Dynamic Optimization with Path Constraints

by

William Francis Feehery

Submitted to the Department of Chemical Engineering
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy in Chemical Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 1998

Author .....
Department of Chemical Engineering
March 5, 1998

Certified by .....
Paul I. Barton
Assistant Professor of Chemical Engineering
Thesis Supervisor

Accepted by .....
Robert Cohen
St. Laurent Professor of Chemical Engineering
Chairman, Committee on Graduate Students

Dynamic Optimization with Path Constraints

by

William Francis Feehery

Submitted to the Department of Chemical Engineering
on March 5, 1998, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy in Chemical Engineering

Abstract

Dynamic optimization problems, also called constrained optimal control problems, are of interest in many areas of engineering. However, numerical solution of such problems is difficult and thus the application of dynamic optimization in process engineering has been limited. The dynamic optimization problems of interest in process engineering typically consist of large systems of differential and algebraic equations (DAEs), and often contain path equality or inequality constraints on the state variables. The objective of this thesis was to improve the efficiency with which large-scale dynamic optimization problems may be solved and to develop improved methods for including path constraints.

The most efficient method for numerical solution of large dynamic optimization problems is the control parameterization method. The cost of solving the dynamic optimization problem is typically dominated by the cost of solving the sensitivity system. The efficiency with which the sensitivity system can be solved is significantly improved with the staggered corrector sensitivity algorithm which was developed and implemented during the course of this thesis.

State variable path constraints are difficult to handle because they can cause the initial value problem (IVP) to be a high-index DAE. An efficient method for numerical solution of a broad class of high-index DAEs, the dummy derivative method, is described and demonstrated. Also described is a method for transforming an equality path-constrained dynamic optimization problem into a dynamic optimization problem with fewer degrees of freedom that contains a high-index DAE, which may be solved using the dummy derivative method.

Inequality path-constrained dynamic optimization problems present special challenges because they contain the additional decisions concerning the order and number of inequality activations and deactivations along the solution trajectory. Such problems are shown to be equivalent to a class of hybrid discrete/continuous dynamic optimization problems. Existence and uniqueness theorems of the sensitivities for hybrid systems are derived. Based on these results, several algorithms are presented for the solution of inequality path-constrained dynamic optimization problems. One of them, the fluctuating index infeasible path algorithm, works particularly well, and its use is demonstrated on several examples.

Thesis Supervisor: Paul I. Barton
Title: Assistant Professor of Chemical Engineering

To Lisa

Acknowledgments

Although this thesis may seem like an individual effort, it reflects an extensive network of support, advice, and friendship given to me by many people.

I would like to thank Professor Paul Barton for his intellectual guidance and friendship. As I look back over the last few years, the chance to work with Paul has been the biggest advantage of my choice to come to MIT. His encouragement, advice, and (sometimes) criticism have kept my research focused and interesting.

It has been fun to work with the other students in the Barton group. In particular, I would like to mention Berit Ahmad, Russell Allgor, Wade Martinson, Taeshin Park, and John Tolsma. In addition, the presence of many visitors to the group during my time here have made it a stimulating place, especially Julio Banga, Christophe Bruneton, Santos Galán, Lars Kreul, and Denis Sédès.

My family has been a great source of love, encouragement, and support, probably even more than they know. My parents have always been there when I needed them, and my having gotten this far on the educational ladder is an excellent reflection on the values that they taught me.

Most of all, thanks to my wife Lisa. I will always think of the period I spent at MIT as the time where I met the most amazing person and we fell in love. Lisa's support and understanding during this project have kept me going– especially at the end as I was writing this thesis.

Finally, I would like to acknowledge financial support in the form of a fellowship from the National Science Foundation, and additional support from the United States Department of Energy.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Dynamic optimization (also called *optimal control*) problems require determining a set of control variable time profiles for a dynamic system that optimizes a given performance measure. Some examples of dynamic optimization problems are determination of optimal operating policies for chemical plants subject to safety, operational, and environmental constraints, determination of the control policy for an industrial mechanical robot, and solution of the minimum time-to-climb problem for an aircraft that is required to stay within a specified flight envelope.

Solution of dynamic optimization problems has been a subject of research for hundreds of years. Bernoulli posed the first dynamic optimization problem, the brachistochrone problem, in 1696. The development of the calculus of variations allowed the derivation of necessary and sufficient conditions for a solution of a dynamic optimization problem. However, these conditions are useful for finding an analytic solution only in restricted cases, and numerical solutions were not attempted until the advent of modern computers.

Even with modern computers, numerical solution of dynamic optimization problems is not an easy task. The necessary and sufficient conditions that can be derived using the calculus of variations indicate if a solution is optimal, but not necessarily how to find an optimal (or even an improved) solution. However, many numerical techniques have been developed which utilize information from the necessary optimality conditions, which have in common that they are computationally costly for all but

the most simple problems. Furthermore, these techniques often require a significant investment of time and effort to set the problem up to be solved.

The focus of much recent process engineering research has been development of techniques for developing and solving numerically large simulations of dynamic systems. A typical oil refinery can be modeled with over 100,000 equations, and even models of complex unit operations can require tens of thousands of equations. Engineers often use these simulations to find or improve feasible operating policies for a given system design. This process is often performed by enumerating various operating policies and comparing them against each other. Such a procedure is time consuming and provides no guarantee of finding the 'best' answer. Finding and improving feasible operating policies can also be done by solving a dynamic optimization problem. Dynamic optimization problems have the advantage that the solution techniques can often provide an optimal answer, at least locally, but the problems are at present difficult to set up and solve.

A simple example of a dynamic optimization problem is the brachistochrone problem that was proposed in 1696 by Johann Bernoulli to challenge the mathematicians of Europe. The story goes that Bernoulli originally specified a deadline of six months, but due to a request by Leibniz extended it to one year. The challenge was delivered to Isaac Newton on January 29, 1697, and before leaving for work the next morning he had invented the calculus of variations and used it to solve the brachistochrone problem. Newton published his solution anonymously, but when Bernoulli saw it, he said "We recognize the lion by his claw"[126]. The problem is to find the shape of a frictionless wire that causes a bead, initially at rest, to move under the influence of gravity to a specified point in minimum time.

A dynamic optimization problem that arises in process engineering is finding optimal operating policies for batch unit operations. Examples include finding a temperature profile that maximize selectivity to the desired product in a batch reaction involving competing kinetics, or a reflux and accumulator dump policy for a batch distillation column that maximizes profit subject to purity and recovery constraints. In a real plant, the profiles determined *a priori* by the solution to the dynamic opti-

mization problem can serve as set point programs for the control systems. Although the literature abounds with analytical and numerical treatments of such problems (see for example [118] for a review), the extension of dynamic optimization to the design of integrated plant-wide operating policy for an entire batch process has only been contemplated in recent years. The benefits of plant-wide dynamic optimization of batch processes were first demonstrated in [10], and more recent work [19, 33] shows that dynamic optimization of relatively sophisticated plant-wide models involving thousands of states is possible.

The dynamic optimization problems that arise in process engineering have two distinct characteristics. First, the dynamic models are composed of sets of differential and algebraic equations (DAEs). The differential equations typically arise from dynamic material and energy balances, while the algebraic equations arise from thermodynamic and kinetic relationships and the physical connectivity of a process flowsheet. Although numerical solution of the types of DAEs that typically arise in dynamic simulation of chemical processes has become reliable in recent years, dynamic optimization problems typically involve (either explicitly or implicitly) more complex forms of DAEs, called *high-index DAEs*, for which numerical solution is still an active research area. Second, dynamic optimization problems that arise in process systems engineering often contain *path constraints* on state variables, which result from physical, safety, and economic limitations imposed on the system. State variable path constraints are not easily handled by dynamic optimization solution techniques that are currently available. In fact, it is shown later in this thesis that state variable path constraints lead to high-index DAEs.

A simple example of a dynamic optimization problem in process engineering is shown in Figure 1-1. A set of reactions takes place inside a pressure-constrained batch reactor. The objective is to maximize the amount of the product created in a fixed amount of time. The controls available are the feed rates of the reactants and the flow rate of the cooling water through the reactor cooling jacket. Since the reactions taking place are exothermic, state variable path constraints must be imposed on the pressure and temperature to keep them within safe limits throughout

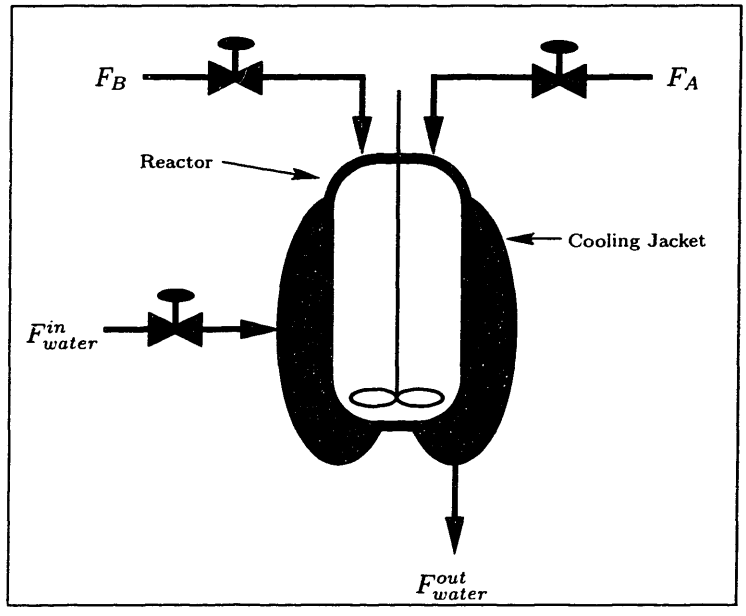

Figure 1-1: *Dynamic Optimization Example*

the entire process.

Although computational techniques for the solution of dynamic optimization problems has been an active research topic for decades (see for example, [24, 82]), local solution of large-scale DAE-based dynamic optimization problems has only recently been demonstrated [33, 142] using a method called *control parameterization.* However, the types of dynamic optimization problems that have been solved using control parameterization have been limited by the requirement that the DAEs not be high-index. Although this was thought to be sufficient for physically meaningful simulation problems, it is shown in this thesis that high-index DAEs are common in dynamic optimization problems. A related problem is that the methods that have been proposed to include state path constraints in the control parameterization method are computationally inefficient and incapable of guaranteeing that the constraints are satisfied to within a specified tolerance.

