Equation 3.2 with zero initial conditions gives,

$$y(t) = \begin{cases} 0, & t \in [0, \theta - T) \\ -2e^{-a(t+2T-\theta)}, & t \in [\theta - T, \theta) \\ 1 - 2e^{-a(t+2T-\theta)}, & t \in [\theta, \infty) \end{cases} \tag{3.7}$$

The ISE for this input is

$$\int_0^{\infty} e^2(t)dt = \frac{2}{a}e^{-2aT} \tag{3.8}$$

[2]This input can be shown to be optimal for a class of modified problems. In the modified problems, input or state penalties are added to Φ, or constraints are placed on u in order to remove the undesirable e^{at} term from u for $t > \theta$.

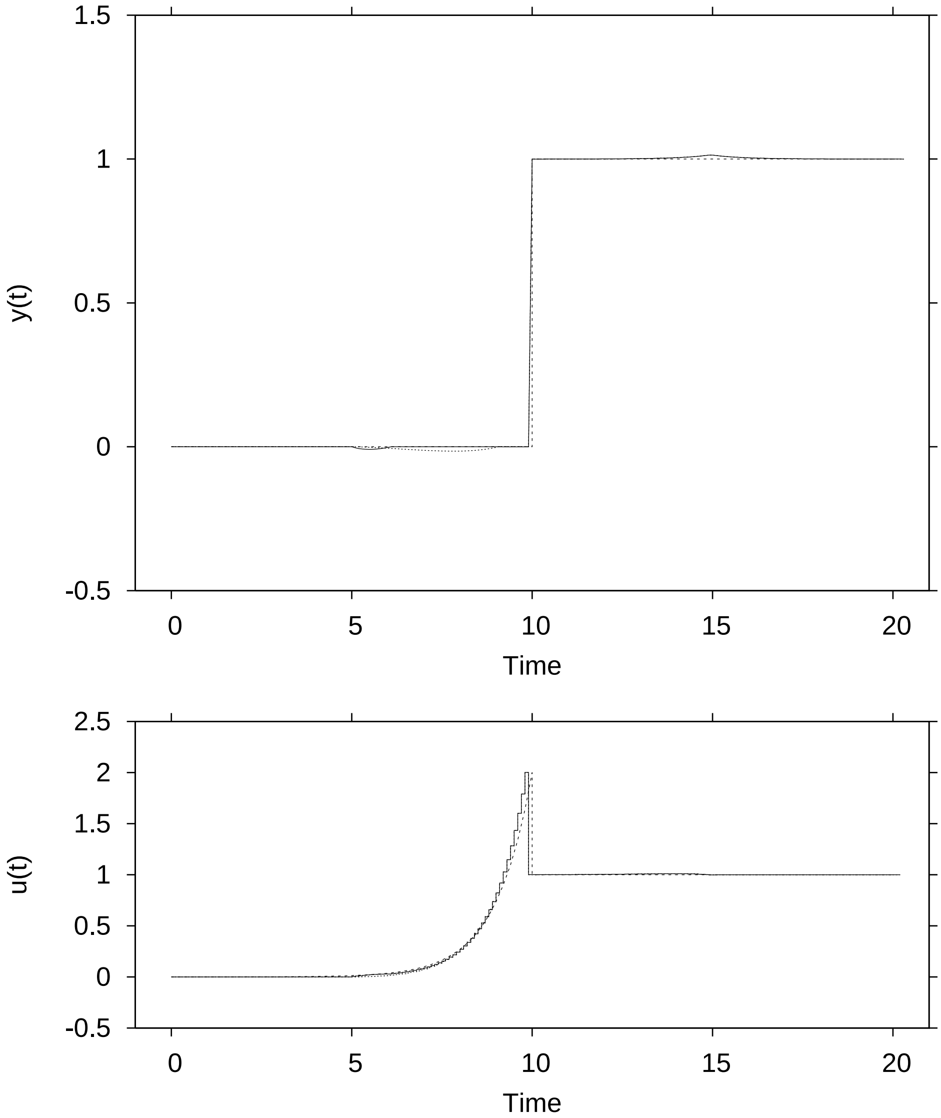

Figure 3.7: Ideal (dashed, $T = \infty$) and actual (dotted, $T = 5.0$, $C = 4.0$; solid, $T = 5.0$, $C = 1.0$) response, $y(t)$, and input, $u(t)$, for the system $(-s+a)/(s+a)$ using predictive DMC with a sample time of 0.1.

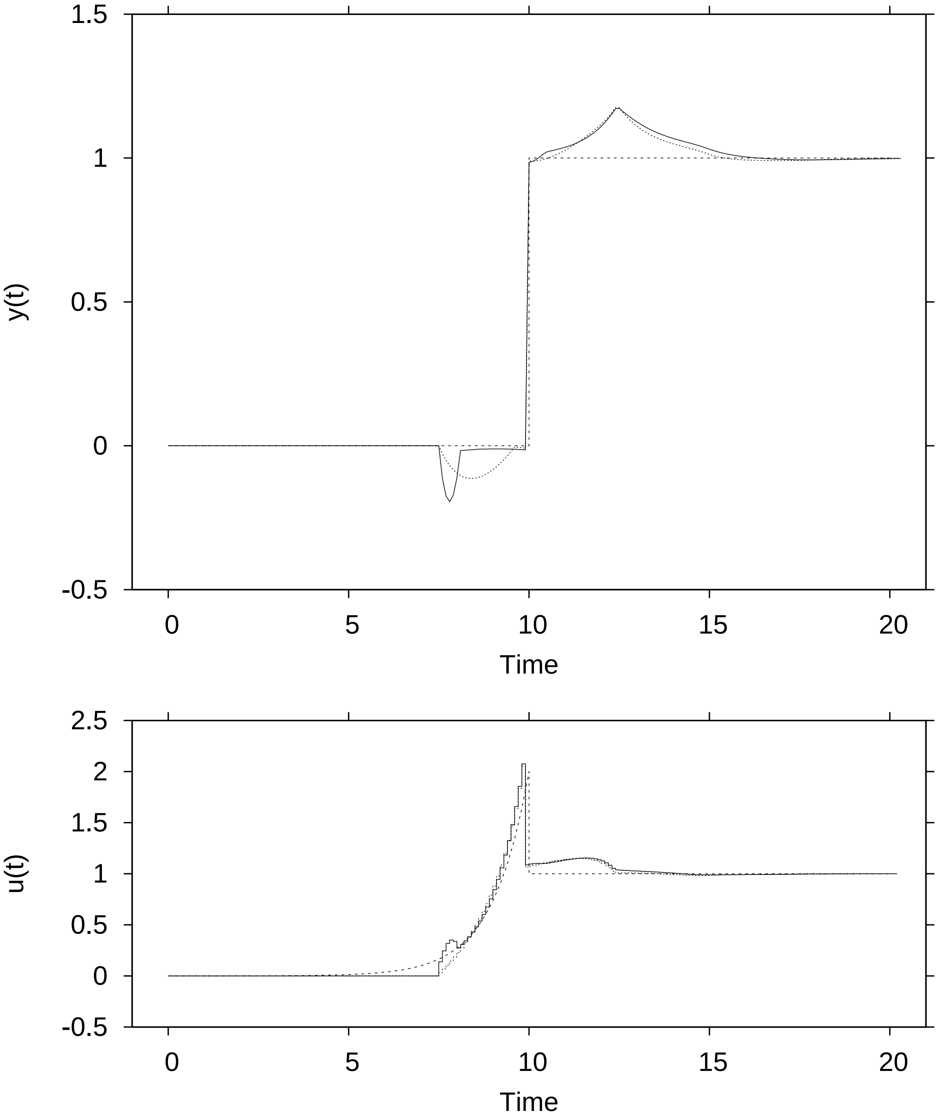

Figure 3.8: Ideal (dashed, $T = \infty$) and actual (dotted, $T = 2.5$, $C = 2.0$; solid, $T = 2.5$, $C = 0.5$) response, $y(t)$, and input, $u(t)$, for the system $(-s+a)/(s+a)$ using predictive DMC with a sample time of 0.1.

Notice that $T = 0$ (no prediction horizon) in Equation 3.8 recovers the optimal feedback controller's ISE. Since the ISE decreases exponentially with T, one does not require a long horizon to get a large benefit from MPC. For example, if $a = 1$, one can choose $T = 2.3$ to remove 99% of the error achieved with the best feedback controller. Notice also that the closer the zero is to the origin, the longer the prediction horizon must be to improve performance.

In designing the MPC controller, one should be careful to define the performance objective so the controller approximates Equation 3.5 rather than Equation 3.3 as the control action for plants with RHP zeros. There are several ways to accomplish this, most of which involve penalizing the input (or states). One could use Equation 1.2, for example, for the objective function. The controller would be stabilizing with even a modest penalty, R. The only criticism of this approach is that there is no real justification for the selection of R. In process control applications, R certainly does *not* reflect the cost of control action. An alternative approach is given in García and Morshedi [24]. They choose the control horizon, C, to be less than T. This is equivalent to placing an *infinite* penalty on the rate of change of the last control moves in the horizon. This has two benefits. First, the need for selecting a value of R is removed. Second, the demand that the control action become constant after sufficient time has elapsed, forces the controller to avoid the input in Equation 3.3, and the system is stable. In practice, it is also probably easier to find a good value of C than to choose an appropriate R.

Figures 3.7 and 3.8 show the performance of the standard DMC algorithm for several choices of control and prediction horizons. In all cases the control horizon is less than the prediction horizon and the controller is stabilizing. Notice that the standard DMC algorithm does a reasonably good job of finding Equation 3.6 as the input. The deviations in the interval $7 < t < 9$ in Figure 3.8 are directly attributable to making $C < T$. The parameterization of the input for $C < T$ (*i.e.* u constant for $t \in (t_0 + C, t_0 + T)$) prevents the controller from finding Equation 3.6 when the time of the setpoint change falls between the ends of the control horizon and the prediction horizon. The optimal u has its large negative jump in this time interval but the DMC controller has u parameterized as a constant. Figure 3.9 shows what the DMC algorithm forecasts for the input when the time of the setpoint change is between the control and prediction horizons. After some time passes, the jump in $u(t)$ occurs for $t < t_0 + C$ and the DMC parameterization of the input can track Equation 3.6 exactly. The effects of the small time of suboptimal behavior of the DMC algorithm can be reduced by making the horizon larger (compare Figures 3.7 and 3.8). Finally, a more flexible solution is to add a penalty term of the form $R(u - u_{ref})^2$ to the objective function in which u_{ref} is given by

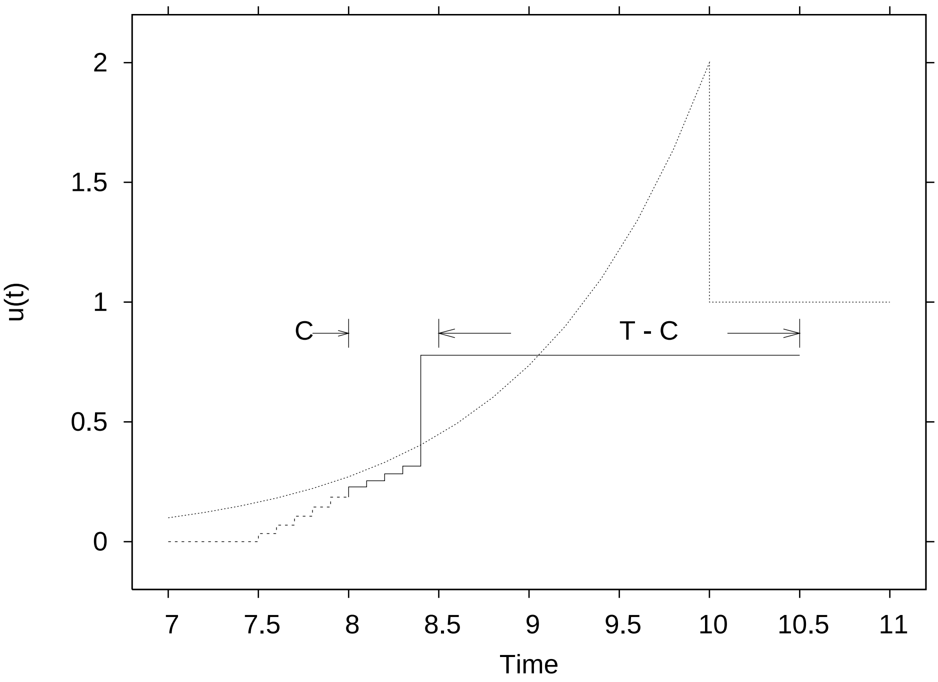

Figure 3.9: Performance of DMC at a given sample time with $T = 2.5$ and $C = 0.5$ for the system $(-s + a)/(s + a)$. The solid line represents the predicted optimal inputs, of which only one will be implemented. The dashed line shows the past inputs that have been used, and the dotted line shows the ideal input.

Equation 3.6 and R is adjustable. In this framework, the controller could have $C = T$ but avoid the unstable input of Equation 3.3.

The multivariable extension of the desirable properties of MPC for handling nonminimum phase elements is straightforward. Several papers in the Second Shell Process Control Workshop [54], for example, implement MPC algorithms on a process with three inputs, two loads, and seven outputs in which 24 of the 35 transfer functions have time delays.

3.4 Unstable Linear Plants

Clarke *et al.* [9] handle unstable plants with predictive control by using a "Controlled Auto-Regressive Integrated Moving Average" (CARIMA) plant model in which the unstable poles are included explicitly in the model. Other researchers have demonstrated stabilization of unstable plants with various MPC algorithms. Mayne and Michalska demonstrate the stabilization of the system

$$\begin{aligned}\dot{x}_1 &= -(1+\varepsilon x_2)x_2 + x_1 u\\ \dot{x}_2 &= \quad (1+\varepsilon x_1)x_1 - 4x_2 u\end{aligned}$$

with $\varepsilon = 0.0001$, 0.5, and 1.0 using a discrete time receding horizon controller.

Hidalgo and Brosilow [29] simulate the control of a styrene polymerization reactor at unstable operating points using a predictive control algorithm that solves for the control action to drive the process model to the setpoint over a finite horizon and filters the resulting controller output to accommodate modelling errors. Brosilow and Cheng [6] apply a similar algorithm to a unstable first-order plus dead time system.

Patwardhan *et al.* [49] apply a simultaneous solution and optimization approach to the control of an unstable continuous stirred tank reactor (CSTR) and compare the results to global linearization techniques. In another application to CSTR models, Brengel and Seider [5] use a controller based on linearizing the process model over the prediction horizon.

Lee *et al.* [39] describe modifications to the standard DMC algorithm to stabilize integrating processes. Their modifications are unable to handle the general case of systems with unstable poles, though it is possible to control some systems by approximating the behavior of the unstable system with an integrating model (see the example given below).

Finally, Schmid and Biegler [65] use a Newton-type control strategy to simulate the control of a fluid catalytic cracking process.

The standard DMC algorithm described above, however, is unable to handle unstable processes due to an unstable transfer function between u_1 and d in Figure 3.1.

To represent an unstable process with the DMC model, one may choose ε to be an extrapolation formula that approximates the unstable growth of the output due to past inputs. Within the framework of the DMC feedback structure, however, one can only choose ε to describe an integrator, not any arbitrary unstable plant. This approximation may work in practice for some systems, but will fail if the plant-model mismatch becomes too large. In order to overcome this deficiency and stabilize arbitrary unstable plants, one must design a feedback that avoids the internal stability problems of the DMC algorithm.

To demonstrate the ability of this modified algorithm to handle unstable processes, consider the plant with a pole at $s = 1$. (*i.e.* q strictly outside the unit circle)

$$\frac{dx}{dt} = x + u$$

with

$$x(t = 0) = 0$$

The sample time for the simulation is 0.1, with control and prediction horizons both equal to 1.0. Figure 3.10 shows the system response and control action for the unconstrained system. In the plot of the output, $x(t)$, the solid line shows the plant output, the dotted line shows the model prediction at each sample time, and the dashed line is the desired behavior. The difference between the plant and model prediction at steady state is due to model error caused by the specific choice of the extrapolation formula, which in this case is $\varepsilon = [\,0\ \cdots\ 0\ -1\ 2\,]$. This represents an integrating process and the model will have two poles at $q = 1$ (again, one arises from the fact that the DMC model is in written in terms of Δu).

The spike in the input at $t = 10.0$ is due to the fact that the system is not constrained and there are no penalties for large changes in the input. This undesirable behavior can be eliminated by the addition of input constraints but is not discussed here because the resulting system would be nonlinear. The smaller spikes are due to the model error introduced by choosing an extrapolation formula for an integrator in the model when the plant has a pole at $s = 1$. Note that the spikes occur at intervals equal to the length of the prediction horizon. The effect of this model error could also be minimized by the introduction of input constraints or penalties.

The limitation of modelling the process as an integrator rather than a system with an unstable pole is due to the step response model implicit in the DMC algorithm. If the step response model is replaced with a state space formulation, this restriction is removed, and much better performance is possible because the model may more closely represent the plant's behavior.

3.5 Nonlinear Algorithms

In this section we will apply the nonlinear optimal control algorithm to an open-loop unstable nonlinear process that exhibits multiple steady-state behavior in order to demonstrate several interesting features of the feedback.

For these examples, the full nonlinear open-loop optimal control problem, NLP3, is solved at each sampling time using the method developed in Section 2.3. A portion of the predicted input is implemented and the plant output is measured. This information may be used with the process model to estimate a disturbance as in the QDMC algorithm or to update the values of the model parameters.