The objective of this thesis is to improve the ease with which dynamic optimization can be used in process engineering on a regular basis. The three specific areas of research were improving the computational efficiency of large-scale dynamic optimization solution methods, developing numerical solution techniques for the types

of high-index DAEs that arise in dynamic optimization problems, and developing the ability to include state variable path constraints on the dynamic optimization problem.

The dynamic optimization problem and a review of numerical solution techniques are presented in Chapter 2. A description of the control parameterization method is also given in Chapter 2. Control parameterization is dependent on the ability to calculate the sensitivity of a DAE model to system parameters, which is described in Chapter 3. A method for efficient numerical sensitivity calculation is given in Chapter 4. Efficient numerical solution of high-index DAEs is described in Chapter 5. Chapters 6 and 7 describe a new method for including state variable equality and inequality path constraints in dynamic optimization problems, and Chapter 8 gives numerical examples of this method. Chapter 9 contains conclusions and suggestions for future research.

Chapter 2

Dynamic Optimization: Theory and Numerical Methods

The solution of dynamic optimization (also known as optimal control) problems requires the determination of control trajectories that optimize some performance measure for a dynamic system. The objective of this chapter is to derive the first-order necessary conditions for the solution to a dynamic optimization problem, and discuss various numerical methods for solving these problems. The control parameterization method, which was the method chosen in this thesis to solve dynamic optimization problems, is described and justified. The dynamic optimization problems discussed in this chapter do not include so-called *additional* state variable path constraints, which are introduced in later chapters. These problems are constrained by the dynamic system, which is assumed to be a differential-algebraic equation (DAE). The control parameterization framework developed here is one that may be extended to handle state variable path constraints later in this thesis.

The following glossary may be useful when reading this chapter. These are brief definitions intended to clarify the discussion in this chapter, and they are explored in greater detail later in this thesis.

Differential-Algebraic Equation (DAE) A DAE is a system of equations that

can be written as

$$f(\dot{z}, z, t) = 0 \tag{2.1}$$

Ordinary differential equations (ODEs) are one class of DAEs. With the exception of ODEs, one characteristic of DAEs is that there are algebraic constraints on the state variables z. These constraints may appear explicitly as in

$$g(\dot{x}, x, y, t) = 0 \tag{2.2}$$

$$h(x, y, t) = 0 \tag{2.3}$$

where $z = (x, y)$, or they may appear implicitly due to singularity of $\frac{\partial f}{\partial \dot{z}}$ when it has no zero rows.

Differential Index of a DAE "The minimum number of times that all or part of (2.1) must be differentiated with respect to t in order to determine $\dot{z}$ as a continuous function of z, t is the index of the DAE" [21]. Reliable numerical techniques exist for the direct solution of DAEs that have index ≤ 1. The index is a *local* quantity– *i.e.*, it is defined at a particular point on the state trajectory and may change at discrete points along the state trajectory.

High-Index DAE According to common convention a DAE that has index > 1. The solution of a high-index DAE is constrained by time derivatives of a non-empty subset of the equations in the DAE. In general, only limited classes of high-index DAEs may be solved directly using standard numerical techniques.

Sufficient Conditions for a DAE to be at most Index-1 It can be shown that a sufficient condition for a DAE to at most index-1 is nonsingularity of the Jacobian of the DAE with respect to the highest order time derivatives of each state variable in the DAE. Therefore, if the DAE contains some differential state variables x and algebraic state variables y and the matrix $\frac{\partial f}{\partial q}$ is nonsingular, where $q = (\dot{x}, y)$, then the index of the DAE is at most one. Note that this is only

a sufficient condition, and DAEs for which this condition does not hold are not necessarily high-index. However, this is a very useful criterion for determining the index of many problems.

Corresponding Extended System A DAE may be transformed into an ODE by differentiating some or all of the equations in the DAE. The DAE and all of the time derivatives that were created during this differentiation are collectively called the corresponding extended system [58].

Consistent Initial Conditions The vectors $z(t_o)$, $\dot{z}(t_0)$ are called consistent initial conditions of (2.1) if they satisfy the corresponding extended system at t_0 [139].

Dynamic Degrees of Freedom Variables z or their time derivatives $\dot{z}$ which can be assigned arbitrary initial values and still allow a consistent initialization of (2.1) are called dynamic degrees of freedom [139]. The number of dynamic degrees of freedom is less than or equal to the number of state variables, and is related to the index of the DAE. In general, the higher the index of the DAE, the greater the number of nonredundant equations in the corresponding extended system, and the lower the number of dynamic degrees of freedom.

Design Degrees of Freedom The number of unknowns which can be specified arbitrarily as design or input quantities is called the number of design degrees of freedom [139]. This quantity is different from the dynamic degrees of freedom because it involves the input variables, the choice of which can affect the number of dynamic degrees of freedom [56, 87, 139].

Dummy Derivative Method A method for solving broad classes of high-index DAEs that was developed in [97] and refined during the course of the work for this thesis. The method works by deriving underlying index-1 DAEs that have the same solution set as the high-index DAE and may be solved using standard numerical techniques.

2.1 DAE Dynamic Optimization Necessary Conditions

There are many texts (*e.g.*, [24, 82]) that describe in detail the theory of dynamic optimization. Most of them derive the necessary conditions for optimality of optimal control problems under the assumption that the dynamic system is described by ordinary differential equations (ODEs). In this section, the more general first-order necessary optimality conditions are presented for dynamic systems that are described by DAEs. The subject of necessary conditions for optimal control for DAEs was also addressed in [36, 145], however both papers focus on the derivation of a Maximum Principle. A Maximum Principle is useful when there are constraints on the control variable, but in this section the dynamic optimization problem is assumed to have no constraints on the control variable in order to demonstrate that the necessary conditions for optimality for a DAE are also DAEs with some interesting properties.

The dynamic optimization problem considered in this section is one where the initial state for the state variables is given (that is, it is not free to be determined by the optimization), the control trajectories are unconstrained, and the state trajectories are constrained only by the DAE.

The dynamic optimization problem described above is expressed mathematically as:

$$\min_{u(t),t_f} J = \psi\left(x\left(t_f\right), t_f\right) + \int_{t_0}^{t_f} L\left(x, u, t\right) dt \tag{2.4}$$

subject to the DAE:

$$f\left(\dot{x}, x, u, t\right) = 0 \tag{2.5}$$

$$\phi\left(\dot{x}\left(t_0\right), x\left(t_0\right), t_0\right) = 0 \tag{2.6}$$

where $J(\cdot), L(\cdot), \psi(\cdot) \rightarrow \mathbb{R}$, $f(\cdot), \phi(\cdot) \rightarrow \mathbb{R}^{m_x}$, $x \in \mathbb{R}^{m_x}$, and $u \in \mathbb{R}^{m_u}$. The state variables x in this formulation include both differential and algebraic state variables.

The DAE (2.5) may have arbitrary differential index v, and (2.6) defines a consistent set of initial conditions. Since the index of (2.5) is not restricted, this formulation may include equality path constraints on state variables.

The function ψ in (2.4) may be expressed as:

$$\psi\left(x\left(t_f\right), t_f\right) = \psi\left(x\left(t_0\right), t_0\right) + \int_{t_0}^{t_f} \frac{d\psi}{dt} dt \tag{2.7}$$

Since it is assumed that the initial time t_0 and state condition $x(t_0)$ are fixed, the objective function may be expressed as:

$$J = \int_{t_0}^{t_f} \bar{L}\left(\dot{x}, x, u, t\right) dt \tag{2.8}$$

where:

$$\bar{L}(\dot{x}, x, u, t) = \frac{d\psi}{dt} + L = \frac{\partial \psi}{\partial t} + \left[\frac{\partial \psi}{\partial x}\right]^T \dot{x} + L \tag{2.9}$$

An augmented objective function is formed by appending the DAE constraints to the objective function by using the *adjoint* variables $\lambda\left(t\right)$:

$$\bar{J} = \int_{t_0}^{t_f} \left[\bar{L}\left(\dot{x}, u, t\right) + \lambda^T\left(t\right) f\left(\dot{x}, x, u, t\right)\right] dt \tag{2.10}$$

It is convenient to define the *Hamiltonian* as:

$$H\left(\dot{x}, x, u, \lambda, t\right) = \bar{L}\left(\dot{x}, x, u, t\right) + \lambda^T\left(t\right) f\left(\dot{x}, x, u, t\right) \tag{2.11}$$

In order to derive the necessary optimality conditions, it is necessary to define the variation of a functional. For the functional:

$$\bar{J} = \int_{t_0}^{t_f} H(\dot{x}, x, u, \lambda, t) dt \tag{2.12}$$

the *increment* of the functional is:

$$\begin{aligned}\Delta\bar{J} = &\int_{t_0}^{t_f} [H(\dot{x}+\delta\dot{x}, x+\delta x, u+\delta u, \lambda+\delta\lambda, t) - H(\dot{x}, x, u, \lambda, t)]\, dt \\ &+ \int_{t_f}^{t_f+\delta t_f} H(\dot{x}, x, u, \lambda, t) dt\end{aligned} \tag{2.13}$$

Expanding the increment in a Taylor series around the point $(\dot{x}(t), x(t), y(t))$ and extracting the terms that are linear in $\delta\dot{x}$, δx, δu, $\delta\lambda$, and δt_f gives the variation of $\bar{J}$:

$$\begin{aligned}\delta\bar{J} = &\int_{t_0}^{t_f} \left[\frac{\partial H}{\partial \dot{x}}\delta\dot{x} + \frac{\partial H}{\partial x}\delta x + \frac{\partial H}{\partial u}\delta u + \frac{\partial H}{\partial \lambda}\delta\lambda\right] dt \\ &+ H(\dot{x}, x, u, \lambda, t)\delta t_f\end{aligned} \tag{2.14}$$

which can be simplified by integrating the first term by parts to obtain:

$$\begin{aligned}\delta\bar{J} = &\int_{t_0}^{t_f} \left[\left[\frac{\partial H}{\partial x} - \frac{d}{dt}\left[\frac{\partial H}{\partial \dot{x}}\right]\right]\delta x + \frac{\partial H}{\partial u}\delta u + \frac{\partial H}{\partial \lambda}\delta\lambda\right] dt \\ &+ \left[\frac{\partial H}{\partial \dot{x}}\right]_{t=t_f} \delta x(t_f) + H(\dot{x}, x, u, \lambda, t)\delta t_f\end{aligned} \tag{2.15}$$