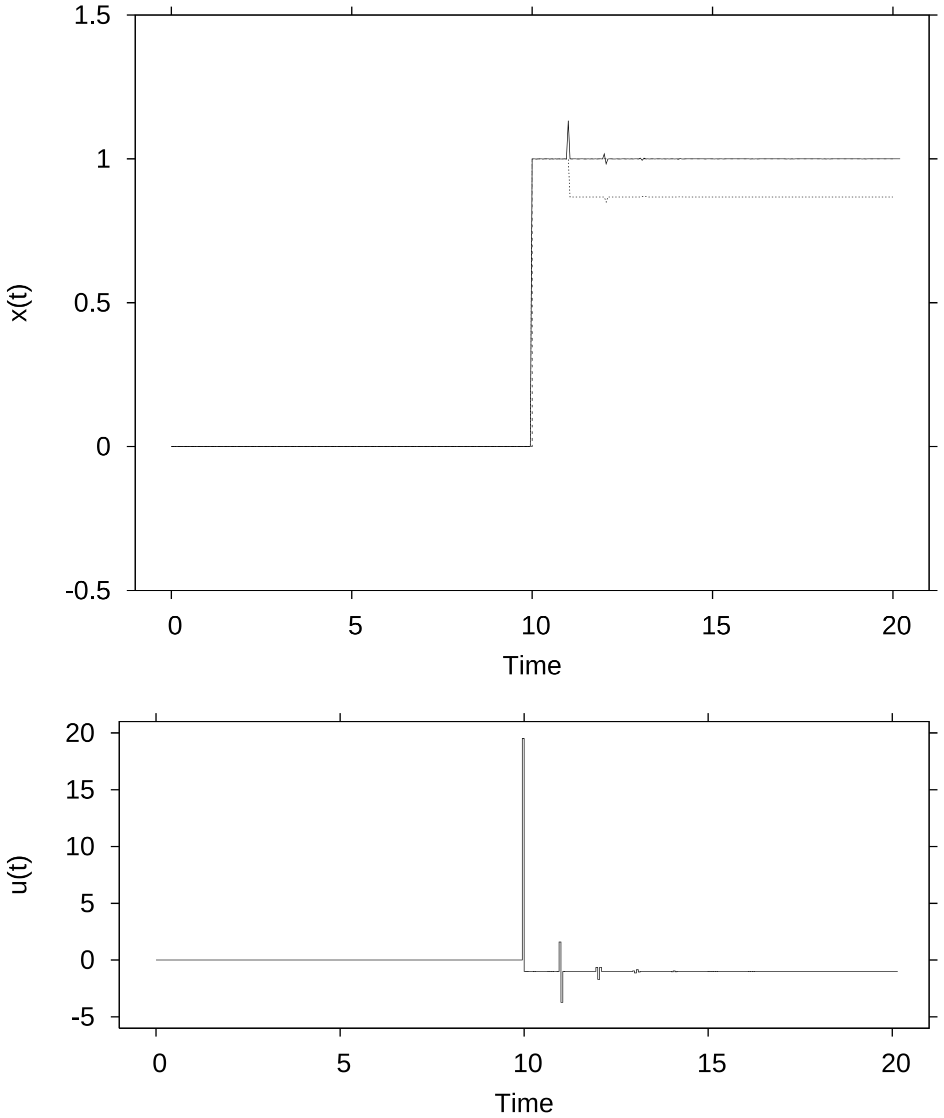

Figure 3.10: Performance of DMC for the plant $\dot{x} = x + u$ with an integrating model. The solid lines show the actual input and output, the dotted line shows the model prediction at each sample time, and the dashed line is the desired behavior.

	open-loop stable steady state	open-loop unstable steady state	washout (stable) steady state
x	1.5122	0.9951	0.0
s	0.1746	1.5301	4.0
D	0.3	0.3	0.3

Table 3.1: Steady state values of the states (biomass, x and substrate, s) for a single value of manipulated variable, D.

3.5.1 Continuous bioreactor

As an example of an open-loop unstable nonlinear process, consider the following model of a continuous bioreactor with substrate inhibition [1]. For some regions of the parameter space, this system exhibits multiple and unstable steady-state behavior. The state equations are

$$\frac{dx}{dt} = (\mu - D)x$$
$$\frac{ds}{dt} = (s_f - s)D - \frac{\mu\, x}{y}$$

$$\mu = \frac{\mu_{\max}\, s}{k_m + s + k_1 s^2}$$

with

$$x(t=0) = x_0 \qquad s(t=0) = s_0$$

The dilution rate, D is manipulated to control the cell mass concentration, x. The other state and parameters are the substrate concentration, s, the specific growth rate, μ, the yield of cell mass y, and the substrate concentration in the feed stream, s_f. The values of the parameters used in the simulation of the plant are $y = 0.4$, $s_f = 4.0$, $\mu_{\max} = 0.53$, $k_m = 0.12$, and $k_1 = 0.4545$. Steady-state solutions of the model equations for these parameter values are given in Table 3.1. The values of the parameters $\mu_{\max}$ and k_m used in the model equations are 20% greater than the values used in the simulation of the plant.

The open loop behavior for the plant at these conditions is shown in Figure 3.11.

For the simulations shown here, the following performance objective is

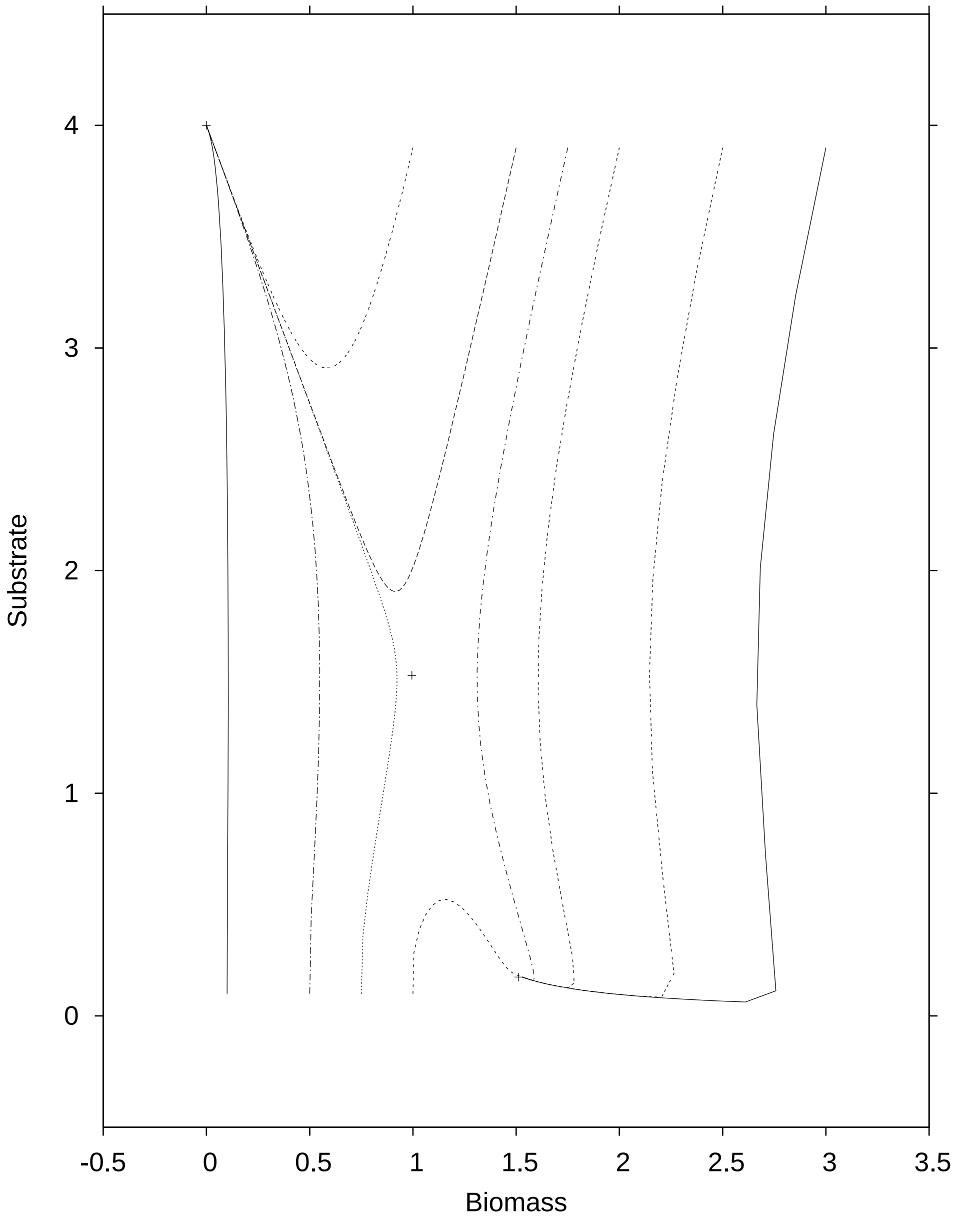

Figure 3.11: Open-loop behavior or the bioreactor model.

chosen to obtain a specified cell mass concentration

$$\Phi = \int_{t}^{t+T} (x_{set} - x)^2 dt' \tag{3.9}$$

and the open-loop optimal control step is solved by approximating the model equations using orthogonal collocation on finite elements. The resulting nonlinear program is solved using `NPSOL`, a Fortran implementation of a successive quadratic programming algorithm [28]. For the specific simulations shown subsequently, the prediction horizon and control horizon are both 0.6, and the sample time is 0.2.

Figure 3.12 shows the response of the system to a setpoint change from the unstable to the stable steady state.

In this case, the general approach is very much like the standard DMC algorithm—the controller provides a sequence of optimal inputs and a small number of these are used to drive the plant. A measurement is obtained, compared to the output predicted by the model, and the difference is fed back to the controller. The model equations are allowed to evolve separately from the plant without any difficulty because the parameter error is small and the setpoint is a stable stationary point.

While this approach is acceptable for open-loop stable linear systems, it is not particularly well suited to nonlinear systems. Even if the only steady state in the entire operating region is stable, the performance will suffer if the model is allowed to move too far from the plant because for nonlinear systems, the plant's behavior will vary throughout the operating region.

To see how this can happen, even for the same small errors in model parameters, consider Figure 3.13, which shows the behavior of the same system when moving from the stable to the unstable setpoint. In the plots of the biomass and substrate profiles, the solid lines show the response of the plant, and the dotted lines show the response of the model to variations in the dilution rate.

Even though the closed-loop system is input-output stable, there are unexpected deviations near $t = 5.0$, and the model prediction bears no resemblance to the plant output. In fact, the model has moved away from the intended setpoint and has come to rest on the other *stable* steady state. Had this system not had a second stable steady state, the difference between the model prediction and the plant output would have grown until the controller eventually failed from numerical difficulties.

In order to overcome this problem, the control scheme is modified to reset the initial condition of the model to the measurements of the states at each time step. As Figure 3.14 shows, the resulting system is stable, and the model does not wander off to some other part of the operating region. Although

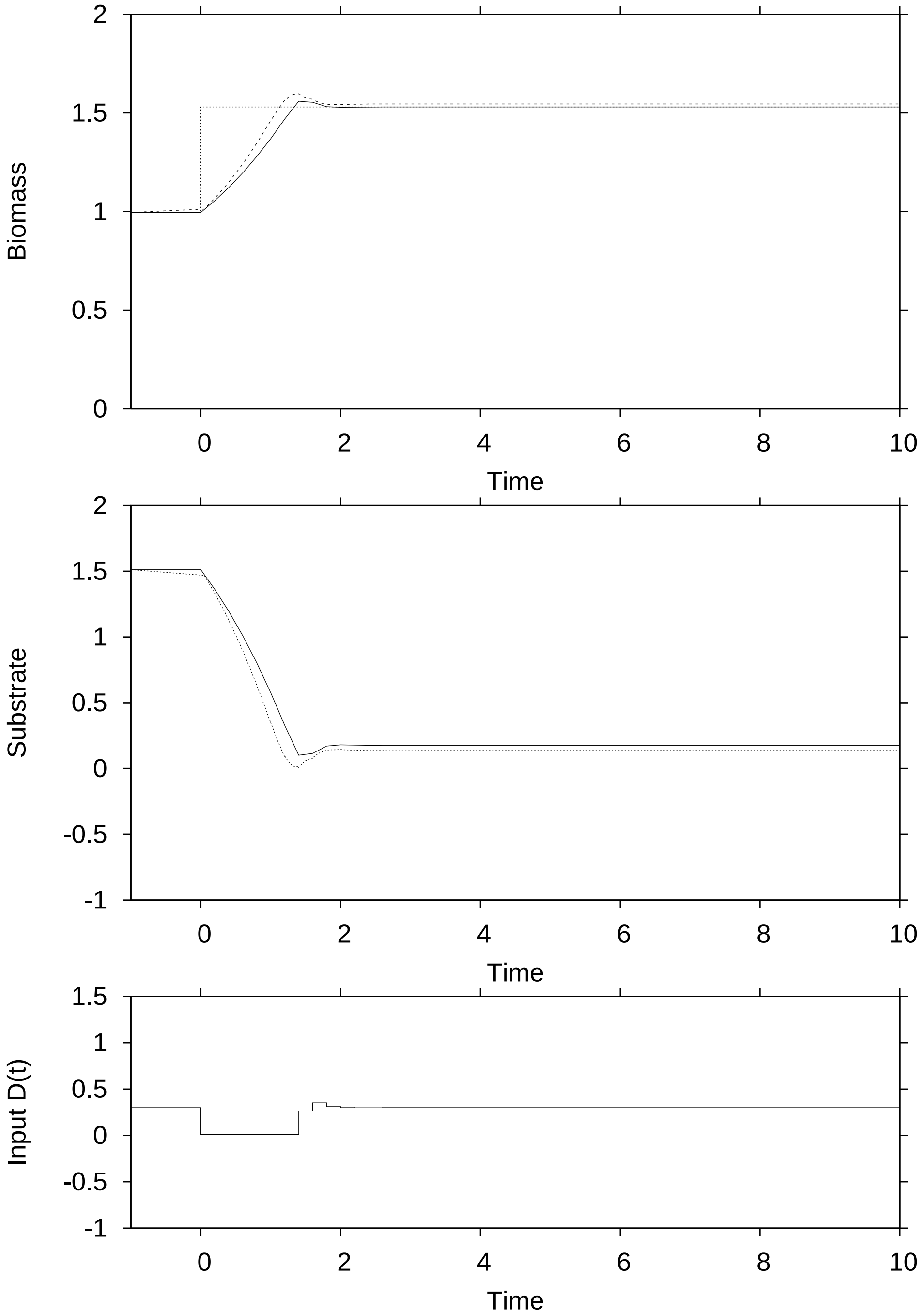

Figure 3.12: Performance of the nonlinear model-predictive controller without resetting the model initial condition for an open-loop stable target. Biomass (solid) and substrate (dotted) concentrations (top and middle) and dilution rate (bottom) for the parameters given in Table 3.1.

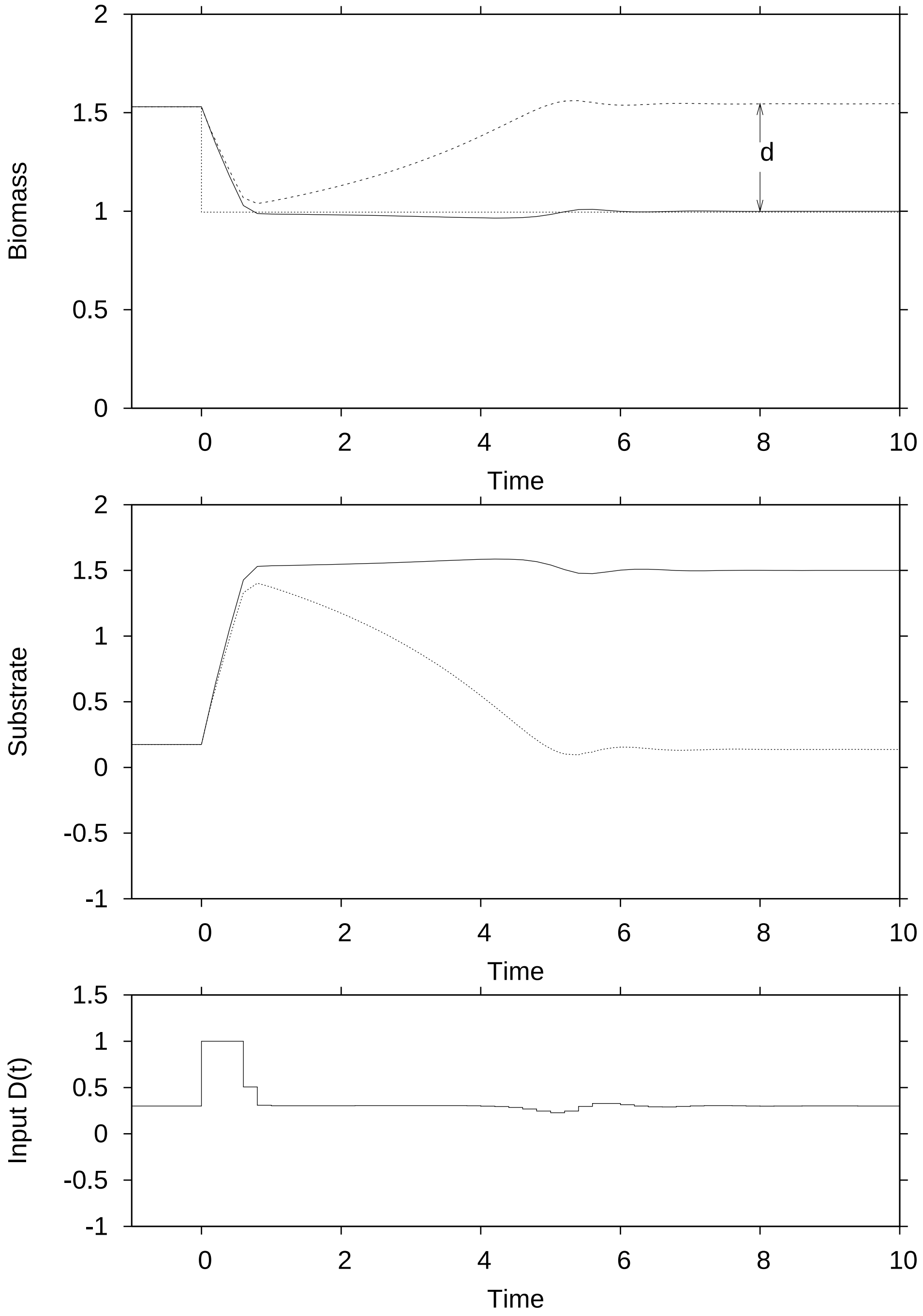

Figure 3.13: Performance of nonlinear model-predictive controller without resetting the model initial condition for an open-loop unstable target. The model prediction reaches a different (stable) steady state than the plant. Biomass (solid) and substrate (dotted) concentrations (top and middle) and dilution rate (bottom) for the parameters given in Table 3.1.