Using the following relation:

$$\delta x(t_f) = \delta x_f - \dot{x}\delta t_f \tag{2.16}$$

with (2.15) yields:

$$\begin{aligned}\delta\bar{J} = &\left[\frac{\partial H}{\partial \dot{x}}\right]_{t=t_f} \delta x_f + \left[H - \frac{\partial H}{\partial \dot{x}}\dot{x}\right]_{t=t_f} \delta t_f \\ &+ \int_{t_0}^{t_f} \left[\left[\frac{\partial H}{\partial x} - \frac{d}{dt}\frac{\partial H}{\partial \dot{x}}\right]\delta x + \frac{\partial H}{\partial u}\delta u + \frac{\partial H}{\partial \lambda}\delta\lambda\right] dt\end{aligned} \tag{2.17}$$

First-order necessary conditions for an optimum can be found by setting the variation

of $\bar{J}$ equal to zero. The conditions are:

$$\left[\frac{\partial H}{\partial x}\right] - \frac{d}{dt}\left[\frac{\partial H}{\partial \dot{x}}\right] = 0 \tag{2.18}$$

$$\frac{\partial H}{\partial u} = 0 \tag{2.19}$$

$$\frac{\partial H}{\partial \lambda} = 0 \tag{2.20}$$

$$\left[\frac{\partial H}{\partial \dot{x}}\delta x\right]_{t=t_f} + \left[H - \frac{\partial H}{\partial \dot{x}}\dot{x}\right]_{t=t_f} \delta t_f = 0 \tag{2.21}$$

Conditions (2.18–2.21) define a two-point boundary value problem in DAEs, as is the case with an ODE embedded system. Condition (2.18) is often called the *costate* equation. These conditions may be simplified. In the expansion of (2.18), the terms that include ψ are:

$$\begin{aligned}\frac{\partial}{\partial x}\left[\frac{\partial \psi}{\partial x}\dot{x} + \frac{\partial \psi}{\partial t}\right] - \frac{d}{dt}\left[\frac{\partial}{\partial \dot{x}}\left[\frac{\partial \psi}{\partial x}\dot{x}\right]\right] \\ = \left[\frac{\partial^2 \psi}{\partial x \partial t} + \frac{\partial^2 \psi}{\partial^2 x}\dot{x}\right] - \left[\frac{\partial^2 \psi}{\partial t \partial x} + \frac{\partial^2 \psi}{\partial^2 x}\dot{x}\right] \\ = 0\end{aligned} \tag{2.22}$$

provided that the second partial derivatives are continuous.

Using (2.11) and (2.22), (2.18–2.21) simplify to:

$$\frac{\partial L}{\partial x} + \lambda^T \frac{\partial f}{\partial x} - \dot{\lambda}^T \frac{\partial f}{\partial \dot{x}} - \lambda^T \frac{d}{dt}\left[\frac{\partial f}{\partial \dot{x}}\right] = 0 \tag{2.23}$$

$$\frac{\partial L}{\partial u} + \lambda^T \frac{\partial f}{\partial u} = 0 \tag{2.24}$$

$$f(\dot{x}, x, u, t) = 0 \tag{2.25}$$

$$\left[\lambda^T \frac{\partial f}{\partial \dot{x}} + \frac{\partial \psi}{\partial x}\right]_{t=t_f} \delta x_f + \left[\frac{\partial \psi}{\partial t} + L + \lambda^T f - \lambda^T \left[\frac{\partial f}{\partial \dot{x}}\right]\dot{x}\right]_{t=t_f} \delta t_f = 0 \tag{2.26}$$

These conditions are a generalization of the conditions that have been reported for dynamic optimization of ODE systems.

2.1.1 ODE Example

The following dynamic optimization problem was taken from [24] and is attributed to Isaac Newton. Essentially, the problem is to find the minimum-drag nose cone shape in hypersonic flow. The problem is:

$$\min_{u(x)} \frac{1}{2}[r(l)]^2 + \int_0^l \frac{ru^3}{1+u^2} dx \tag{2.27}$$

subject to:

$$\frac{dr}{dx} + u = 0 \tag{2.28}$$

$$r(0) = r_0 \tag{2.29}$$

where x is the axial distance from the point of maximum radius, r is the radius of the body, $r(0)$ is the maximum radius, and l is the length of the body.

The optimality conditions (2.23–2.26) reduce in this problem to:

$$\frac{d\lambda}{dx} = \frac{u^3}{1+u^2} \tag{2.30}$$

$$\lambda = \frac{ru^2(3+u^2)}{(1+u^2)^2} \tag{2.31}$$

$$\frac{dr}{dx} + u = 0 \tag{2.32}$$

$$r(0) = r_0 \tag{2.33}$$

$$\lambda(l) = -r(l) \tag{2.34}$$

These are the same optimality conditions as those derived in [24] except that the sign of λ in (2.31) and (2.34) is different. This difference occurs because the usual derivation of conditions (2.23–2.26) for ODEs assumes that the ODEs have the form $\dot{x} = f(x, u, t)$, while the derivation of (2.23–2.26) made the more general assumption that the DAE has the form $f(\dot{x}, x, u, t) = 0$. However, it can be shown that the solution for the state variable given in [24] also satisfies (2.30–2.34), although the adjoint variable trajectories are different.

Note that even though the original problem is described by an ODE, the optimality conditions are an index-1 DAE (when $u \neq 0$), since (2.31) contains no time derivative variables. In general, all dynamic optimization problems described by ODEs have optimality conditions that are DAEs.

2.1.2 DAE Example #1

It is possible to turn the ODE example of the previous section into an index-1 DAE, and thus check the validity of the optimality conditions, by writing:

$$\min_{u(x)} a(l) + \int_0^l b dx \tag{2.35}$$

subject to:

$$\frac{dr}{dx} + u = 0 \tag{2.36}$$

$$a = \frac{1}{2} r^2 \tag{2.37}$$

$$b = \frac{ru^3}{1+u^2} \tag{2.38}$$

$$r(0) = r_0 \tag{2.39}$$

The optimality conditions for this problem (neglecting the condition (2.26) for the moment) reduce to (2.36–2.39) and:

$$\frac{d\lambda_1}{dx} = -\frac{u^3 \lambda_2}{1+u^2} - \frac{1}{2}\lambda_3 \tag{2.40}$$

$$\lambda_2 = -1 \tag{2.41}$$

$$\lambda_3 = 0 \tag{2.42}$$

$$\lambda_1 = -\frac{ru^2(3+u^2)}{(1+u^2)^2}\lambda_2 \tag{2.43}$$

If (2.41–2.42) are substituted into (2.40) and (2.43), the result is the same as (2.30–2.31).

The boundary condition (2.26) reduces to:

$$\lambda_1(l) = 0 \tag{2.44}$$

$$1 = 0 \tag{2.45}$$

which apart from being inconsistent, does not agree with the results of the ODE example. The problem is that the variable a appears in ϕ but $\dot{a}$ does not appear in the DAE (this is further discussed in Section 2.2 below). If (2.37) is replaced with its time differential, the boundary condition reduces to:

$$\lambda_1(l) = r(l)\lambda_3(l) \tag{2.46}$$

$$\lambda_3(l) = -1 \tag{2.47}$$

which reduces to the result obtained in the ODE example.

2.1.3 DAE Example #2

The following dynamic optimization contains a high-index (index-3) DAE:

$$\min_{u(t)} \left(x(t_f) - \frac{1}{2} \right)^2 \tag{2.48}$$

subject to:

$$\dot{x} = v \tag{2.49}$$

$$\dot{y} = w \tag{2.50}$$

$$\dot{v} + Tx = u^2 y \tag{2.51}$$

$$\dot{w} + Ty = -1 + u^2 x \tag{2.52}$$

$$x^2 + y^2 = 1 \tag{2.53}$$

$$x(t_0) = 0 \tag{2.54}$$

$$\dot{y}(t_0) = 0 \tag{2.55}$$

The costate equations are:

$$\dot{\lambda}_1 = T\lambda_3 - u^2\lambda_4 + 2x\lambda_5 \tag{2.56}$$

$$\dot{\lambda}_2 = -u^2\lambda_3 + T\lambda_4 + 2y\lambda_5 \tag{2.57}$$

$$\dot{\lambda}_3 = -\lambda_1 \tag{2.58}$$

$$\dot{\lambda}_4 = -\lambda_2 \tag{2.59}$$

$$x\lambda_3 + y\lambda_4 = 0 \tag{2.60}$$

and (2.24) simplifies to:

$$uy\lambda_3 + ux\lambda_4 = 0 \tag{2.61}$$

The index of the combined system (2.49–2.53) and (2.56–2.61) is 3. A degree of freedom analysis indicates that there should be two initial time conditions and two final time conditions. However, (2.26) gives:

$$\lambda_1(t_f) = 1 - 2x(t_f) \tag{2.62}$$

$$\lambda_2 = 0 \tag{2.63}$$

$$\lambda_3 = 0 \tag{2.64}$$

$$\lambda_4 = 0 \tag{2.65}$$

which cannot be correct. As discussed in Section 2.2 the boundary conditions must be derived using an equivalent non-high index formulation of the DAE.

Another way to find optimality conditions for this problem is to use the methods

of Chapter 5 to derive an equivalent index-1 DAE for (2.49–2.53):

$$\dot{x} = v \tag{2.66}$$

$$\dot{y} = w \tag{2.67}$$

$$m\dot{v} + Tx = u^2 y \tag{2.68}$$

$$m\dot{w} + Ty = -1 + u^2 x \tag{2.69}$$

$$x^2 + y^2 = 1 \tag{2.70}$$

$$x\dot{x} + y\bar{y} = 0 \tag{2.71}$$

$$\dot{x}^2 + \bar{y}^2 + x\dot{v} + y\bar{w} = 0 \tag{2.72}$$

where $\bar{y}$ and $\bar{w}$ are dummy algebraic variables that have been introduced in place of $\dot{y}$ and $\dot{w}$. For this system, the costate equations are:

$$\dot{\lambda}_1 = T\lambda_3 - u^2\lambda_4 + 2x\lambda_5 + \dot{x}\lambda_6 + \dot{v}\lambda_7 - x\dot{\lambda}_6 - 2\dot{x}\dot{\lambda}_7 \tag{2.73}$$

$$-u^2\lambda_3 + T\lambda_4 + 2y\lambda_5 + \bar{y}\lambda_6 + \bar{w}\lambda_7 = 0 \tag{2.74}$$

$$\dot{\lambda}_3 = -\lambda_1 - x\dot{\lambda}_7 - \dot{x}\lambda_7 \tag{2.75}$$

$$\lambda_2 = 0 \tag{2.76}$$

$$x\lambda_3 + y\lambda_4 = 0 \tag{2.77}$$

$$\lambda_2 + y\lambda_6 + 2\bar{y}\lambda_7 = 0 \tag{2.78}$$

$$\lambda_4 + y\lambda_7 = 0 \tag{2.79}$$

and (2.24) gives:

$$uy\lambda_3 + ux\lambda_4 = 0 \tag{2.80}$$

Interestingly, in this case the combined system (2.66–2.80) is index-2, even though the DAE (2.66–2.72) is index-1. A degree of freedom analysis indicates that there are two initial and two final time boundary conditions, as expected.