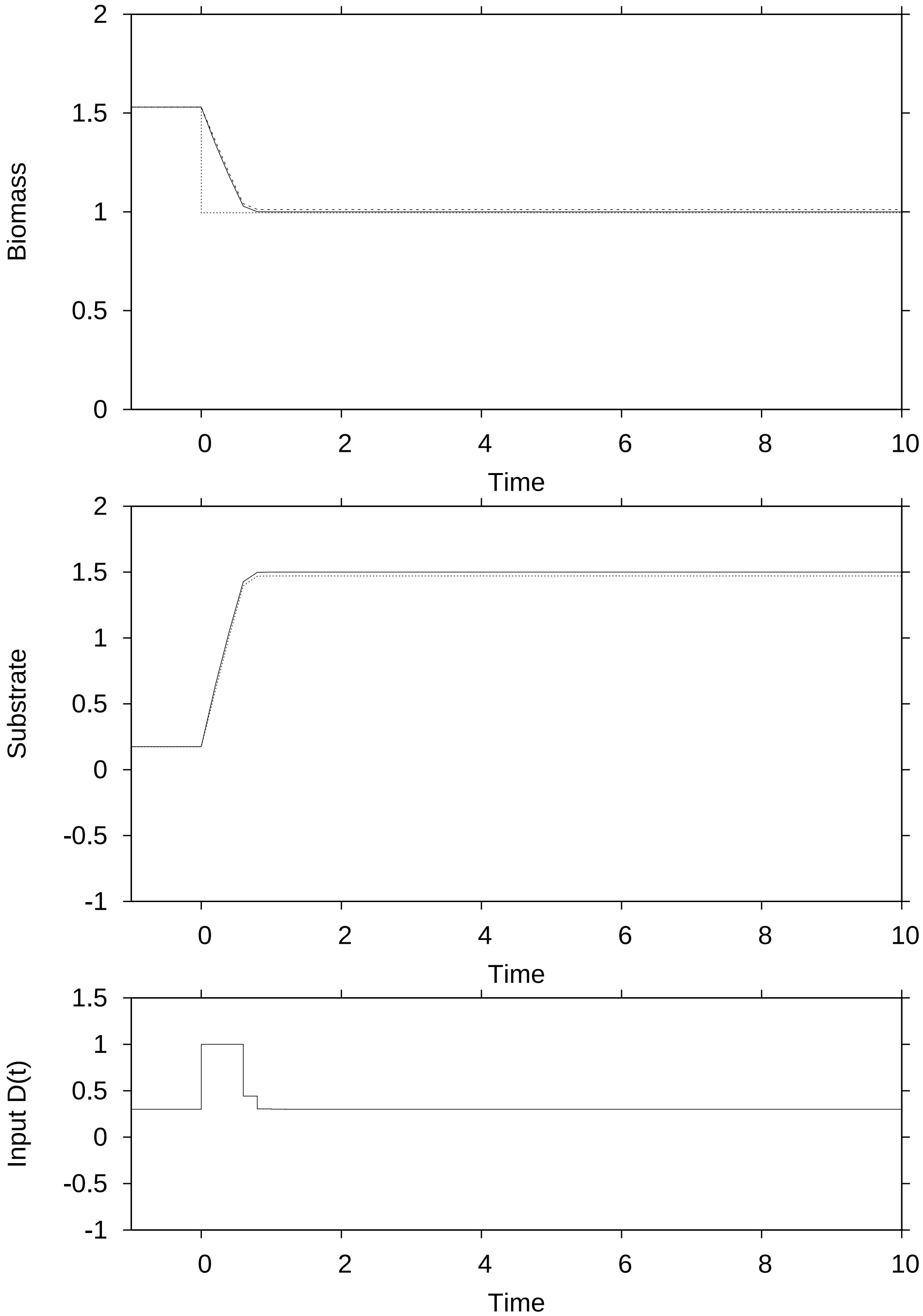

Figure 3.14: Performance of nonlinear model-predictive controller with resetting the model initial condition for an open-loop unstable target. The model prediction remains near the plant output. Biomass (solid) and substrate (dotted) concentrations (top and middle) and dilution rate (bottom) for the parameters given in Table 3.1.

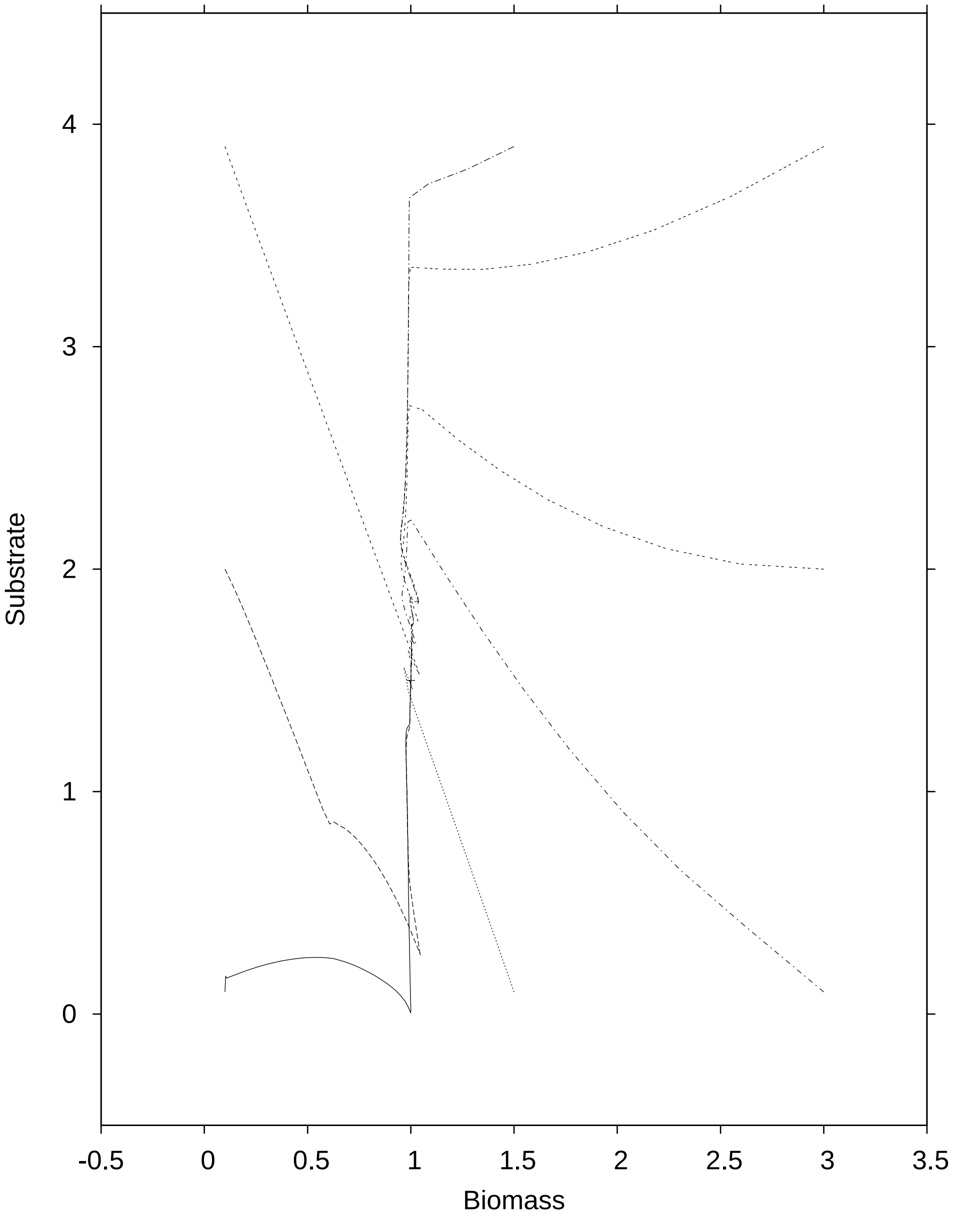

Figure 3.15: Closed-loop behavior for output step disturbance model only.

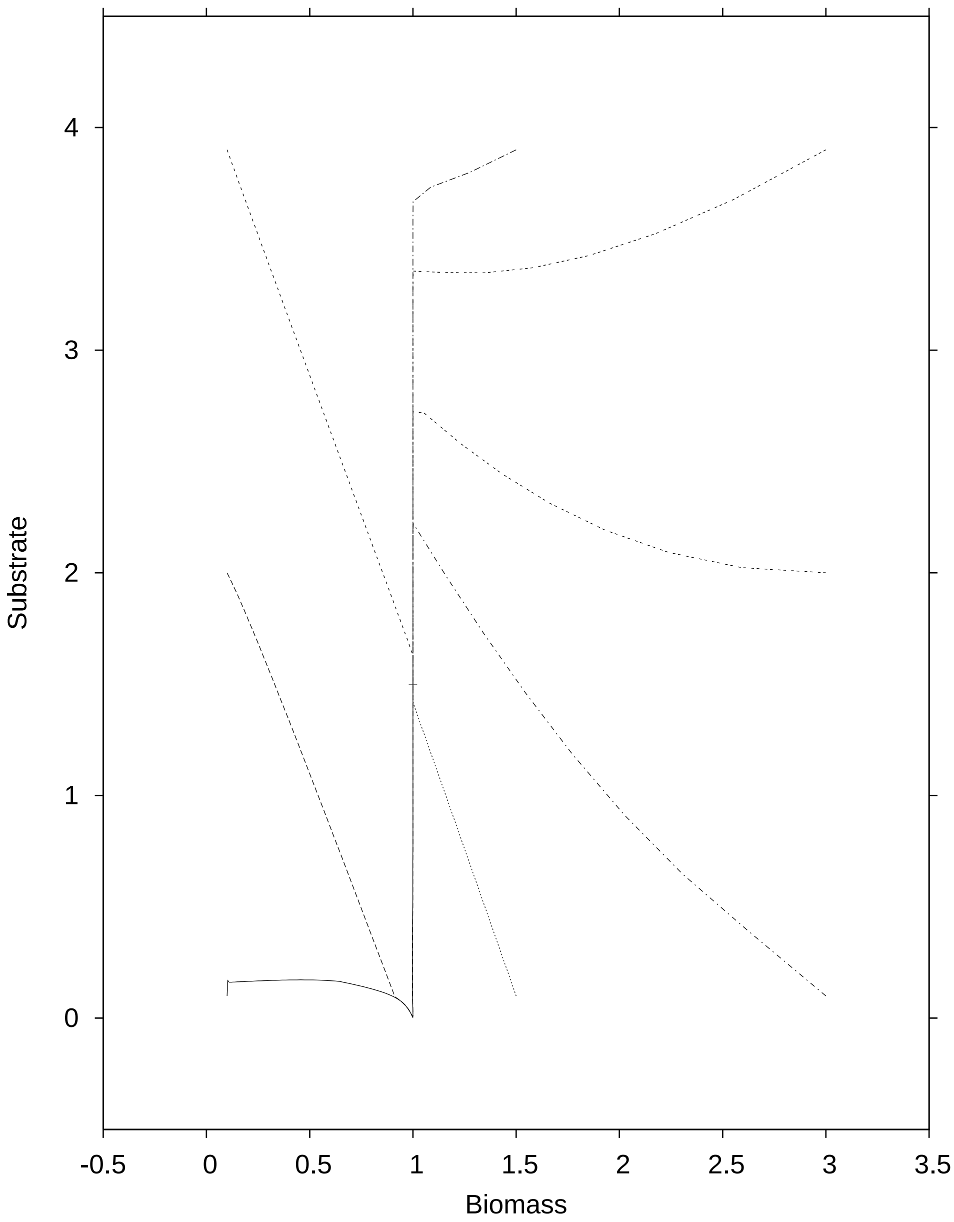

Figure 3.16: Closed-loop behavior with state estimation and output step disturbance model.

both states are assumed to be measured and measurement noise is assumed negligible this is not always true in practice. In such cases, it is reasonable to assume that the model initial conditions may be obtained from a state estimation scheme.

Figures 3.15 and 3.16 show the behavior of these algorithms from a number of initial conditions. Both controllers are able to drive the process to the desired setpoint in approximately the same length of time, but the performance is improved by resetting the initial conditions of the model equations to the measured output. This improved performance is due to the fact that the model remains close to the process and more accurately reflects its behavior.

The light in the north had mysteriously vanished,
but the correspondent took his course from the wide awake captain.

— Stephen Crane, *The Open Boat* (1897)

Chapter 4

Conclusions and Future Directions

4.1 A Final Example

Although NMPC has been shown to be effective in solving a number of problems, several important issues remain unresolved. Simulation studies show that the NMPC is able to stabilize some processes, but there is no general result that will guarantee stability of the feedback for open-loop unstable processes. The robustness of these algorithms to modelling error is also not well understood.

To demonstrate once more some of the strengths and weaknesses of nonlinear model-predictive control and to provide a possible starting point for future explorations of these methods, consider the following set of nonlinear differential equations[1]

$$\begin{aligned} \dot{x} &= u \\ \dot{y} &= x \\ \dot{z} &= x^3 \end{aligned} \tag{4.1}$$

with $x(t = 0) = x_0$, $y(t = 0) = x_0$, and $z(t = 0) = x_0$ and the setpoint $x = y = z = 0$.

It is interesting to note that this system is of the form

$$\dot{x} = f(x) + g(x)u$$

[1]This example was suggested by Professor Hermes of the Department of Mathematics at the University of Colorado, Boulder.

but it cannot be transformed into a linear and controllable system via state feedback and coordinate transformation, because it fails to satisfy

$$\text{Rank}\left[g(x), ad_f g(x), \ldots, ad_f^{n-2} g(x), ad_f^{n-1} g(x)\right] = n$$

where n is the dimension of the system and $ad_f^k g(x)$ is defined as

$$ad_f^k g(x) \equiv \left[f, ad_f^{k-1} g\right](x) \tag{4.2}$$

for and $k \geq 1$ and $ad_f^k g(x) = g(x)$

$$[f, g](x) = \frac{\partial g}{\partial x} f(x) - \frac{\partial f}{\partial x} g(x)$$

is the Lie bracket of f and g [30].

It is also impossible for this system to approach the origin directly from the regions with $y < 0$ and $z > 0$ or $y > 0$ and $z < 0$ because $dz/dy = x^2$ is always positive. This model structure forces the controller to move the system in ways that are not always obvious.

The finite-horizon predictive controller can be effective for controlling the system, but there are a number of interesting problems associated with it. For example, there are multiple optima for the controller calculation depending on the initial guess for the input profile. The particular local optimum that is obtained probably depends on the controller's horizon length. A long horizon might help the controller to find the global solution, though it may not be a simple matter to determine how long that horizon should be. It might be possible to determine this along with solving the optimization, but that introduces another level of difficulty for the controller because it becomes necessary to determine a trade off between the least cost and minimum time objectives.

Figure 4.1 shows the behavior of the closed-loop system for eight starting points around the origin for the objective function

$$\Phi = \sum_{i=1}^{N} x_i^2 + y_i^2 + z_i^2$$

with the additional constraints

$$x_N = y_N = z_N = 0.0$$

enforced at the end of the prediction horizon. The values of x_i, y_i, and z_i are predicted values of the states at the sampling times, and with a sampling time

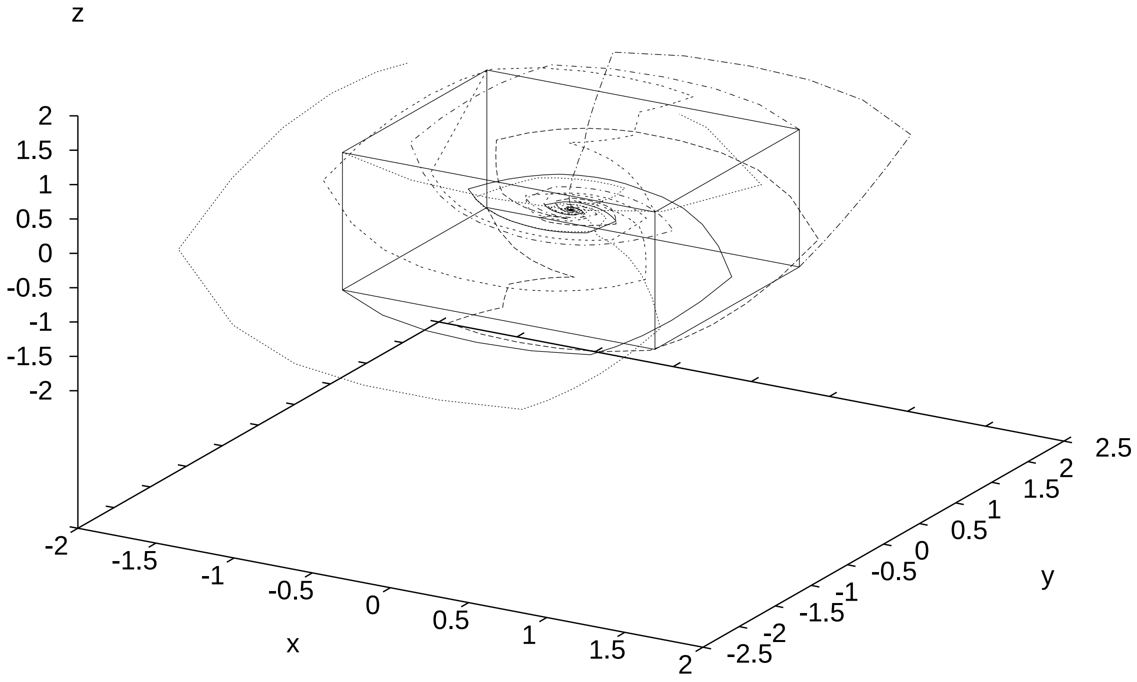

Figure 4.1: Global response of the controller for the model given by Equation 4.2.

of 1.0 time units and piecewise constant inputs. In all cases, the control and prediction horizons were chosen (arbitrarily) as $N = 10$ sampling times.

The controller appears to have no significant trouble in eventually driving the system to the origin. The path taken does not always appear to be very direct, and this is probably a result of the structural restrictions of the process noted above. It may also be that the controller might have found a more direct path given a different initial estimate of the optimal solution, or a different objective function. Of course, the task of selecting an objective function to achieve "good performance" is always a problem with optimization-based methods.

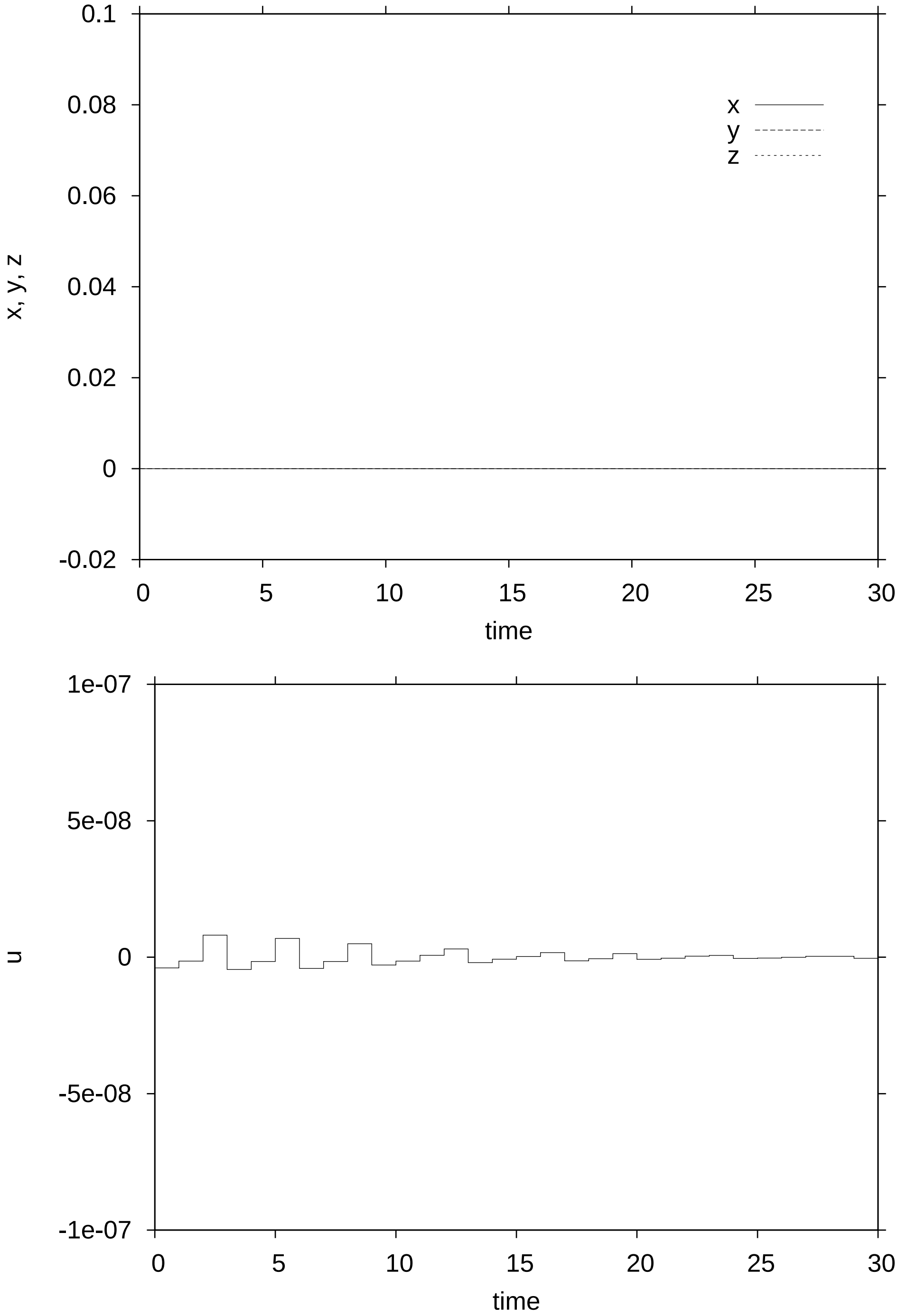

Figure 4.2: State and manipulated variable profiles.

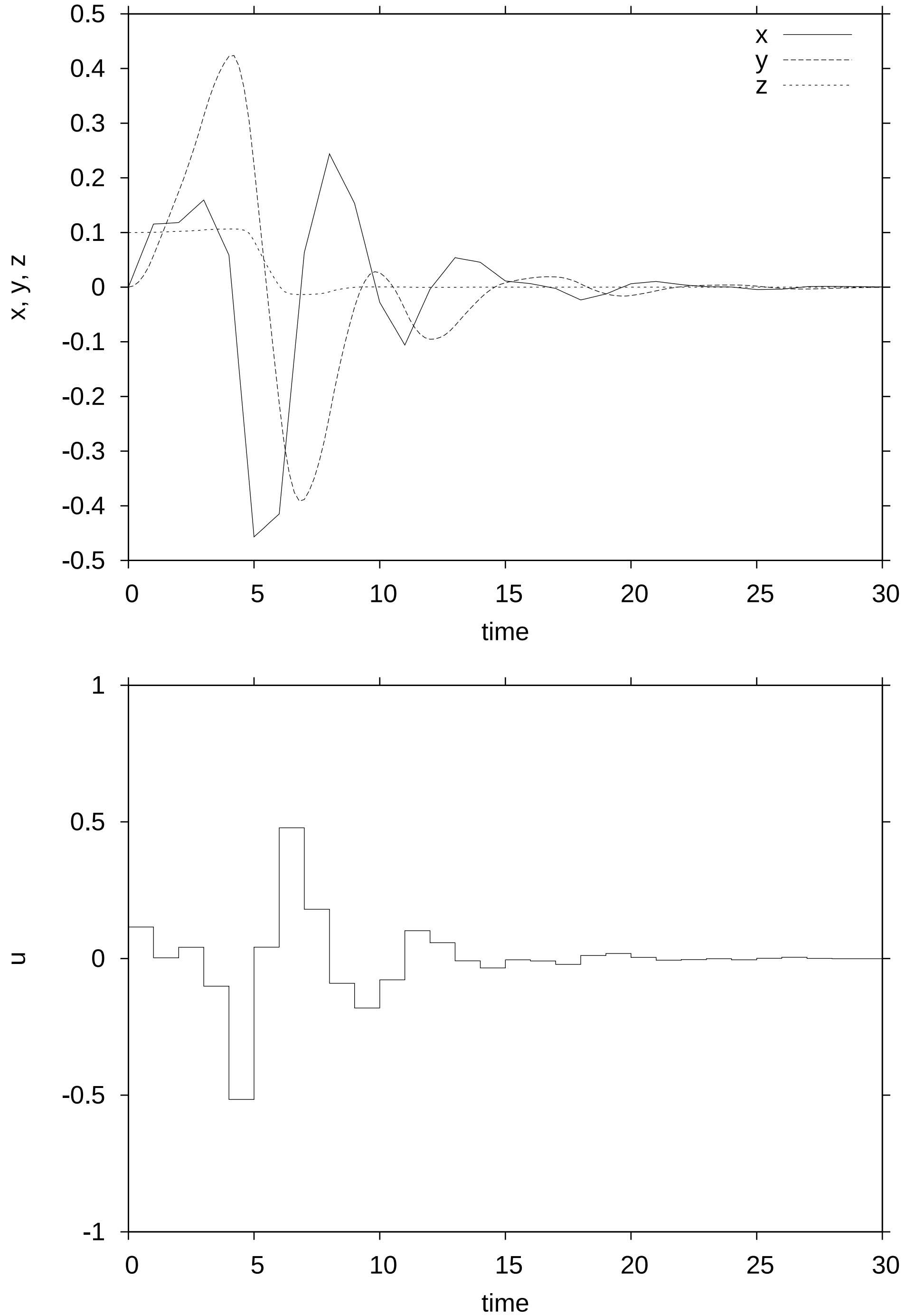

Figure 4.3: State and manipulated variable profiles.

The constraints imposed at the end of the prediction horizon were required to ensure that the system eventually reached the setpoint. Attempts to solve the problem without them sometimes resulted in steady-state offset for z. For an infinite horizon controller, the offset would not be tolerated. For this problem, adding constraints at the end of the prediction horizon is equivalent to solving the controller subproblem on an infinite horizon because once the states are at zero, they will remain there with no further control action.

The closed-loop behavior of the system for the initial condition $x_0 = y_0 = 0.0$ and $z_0 = 0.1$ is shown in Figure 4.2 for the controller without final time constraints and in Figure 4.3 for the controller with the final time constraint. In the first case, the controller will tolerate a small amount of offset in z over the finite horizon because to eliminate it would require x and y to move away from zero, and the cost for doing so too great.

In the second case, the controller is required to move all the states to the origin by the end of the prediction horizon by the introduction of hard constraints. Because of this, the controller moves x and y away from zero in order to eliminate the offset for z. Although the controller has achieved the setpoint for all the states, doing so has required a significant deviation for the other states. Given this behavior, one may decide that it is reasonable to tolerate the offset.

The existence of multiple local minima is another interesting feature of this example. Figure 4.4 shows the series of solutions obtained by the NMPC controller for the objective function, sampling time, and control and prediction horizons given above. Each point represents the first manipulated variable that the controller computes for a given starting point.

The sequence of solutions shown in Figure 4.4 was obtained by solving the controller equations for point 1 in the figure using the simultaneous solution techniques described in Chapter 2. This represents *an* optimal solution for the given objective function and starting point $x_0 = y_0 = 1.0$, $z_0 = -1.0$. This solution was then used as the starting point for the state and manipulated variable profiles for the next calculation, moving from the point 1 to the point 2 and then back toward 1 again.

It is important to remember that the initial condition includes the complete state and manipulated variable profiles from the previous run, not just the initial x, y, and z. In solving each problem, only the initial value of z was changed from the previous solution. This small change in the initial z introduces a slight infeasibility at the start of each calculation that in the region of $z = 0.03$ with z decreasing, is sufficient to move the controller to a different local minimum.

The initial infeasibility for this problem could be removed by adjusting

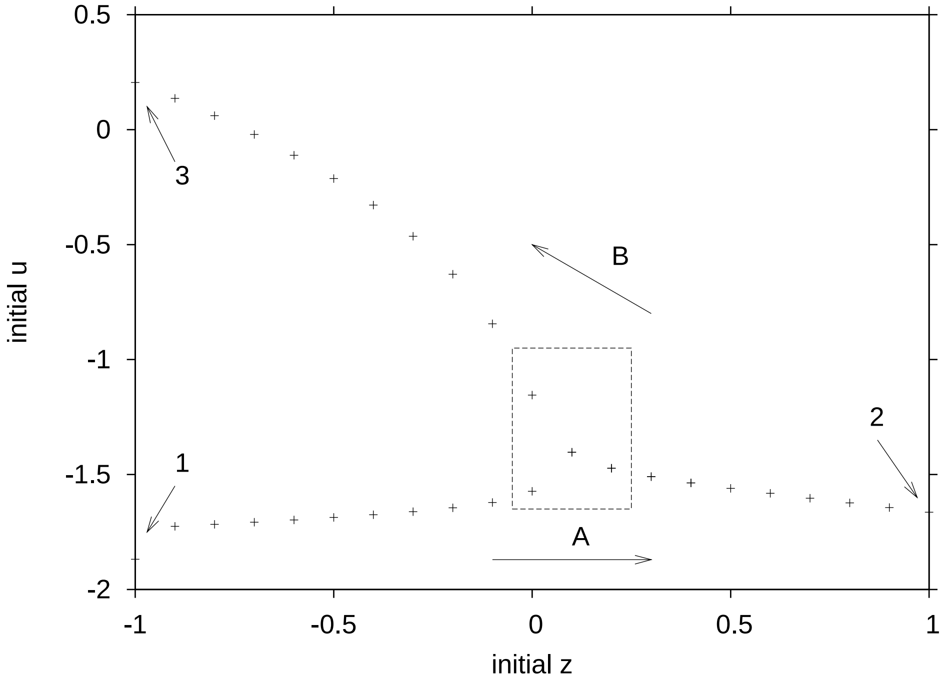

Figure 4.4: Initial u for various initial conditions ($x_0 = y_0 = 1.0$).

the state or manipulated variable profiles (or both), but it is not likely that doing so would change the overall results shown here. In addition, because most nonlinear programming codes allow the initial guess to be infeasible it is usually not necessary to take extra care to begin from a feasible point.

The closed-loop behavior for the system starting from the optimal solutions corresponding to the points labeled 1 and 3 in Figure 4.4 is shown in Figures 4.6–4.8. For these two cases, the initial condition for the states (at $t = t_0$ only, not over the entire prediction horizon) is identical, but the different initial guesses for the input and state variable profiles causes the system to behave in very different ways even though each controller calculation finds an optimal solution over the finite horizon. Such behavior is obviously undesirable, and the problem of finding global optimal solutions deserves further attention.

Note that both of these simulations begin from optimal points, with initial guesses for the input and state profiles that are both feasible and consistent, so the problem of starting out infeasible is not an issue here. Beginning from the same point with a model and integration method of similar accuracy, a simultaneous solution method would result in the same behavior.

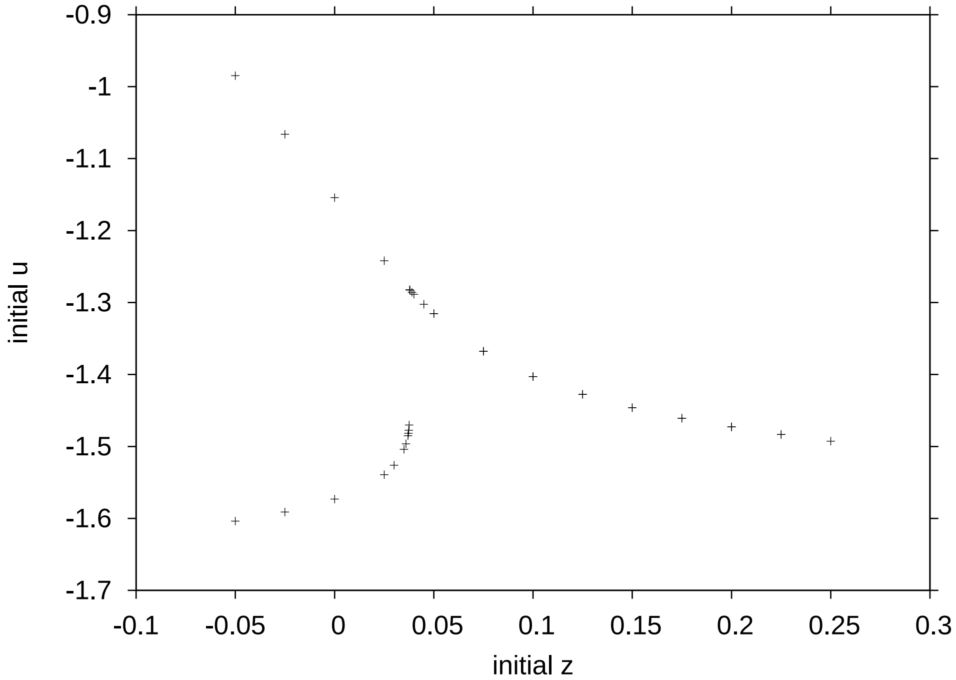

Figure 4.5: Initial u for various initial conditions ($x_0 = y_0 = 1.0$).

	x	y	z	total
Figure 4.7	4.6526	4.0788	9.0554	17.7868
Figure 4.8	2.7996	14.4102	0.3009	17.5108

Table 4.1: Sum of squares deviations at the sampling points for the example of Section 4.1.

It is also interesting that although the manipulated variable profile shown in Figure 4.7 seems quite irregular, it does not change drastically if the initial starting point is moved slightly.

Finally, in terms of overall performance, the two controllers are almost identical even though the output and control action shown in Figure 4.8 has a much more pleasing appearance than that of Figure 4.7. The overall objective function, summarized in Table 4.1 and computed from the *plant* outputs over the 30 time units shown in the plots, is essentially the same for both cases.

There are many other approaches one might attempt for this problem besides adding final time constraints to this, or solving it as a minimum time problem with an associated performance index. For example, it may be de-

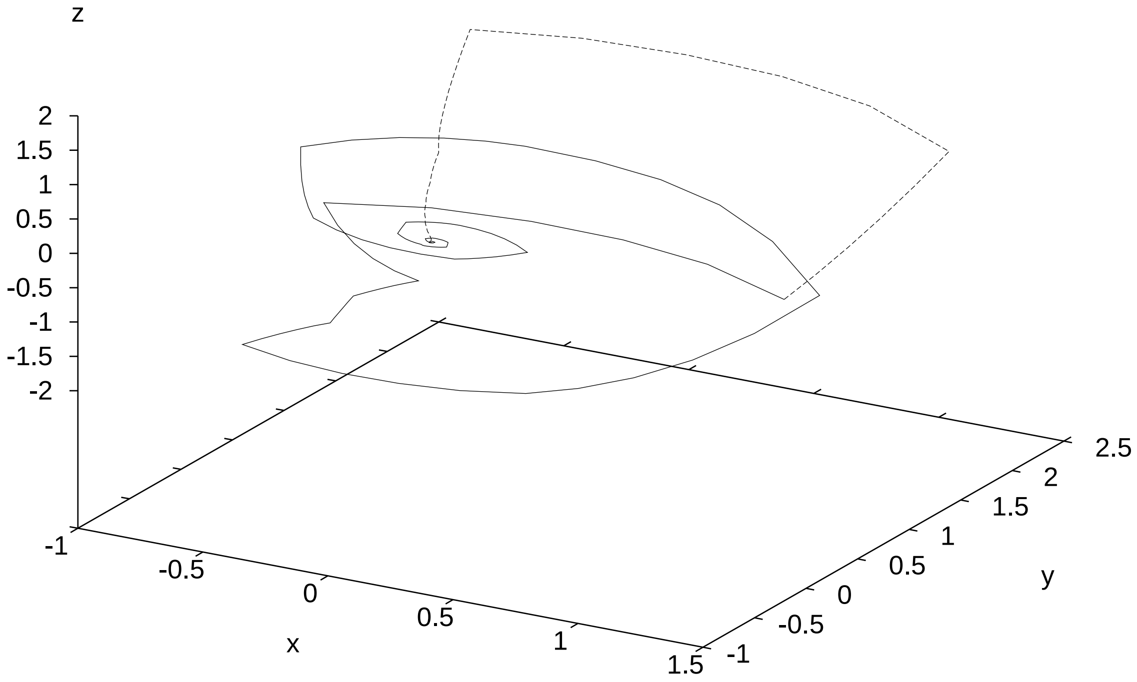

Figure 4.6: State trajectories for two different initial guesses.

sirable to penalize the input, or weight the terms in the objective function differently. There are many open issues related to tuning the model-predictive controller and the development of efficient algorithms for global minimization.