Applying the boundary condition (2.26) gives:

$$2x(t_f) - 1 + \lambda_1(t_f) + x(t_f)\lambda_6(t_f) + 2\dot{x}(t_f)\lambda_7(t_f) = 0 \tag{2.81}$$

$$\lambda_3(t_f) + x(t_f)\lambda_7(t_f) = 0 \tag{2.82}$$

which is the correct number of final time conditions. This example leads to a conjecture that it is possible to derive optimality conditions for a high-index DAE directly, but that the final-time boundary conditions must be derived using an equivalent non-high index formulation of the high-index DAE.

2.1.4 DAE Example #3

Consider the following DAE dynamic optimization problem:

$$\min_{\theta(t), t_f} t_f \tag{2.83}$$

subject to:

$$\dot{x} = u \tag{2.84}$$

$$\dot{y} = v \tag{2.85}$$

$$\dot{u} = F\sin(\theta) \tag{2.86}$$

$$\dot{v} = g - F\cos(\theta) \tag{2.87}$$

$$\tan(\theta) = \frac{v}{u} \tag{2.88}$$

$$[x(0), y(0), \dot{x}(0)] = [0, 1, 0]$$

This problem is a version of the brachistochrone problem due to [14] which is further discussed in Chapter 6. This problem is used in the next section to demonstrate the different types of boundary conditions that are possible in dynamic optimization problems. The index of the DAE in this problem is two if θ is selected as the control.

The optimality condition (2.23) reduces to:

$$\dot{\lambda}_1 = 0 \tag{2.89}$$

$$\dot{\lambda}_2 = 0 \tag{2.90}$$

$$\dot{\lambda}_3 = -\lambda_1 + \lambda_5 \frac{v}{u^2} \tag{2.91}$$

$$\dot{\lambda}_4 = -\lambda_2 - \lambda_5 \frac{1}{u} \tag{2.92}$$

$$0 = -\lambda_3 \sin(\theta) + \lambda_4 \cos(\theta) \tag{2.93}$$

Note that this condition is also a DAE.

Condition (2.24) reduces to:

$$\lambda_3 F \cos(\theta) + \lambda_4 F \sin(\theta) + \lambda_5 \sec^2(\theta) = 0 \tag{2.94}$$

To obtain boundary conditions for this problem, it is necessary to consider both the index of the DAE (2.84–2.88), and the index of the extended DAE that is formed by (2.84–2.94) so that the correct dynamic degrees of freedom may be determined. The dynamic degrees of freedom of the original DAE determines the number of initial conditions that may be specified, and the dynamic degrees of freedom of the extended system determine the number of final conditions that may be specified. This result has not been previously noted in the literature, and it is further discussed in the next section.

2.2 Boundary conditions

The exact form of the boundary condition (2.21) depends on the classes of constraints imposed on the dynamic optimization problem. This section presents the boundary conditions for several different classes of constraints. The approach followed is generally that of [82], but here the results are valid for arbitrary-index DAEs.

The number of boundary conditions depends on the form of (2.5), and is related to r_d, the dynamic degrees of freedom of the DAE. Several observations may be made about the number of boundary conditions that may be imposed:

- The total number of boundary conditions that may be imposed is $r_{overall} + 1$, where $r_{overall}$ is the dynamic degrees of freedom of the extended system (2.23–2.25). The number of boundary conditions is one more than $r_{overall}$ because one condition must be imposed to define t_f.

- The number of boundary conditions can be dependent on the time at which the conditions are defined, since $r_{overall}$ can change over the time domain.

- Typically, the boundary conditions consist of initial conditions on the DAE and end-point conditions on the costate equations. In this case, the number of initial conditions for the DAE is r_d at $t = t_0$. The number of end-point conditions on the costate equations is $r_{overall} - r_d$ at $t = t_f$.

- It is not possible to make general statements concerning either the dependence of the index of (2.23–2.25) on the index of (2.25) or the dependence of $r_{overall}$ on r_d.

The final-time boundary conditions are obtained by customizing (2.26) to the particular type of side conditions in the dynamic optimization problem. It is important to note that although (2.23–2.25) can be applied to DAEs of arbitrary index, (2.26) cannot be correctly applied directly to high-index DAEs. Consistent initial conditions for a high-index DAE can only be obtained with the corresponding extended system for the high-index DAE, and likewise (2.26) can only be used with the corresponding

extended system for the DAE (2.23–2.25). Therefore, for the purposes of this section, the DAE f is assumed to have index at most one. This restriction is allowable because the methods of Chapter 5 can be used to derive an equivalent index-1 DAE for the high-index DAEs of interest in this thesis.

2.2.1 Problems with t_f fixed

Fixing t_f provides the boundary condition on the final time, and allows δt_f to be set to zero in (2.26). There are several possibilities for the other boundary conditions, depending on the form of the dynamic optimization problem.

- *Final state free*: Since the final state is unspecified, $\delta x(t_f)$ is arbitrary, and therefore the end-point boundary conditions must satisfy:

$$\left[\lambda^T \frac{\partial f}{\partial \dot{x}} + \frac{\partial \psi}{\partial x}\right]_{t=t_f} \delta x_f = 0 \tag{2.95}$$

  Note that when:

$$\left[\lambda^T \frac{\partial f}{\partial \dot{x}_i} + \frac{\partial \psi}{\partial x_i}\right]_{t=t_f} = 0 \tag{2.96}$$

  that δx_{fi} is arbitrary.

- *Final state constrained*: If the final state is constrained to lie on a hypersurface defined by $m(x(t_f)) = 0$, where $m : \mathbb{R}^{n_x} \rightarrow \mathbb{R}^{n_m}$, $1 \leq n_m \leq r_d - 1$, and if it is assumed that $m(x(t_f))$ is neither inconsistent nor redundant, then $\delta x(t_f)$ must be normal to each of the gradient vectors:

$$\frac{\partial m_i}{\partial x} \qquad i = 1 \ldots n_m \tag{2.97}$$

and therefore it can be shown [82] that the boundary conditions are:

$$\left[\lambda^T \frac{\partial f}{\partial \dot{x}} + \frac{\partial \psi}{\partial x}\right]_{t=t_f} = \sum_{i=1}^{n_m} \nu_i \frac{\partial m_i}{\partial x} \tag{2.98}$$

$$m(x(t_f)) = 0 \tag{2.99}$$

where $\nu \in \mathbb{R}^{n_m}$ is a vector of undetermined multipliers.

- *Final state specified*: This is a special instance of the preceding case. However, here the dimensionality of the hypersurface is equal to r_d, and therefore the boundary conditions:

$$x(t_f) = x_f \tag{2.100}$$

apply. The boundary condition (2.26) applies also, but for every equation in (2.100), the corresponding $\delta x = 0$ in (2.26). Therefore, if $r_{overall} > 2r_d$ some of the boundary conditions will come from (2.26).

2.2.2 Problems with t_f free

When t_f is free, the assumption that $\delta t_f = 0$ can no longer be made.

- *Final state fixed or final state free*: These cases have the same boundary conditions as their counterpart discussed above except that there is an additional boundary condition:

$$\left[\frac{\partial \psi}{\partial t} + L + \lambda^T f - \lambda^T \left[\frac{\partial f}{\partial \dot{x}}\right] \dot{x}\right]_{t=t_f} = 0 \tag{2.101}$$

- *Final state constrained to lie on a moving point*: In this case, $x(t_f)$ must lie on the final point defined by $\theta(t_f)$. This case is complicated by the fact that no more than r_d state conditions may be simultaneously imposed, and r_d will be less than m_x if the DAE is index-1. The dimension of the vector function θ is

equal to r_d, and the variations of δx_f and δt_f are related by:

$$P\delta x_f = \frac{d\theta}{dt}\delta t_f \tag{2.102}$$

where P is a permutation matrix which maps $\mathbb{R}^{m_x} \rightarrow \mathbb{R}^{r_d}$. The boundary condition for this problem is therefore:

$$\left[\left[\lambda^T\frac{\partial f}{\partial \dot{x}} + \frac{\partial \psi}{\partial x}\right]P\frac{d\theta}{dt}\right]_{t=t_f} + \left[\frac{\partial \psi}{\partial t} + L + \lambda^T f - \lambda^T\left[\frac{\partial f}{\partial \dot{x}}\right]\dot{x}\right]_{t=t_f} = 0 \tag{2.103}$$

$$Px(t_f) = \theta(t_f) \tag{2.104}$$

Note that (2.102) will not define all δx if r_d is less than m_x. Equation (2.95) holds for all undefined elements of δx, and may give additional boundary conditions.

- *Final state constrained by a surface*: Again, the final state is constrained by the equations $m(x(t_f)) = 0$, $m : \mathbb{R}^{m_x} \rightarrow \mathbb{R}^{n_m}$, $1 \leq n_m \leq r_{overall} - r_d - 1$ at t_f. Since the final time is free, the boundary conditions are the same as the corresponding case with fixed t_f with the addition of the boundary condition (2.101).

- *Final state constrained by a moving surface*: This is the most complicated case, where the final state is constrained by the moving surface $m(x,t) = 0$, $m : \mathbb{R}^{m_x+1} \rightarrow \mathbb{R}^{n_m}$, $1 \leq n_m \leq r_{overall} - r_d - 1$ at t_f. This situation is similar to the preceding case, except that the vector $[\delta x(t_f) \mid \delta t_f]$ is normal to each of the gradient vectors $[\frac{\partial m_i}{\partial x} \mid \frac{\partial m_i}{\partial t}]$. Therefore, the boundary conditions are:

$$\left[\lambda^T\frac{\partial f}{\partial \dot{x}} + \frac{\partial \psi}{\partial x}\right]_{t=t_f} = \sum_{i=1}^{n_m}\nu_i\frac{\partial m_i}{\partial x} \tag{2.105}$$

$$m(x(t_f)) = 0 \tag{2.106}$$

$$\left[\frac{\partial \psi}{\partial t} + L + \lambda^T f - \lambda^T\left[\frac{\partial f}{\partial \dot{x}}\right]\dot{x}\right]_{t=t_f} = \sum_{i=1}^{n_m}\nu_i\frac{\partial m_i}{\partial t} \tag{2.107}$$

2.2.3 Problems with algebraic variables in ψ

As shown in Section 2.1.2, if the function ψ in (2.4) contains algebraic state variables, special treatment is required to derive the boundary conditions with (2.26). The difficulty is that the time derivatives of algebraic variables do not appear explicitly in the DAE, but $\partial H/\partial \dot{x}$ will be nonzero with respect to the time derivatives of any algebraic variables for which the corresponding elements of $\partial \psi/\partial x$ are nonzero.