4.2 Concluding Remarks

Model-predictive control has been demonstrated to be a useful method for solving batch and continuous time problems. The open-loop control calculation has been solved using nonlinear programming techniques by approximating the dynamic constraints with algebraic equations using orthogonal collocation

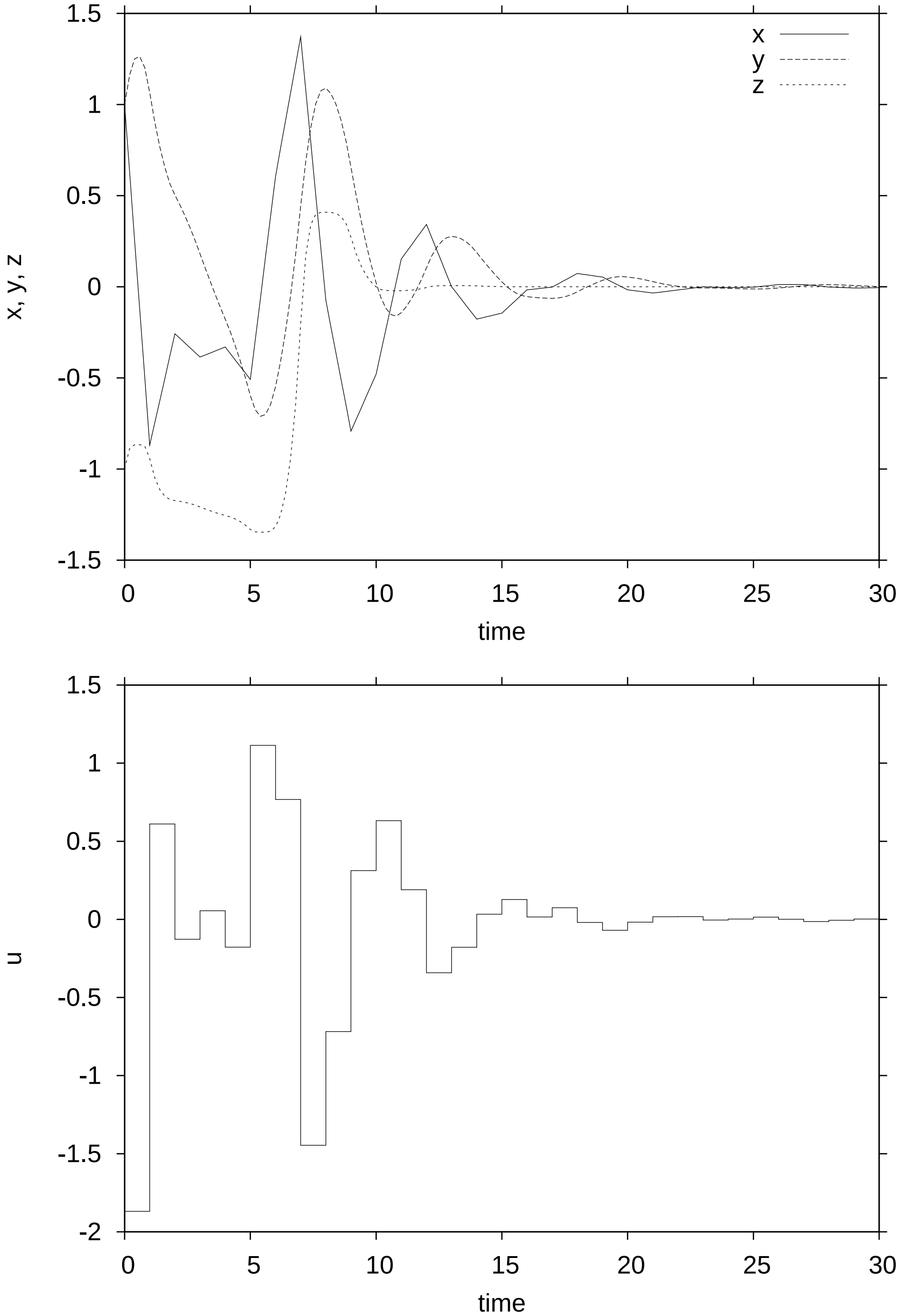

Figure 4.7: State and manipulated variable profiles.

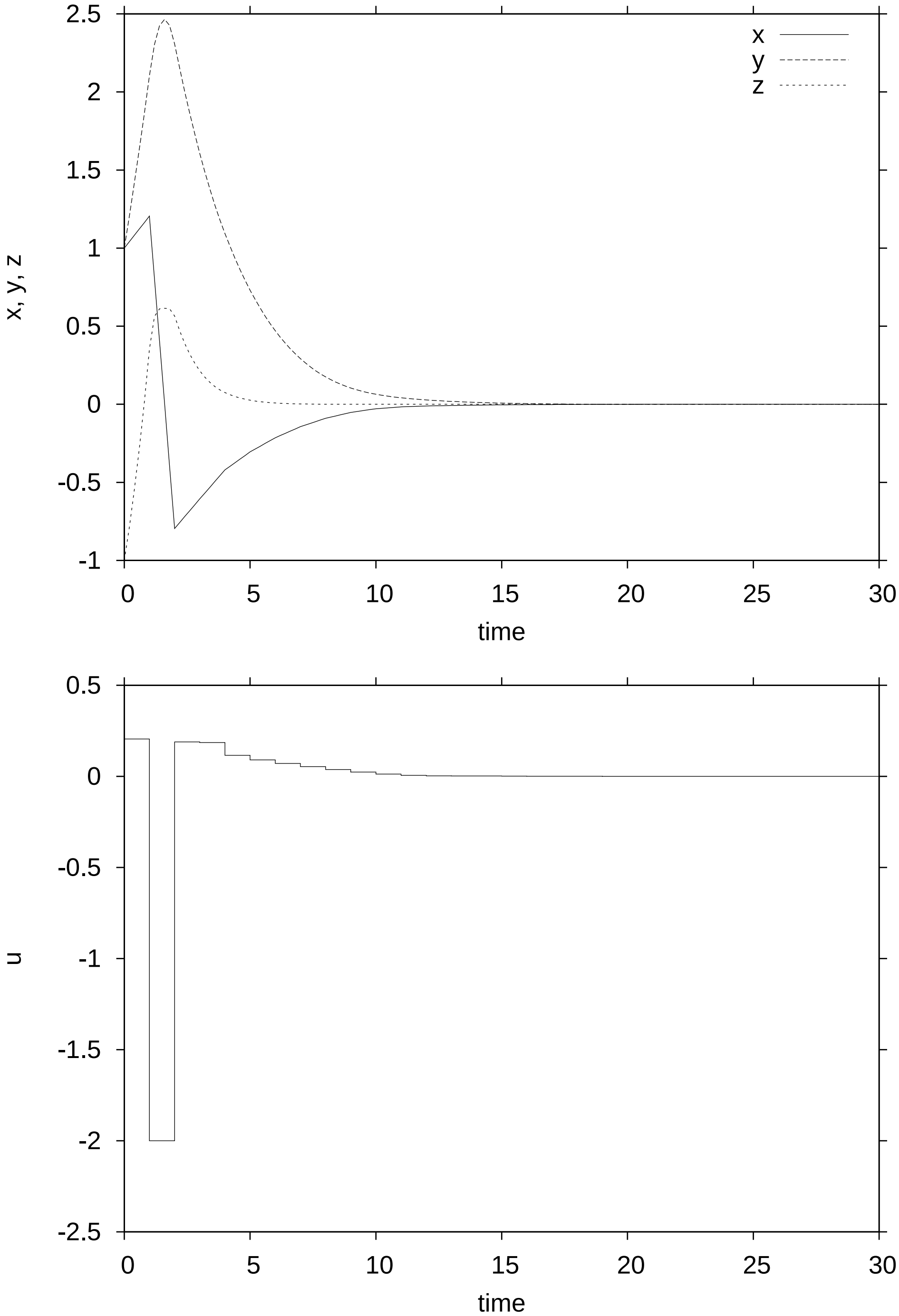

Figure 4.8: State and manipulated variable profiles.

on finite elements. Elimination of the dynamic constraints by solving them as a separate problem was also discussed.

Both of these solution techniques offer certain advantages and have a number of weaknesses. Neither is clearly superior for all process models of interest. The sequential approach provides a convenient method for solving stiff systems of differential-algebraic equations accurately but requires manipulated variables to be parameterized as piecewise constant functions in order to efficiently compute gradients for the optimization algorithm. The simultaneous approach allows the manipulated variables to be parameterized by any piecewise continuous function, but makes it more difficult to compute bounds on the accuracy of the modelling solution.

Both methods require models that can be evaluated for any set of inputs, and in the case of the simultaneous approach, any combination of inputs or states within the simple bounds. Ensuring that it is always possible to evaluate the modelling equations requires careful consideration of the properties of the process model prior to implementation.

Using the simultaneous optimization and model solution approach simplifies the sensitivity analysis of the nonlinear program used to compute control moves. This analysis makes it possible to determine the sensitivity of the optimal solution of the dynamic optimization with respect to changes in model parameters, and an expression has been developed for determining bounds on the manipulated variables corresponding to a given change in the performance index.

A new set of computational tools, described in detail in Appendix A, has been developed to solve the open-loop nonlinear optimal control calculation and simulate the closed-loop behavior of the nonlinear model-predictive controller. These tools, together with the implementation of the QDMC controller described in the appendix, have been used to solve all of the example problems discussed in this dissertation.

In Chapter 3 the properties of feedback systems based on these optimal control techniques were examined using the QDMC controller. A new, clear and concise block diagram has been developed for this linear model-predictive control algorithm, from which it is possible to obtain the QDMC closed-loop transfer function in terms of the shift operator, q.

Advances in computing technology have made it possible to consider NMPC as a real alternative to other control schemes, even when there are obvious deficiencies in the implementation of the underlying optimization algorithms. In the case of the simultaneous solution technique described in Section 2.3, there is an opportunity to significantly improve the efficiency of the algorithm by taking advantage of the sparse structure of the optimization.

Finally, apart from any modifications of the optimization algorithm, or any major improvements in computing hardware, practicing engineers must be able to construct low order nonlinear models that capture the key features of the process to be controlled in order for NMPC to be successful. Currently, there are few, if any, convenient methods for constructing such nonlinear process models. The development of tools to aid in model building is perhaps more important to the acceptance of NMPC than any refinements of the control algorithms.

Someday baby...

— B. B. King, *Live in the Cook County Jail* (1970)

Appendix A

Computational Tools

This appendix documents a set of programs developed for the minimization of nonlinear functions subject to differential-algebraic constraints, and two programs for the solution of linear model-predictive control problems using an algorithm much like Shell's Dynamic Matrix Control. These tools were developed in order to solve batch and continuous optimal control problems, but they are not necessarily limited to that class of computations.

The complete set of programs is available from the author. They are provided "as is" and without any express or implied warranties, including, without limitation, the implied warranties of merchantability and fitness for a particular purpose.

A.1 Nonlinear MPC Software

The software for solving nonlinear MPC problems consists of a collection of (mostly standard) Fortran 77 programs that communicate via data files. The original intent of this design was to make each of the individual programs smaller and easier to understand and maintain. After struggling with a monolithic design, it appeared that breaking the problem up into a number of smaller pieces would lead to much faster development and greater flexibility. Unfortunately, the limitations of programming in Fortran make this very difficult if not simply impossible. This and other programming language issues are discussed in more detail below. The remainder of this section describes the programs, required data files, user-supplied subroutines, and concludes with an example problem.

A.1.1 The programs and their data files

The following sections briefly describe the various programs and the required input data files. The input and output files have names of the form `problem.extension`, where `problem` is the name of the particular problem being solved and `.extension` is a mnemonic for the information the file contains. For example, the file containing problem dimensions and other general information has the extension `.dim`. All the files corresponding to one problem are given the same base name so that they may be easily grouped together.

In the following descriptions, a list of filename extensions is given to indicate which files are used by each program for input and output. A summary of the information contained in each of the individual files follows the program descriptions.

soopt. This program attempts to solve the optimal control problem described in Chapter 2 using a simultaneous optimization and solution approach based on orthogonal collocation on finite elements. The resulting nonlinear program is solved using `NPSOL` [28]. The user-supplied subroutines that `soopt` requires are described in Section A.1.1 below.
usage: `soopt [-fvgh] [-o|e] name`

- **-f** Overwrite existing output files.
- **-v** Print version number and exit.
- **-g** Compute gradient info for sosens.
- **-h** Print this help message and exit.
- **-o** Solve controller optimization.
- **-e** Solve estimation problem.

Input files: `.dim`, `.oic`, `.oms`, `.mst`, `.ymeas`, `.umeas`, `.par`
Output files: `.ypr`, `.upr`, `.grd`, `.otr`, `.par`

soplnt. Given a manipulated variable profile over a finite element mesh, `soplnt` solves the system of differential-algebraic equations describing the plant using `DASSL` [51]. The user-supplied subroutines that `soopt` requires are described in Section A.1.1 below.
usage: `soplnt [-fvh] name`

-f Overwrite existing output files.

-v Print version number and exit.

-h Print this help message and exit.

Input files: `.dim`, `.oms`, `.upr`, `.pic`, `.stm`
Output files: `.ypl`

soexec. This program combines the functionality of several of the other programs so that continuous-time simulations may be performed. Simulations begin by solving an optimal control problem with subroutines from `soopt`. Given the optimal manipulated variable from that calculation, the plant output is computed with `soplnt`. The initial conditions and new element meshes for the next simulation are computed with `updic` and `updmsh` respectively. To be complete, the optimizer should be called again to (optionally) compute state or parameter estimates, but that is not yet implemented.

The various parts of this program communicate via files, in exactly the same manner as if the computations were carried out by running the individual programs in sequence, and intermediate data is stored in an archive file with `archiv`.
usage: `soexec [-fvsidh] [-n num] [-l step] [-r step] name`

-f Overwrite existing output files.

-v Print version number and exit.

-s Compute sensitivity information.

-i Reset model initial condition.

-d Compute output disturbance term.

-h Print this help message and exit.

-l step
Set the step for the left boundary to **step**.

-r step
Set the step for the right boundary to **step**.

Input files: `.dim`, `.oic`, `.oms`, `.mst`, `.ymeas`, `.umeas`, `.par`
Output files: `.ypr`, `.upr`, `.grd`, `.otr`, `.par`

archiv. This program stores intermediate data in a file for later processing by `plfgen` for plotting. When used by `soexec`, it stores all the information available, which includes the data required to plot the closed-loop behavior as well as any intermediate predictions.
usage: `archiv [-fvh] -[o|e] name`

- **-f** Overwrite existing archive file.
- **-v** Print version number and exit.
- **-h** Print this help message and exit.
- **-o** Work on optimization files.
- **-e** Work on estimation files.

Input files: `.dim`, `.oms`, `.ypl`, `.ypr`, `.upr`, `.sns`
Output files: `.arc`

plfgen. This program reads the archive file produced by `archiv` and generates files containing columns of data suitable for plots of the system response as a function of time or as a phase portrait.
usage: `plfgen [-fvh] [-n num] name`

- **-f** Overwrite existing output files.
- **-v** Print version number and exit.
- **-h** Print this help message and exit.
- **-n num**
  Set the number of points to plot in each interval to **num**.

Input files: `.dim`, `.arc`
Output files: `.yplt`, `.uplt`, `.ypplt`, `.ysplt`, `.usplt`, `.mvsplt`

sosens. This program reads gradient data optionally produced by `soopt` and computes the sensitivity derivatives described in Section 2.3.2.
usage: `sosens [-fvh] name`

- **-f** Overwrite existing output files.
- **-v** Print version number and exit.
- **-h** Print this help message and exit.

Input files: `.grd`
Output files: `.sns`

updic. Given the output files produced by `soopt` for the simulation of the controller and process model, and `soplnt` for the simulation of the plant, this program updates the initial conditions for the optimization and for the model and plant simulations. This program is normally used by `soexec` for continuous time simulations.
usage: `updic [-fvpmidh] [-l step] [-r step] name`

-f Overwrite existing output files.

-v Print version number and exit.

-p Update plant initial condition.

-m Update model initial condition.

-i Reset model initial condition.

-d Compute output disturbance term.

-h Print this help message and exit.

-l step
Set the step for the left boundary to **step**.

-r step
Set the step for the right boundary to **step**.

Input files: `.dim`, `.oms`, `.ypl`, `.oic`, `.ypr`, `.upr`, `.ost`, `.mst`
Output files: `.oic`, `.pic`, `.mst`

updmsh. Given a step to move forward, `updmsh` updates the finite element meshes for the optimization and plant simulation.
usage: `updmsh [-fvh] [-o|e] [-l step] [-r step] name`

-f Overwrite existing output files.

-v Print version number and exit.

-h Print this help message and exit.

-o Work on optimization files.

-e Work on estimation files.

-l step
Set the step for the left boundary to **step**.

-r step
Set the step for the right boundary to **step**.

Input files: `.dim`, `.oms`
Output files: `.oms`

User-supplied subroutines. There are a number of subroutines that must be written to describe the system being solved and then compiled and linked with the driver programs described above. This section provides only a brief introduction to each of the routines and does not attempt to explain the use of each parameter in detail. Input files and complete subroutines for an example problem are given after the general descriptions of the user-supplied subroutines and input files.

Even the casual reader will probably notice that all of these subroutines have far too many parameters in their argument lists. This problem is addressed in Section A.3 below.

Subroutines required by `soopt`. The following sections briefly describe the user-supplied subroutines required to solve the optimal control problem. The subroutines that compute gradient information are optional, provided that the appropriate options have been set in the input files.

fcn. This is the user-supplied subroutine that evaluates the right hand side functions that define the set of differential-algebraic equations. The argument list is very similar to the argument list for the residual function required by `DASSL`, augmented with vectors to give values of additional decision variables, `x`, the manipulated variables, `u`, and, if they are available, measurements of past state and manipulated variables, `ymeas` and `umeas`.

```
      subroutine fcn (x, nx, y, nde, u, nu, t, rpar, ipar, fvect,
     $     ymeas, umeas, nym, num, nstate, mode, optest)
*
      integer          nx, nde, nu, ipar(*), nym, num, nstate, mode
      logical          optest
      double precision x(*), y(*), u(*), t, rpar(*), fvect(*),
     $                 ymeas(*), umeas(*)
```

fcnj. This is the user-supplied subroutine that evaluates the augmented Jacobian matrix of the functions defined in the subroutine `fcn`. This routine is optional if finite differences are being used to compute gradients of the constraints.