The boundary condition (2.26) can be correctly applied if a subset of the equations DAE $f(\dot{x}, x, u, t)$ is replaced with its time derivatives for the purposes of (2.26). It is valid to do this because all time derivatives of $f(\dot{x}, x, u, t)$ must hold at t_0.

2.2.4 Boundary condition examples

The dynamic degrees of freedom of the DAE (2.84–2.88) and the extended DAE (2.84–2.93) are three and seven, respectively. Three initial conditions were defined, and therefore, four final time conditions remain to be specified. The implications of some of the various boundary condition types can be demonstrated by considering the following cases:

- *Case 1: Final condition is $x(t_f) = x_f$:*

  This is the case where the final state is constrained by a hypersurface. Since the final time is free, there is an additional degree of freedom that is determined by the dynamic optimization problem (2.83–2.88). The final time condition (2.101) reduces to:

$$\left[1 - \lambda_1 u - \lambda_2 v - \lambda_3 F \sin(\theta) - \lambda_4 F \cos(\theta) + \lambda_5 \left(\tan(\theta) - \frac{v}{u}\right)\right]_{t=t_f} = 0 \quad (2.108)$$

The optimality conditions (2.98–2.99) reduce to:

$$\lambda_1(t_f) = \nu_1 \tag{2.109}$$

$$\lambda_2(t_f) = 0 \tag{2.110}$$

$$\lambda_3(t_f) = 0 \tag{2.111}$$

$$\lambda_4(t_f) = 0 \tag{2.112}$$

$$x(t_f) = x_f \tag{2.113}$$

Note that the undetermined multiplier ν_1 has been added to the dynamic optimization problem, and there are now four remaining dynamic degrees of freedom.

- *Case 2: Final condition is* $x(t_f) = a(t_f)$, $y(t_f) = b(t_f)$, $v(t_f) = 0$:

  This is the case where the final state must lie on a moving point. Assuming the state variable vector for this problem is $[x, y, u, v, F]$, the permutation matrix P for this problem is:

$$P = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix} \tag{2.114}$$

  and therefore, (2.102) becomes:

$$\delta x(t_f) = a'(t_f)\delta t_f \tag{2.115}$$

$$\delta y(t_f) = b'(t_f)\delta t_f \tag{2.116}$$

$$\delta v(t_f) = 0 \tag{2.117}$$

Note that δu and δF can take arbitrary values and still satisfy the boundary

condition. Equations (2.103–2.104) reduce to:

$$\begin{aligned}[\lambda_1 a' + \lambda_2 b' + 1 - \lambda_1 u - \lambda_2 v - \lambda_3 F\sin(\theta) - \\ \lambda_4 F\cos(\theta) - \lambda_5\left(\tan(\theta) - \frac{v}{u}\right)\Big]_{t=t_f} = 0\end{aligned} \tag{2.118}$$

$$x(t_f) = a(t_f) \tag{2.119}$$

$$y(t_f) = b(t_f) \tag{2.120}$$

$$\dot{y}(t_f) = 0 \tag{2.121}$$

There is still one more final time condition that must be defined. Equation (2.95) is applied to the undefined elements of δx to obtain the final boundary condition:

$$\lambda_3(t_f) = 0 \tag{2.122}$$

- *Case 3: Final condition is* $x^2(t_f) + y^2(t_f) = R(t_f)^2$:

  In this case, the final state is constrained to lie on a circle that is expanding with time (*i.e.*, $R(t)$ is a scalar valued function of time). Equations (2.105–2.107) reduce to:

$$\lambda_1(t_f) = 2\nu_1 x(t_f) \tag{2.123}$$

$$\lambda_2(t_f) = 2\nu_1 y(t_f) \tag{2.124}$$

$$\lambda_3(t_f) = 0 \tag{2.125}$$

$$\lambda_4(t_f) = 0 \tag{2.126}$$

$$x^2(t_f) + y^2(t_f) = R(t_f)^2 \tag{2.127}$$

$$\begin{aligned}[1 - \lambda_1 u - \lambda_2 v - \lambda_3 F\sin(\theta) \\ - \lambda_4 F\cos(\theta) + \lambda_5(\tan(\theta) - \frac{v}{u}\Big]_{t=t_f} = -\nu_1\frac{\partial R}{\partial t}\end{aligned} \tag{2.128}$$

- *Case 4: Fixed final time with objective function* x^2:

In this case, the objective function (2.83) is changed to:

$$\min_{\theta(t)} x^2(t_f) \tag{2.129}$$

and the final time is not a decision variable. Equations (2.89–2.94) apply unchanged to this problem, and (2.95) gives the boundary conditions:

$$\lambda_1 + 2x = 0 \tag{2.130}$$

$$\lambda_2 = 0 \tag{2.131}$$

$$\lambda_3 = 0 \tag{2.132}$$

$$\lambda_4 = 0 \tag{2.133}$$

2.3 DAEs and Dynamic Optimization Problems

The previous section demonstrated that it is necessary to know $r_{overall}$ at t_f in order to determine a valid set of end-point boundary conditions. It is also necessary to know r_d at t_0 in order to determine consistent initial conditions for the DAE. This section shows that in fact the necessary conditions for optimality are naturally DAEs, and therefore the solution of dynamic optimization problems requires the ability to handle DAEs. This situation is true even for many problems where the dynamic model is described by an ODE.

The extended system (2.23–2.25) is a DAE with an index of at least one, even if (2.25) is an ODE, which may be observed in the example (2.27–2.34). The extended system must be at least index-1 because it contains algebraic variables u.

To show that the index of the extended system can be greater than one, even for an ODE-embedded system, consider the following system that was discussed in [24, 90].

$$\min_u \Phi(x(t_f)) \tag{2.134}$$

subject to:

$$\dot{x} = f(x) + g(x)u \tag{2.135}$$

This problem is a very simple ODE system which is linear in the control variable u, and possibly nonlinear in the state variable x. The necessary conditions (2.23–2.26) simplify to generate the extended system:

$$\dot{\lambda} = -\left[\frac{\partial f}{\partial x} + \left[\frac{\partial g}{\partial x}\right] u\right] \lambda \tag{2.136}$$

$$\lambda g(x) = 0 \tag{2.137}$$

$$\dot{x} = f(x) + g(x)u \tag{2.138}$$

$$\lambda(t_f) = -\frac{\partial \Phi(t_f)}{\partial x} \tag{2.139}$$

The Jacobian of (2.136–2.138) with respect to the highest order time derivatives $(\dot{\lambda}, \dot{x}, u)$ is:

$$\begin{bmatrix} 1 & 0 & \left[\frac{\partial g}{\partial x}\right]\lambda \\ 0 & 0 & 0 \\ 0 & 1 & -g \end{bmatrix} \tag{2.140}$$

which is singular. However, if (2.137) is replaced with its time derivative:

$$\lambda \frac{\partial g}{\partial x}\dot{x} + \dot{\lambda} g(x) = 0 \tag{2.141}$$

the Jacobian of the underlying DAE consisting of (2.136), (2.141) and (2.138) is:

$$\begin{bmatrix} 1 & 0 & \left[\frac{\partial g}{\partial x}\right]\lambda \\ g & \left[\frac{\partial g}{\partial x}\right]\lambda & 0 \\ 0 & 1 & -g \end{bmatrix} \tag{2.142}$$

which is structurally nonsingular. Since an algebraic state variable u is still present in the underlying DAE, the index of the underlying DAE must be at least one. Therefore, the structural index of the original system (2.136–2.138) was two. However, it can be shown that matrices of the form of (2.140) are always singular. This singularity *may* indicate that the index of the underlying DAE is higher than two, but singularity of (2.140) is not a sufficient condition for a DAE to be high-index. This fact was overlooked in [90], however, algebraic rearrangement can be used to show that (2.136–2.138) is in fact index-3, as reported in that paper. To see this, substitute (2.136) and (2.138) into (2.141) to get:

$$\lambda \frac{\partial g}{\partial x} f(x) - \lambda \frac{\partial f}{\partial x} g(x) \tag{2.143}$$

since the variable u does not appear, a further differentiation of (2.136), (2.141) and (2.138) is required to derive the corresponding extended system.

It is interesting to see whether (2.26) has given the correct number of final time

boundary conditions for this problem. The corresponding extended system for this DAE consists of seven equations, (2.136–2.138), (2.141), and the time derivatives of (2.136), (2.141) and (2.138). The variables incident in the corresponding extended system are $\ddot{x}, \dot{x}, x, \ddot{\lambda}, \dot{\lambda}, \lambda, \dot{u}$, and u. Since there are eight incident variables and seven equations in the corresponding extended system, there should be one boundary condition. That boundary condition is the final-time condition given by (2.139), which implies no initial conditions may be specified for this problem.

The bottom line is that dynamic optimization problems are inherently boundary value problems in DAEs, not ODEs (even if the dynamic system is an ODE). Therefore, solving a dynamic optimization problem requires knowledge about the dynamic degrees of freedom of both the original dynamic system and the DAE defined by the optimality conditions in order to formulate valid boundary conditions, and the ability to solve the DAE that is embedded in the two-point boundary value problem either explicitly or implicitly. This fact has not been formally recognized in any of the work that has been done on development of methods to solve dynamic optimization problems. In fact, all of the methods that have been developed handle the DAEs implicitly, both because these problems were not recognized as DAE problems, and because advances that allowed direct numerical solution of DAEs have been made only recently.

2.4 Solution algorithms for Dynamic Optimization Problems

The optimality conditions presented in the previous section are of little practical use by themselves. They give conditions that must be satisfied at the optimum, but they are not a method for finding the optimum. In fact, they define a two-point boundary value problem, which is well known to be difficult to solve numerically. This section gives a brief overview of the various methods that have been developed to solve dynamic optimization problems.

There is an extensive literature on numerical methods for the solution of dynamic optimization problems, which fall into three classes. The *dynamic programming* method was described in [18] and the approach was extended to include constraints on the state and control variables in [91]. *Indirect* methods focus on obtaining a solution to the classical necessary conditions for optimality (2.18–2.21) which take the form of a two-point boundary value problem. There are many examples of such methods, for example [24, 37, 41, 42, 80, 100, 111]. Finally, *direct* methods transform the infinite-dimensional dynamic optimization problem into a finite dimensional mathematical programming problem, typically a nonlinear program (NLP).

The dynamic programming method is based on Bellman's principal of optimality, which can be used to derive the Hamilton-Jacobi-Bellman equation:

$$0 = \frac{\partial J}{\partial t} + \min_{u(t)} \left\{ L + \frac{\partial J}{\partial x} f(x,u,t) \right\} \tag{2.144}$$

$$J(x(t_f),t_f) = \psi(x(t_f),t_f) \tag{2.145}$$

which must hold at the optimum, assuming that the DAE is index-0 and may be written as $\dot{x} = f(x,u,t)$. This is a partial differential equation and in practice it is very difficult to solve except in certain fortuitous cases [133].

The indirect methods are so termed because they attempt to find solutions to the two-point boundary value problem (2.18–2.21), thus indirectly solving the dynamic optimization problem (2.4–2.6). There are many variations on indirect methods, but

they are generally iterative methods that use an initial guess to find a solution to a problem in which one or more of (2.18–2.21) is not satisfied. This solution is then used to adjust the initial guess to attempt to make the solution of the next iteration come closer to satisfying all of the necessary conditions.

One of the common variations on the indirect method is the *steepest descent algorithm* (for example, [82]). In this method, the state equation (2.20) is integrated forward using a guess for the control profile, and then the costate equation (2.18) is integrated backward. Equation (2.19) is then used locally to find a steepest descent direction for u at a discrete number of points, and globally as a termination criterion. The main disadvantage of this method is the need to backward-integrate the costate equations, which can be inefficient and/or unstable for both DAEs [142] and ODEs. Also, it is not easy to set up the problem, since the costate equations need to be generated, and convergence is slow for many problems.

Backward integration of the costate equations may be avoided by using a *direct shooting method*, in which the initial values of the adjoint variables $\lambda(t_0)$ are guessed initially, and updated iteratively [99]. There are some problems with this method, including a small region of convergence to the optimum, the need to formulate the costate equations, and the difficulties in choosing reasonable initial guesses for $\lambda(t_0)$, which may or may not have an obvious physical meaning. The *multiple shooting method* was proposed to extend the small convergence region of the direct shooting method [39, 79, 122]. Multiple shooting algorithms transform the problem into a multipoint boundary value problem for the state and adjoint variables. *Homotopy methods* have also been proposed [144] as a response to the small region of convergence, and automatic symbolic differentiation has been used to generate the costate equations [129]. Other indirect techniques reported have been the *finite difference method* [89] which is essentially a low-order collocation method, and standard *collocation techniques* [4].

There are two general strategies within the framework of the direct method. In the *sequential method*, often called *control parameterization*, the control variables are discretized over finite elements using polynomials [23, 105, 104, 115, 120, 128, 142, 143]

or in fact any suitable basis functions. The coefficients of the polynomials and the size of the finite elements then become decision variables in a master nonlinear program (NLP). Function evaluation is carried out by solution of an initial value problem (IVP) of the original dynamic system, and gradients for a gradient-based search may be evaluated by solving either the adjoint equations (*e.g.*, [128]) or the sensitivity equations (*e.g.*, [142]). In the *simultaneous strategy* both the controls and the state variables are discretized using polynomials on finite elements, and the coefficients and element sizes become decision variables in a much larger NLP (*e.g.*, see [12, 61, 85, 92, 106, 132, 137, 147]). Unlike control parameterization, the simultaneous method does not require the solution of IVPs at every iteration of the NLP.

Direct methods have been demonstrated to work automatically and reliably for very large problems. Within direct methods, there has been significant discussion in the literature about the relative advantages of simultaneous and sequential strategies. The main areas of contention are the ease of use, the computational cost in obtaining a solution, and the ability to handle state variable path constraints.

The sequential strategy is easier to use on large problems than the simultaneous strategy because it uses a numerical IVP solver to enforce (2.5), rather than including it as a set of constraints in the NLP. Including these constraints in the NLP in the sequential strategy causes the NLP to grow explosively with the size of the DAE, the time horizon of interest, and the excitation of high-frequency responses. Solution of such large NLPs requires special techniques and careful attention to determining an initial guess for the optimization parameters that leads to convergence of the NLP algorithm. On the other hand, solution of the IVP in the sequential strategy can take advantage of the highly refined heuristics available in state-of-the-art DAE integrators. In fact, it can be shown that if collocation is used for the discretization in the simultaneous method, the problem is equivalent to performing a fully implicit Runge-Kutta integration [142], which is not as efficient as the BDF method for DAEs [21].

It is not clear from the literature discussions which strategy requires less computational cost. On the one hand, it is expensive to solve the extremely large NLPs that

result from the simultaneous method. On the other hand, an efficient search using the sequential strategy requires gradients of the objective function, which can also be very expensive to obtain. This issue is addressed in Chapter 4, where an efficient method for calculating the sensitivities of the state variables is described, which can then be used to calculate the gradients.

Until now, the simultaneous strategy has had the clear advantage in handling constraints on the state variables. Such constraints may be handled by including them directly as additional constraints in the NLP. However, the theoretical properties of the discretization employed in the simultaneous strategy break down for nonlinear DAEs which have index > 2 (at least locally; for example, during an activation of a path inequality constraint) [90]. Attempts to handle state path constraints in the sequential strategy have relied on including some measure of their violation indirectly as point constraints in the NLP, rather than directly in the IVP subproblem [84, 133, 142]. This method was used because it was previously not possible to solve numerically the high-index DAEs which resulted from appending the path constraints to the IVP problem. In fact, the use of a dynamic optimization formulation with such point constraints has been proposed as a method for solving high-index DAEs [76], although this method is extraordinarily costly.

This work focuses on the control parameterization method. This choice was based on the fact that many problems in chemical engineering are modeled with large systems of equations, and the control parameterization method appears to be the most easily applied and reliable method for these problems. The reasons for this include:

- Direct methods are more easily implemented than indirect methods because there is no need to generate the additional costate equations. Generation of the costate equations is a significant problem for large dynamic optimization problems, and more than doubles the size of the DAE system that must be solved.

- It is extremely difficult to find numerical solutions to the two-point boundary value problem that indirect methods are dependent upon if m_x is large. In fact,

we were unable to find any examples in the literature of the use of indirect methods to solve a problem with $m_x > 50$. On the other hand, there are reported examples of the direct method being used to solve problems with thousands of state variables [33].

- The sequential direct method results in fairly small dense NLPs, but potentially large sparse IVPs. Simultaneous direct methods do not require the solution of an IVP, but require large sparse NLPs. Research into large-scale NLP solution methods is currently a very active research area but it is still very difficult to solve large NLPs from poor initial guesses. On the other hand, solution of large sparse IVPs has become a standard numerical mathematical tool, and it is much easier to provide good initial guesses for the small number of parameters in the master NLP.

- The developments of this thesis dramatically increases the efficiency of control parameterization through the staggered corrector algorithm (see Chapter 4) and develops novel methods to solve problems that include path constraints.

2.5 Optimality conditions for direct methods

The aim of direct methods is to transform the infinite-dimensional problem (2.147–2.149) into a finite dimensional problem. This is accomplished by constraining the control functions $u(t)$ to lie in a subspace of function space that is characterized by a finite number of parameters. In other words, the control trajectories are described by some function:

$$u = u(p, t) \tag{2.146}$$

where $p \in \mathbb{R}^{n_p}$ is a set of parameters. Hence, as shown in [63] the solution of (2.23–2.26) found by direct methods will be suboptimal to those found by an analytic solution to the necessary conditions (2.4–2.6).

The use of control parameterization turns the dynamic optimization problem into a parameter optimization problem:

$$\min_{p,t_f} J = \psi\left(x\left(p, t_f\right), t_f\right) + \int_{t_0}^{t_f} L\left(x(p,t), u(p,t), t\right) dt \tag{2.147}$$

subject to the DAE:

$$f\left(\dot{x}(p,t), x(p,t), u(p,t), t\right) = 0 \qquad \forall t \in [t_0, t_f] \tag{2.148}$$

$$\phi\left(\dot{x}\left(t_0\right), x\left(t_0\right), t_0\right) = 0 \tag{2.149}$$

which may be solved as a mathematical program.

The necessary optimality conditions for (2.147–2.149) may be derived using Lagrange multipliers [16]:

$$\frac{\partial J}{\partial \dot{x}}\frac{\partial \dot{x}}{\partial p} + \frac{\partial J}{\partial x}\frac{\partial x}{\partial p} + \frac{\partial J}{\partial u}\frac{\partial u}{\partial p} + \left[\lambda^T\left[\frac{\partial f}{\partial \dot{x}}\frac{\partial \dot{x}}{\partial p} + \frac{\partial f}{\partial x}\frac{\partial x}{\partial p} + \frac{\partial f}{\partial u}\frac{\partial u}{\partial p}\right]\right]_{\forall t \in [t_0, t_f]} = 0 \tag{2.150}$$

$$f\left(\dot{x}(p,t), x(p,t), u(p,t), t\right) = 0 \qquad \forall t \in [t_0, t_f] \tag{2.151}$$

$$\phi\left(\dot{x}\left(t_0\right), x\left(t_0\right), t_0\right) = 0 \tag{2.152}$$

Note that this problem is still infinite dimensional because (2.150–2.151) are enforced for $t \in [t_0, t_f]$.

The simultaneous direct method approximates (2.150–2.151) at discrete points in the interval $[t_0, t_f]$. Doing this requires that the state variables as well as the control variables are parameterized using:

$$x = x(\bar{p}, t) \tag{2.153}$$

$$\dot{x} = x'(\bar{p}, t) \tag{2.154}$$

These state variable approximations are then substituted into (2.150–2.151), and the result evaluated at a set of discrete points along the solution trajectory.