The following matrices of partial derivative are required:

$$\mathtt{pdx} = \frac{\partial\,\mathtt{fvect}}{\partial \mathtt{x}} \qquad \mathtt{pdy} = \frac{\partial\,\mathtt{fvect}}{\partial \mathtt{y}} \qquad \mathtt{pdu} = \frac{\partial\,\mathtt{fvect}}{\partial \mathtt{u}}$$

The subroutines `fcn` and `fcnj` are called repeatedly for different values of t over the prediction horizon. In these functions, the user does not have access to the values of the states, and manipulated variables over the entire prediction horizon, but only at a single point.

```
      subroutine fcnj (x, nx, y, nde, u, nu, t, rpar, ipar, pdx,
     $     pdy, pdu, ldpd, ymeas, umeas, nym, num, nstate, mode,
     $     optest)
*
      integer          nx, nde, nu, ipar(*), ldpd, nym, num, nstate,
     $                 mode
      logical          optest
      double precision x(*), y(*), u(*), t, rpar(*), pdx(ldpd,*),
     $                 pdy(ldpd,*), pdu(ldpd,*), ymeas(*), umeas(*)
```

obj. Given a ton of information about the problem, the user is asked to sort it all out and return the value of the objective function. Since the objective function may depend on values of the state, manipulated, and other variables over the entire prediction horizon, this routine is given access to all of this information. For example, each row of the matrix `y` contains the values of the state variables at each collocation point on the finite element grid.

```
      subroutine obj
     $  (
     $  x,      nx,     y,      ldy,    nde,   u,      ldu,
     $  nu,     yelbnd, uelbnd, nyel,   nuel,  yroot,  uroot,
     $  inyrhb, inulhb, inurhb, ncpb,   nurpb, rhntrp, yquad,
     $  yset,   uset,   uquad,  uintrp, rpar,  ipar,   objf,
     $  ymeas,  ldym,   umeas,  ldum,   nym,   num,    nstate,
     $  mode,   setpnt, optest
     $  )
*
      integer
     $
     $  nx,      ldy,  nde,  ldu, nu,  nyel,   nuel, ncpb, nurpb,
     $  ipar(*), ldym, ldum, nym, num, nstate, mode
*
      double precision
     $
     $  x(*),            y(ldy,*), u(ldu,*), yelbnd(*),
     $  uelbnd(*),       yroot(*), uroot(*), rhntrp(*),
     $  yquad(*),        uquad(*), yset(*),  uset(*),
```

```
     $  uintrp(nurpb,*), rpar(*),  objf,     ymeas(ldym,*),
     $  umeas(ldum,*)
*
      logical  inyrhb, inulhb, inurhb, setpnt, optest
```

objgrd. Given the same information as `obj`, the user must supply the gradient of the objective function. This routine is not required if the finite difference option is selected for the objective function gradient.

```
      subroutine objgrd
     $  (
     $  x,      nx,     y,      ldy,    nde,   u,      ldu,
     $  nu,     yelbnd, uelbnd, nyel,   nuel,  yroot,  uroot,
     $  inyrhb, inulhb, inurhb, ncpb,   nurpb, rhntrp, yquad,
     $  uquad,  yset,   uset,   uintrp, rpar,  ipar,   objf,
     $  grad,   ymeas,  ldym,   umeas,  ldum,  nym,    num,
     $  nstate, mode,   setpnt, optest
     $  )
*
      integer
     $
     $  nx,      ldy,  nde,  ldu, nu,  nyel,   nuel, ncpb, nurpb,
     $  ipar(*), ldym, ldum, nym, num, nstate, mode
*
      double precision
     $
     $  x(*),            y(ldy,*),      u(ldu,*), yelbnd(*),
     $  uelbnd(*),       yroot(*),      uroot(*), rhntrp(*),
     $  yquad(*),        uquad(*),      yset(*),  uset(*),
     $  uintrp(nurpb,*), rpar(*),       objf,     grad(*),
     $  ymeas(ldy,*),    umeas(ldum,*)
*
      logical  inyrhb, inulhb, inurhb, setpnt, optest
```

constr. Also given all the information about the state of the solution over the entire prediction horizon, this routine may be used to specify additional algebraic constraints for the optimization.

```
      subroutine constr
     $  (
     $  x,      nx,     y,     ldy,    nde,    u,     ldu,    nu,
     $  yelbnd, uelbnd, nyel,  nuel,   yroot,  uroot, inyrhb, ncpb,
     $  nurpb,  rhntrp, yquad, uquad,  yset,   uset,  uintrp, rpar,
     $  ipar,   ymeas,  ldym,  umeas,  ldum,   nym,   num,    nac,
     $  c,      nstate, mode,  setpnt, optest
     $  )
*
```

```
      integer
     $
     $  nx,      ldy,  nde,  ldu, nu,  nyel, nuel,   ncpb, nurpb,
     $  ipar(*), ldym, ldum, nym, num, nac,  nstate, mode
*
      double precision
     $
     $  x(*),            y(ldy,*),  u(ldu,*),
     $  yelbnd(*),       uelbnd(*), yroot(*),
     $  uroot(*),        rhntrp(*), yquad(*),
     $  uquad(*),        yset(*),   uset(*),
     $  uintrp(nurpb,*), rpar(*),   ymeas(ldym,*),
     $  umeas(ldum,*),   c(*)
*
      logical  inyrhb, setpnt, optest
```

congrd. This routine may be used to supply gradient information for the additional constraints defined in `constr`. This routine is not required if finite differences are being used to compute the constraint gradients.

```
      subroutine congrd
     $  (
     $  x,      nx,     y,      ldy,    nde,    u,     ldu,
     $  nu,     yelbnd, uelbnd, nyel,   nuel,   yroot, uroot,
     $  inyrhb, ncpb,   nurpb,  rhntrp, yquad,  uquad, yset,
     $  uset,   uintrp, rpar,   ipar,   ymeas,  ldym,  umeas,
     $  ldum,   nym,    num,    nac,    pdx,    pdy,   pdu,
     $  ldpd,   nstate, mode,   setpnt, optest
     $  )
*
      integer
     $
     $  nx,      ldy,  nde,  ldu, nu,  nyel, nuel, ncpb,   nurpb,
     $  ipar(*), ldym, ldum, nym, num, nac,  ldpd, nstate, mode
*
      double precision
     $
     $  x(*),            y(ldy,*),    u(ldu,*),
     $  yelbnd(*),       uelbnd(*),   yroot(*),
     $  uroot(*),        rhntrp(*),   yquad(*),
     $  uquad(*),        yset(*),     uset(*),
     $  uintrp(nurpb,*), rpar(*),     ymeas(ldym,*),
     $  umeas(ldum,*),   pdx(ldpd,*), pdy(ldpd,*),
     $  pdu(ldpd,*)
*
      logical  inyrhb, setpnt, optest
```

optic. This routine is called once before the optimization starts and may be used to revise the initial condition for the differential-algebraic equations that is given in the file `.oic`. It also provides a convenient place for the user to read additional information or perform other initial computations before the optimization routine is called.

```
      subroutine optic (x, y, ldy, uu, ldu, nyel, nuel, ncpb, nurpb,
     $     nx, nde, nu, inesty, inestu, rpar, ipar, err)
*
      integer  ldy, ldu, nyel, nuel, ncpb, nurpb, nx, nde, nu, ipar(*)
      double precision  x(*), y(ldy,*), uu(ldu,*), rpar(*)
      logical  inesty, inestu, err
```

Subroutines required by `soplnt`. The following sections briefly describe the user-supplied subroutines required to simulate the response of the plant. The subroutines that compute gradient information are optional, provided that the appropriate options have been set in the input files.

dfcn. Given values of `t`, `u`, `y`, and `yprime`, this subroutine must provide the residual vector `delta (t, y, yprime)` as required by `DASSL`.

```
      subroutine dfcn (t, y, yprime, delta, ires, rpar, ipar, u, nu,
     $     x, nx, neq)
*
      integer           ires, ipar(*), nu, nx, neq
      double precision  t, y(*), yprime(*), delta(*), rpar(*), u(*),
     $     x(*)
```

dfcnj. This subroutine must provide the partial derivatives of the residual vector `pd (t, y, yprime)` as required by `DASSL`.

```
      subroutine dfcnj (t, y, yprime, pd, cj, rpar, ipar, u, nu, x, nx,
     $     neq)
*
      integer           ipar(*), nu, nx, neq
      double precision  t, y(*), yprime(*), pd(neq,*), cj, rpar(*),
     $     u(*), x(*)
```

plnic. This routine is called once before the plant simulation starts and may be used to revise the initial condition for the differential-algebraic equations that is given in the file `.pic`. It also provides a convenient place for the user to read additional information or perform other initial computations before the DAE solver is called.

```
      subroutine plnic (neq, t, y, yprime, uu, rpar, ipar, err)
*
      integer          neq, ipar(*)
      double precision  t, y(*), yprime(*), uu(*), rpar(*)
      logical          err
```

Input file formats. This section briefly describes the formats of the various input files used by the programs described above. Some of these files are also output files for some of the programs.

.dim. Dimension file, 3 possible sections separated by keywords.
Optimization section (keyword: optim)

`nipar` An integer specifying the number of integer-valued parameters to read for this problem.

`nrpar` An integer specifying the number of real-valued parameters to read for this problem.

`nde` An integer specifying the number of differential-algebraic equations.

`nu` An integer specifying the number of manupulated variables.

`nx` An integer specifying the number of additional decision variables.

`nac` An integer specifying the number of purely algebraic constraints (this is *not* the number of algebraic equations in the differential-algebraic equations).

`ncol` An integer specifying the number of interior collocation points in each element.

`inyrhb` A logical flag specifying whether or not the right hand boundary of each finite element should be included as a collocation point for the state variables.

`nuroot` An integer specifying the number of interior collocation points in each element.

`inulhb` A logical flag specifying whether or not the left hand boundary of each finite element should be included as an interpolation point for the manipulated variables.

inurhb A logical flag specifying whether or not the right hand boundary of each finite element should be included as an interpolation point for the manipulated variables.

nyel An integer specifying the number of finite elements to use for the state variable grid.

nuel An integer specifying the number of finite elements to use for the manipulated variable grid.

nym An integer specifying the number of state variable measurements are available in the `.ymeas` file.

num An integer specifying the number of manipulated variable measuerments are available in the `.umeas` file.

setpnt A logical flag specifying whether or not

mvmesh A logical flag specifying whether or not the mesh will be modified during the course of solving the optimization. This flag should be given a true value if, for example, a minimum time problem is being solved.

maxtry An integer specifying the maximum number of times to attempt to solve a particular optimization problem. This parameter is often useful because it frequently happens that restarting `NPSOL` will allow it to find an optimal solution after terminating prematurely.

npar An integer specifying the number of real-valued parameters to read from the `.par`. These values would normally be the parameters determined from a parameter estimation calculation (`soopt` writes the `.par` file when computing parameter estimates).

xlo A real-valued vector of length `nx` specifying the lower bounds for the additional decision variables, `x`. The `xlo` vector is only required if $\mathtt{nx} > 0$

xup A real-valued vector of length `nx` specifying the upper bounds for the additional decision variables, `x`. The `xup` vector is only required if $\mathtt{nx} > 0$

`daeflg` A vector of `nde` logical flags specifying whether the corresponding differential-algebraic equations are either differential equations (`daeflg` = true) or algebraic (`daeflg` = false). The `daeflg` vector is only required if `nde` > 0

`ylo` A real-valued vector of length `nde` specifying the lower bounds for the state variables, `y`. The `ylo` vector is only required if `nde` > 0

`yup` A real-valued vector of length `nde` specifying the upper bounds for the state variables, `y`. The `yup` vector is only required if `nde` > 0

`ulo` A real-valued vector of length `nu` specifying the lower bounds for the manipulated variables, `u`. The `ulo` vector is only required if `nu` > 0

`uup` A real-valued vector of length `nu` specifying the upper bounds for the manipulated variables, `u`. The `uup` vector is only required if `nu` > 0

`aclo` A real-valued vector of length `nac` specifying the lower bounds for the purely algebraic constraints. The `aclo` vector is only required if `nac` > 0

`acup` A real-valued vector of length `nac` specifying the upper bounds for the purely algebraic constraints. The `acup` vector is only required if `nac` > 0

`rpar` A real-valued vector of length `nrpar` $-$ `npar` specifying the values of the parameters not read from the `.par` file. The `rpar` vector is only required if `nrpar` $-$ `npar` > 0

`ipar` An integer-valued vector of length `nipar` specifying the values of the integer parameters for this problem. The `ipar` vector is only required if `nipar` > 0

`fdobjg` A logical flag specifying whether or not the gradient of the objective function should be obtained via finite difference approximation. If `fdobjg` is true, gradients are approximated and the user need not supply a function to compute them.

fdcong A logical flag specifying whether or not the gradient of the constraint functions should be obtained via finite difference approximation. If `fdobjg` is true, gradients are approximated and the user need not supply a function to compute them.

fdnpsol A logical flag specifying whether or not `npsol` or `soopt` should find the gradients of the objective and constraint functions via finite difference approximation. If either of the flags `fdobjg` or `fdcong` are false, this flag is ignored. If `fdnpsol`, `fdobjg`, and `fdcong` are all true, gradients are approximated and the user need not supply functions to compute them.

Estimation section (keyword: estim)

Identical format as for the optimization section but need not describe a problem of the same dimensions.

Plant section (keyword: plant)

nipar An integer specifying the number of integer-valued parameters to read for this problem.

nrpar An integer specifying the number of real-valued parameters to read for this problem.

neq An integer specifying the number of differential-algebraic equations.

nu An integer specifying the number of manupulated variables.

nx An integer specifying the number of additional decision variables.

nuroot An integer specifying the number of interior collocation points in each element.[1]

inulhb A logical flag specifying whether or not the left hand boundary of each finite element should be included as an interpolation point for the manipulated variables.

inurhb A logical flag specifying whether or not the right hand boundary of each finite element should be included as an interpolation point for the manipulated variables.

[1] It is is unfortunate that this variable (as well as some others) needs to be specified in two places, and that both values must agree. This problem should be fixed in a future release.

nuel An integer specifying the number of finite elements to use for the manipulated variable grid.

nsteps An integer specifying the number of uniformly spaced points at which to write the values of the states to the output file `.ypl`.

stmvec A logical flag specifying whether or not to read the set of output points from the `.stm` file. This parameter allows the user to specify a set of non-uniformly spaced output points.

info An integer vector of length 15 used to pass options to the differential-algebraic system solver `DASSL`. The documentation for `DASSL` contains a complete description of the meanings of these flags.

nyel An integer specifying the number of finite elements to use for the state variable grid.

ncol An integer specifying the number of interior collocation points in each element.

inyrhb A logical flag specifying whether or not the right hand boundary of each finite element should be included as a collocation point for the state variables.

rtol If `info`(2) = 1, `rtol` is a vector of length `neq`, and a scalar otherwise. The documentation for `DASSL` includes a complete description of the proper use of this variable.

atol If `info`(2) = 1, `atol` is a vector of length `neq`, and a scalar otherwise. The documentation for `DASSL` includes a complete description of the proper use of this variable.

rpar A real-valued vector of length `nrpar` specifying the values of the integer parameters for this problem. The `rpar` vector is only required if `nrpar` > 0

ipar An integer-valued vector of length `nipar` specifying the values of the integer parameters for this problem. The `ipar` vector is only required if `nipar` > 0

tstop A real-valued parameter specifying a point past which `DASSL` should not integrate. The variable `tstop` is only required if `info`(4) = 1 `hmax` A real-valued parameter specifying the maximum step size for `DASSL`. The variable `hmax` is only required if $\texttt{info}(7) = 1$

`hzero` A real-valued parameter specifying the initial step size for `DASSL`. The variable `hzero` is only required if $\texttt{info}(8) = 1$

Other files required for the optimal control or estimation calculation:

.oms. Mesh file, only required if $\texttt{nuel} > 0$ or $\texttt{nyel} > 0$.

`ymesh` A real-valued vector of `nyel+1` elements specifying the boundaries of the finite element grid for the state variables.

`umesh` A real-valued vector of `nuel+1` elements specifying the boundaries of the finite element grid for the manipulated variables.

Model parameter file, only required if $\texttt{npar} > 0$ (.par):

`par` A real-valued vector of length `npar` providing the values of the model parameters.

.ost. Setpoint file, only required if `setpnt` is true.

.oic. Initial condition file.

`yinitial` A logical flag specifying whether or not a complete initial profile is provided for the state variables. If this flag is false, only the initial values of the states should be provided; the initial condition over the prediction horizon will be assumed constant at these values.

`y` If `yinitial` is false, this is a real-valued vector of initial conditions for the state variables. If `yinitial` is false, this is a real-valued matrix of initial conditions with each column corresponding to one of the `nde` differential-algebraic equations. Each column should then contain N_y elements, where N_y is given by

$$N_y = \texttt{nyel} \times (\texttt{ncpb} - 1) + 1$$

where

$$\texttt{ncpb} = \begin{cases} \texttt{ncol} + 1 & \text{if } \texttt{inyrhb} \text{ is true} \\ \texttt{ncol} & \text{otherwise} \end{cases}$$

`uinitial` A logical flag specifying whether or not a complete initial profile is provided for the manipulated variables. If this flag is false, only the initial values of the manipulated variables should be provided; the initial condition over the control horizon will be assumed constant at these values.

`u` If `uinitial` is false, this is a real-valued vector of initial conditions for the manipulated variables. If `uinitial` is false, this is a real-valued matrix of initial conditions with each column corresponding to one of the `nu` manipulated variables. Each column should then contain N_u elements, where N_u is given by

$$N_u = \texttt{nuel} \times (\texttt{nurbp} - 1) + 1$$

where

$$\texttt{nurpb} = \begin{cases} \texttt{nuroot} + 1 & \text{if } \texttt{inurhb} \text{ is true} \\ \texttt{nuroot} & \text{otherwise} \end{cases}$$

.ypl. Output measurement, only required if $\texttt{nym} > 0$.

`ymeas(1:nde,1:nymeas)` dm

Input measurement, only required if $\texttt{num} > 0$ (.upl):

`umeas(1:nu,1:numeas)` dm

.pic. Initial condition file.

`y` A real-valued vector of length `neq` providing the initial values of the state variables for the differential-algebraic equation solver.

`dydt` A real-valued vector of length `neq` providing the initial values of the time derivatives of state variables for the differential-algebraic equation solver.

.upr. Manipulated variables.