The sequential direct method creates a finite dimensional problem from (2.150–2.152) by transforming it into the following problem:

$$\frac{\partial J}{\partial \dot{x}}\frac{\partial \dot{x}}{\partial p} + \frac{\partial J}{\partial x}\frac{\partial x}{\partial p} + \frac{\partial J}{\partial u}\frac{\partial u}{\partial p} = 0 \tag{2.155}$$

$$\frac{\partial f}{\partial \dot{x}}\frac{\partial \dot{x}}{\partial p} + \frac{\partial f}{\partial x}\frac{\partial x}{\partial p} + \frac{\partial f}{\partial u}\frac{\partial u}{\partial p} = 0 \quad \forall t \in [t_0, t_f] \tag{2.156}$$

$$\left[\frac{\partial \phi}{\partial \dot{x}}\frac{\partial \dot{x}}{\partial p} + \frac{\partial \phi}{\partial x}\frac{\partial x}{\partial p}\right]_{t=t_0} = 0 \tag{2.157}$$

$$f\left(\dot{x}(p,t), x(p,t), u(p,t), t\right) = 0 \qquad \forall t \in [t_0, t_f] \tag{2.158}$$

$$\phi\left(\dot{x}\left(t_0\right), x\left(t_0\right), t_0\right) = 0 \tag{2.159}$$

An IVP solver is used to discretize (2.156–2.159).

A diagram illustrating the general algorithm for solving a dynamic optimization problem using the sequential direct method with an NLP solver is shown in Figure 2-1.

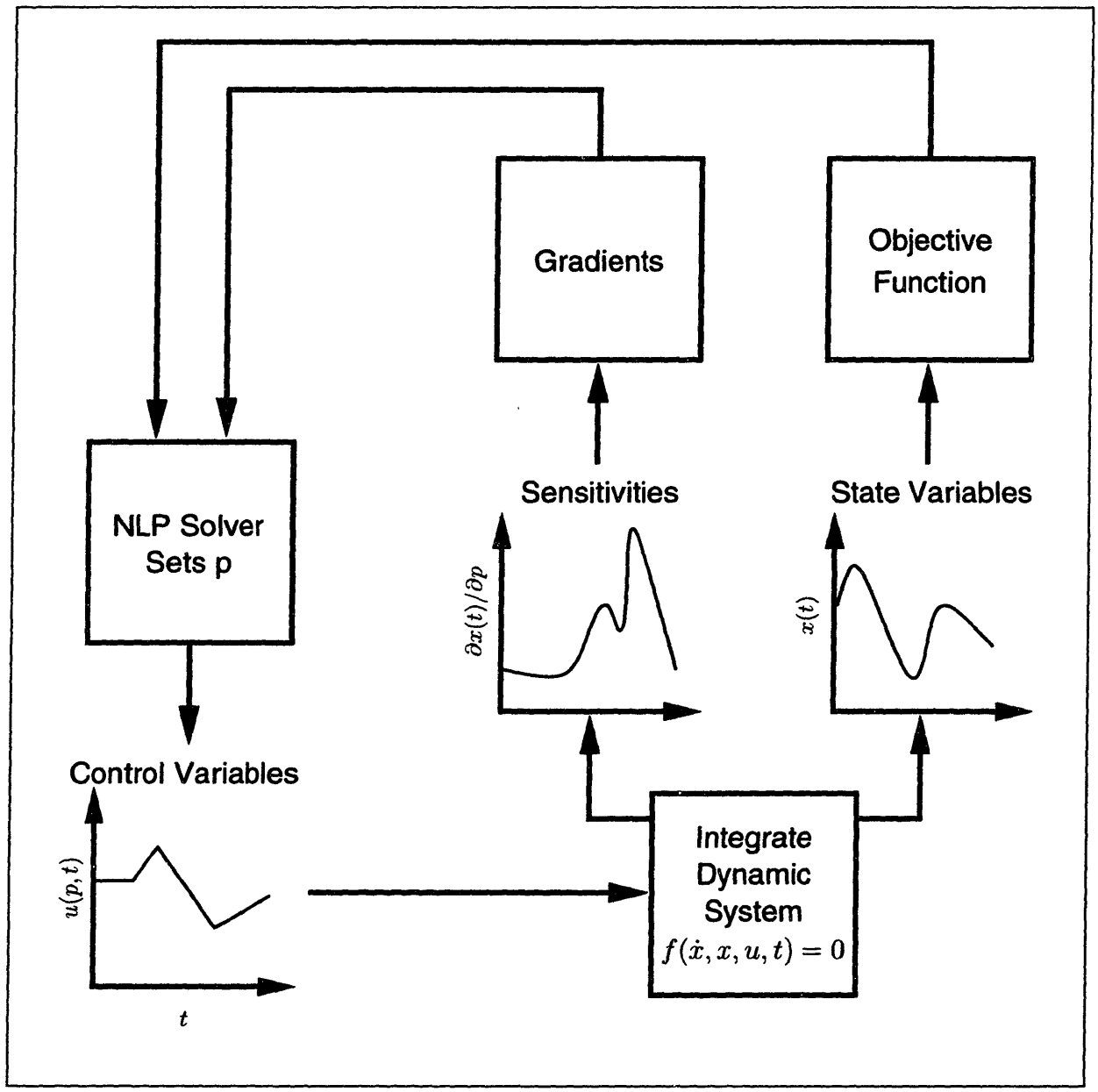

Figure 2-1: *The control parameterization algorithm*

2.6 A general control parameterization problem

The control parameterization problem (2.155–2.159) presented in the previous section was used to describe how the method works. This section presents a more general control parameterization formulation that allows the solution of a broad range of dynamic optimization problems.

2.6.1 Lagrange approximation of control variables

The control parameterization method approximates the control profiles using polynomials over a set of N_{FE} finite elements of length $h_i = t_i - t_{i-1}$, $i = 1 \dots N_{FE}$. For convenience in bounding the control profiles, the controls are parameterized using Lagrange polynomials. For control variable u_j in finite element i the Lagrange approximation is:

$$u_j^i(\tau^{(i)}) = \sum_{k=1}^{M+1} u_{ijk} \phi_k^{(M)}\left(\tau^{(i)}\right) \qquad \forall \tau^{(i)} \in [0,1] \tag{2.160}$$

$$i = 1 \dots N_{FE} \quad j = 1 \dots m_u \tag{2.161}$$

where:

$$\phi_k^{(M)}\left(\tau^{(i)}\right) = \begin{cases} 0 & \text{if } M = 0 \\ \prod_{\substack{l=1 \\ l \neq k}}^{M+1} \dfrac{\tau - \tau_l}{\tau_k - \tau_l} & \text{if } M \geq 0 \end{cases} \tag{2.162}$$

and $\tau^{(i)}$ is the normalized time in the finite element i:

$$\tau^{(i)}(t) = \frac{t - t_{i-1}}{t_i - t_{i-1}} \tag{2.163}$$

The main advantage of Lagrange polynomials is that

$$u_j^i(\tau^{(i)}) = u_{ijk} \qquad \text{if } \tau^{(i)}(t) = \tau_l \tag{2.164}$$

Therefore, bounding the control parameters u_{ijk} is sufficient to bound the control functions at the points τ_l. Bounding the profile at these points is sufficient to guarantee bounding of the entire control profile for $M \leq 1$. It is possible to bound control profiles with $M > 1$, but this requires the use of more complicated constraints that are functions of the parameters. In practice, however, bounding higher order polynomials can be done by ensuring that N_{FE} is large.

Theoretically, the choice of the set of points τ_l does not affect the solution because all polynomial approximations of the same order are equivalent. However, a poor choice for τ_l can result in an ill-conditioned approximation of the control function. Ideally, $\tau_1 = 0$ and, if $M \geq 0$, $\tau_{M+1} = 1$ so that the control may be more easily bounded. The other τ_l may be set at collocation points or at equidistant intervals. The 'best-conditioned' approximation is given by selecting collocation points, but in practice equidistant points are easier to use and do not seem to give a noticeable decrease in the performance of the algorithm.

2.6.2 The dynamic optimization problem

The goal here is not to present the most general formulation of the control parameterized problem, but rather to show a formulation that is general enough for the purposes of this thesis. Note that this formulation does not include path constraints on state variables, which are discussed in Chapter 6.

The control parameterization formulation used in this thesis is:

$$\min_{u_{ijk}, t_i} J\left(\dot{x}(u_{ijk}, t_i), x(u_{ijk}, t_i), u(u_{ijk}, t_i), v, t_f\right) \tag{2.165}$$

subject to:

Initial time point constraints:

$$\phi(\dot{x}(u_{111}), x(u_{111}), u(u_{111}), t_0) = 0 \tag{2.166}$$

Final time point constraints:

$$r^{eq}(\dot{x}(u_{ijk}, t_i), x(u_{ijk}, t_i), u(u_{ijk}, t_i), t_f) = 0 \quad r^{eq}(\cdot) \rightarrow \mathbb{R}^{n_{req}} \tag{2.167}$$

$$r^{ineq}(\dot{x}(u_{ijk}, t_i), x(u_{ijk}, t_i), u(u_{ijk}, t_i), t_f) \leq 0 \quad r^{ineq}(\cdot) \rightarrow \mathbb{R}^{n_{rineq}} \tag{2.168}$$

Control parameter bounds:

$$u_{ijk}^L \leq u_{ijk} \leq u_{ijk}^U \tag{2.169}$$

Time invariant bounds:

$$v^L \leq v \leq v^U \qquad v \in \mathbb{R}^{m_v} \tag{2.170}$$

Finite element size constraints:

$$h^{min} \leq t_i - t_{i-1} \leq h^{max} \tag{2.171}$$

$$t_{N_{FE}} = t_f \tag{2.172}$$

The values of the variables are obtained by solving the DAE:

$$f(\dot{x}(u_{ijk}, t_i), x(u_{ijk}, t_i), u(u_{ijk}, t_i, t), v, t) = 0 \tag{2.173}$$

subject to the initial conditions (2.166), the control parameterization (2.160).

Transferring the values of the state variables from one finite element to the next requires *junction conditions.* On the boundaries between finite elements, these junction

conditions have the form:

$$\begin{aligned}\Phi_i(&\dot{x}(u_{ijk}, t_{j\neq i}, t_i^-), x(u_{ijk}, t_{j\neq i}, t_i^-), u(u_{ijk}, t_{j\neq i}, t_i^-), t_i^-, \\ &\dot{x}(u_{ijk}, t_{j\neq i}, t_i^+), x(u_{ijk}, t_{j\neq i}, t_i^+), u(u_{ijk}, t_{j\neq i}, t_i^+), t_i^+) = 0\end{aligned} \tag{2.174}$$

where $\Phi(\cdot) \rightarrow \mathbb{R}^{r_d}$.