`u` A real-valued matrix of `nu` columns and N_u rows, where the value of N_u has been defined above, providing the values of the manipulated variables over the time defined in the mesh file, `.oms`. This file is normally produced by the program `soopt` in solving an optimization, but may be created by hand in order to solve the plant equations for a given sequence of inputs.

.stm. Solution times for the plant simulation. This file allows the user to specify a set of unevenly spaced points at which to compute the solution of the plant equations.

`soltim` A real-valued vector of length `nsteps` providing the times for which a solution of the differential-algebraic equations should be computed.

A.1.2 An example problem

This section contains the complete Fortran source and associated data files required to solve the batch reactor problem described in Section 2.3.3, including the simulation of the plant, which was not shown in the previous discussion.

Since there is no need to specify additional algebraic constraints, and there is no need to compute a set of initial conditions different from what is read from the `.oic` file, the subroutines `constr`, `congrd`, and `optic` are not supplied.

User-supplied subroutines. The following subroutines define the process model and objective function used by the controller.

```
      subroutine fcn (x, nx, y, nde, u, nu, t, rpar, ipar, fvect,
     $     ymeas, umeas, nym, num, nstate, mode, optest)
*
      integer           nx, nde, nu, ipar(*), nym, num, nstate, mode
      logical           optest
      double precision  x(*), y(*), u(*), t, rpar(*), fvect(*),
     $                  ymeas(*), umeas(*)
*
*-----------------------------------------------------------------------
*
      double precision  alpha, p, uu
*
* -- Evaluate the right hand side functions that define the
* -- set of first order ordinary differential equations.
*
      alpha = rpar(1)
      p     = rpar(2)
      uu    = u(1)
*
      fvect (1) = -uu * (1.0 + alpha * (uu**(p - 1.0))) * y(1)
*
      fvect (2) =  uu * y(1)
*
      return
      end
      subroutine fcnj (x, nx, y, nde, u, nu, t, rpar, ipar, pdx,
```

```
     $     pdy, pdu, ldpd, ymeas, umeas, nym, num, nstate, mode,
     $     optest)
*
      integer           nx, nde, nu, ipar(*), ldpd, nym, num, nstate,
     $                  mode
      logical           optest
      double precision  x(*), y(*), u(*), t, rpar(*), pdx(ldpd,*),
     $                  pdy(ldpd,*), pdu(ldpd,*), ymeas(*), umeas(*)
*
*-----------------------------------------------------------------------
*
      double precision  alpha, p, uu
*
* -- Evaluate the matrix of the partial derivatives of the
* -- NDE differential equation functions with respect to the NX
* -- decision variables.
*
      alpha = rpar(1)
      p     = rpar(2)
      uu    = u(1)
*
      pdu(1,1) = -(1.0 + alpha*p*(uu**(p - 1.0))) * y(1)
*
      pdu(2,1) = y(1)
*
      pdy(1,1) = -uu * (1.0 + alpha*(uu**(p - 1.0)))
*
      pdy(2,1) =  uu
*
      return
      end
      subroutine obj
     $  (
     $  x,      nx,     y,      ldy,    nde,   u,      ldu,
     $  nu,     yelbnd, uelbnd, nyel,   nuel,  yroot,  uroot,
     $  inyrhb, inulhb, inurhb, ncpb,   nurpb, rhntrp, yquad,
     $  yset,   uset,   uquad,  uintrp, rpar,  ipar,   objf,
     $  ymeas,  ldym,   umeas,  ldum,   nym,   num,    nstate,
     $  mode,   setpnt, optest
     $  )
*
      integer
     $
     $  nx,      ldy,  nde,  ldu, nu,  nyel,   nuel, ncpb, nurpb,
     $  ipar(*), ldym, ldum, nym, num, nstate, mode
*
      double precision
     $
     $  x(*),           y(ldy,*), u(ldu,*), yelbnd(*),
     $  uelbnd(*),      yroot(*), uroot(*), rhntrp(*),
```

```
     $  yquad(*),        uquad(*), yset(*),  uset(*),
     $  uintrp(nurpb,*), rpar(*),  objf,     ymeas(ldym,*),
     $  umeas(ldum,*)
*
      logical  inyrhb, inulhb, inurhb, setpnt, optest
*
*-----------------------------------------------------------------------
*
      integer  ntemp, i, j
*
* -- Begin.
*
      if (inyrhb) then
*
        ntemp = nyel * (ncpb - 1) + 1
*
        objf = -y(2, ntemp)

      else
        write (*,*) 'This case is more complicated than it''s worth.'
        stop
      end if
*
      return
      end
      subroutine objgrd
     $  (
     $  x,      nx,     y,      ldy,    nde,   u,      ldu,
     $  nu,     yelbnd, uelbnd, nyel,   nuel,  yroot,  uroot,
     $  inyrhb, inulhb, inurhb, ncpb,   nurpb, rhntrp, yquad,
     $  uquad,  yset,   uset,   uintrp, rpar,  ipar,   objf,
     $  grad,   ymeas,  ldym,   umeas,  ldum,  nym,    num,
     $  nstate, mode,   setpnt, optest
     $  )
*
      integer
     $
     $  nx,      ldy,  nde,  ldu, nu,  nyel,   nuel, ncpb, nurpb,
     $  ipar(*), ldym, ldum, nym, num, nstate, mode
*
      double precision
     $
     $  x(*),            y(ldy,*),      u(ldu,*), yelbnd(*),
     $  uelbnd(*),       yroot(*),      uroot(*), rhntrp(*),
     $  yquad(*),        uquad(*),      yset(*),  uset(*),
     $  uintrp(nurpb,*), rpar(*),       objf,     grad(*),
     $  ymeas(ldy,*),    umeas(ldum,*)
*
      logical  inyrhb, inulhb, inurhb, setpnt, optest
*
```

```
*-----------------------------------------------------------------------
*
      integer    ldtemp,     nrtemp
      parameter (ldtemp = 2, nrtemp = 200)
*
      integer           i, j, n, ntemp
      double precision  tmpgrd(ldtemp,nrtemp)
      logical           load
*
* -- Begin.
*
      if (inyrhb) then
*
        n = nx + nyel * nde * (ncpb - 1) + nuel * nu * nurpb
*
        ntemp = nx + nyel * nde * (ncpb - 1)
*
        grad (ntemp) = -1.0
*
      else
        write (*,*) 'This case is more complicated than it''s worth.'
        stop
      end if
*
      return
      end
```

The following subroutines define the model for the plant. The parameters `alpha` and `p` used in these subroutines need not have the same values as were used above in the computation of the optimal temperature profile.

Since there is no need to compute a set of initial conditions different from what is read from the `.pic` file, the subroutine `optic` is not supplied.

```
      subroutine dfcn (t, y, yprime, delta, ires, rpar, ipar, u, nu,
     $     x, nx, neq)
*
      integer           ires, ipar(*), nu, nx, neq
      double precision  t, y(*), yprime(*), delta(*), rpar(*), u(*),
     $     x(*)
*
*-----------------------------------------------------------------------
*
      double precision  alpha, p
*
* -- Begin.
*
      alpha = rpar(1)
      p     = rpar(2)
*
      delta(1) = yprime(1) + u(1)*(1.0 + alpha*(u(1)**(p - 1.0)))*y(1)
```

```
*
      delta(2) = yprime(2) - u(1)*y(1)
*
      return
      end
      subroutine dfcnj (t, y, yprime, pd, cj, rpar, ipar, u, nu, x, nx,
     $     neq)
*
      integer           ipar(*), nu, nx, neq
      double precision  t, y(*), yprime(*), pd(neq,*), cj, rpar(*),
     $     u(*), x(*)
*
*---------------------------------------------------------------------
*
      double precision  alpha, p
*
* -- Begin.
*
      alpha = rpar(1)
      p     = rpar(2)
*
      pd(1,1) = u(1) * (1.0 + alpha*(u(1)**(p - 1.0))) + cj
*
      pd(2,1) = -u(1)
*
      pd(1,2) = 0.0
*
      pd(2,2) = cj
*
      return
      end
```

Input files for the optimal control calcluation. All of the following files (except `problem.oop`) must be provided.

The dimension file, `problem.dim`, specifies the problem size in terms of the number of differential-algebraic equations, inputs, finite elemente, collocation points, and so on.

```
optim -----------------------------------------------------------------
0                              nipar
3                              nrpar
2                              nde
1                              nu
0                              nx
0                              nac
0                              ncol
t                              inyrhb
0                              nuroot
```

```
f                                inulhb
t                                inurhb
20                               nyel
20                               nuel
0                                nym
0                                num
f                                setpnt
f                                mvmesh
5                                maxtry
2                                npar
t t                              daeflg(1:nde)
0.0 0.0                          ylo(1:nde)
1.0 1.0                          yup(1:nde)
0.0                              ulo(1:nu)
5.0                              uup(2:nu)
0.0                              rpar(npar+1:nrpar)
f                                fdobjg
f                                fdcong
f                                fdnpsol

plant -----------------------------------------------------------------
0                                nipar
2                                nrpar
2                                neq
1                                nu
0                                nx
0                                nuroot
f                                inulhb
t                                inurhb
20                               nuel
200                              nsteps
f                                stmvec
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0    info
20                               nyel
0                                ncol
t                                inyrhb
1.0d-12                          rtol
1.0d-12                          atol
0.5 2.0                          rpar(1:nrpar)
```

Note that `rpar(1)` to `rpar(npar)` are read from the file `<name>.par` (these are the values that will be updated when when soopt is used for parameter estimation)

The initial condition file, `problem.oic`, specifies the initial estimate of the solution to the optimization problem. The logical flags control whether the solution is given over the entire horizon. If not, the initial profiles are assumed to be constant at the initial values.

```
f                  inesty
```

```
1.0  0.0          y(1:nde,1)
t                 inestu
5.0               u(1:nu,1)
```

The mesh file, `problem.oms`, specifies the finite element grid for the prediction horizon.

```
0.00  0.05  0.10  0.15  0.20  0.25  0.30
0.35  0.40  0.45  0.50  0.55  0.60  0.65
0.70  0.75  0.80  0.85  0.90  0.95  1.00        yelbnd(1:nyel+1)
0.00  0.05  0.10  0.15  0.20  0.25  0.30
0.35  0.40  0.45  0.50  0.55  0.60  0.65
0.70  0.75  0.80  0.85  0.90  0.95  1.00        uelbnd(1:nuel+1)
```

The parameter file `problem.par`, specifies the modelling parameters.

```
0.5  2.0      rpar(1:npar)
```

Finally, the following file, `problem.oop`, provides optional inputs to `NPSOL`. The complete syntax and list of available options is described in the `NPSOL` Users' Guide [28]

```
Begin
  Nolist
  Feasibility Tolerance = 1.0e-12
  Optimality Tolerance = 1.0e-12
End
```

A.1.3 Installation

The installation procedure for this software is outlined here to help the reader to determine whether it would be useful for him to obtain a copy of the software and try to install it. Complete installation details are included in the distribution.

It should be very easy to install this software if you are using a machine running some variant of the Unix operating system. If not, you should still be able to get this set of programs to run but you will need to do a bit of extra work.

You will need to have `NPSOL`, `LINPACK`, `DASSL`, and the `BLAS` installed somewhere on your system. `LINPACK`, `DASSL`, and the `BLAS` are available from the netlib mail server and via anonymous ftp from `ftp.che.utexas.edu` and `research.att.com` (for the latter, login as *netlib* instead of anonymous). `NPSOL` is available from

Software Distribution Center
350 Cambridge Avenue, Suite 250

Stanford University
Palo Alto, CA 94306
(415) 723-0651

In 1989, the NPSOL license fee was $300 for academic sites and $3600 for commercial sites.

These programs were all developed on machines running Unix-like operating systems, and have been installed and used without modification on a variety of systems, including a Cray Y-MP8/864, an IBM RS/6000, a Convex C-220, a DECsystem 5810, and a VAXstation 3200. It should be very easy to install them on other machines running a Unix-based operating system. Some work will surely be required if these programs are to be installed on a system that is much different than Unix, such as DEC's VMS or IBM's VM/MVS, though the necessary changes should be straightforward.

The following step should allow one to complete the installation on a system running some version of the Unix operating system.

- Some of the files in the `sysdep` directory may need to be modified. Of particular interest:
  - The file `stdio.h` defines the Fortran logical unit numbers for the standard output, standard input, and standard error streams. If these are modified, some of the other include files in the sysdep directory that define unit numbers may have to change as well.
  - The file `chngcs.f` depends on the character set being ASCII. If your system does not use the ASCII collating sequence, you can use the standard-conforming subroutine provided in the file `chngcs-std.f`. This portable version is not very efficient, and even though it is not called very often, you may wish to provide a more efficient version anyway, just because.

    If your system is monocase, unplug it and go shopping for some newer hardware.
  - The file `getarg-std.f` contains portable definitions for the Unix f77 functions `getarg()` and `iargc()` that work by prompting the user for the command line arguments rather than getting them directly from the environment (there is no portable way to do this—the prompting versions provided are about as close as one can get). It may be useful to provide implementations of these functions if your system does not have them but does have a way to get command line arguments from the environment.

Thanks to Michael Lemke of the Department of Astronomy at The University of Texas at Austin, there are implementations of these functions for the VMS operating system in the file `getarg-vms.for`.

- These programs depend heavily on the nonstandard 'include' statement and usually attempt to include files from other directories using statements like

```
include '../include/file.h'
```

  If your system has a different way of specifying directory names, you may need to change this. For example, under VMS, the above line becomes something like

```
include '[-.include]file.h'
```

  If your compiler does not handle the include statement, you can always simply replace each instance of an include statement with the contents of the file it references.

A.2 Linear MPC Software

The software for solving linear MPC problems implements the QDMC algorithm described in Section 3.1.1. A dynamic simulation program, `sim`, may be used to generate step response data for a nonlinear system for use with the QDMC controller. The program `dmc` implements the contoller and allows one to simulate the performance of the linear controller for linear or nonlinear process models.

Both programs are written in C and rely on Fortran libraries for the numerical compuations. This mixture of source languages does not present a major difficulty so long as it is possible to use the program `f2c` [20] to convert the Fortran source to C on the target system.

A.2.1 The programs and their data files

dmc. This program is an imlementation of the QDMC algorithm discussed in Section 3.1.1.
usage: `dmc [-cghklrv] [-b file] [-i file] [-m method] [-n steps] [-o file] [-s start] infile outfile`

-c Compute additive output disturbance term.

-h Print this help message and exit.

-l Use linear extrapolation.

-v Verbose mode.

-b file
Give name of bounds file (dmc.bnd).

-i file
Give name of intermediate output file.

-m method
Select method to solve QP (lssol).

-n steps
Specify number of steps to simulate (20).

-o file
Give name of option file (dmc.opt).

-s tbegin
Specify starting time for simulation (0.0).

sim. This program allows the user to compute the solution to a set of differential-algebraic equations given a piecewise constant input profile and is convenient for generating the step response coefficients used by the DMC simulator.
usage: `sim [-hv] [-n steps] [-b begin] [infile]`

-h Print this help message and exit.

-v Verbose Mode.

-n steps
Specify the number of steps to simulate.

-b begin
Specify the beginning time for the simulation.

infile
Name of file to read for simulation parameters. The standard input is read if **infile** is not specified.

User-supplied subroutines.

```
/*
 * plant.c
 */
#include <d-f-alloc.h>

extern int getu ();

/*
 * Solve the 'real' system of equations.
 */
int
dmc_plant (first, t, tout, y)
     int first;
     double t, tout;
     double_f_vector *y;
{
  /*
   * User code begins here.  Given the initial time t, and the output
   * time tout, solve for the output y given the input u.  The value
   * of u at any time between t and tout is available via the function
   *
   *   int getu (double t, double_f_vector *u).
   */

  return 0;
}
```

dmc_plant.

Input file formats. Input file for `dmc`:

- `nu` Number of inputs.
- `ny` Number of outputs.
- `n` Control horizon.
- `p` Prediction horizon.
- `tsamp` Sampling time.
- `lambda` Vector of length `ny`
- `gamma` Vector of length `nu`.
- `step_resp` Columns of the dynamic matrix. Each column of data provided here corresponds to one of the process inputs. Within each column, the first `p` elements correspond to the first process output, the second `p` elements correspond to the second process output, and so on.

Input file for `sim`:

- `nu` Number of inputs.
- `ny` Number of outputs.
- `tsamp` Sampling time.
- `y` Initial values of the states or outputs in the process model.
- `t, u(t)` Data sets representing time and manipulated variable values at the given time that specify a sequence of piecewise constant inputs. As a minimum, two sets of data must be provided.

A.2.2 Example problem

This section contains the copmlete C source and associated data file required to solve the example problem described in section 3.3.2.

User-supplied subroutines. The following code is called by `sim` to simulate the process. This version uses the DAE solver `DASSL`.

```
#include <d-f-alloc.h>
#include "gripes.h"

extern int ddassl_ (), res (), jac (), getu ();

/*
 * Solve the 'real' system of equations.  This version uses Petzold's
 * differential-algebraic system solver dassl.
 */
int
dmc_plant (first, t, tout, y)
     int first;
     double t, tout;
     double_f_vector *y;
{
  double u[1], rpar, rtol = 1.0e-6, atol = 1.0e-6;
  int i, idid, ipar, neq = 1, lrw = 100, liw = 100;
  static int info[15], iwork[100];
  static double x[1], xprime[1], rwork[100];

  if (first)
    {
      x[0] = 0.0;

      for (i = 0; i < 15; i++)
```

```
        info[i] = 0;

      info[3] = 1;  /* Don't integrate past rwork[0] = tout. */
    }

  rwork[0] = tout;

  ddassl_ (res, &neq, &t, x, xprime, &tout, info, &rtol, &atol,
           &idid, rwork, &lrw, iwork, &liw, &rpar, &ipar, jac);

  if (getu (tout, u))
    GripeGettingU (t);

  y->v[0] = x[0] - u[0];

  return 0;
}

/*
 * Form residual.  r = f(xdot, x, p).
 */
int
res (t, x, xprime, delta, ires, rpar, ipar)
     int *ires, *ipar;
     double *t, *x, *xprime, *delta, *rpar;
{
  double u[1];

  if (getu (*t, u))
    GripeGettingU (t);

  delta[0] = xprime[0] + x[0] - 2.0 * u[0];

  return 0;
}

/*
 * Form modified Jacobian of the residual.  J' = J - I cj. It is
 * just a dummy routine because we're letting dassl use a finite
 * difference approximation for J'.
 */
int
jac (t, x, xprime, pd, cj, rpar, ipar)
     int *ipar;
     double *t, *x, *xprime, *pd, *cj, *rpar;
{
  return 0;
}
```

Input files. The code given above, together with the rest of the `dmc` program, and the input files show below were used to create the plot shown in Figure 3.8.

The following input file is used by `sim` and specifies the number of inputs, outputs, sampling time, initial values for the output, and a set of manipulated variable values at specified times.

```
1 1
0.1
0.0
0.0 1.0
200.0 1.0
```

The following intput file is used by `dmc` and specifies the number of inputs and outputs, steps in the control and prediction horizons, sampling time, output and input penalties, and step response coefficients.

```
 1  1  5  25
 0.1
 1.0
 0.0
-0.809674  -0.637461  -0.481637  -0.340641  -0.213062
-0.097624   0.006829   0.101341   0.186860   0.264241
 0.334257   0.397611   0.454936   0.506806   0.553740
 0.596207   0.634633   0.669403   0.700863   0.729330
 0.755088   0.778394   0.799483   0.818564   0.835830
```

A.2.3 Installation

These programs should be easy to install on any Unix-based system and it should be possible to install them on other systems, though doing so may require a bit more work. They depend on being able to call Fortran from C, and they assume that the Fortran routines have been translated with the Fortran to C translator `f2c`. For systems whose Fortran compilers create object files that are the same as those created by `f2c` and a C compiler, it should be possible to link directly to the Fortran-compiled objects. Other systems will probably require direct use of `f2c`.

A.3 A Note About Languages

The development of the software required to solve the control problems of interest in this dissertation began at a time when it was generally assumed that engineers wrote programs in Fortran and that was that.[2] The programs described here have gone through several major revisions but will probably never be as clean or as simple to use as the author would like. There are several reasons for this that are worth noting.

The primary problem is that there is a significant amount of data that these programs must deal with and there is no convenient way to handle anything but the simplest of data structures in Fortran. This inability to hide data forces too much detail to become visible at all levels of the program, making them difficult to write and maintain. This difficulty translates directly into a larger portion of time spent trying to find programming errors rather than using the programs to discover new properties of the algorithms, and it makes it very difficult to modify them to solve new classes of problems.

For example, the data required to perform a complete simulation includes the past inputs (with and without noise added), outputs with the corresponding time horizon, predicted inputs and outputs with the corresponding time horizon, and model parameter estimates. This information just represents the *visible* data that a reasonable simulator should be able to conveniently present to the user at any given sample time. Along with this information, one must keep track of the dimensions of all the various arrays, and the flags to indicate what operations to perform. In Fortran, this means that a much larger number of variables will be visible, making the programs harder to understand and increasing the possiblity for error.

Using languages such as C, C++, or Pascal that support higher levels of abstraction through their ability to define more complicated data structures would allow unimportant details to be hidden and would reduce the large number of parameters in the argument lists of the user-supplied subroutines.[3] Unfortunately, none of these languages are normally used for solving numerical problems and there are not as many numerical libraries written in these languages as there are written in Fortran. However, with a Fortran to C translator (several exist and one is freely available) it is possible to make use of the large number of reasonably good numerical libraries written in Fortran from C or C++ programs without greatly sacrificing execution speed or portability.

Writing the QDMC simulation code in C was an experiment in doing this, though the author believes that a much cleaner solution would be possible with a language like C++, because the C++ language tends to encourage working at

[2] Perhaps this is still true, but things are beginning to change, for the better, one would hope.

[3] Fortran does have the `COMMON` block, but all the names remain visible, so the problem is not really solved, and passing arrays in `COMMON` requires one to specify an upper bound on the dimension of the array.

a much higher level of abstraction than C. Starting a project of this size in Fortran would definitely *not* be recommended.

The program `f2c` *is a* horror, *based on ancient code and hacked unmercifully. Users are only supposed to look at its C output, not at its appalling inner workings.*

— S. I. Feldman, David M. Gay, Mark W. Maimone, and N. L. Schryer, *A Fortran-to-C Converter* (1990)

Bibliography

[1] Andrews, J. F. A mathematical model for the continuous culture of microorganisms utilizing inhibitory substrates. *Biotech. Bioeng.*, 10(707–723), 1968.

[2] Biegler, L. T. Solution of dynamic optimization problems by successive quadratic programming and orthogonal collocation. *Comput. Chem. Eng.*, 8(3/4):243–248, 1984.

[3] Biegler, L. T. Improved infeasible path optimization for sequential modular simulators—I: The interface. *Comput. Chem. Eng.*, 9(3):245–256, 1985.

[4] Bless, R. R. and D. H. Hodges. Finite element solution of optimal control problems with inequality constraints. In *Proceedings of the 1990 American Control Conference*, pages 242–247, May 1990.

[5] Brengel, D. D. and W. D. Seider. Multistep nonlinear predictive controllers. In *Proceedings of the 1989 American Control Conference*, pages 1100–1101, June 1989.

[6] Brosilow, C. and C. M. Cheng. Model predictive control of unstable systems. AIChE National Meeting, New York, NY, November 1987.

[7] Brusch, R. G. A nonlinear programming approach to space shuttle trajectory optimization. *J. Optim. Theory Appl.*, 13(1):94–118, 1974.

[8] Caracotsios, M. and W. E. Stewart. Sensitivity analysis of initial value problems with mixed ODEs and algebraic equations. *Comput. Chem. Eng.*, 9(4):359–365, 1985.

[9] Clarke, D. W., C. Mohtadi, and P. S. Tuffs. Generalized predictive control—Part I. The basic algorithm. *Automatica*, 23(2):137–148, 1987.

[10] Cuthrell, J. E. *On the Optimization of Differential-Algebraic Systems of Equations in Chemical Engineering.* PhD thesis, Carnegie-Mellon University, Pittsburgh, 1986.

[11] Cuthrell, J. E. and L. T. Biegler. Simultaneous solution and optimization of process flowsheets with differential equation models. In *Process*

Systems Engineering PSE '85: The Use of Computers in Chemical Engineering, pages 13–23. The Institution of Chemical Engineers Symposium Series No 92, 1985.

[12] Cuthrell, J. E. and L. T. Biegler. On the optimization of differential-algebraic systems. *AIChE J.*, 33(8):1257–1270, 1987.

[13] Cuthrell, J. E. and L. T. Biegler. Simultaneous optimization and solution methods for batch reactor control profiles. *Comput. Chem. Eng.*, 13(1/2):49–62, 1989.

[14] Cutler, C. R. and B. L. Ramaker. Dynamic matrix control—a computer control algorithm. In *Proceedings of the Joint Automatic Control Conference*, 1980.

[15] Davis, P. J. and P. Rabinowitz. *Methods of Numerical Integration*. Academic Press, Orlando, Florida, 2nd edition, 1984.

[16] Eaton, J. W. and J. B. Rawlings. Feedback control of chemical processes using on-line optimization techniques. *Comput. Chem. Eng.*, 14(4/5):469–479, 1990.

[17] Eaton, J. W., J. B. Rawlings, and T. F. Edgar. Model-predictive control and sensitivity analysis for constrained nonlinear processes. In McAvoy, T. J., Y. Arkun, and E. Zafiriou, editors, *Proceedings of the 1988 IFAC Workshop on Model Based Process Control*, pages 129–135, Oxford, 1989. Pergamon Press.

[18] Economou, C. G., M. Morari, and B. Palsson. Internal model control. 5. extension to nonlinear systems. *Ind. Eng. Chem. Proc. Des. Dev.*, 25:403–411, 1986.

[19] Economou, C. G. *An Operator Theory Approach to Nonlinear Controller Design*. PhD thesis, California Institute of Technology, 1986.

[20] Feldman, S. I., D. M. Gay, M. W. Maimone, and N. L. Schryer. A Fortran-to-C converter. Computing Science Technical Report 149, AT&T Bell Laboratories, Murray Hill, NJ, May 1990.

[21] Fiacco, A. V. *Introduction to Sensitivity Analysis in Nonlinear Programming*. Academic Press, New York, 1983.

[22] Fletcher, R. *Practical Methods of Optimization*. John Wiley & Sons, New York, 1987.

[23] García, C. E. and M. Morari. Internal model control. 1. A unifying review and some new results. *Ind. Eng. Chem. Proc. Des. Dev.*, 21:308–323, 1982.

[24] García, C. E. and A. M. Morshedi. Quadratic programming solution of dynamic matrix control (QDMC). *Chem. Eng. Commun.*, 46:73–87, 1986.

[25] García, C. E., D. M. Prett, and M. Morari. Model predictive control: Theory and practice—a survey. *Automatica*, 25(3):335–348, 1989.

[26] Gattu, G. and E. Zafiriou. Nonlinear quadratic dynamic matrix control with state estimation. Submitted to *Ind. and Eng. Chem. Res.*, June 1991.

[27] Gill, P. E. and W. Murray. The numerical solution of a problem in the calculus of variations. In Bell, D. J., editor, *Recent Mathematical Developments in Control*, pages 97–122, New York, 1973. Academic Press.

[28] Gill, P. E., W. Murray, M. A. Saunders, and M. H. Wright. User's guide for SOL/NPSOL (Version 4.0): A Fortran package for nonlinear programming, technical report SOL 86-2. Technical report, Systems Optimization Laboratory, Department of Operations Research, Stanford University, 1986.

[29] Hidalgo, P. M. and C. B. Brosilow. Nonlinear model predictive control of styrene polymerization at unstable operating points. *Comput. Chem. Eng.*, 14(4/5):481–494, 1990.

[30] Isidori, A. *Nonlinear Control Systems*. Springer-Verlag, Berlin, 2nd edition, 1989.

[31] Jang, S.-S., B. Joseph, and H. Mukai. On-line optimization of constrained multivariable chemical processes. *AIChE J.*, 33(1):26–35, January 1987.

[32] Jänsh, C. and M. Paus. Aircraft trajectory optimization with direct collocation using movable gridpoints. In *Proceedings of the 1990 American Control Conference*, pages 262–267, May 1990.

[33] Johnson, I. L. Optimization of the solid-rocket assisted space shuttle ascent trajectory. *J. Spacecraft*, 12(12):765–769, December 1975.

[34] Jones, A. G. Optimal operation of a batch cooling crystallizer. *Chem. Eng. Sci.*, 29:1075–1087, 1974.

[35] Kleinman, D. L. An easy way to stabilize a linear constant system. *IEEE Trans. Auto. Cont.*, 15(12):692, December 1970.

[36] Kwon, W. H., A. M. Bruckstein, and T. Kailath. Stabilizing state-feedback design via the moving horizon method. *Int. J. Control*, 37(3):631–643, 1983.

[37] Kwon, W. H. and A. E. Pearson. A modified quadratic cost problem and feedback stabilization of a linear system. *IEEE Trans. Auto. Cont.*, 22(5):838–842, October 1977.

[38] Lee, E. B. and L. Markus. *Foundations of Optimal Control Theory.* John Wiley and Sons, New York, 1967.

[39] Lee, J. H., M. Morari, and C. E. García. State-space interpretation of model predictive control. *Automatica*, 30(4):707–717, 1994.

[40] Li, S., K. Y. Lim, and D. G. Fisher. A state space formulation of model predictive control. *AIChE J.*, 35(2):241–249, February 1989.

[41] Li, W. C. and L. T. Biegler. Process control strategies for constrained nonlinear systems. *Ind. Eng. Chem. Res.*, 27:1421–1433, 1988.

[42] Mayne, D. Q. and H. Michalska. Receding horizon control of nonlinear systems. *IEEE Trans. Auto. Cont.*, 35(7):814–824, July 1990.

[43] Mehra, R. K., R. Rouhani, J. Eterno, J. Richalet, and A. Rault. Model algorithmic control: Review and recent development. In *Engineering Foundation Conference on Chemical Process Control II*, pages 287–310, 1982.

[44] Morshedi, A. M. Universal dynamic matrix control. In Morari, M. and T. J. McAvoy, editors, *Chemical Process Control—CPC III*, pages 547–577, Amsterdam, 1986. Third International Conference on Chemical Process Control, CACHE/Elsevier.

[45] Neuman, C. P. and A. Sen. A suboptimal control algorithm for constrained problems using cubic splines. *Automatica*, 9:601–613, 1973.

[46] Neuman, C. P. and A. Sen. Weighted residual methods and the suboptimal control of distributed parameter systems. *Int. J. Control*, 18(6):1291–1301, 1973.

[47] Nueman, C. P. and A. Sen. Weighted residual methods in optimal control. *IEEE Trans. Auto. Cont.*, 19(2):67–69, 1974.

[48] Patwardhan, A. A., J. B. Rawlings, and T. F. Edgar. Model predictive control of nonlinear processes in the presence of constraints. In *IFAC/IEEE Symposium on Nonlinear Control Systems Design*, pages 456–460, 1989.

[49] Patwardhan, A. A., J. B. Rawlings, and T. F. Edgar. Nonlinear model predictive control. *Chem. Eng. Commun.*, 87:123–141, 1990.

[50] Peterson, T., E. Hernandez, Y. Arkun, and F. J. Schork. A nonlinear DMC algorithm and its application to a semi-batch polymerization reactor. Accepted for publication in *Chem. Eng. Sci.*, 1991.

[51] Petzold, L. R. A description of DASSL: A differential-algebraic system solver. In Stepleman, R. S., editor, *Scientific Computing*, pages 65–68, Amsterdam, 1983. North-Holland.

[52] Pollard, G. P. and R. W. H. Sargent. Off line computation of optimum controls for a plate distillation column. *Automatica*, 6:59–76, 1970.

[53] Prett, D. M. and R. D. Gillette. Optimization and constrained multivariable control of a catalytic cracking unit. In *Proceedings of the Joint Automatic Control Conference*, pages WP5–C, San Francisco, CA, 1980.

[54] Prett, D. M., C. E. García, and B. L. Ramaker, editors. *The Second Shell Process Control Workshop: Solutions to the Shell Standard Control Problem*. Butterworths, Boston, 1989.

[55] Propoi, A. I. Use of linear programming methods for synthesizing sampled-data automatic systems. *Automn. Remote Control*, 24(7):837–844, July 1963.

[56] Psiaki, M. L. and K. Park. Trajectory optimization for real-time guidance: Part 1, time-varying LQR on a parallel processor. In *Proceedings of the 1990 American Control Conference*, pages 248–253, May 1990.

[57] Rawlings, J. B. and J. W. Eaton. Optimal control and model identification applied to the shell standard control problem. In *Shell Process Control Workshop II Proceedings*, December 12–16 1988.

[58] Rawlings, J. B. and K. R. Muske. Stability of constrained receding horizon control. *IEEE Trans. Auto. Cont.*, 38(10):1512–1516, October 1993.

[59] Ray, W. H. *Advanced Process Control*. McGraw-Hill, New York, 1981.

[60] Renfro, J. G. *Computational Studies in the Optimization of Systems Described by Differential/Algebraic Equations.* PhD thesis, University of Houston, Houston, Texas, 1986.

[61] Renfro, J. G., A. M. Morshedi, and O. A. Asbjørnsen. Simultaneous optimization and solution of systems described by differential/algebraic equations. *Comput. Chem. Eng.*, 11(5):503–517, 1987.

[62] Richalet, J., A. Rault, J. L. Testud, and J. Papon. Model predictive heuristic control: Applications to industrial processes. *Automatica*, 14:413–428, 1978.

[63] Ricker, N. L. Use of quadratic programming for constrained internal model control. *Ind. Eng. Chem. Proc. Des. Dev.*, 24:925–936, 1985.

[64] Sargent, R. W. H. and G. R. Sullivan. The development of an efficient optimal control package. In Stoer, J., editor, *Lecture Notes in Control and Information Sciences 7*, pages 158–168, Berlin, 1978. 8th IFIP Conference on Optimization Techniques, Springer-Verlag.

[65] Schmid, C. and L. T. Biegler. Application of multistep newton-type controllers to fluid catalytic cracking. In *Proceedings of the 1990 American Control Conference*, pages 581–586, May 1990.

[66] Tabak, D. and B. C. Kuo. Application of mathematical programming in the design of optimal control systems. *Int. J. Control*, 10(5):545–552, 1969.

[67] Thomas, Y. A. Linear quadratic optimal estimation and control with receding horizon. *Electron. Lett.*, 11:19–21, January 1975.

[68] Tsang, T. H., D. M. Himmelblau, and T. F. Edgar. Optimal control via collocation and non-linear programming. *Int. J. Control*, 21(5):763–768, 1975.

[69] Uppal, A., W. H. Ray, and A. B. Poore. On the dynamic behavior of continuous stirred tank reactors. *Chem. Eng. Sci.*, 29:967–985, 1974.

[70] Villadsen, J. and M. L. Michelsen. *Solution of Differential Equation Models by Polynomial Approximation.* Prentice-Hall, Englewood Cliffs New Jersey, 1978.

[71] Villadsen, J. V. and W. E. Stewart. Solution of boundary-value problems by orthogonal collocation. *Chem. Eng. Sci.*, 22:1483–1501, 1967.

[72] Witkowski, W. R. and J. B. Rawlings. Modelling and control of crystallizers. In *Proceedings of the 1987 American Control Conference*, pages 1400–1405, Minneapolis, Minnesota, June 1987. American Control Conference.

[73] Zafiriou, E. Robust model predictive control of processes with hard constraints. *Comput. Chem. Eng.*, 14(4/5):359–371, 1990.

Scholarly writing is distinguished from all other kinds by its punctilious acknowledgement of sources. This acknowledgement is not just an empty form.

— Mary-Claire van Leunen, *A Handbook for Scholars* (1978)

Vita

John Wesley Eaton, the son of John Roberson Eaton and Barbara Fessler Eaton, was born in Fort Worth, Texas, on March 18, 1963. He graduated from Robert Service High School, Anchorage, Alaska in June 1981, and worked as a photographer for the Anchorage Times during the following summer. In September 1981 he entered the University of Missouri at Columbia, claiming journalism as his major field of study. He transferred to Oregon State University the following year and received the degree of Bachelor of Science in Chemical Engineering in June 1985. During the summers of 1982, 1983, and 1984 he worked for oilfield service companies based in scenic Kenai, Alaska. In September 1986 he entered the Graduate School of The University of Texas to continue his study of chemical engineering. Rest assured that time otherwise unaccounted for was well spent.

Permanent address: 3000 Guadalupe # 301
Austin, Texas 78705

This dissertation was typeset by the author using the LaTeX document preparation system.