For most physically meaningful ODEs and index-1 DAEs the junction conditions (2.174) have the form

$$x_n^+ = x_n^- \qquad \forall n \in \Gamma \tag{2.175}$$

where Γ is an index set of the state variables which have time derivatives that explicitly appear in the DAE. However, for high-index DAEs or DAEs where the index fluctuates between boundaries equation (2.175) is not valid, and methods such as those described in [141] must be used to determine appropriate forms of the transfer conditions.

The control parameterization formulation presented above is simpler than the multi-stage formulation given in [142]. The multi-stage formulation allows the time domain to be divided into subdomains which have end times that may or may not coincide with finite element boundaries. The functional form of the DAE is permitted to change from one stage to the next. However, it is worth pointing out that the multi-stage formulation requires the stage sequence to be fixed, rather than be defined by implicit state events as in [15, 110]. The significance of this limitation is further discussed in Chapter 7.

2.6.3 Gradient evaluation

Efficient control parameterization requires the gradients of the objective functions and NLP constraints with respect to the optimization parameters. Evaluating these gradients requires values for the *sensitivity variables* $\frac{\partial \dot{x}}{\partial p}$ and $\frac{\partial x}{\partial p}$, where $p = \{u_{ijk}, t_i, v\}$.

There are three basic methods for obtaining the sensitivity variables: finite differences, adjoint equations, and direct solution of the sensitivity equation. Of these, direct solution of the sensitivity equation is the preferred method because of its accu-

racy and the high computational efficiency that may now be achieved when solving the combined DAE and sensitivity system [49, 94, 123, Chapter 4].

The sensitivity equations associated with (2.173) are obtained by differentiating (2.173) with respect to the optimization parameters p:

$$\frac{\partial f}{\partial \dot{x}}\frac{\partial \dot{x}}{\partial p} + \frac{\partial f}{\partial x}\frac{\partial x}{\partial p} = -\frac{\partial f}{\partial u}\frac{\partial u(p,t)}{\partial p} - \frac{\partial f}{\partial p}, \quad \forall t = [t_0, t_f] \tag{2.176}$$

which is in itself a DAE that has the initial conditions:

$$\frac{\partial \phi}{\partial \dot{x}}\frac{\partial \dot{x}}{\partial p} + \frac{\partial \phi}{\partial x}\frac{\partial x}{\partial p} = -\frac{\partial \phi}{\partial u}\frac{\partial u(p,t_0)}{\partial p} - \frac{\partial \phi}{\partial p}, \quad t = t_0 \tag{2.177}$$

There are also junction conditions for the sensitivity equation that are valid at the same times t_i as the junction conditions for the DAE, which are discussed in Chapter 3.

Solving (2.176–2.177) requires the partial derivatives of the control functions with respect to the parameters, which are [142]:

$$\frac{\partial u_j^i}{\partial u_{i'j'k}} = \phi_k^M(\tau^i)\delta_{ii'}\delta_{jj'} \quad k = 1 \ldots M+1 \tag{2.178}$$

$$\frac{\partial u_j^i}{\partial t_{i'}} = \begin{cases} -\dfrac{t_i - t}{(t_i - t_{i-1})^2}\displaystyle\sum_{k=1}^{M+1} u_{ijk}\frac{d\phi_k^{(M)}}{d\tau^i} & \text{for } i' = i-1,\ i > 1 \\ -\dfrac{t - t_{i-1}}{(t_i - t_{i-1})^2}\displaystyle\sum_{k=1}^{M+1} u_{ijk}\frac{d\phi_k^{(M)}}{d\tau^i} & \text{for } i' = i \\ 0 & \text{otherwise} \end{cases} \tag{2.179}$$

2.7 ABACUSS Dynamic Optimization Input Language

State variable path-constrained optimization has been implemented within the ABACUSS[1] large-scale equation-oriented modeling system. The input language in ABACUSS has been extended to permit the representation of large-scale path-constrained dynamic optimization systems in a compact manner.

Examples of the input language are given in Figure 2-2 and Appendix A. Variable types are declared in the DECLARE block. The equations describing the DAE are given in the MODEL block. The OPTIMIZATION block is used to define a dynamic optimization using the MODEL. Note that it is possible to move from a simulation to an optimization in a seamless manner.

The sections in the optimization block are:

PARAMETER Defines the parameters that are used in the dynamic optimization that are not defined in the model.

UNIT Defines the MODELs to use in the current dynamic optimization.

VARIABLE Defines variables that are used in the dynamic optimization that are not defined in the models.

OBJECTIVE Defines the objective function to be MINIMIZEd or MAXIMIZEd in terms of variables defined in the UNIT or the VARIABLE section.

SET Sets values for any parameters defined in the UNIT or PARAMETER section

INEQUALITY Defines any path inequalities in terms of variables defined in the UNIT or the VARIABLE section.

CONTROL Indicates that the listed variables are control variables. The first function in the triple notation defines the initial guess, the second and third define the lower and upper bounds for the control, respectively.

TIME_INVARIANT Indicates that the listed variable are time invariant optimization parameters. The first number in the triple notation defines the initial guess,

[1]ABACUSS (Advanced Batch And Continuous Unsteady-State Simulator) process modeling software, a derivative work of gPROMS software, ©1992 by Imperial College of Science, Technology, and Medicine.

```
DECLARE
  TYPE
    NoType = 0.0 : -1E9 : 1E9

END

MODEL brachistochrone

PARAMETER
  G                AS REAL

VARIABLE
  X                AS NoType
  Y                AS NoType
  V                AS NoType
  Theta            AS NoType
  Constr           AS NoType

EQUATION

 $X = V*COS(Theta) ;
 $Y = V*SIN(Theta) ;
 $V = G*SIN(Theta) ;
 Constr = -0.40*X-0.30 ;

END # brachistochrone

OPTIMIZATION Brach

  PARAMETER
    B                    AS REAL
    A                    AS REAL

  UNIT
    kraft AS brachistochrone

  VARIABLE
    Final_Time AS NoType

  OBJECTIVE
    MINIMIZE Final_Time ;

  SET
    WITHIN kraft DO
      G     := -1.0 ;
    END
    A       := -0.40;
    B       :=  0.30;

  INEQUALITY
    WITHIN kraft DO
      Y>=A*X-B ;
    END

  CONTROL
    WITHIN kraft DO
     Theta := -1.6+1.6/0.7*TIME: -1.6: 0.0 ;
    END

  TIME_INVARIANT
    Final_Time := 0.68 : 0.0 : 2.0 ;

  INITIAL
    WITHIN kraft DO
      X=0.0 ;
      Y=0.0 ;
      V=0.0;
    END

  FINAL
     WITHIN kraft DO
       X=1.1 ;
     END

  SCHEDULE
    CONTINUE FOR Final_Time

END # Brach
```

Figure 2-2: *Example of ABACUSS input file for constrained dynamic optimization*

the second and third define the lower and upper bounds for the variable, respectively.

INITIAL Gives additional equations that define the initial condition for the problem, possibly as a function of the optimization parameters.

FINAL Defines point constraints that must be obeyed at the final time.

SCHEDULE Indicates the time domain of the dynamic optimization problem.

Chapter 3

Parametric Sensitivity Functions for Hybrid Discrete/Continuous Systems

This chapter is a summary of a paper [55] of the same title that was coauthored with Santos Galán. The results of this paper are included here because the ability to handle sensitivity functions for hybrid discrete/continuous systems is necessary for the development of the methods to handle inequality path-constrained dynamic optimization problems discussed in Chapter 7.

Parametric sensitivity analysis is the study of the influence of variations in the parameters of a model on its solution. It plays an important role in design and modeling, is used for parameter estimation and optimization, and is extensively applied in the synthesis and analysis of control systems.

The variations of the parameters can be differential or finite. Here only the former case is considered, which approximates linearly the variation of the solution. Two kinds of sensitivity analysis can be distinguished, depending on whether the variation is finite-dimensional (lumped) or it is a function (distributed). In the former case, which is the only one considered in this chapter, the sensitivities are ordinary partial derivatives. In the second, as in structural systems, functional derivatives are needed.

In process design, sensitivity information provides an elucidation of the influence

of design changes, without requiring trial and error. In modeling, sensitivity analysis reveals the relative importance of every parameter, giving arguments for simplifying the model or guiding new experiments. Sensitivities are also used in parameter estimation for error analysis, and in gradient computation for dynamic optimization.

There has been great interest in the application of sensitivity information in control system design. Sensitivity analysis is necessary because the parameters are subject to inaccuracy in data, models and implementation, and because they can deviate with time. Furthermore, sensitivity analysis is essential for adaptive systems. Early reviews in this area are given in [83, 136].

Here, the term "hybrid" will refer to the combined existence and interaction of continuous and discrete phenomena. The continuous part is usually modeled by differential-algebraic equations (DAEs) and the discrete behavior by finite automata. A precise definition of the systems considered in this chapter follows in Section 3.1. Hybrid systems pose a problem for the calculation of sensitivities because the sensitivities are not defined in general when changing from one continuous subsystem to another.

The study of the sensitivity of control systems experienced intense development in the late 1960s, mostly in Eastern Europe. Some authors have dealt with the subject of sensitivities in discontinuous systems, which are frequent in control systems. Tsypkin claims in a short note [138] to have obtained variational equations for relay systems in 1955. Using an obscure variational notation, De Backer [35] derives the "jump conditions" for the sensitivities with respect to the initial conditions of the initial value problem (IVP) in ordinary differential equations (ODEs) of an autonomous system that changes to a second vector field at a time when a state event (*i.e.*, an event which is a function only of the state variables) is satisfied. With the intention of extending system identification to systems with discontinuities, Russ [125] rediscovers De Backer's result. While Russ broadened De Backer's result to cover nonautonomous systems and state events that are functions of time and the parameters, there are several drawbacks to this methodology: the derivation is unnecessarily complex, the representation is inconvenient, and hybrid phenomena are not well understood, as

evidenced by the imposition of an obviously incorrect anullation of components of the gradient of the discontinuity function at the switching times.

In a concise paper, Rozenvasser [124] derived the sensitivity functions of discontinuous systems modeled by a given sequence of explicit ODE vector fields with explicit or implicit switching times. Compared to the above works, it seems that the derivation is a relatively straightforward calculus exercise. This is the most general result we know for sensitivities of hybrid systems and ironically, until now, has been entirely neglected in the subsequent literature. In fact, we derived Rozenvasser's results independently, only subsequently stumbling upon [124] by chance.

This chapter extends Rozenvasser's results in several ways. First, the discrete aspects of the system model are significantly generalized in line with modern notions of hybrid systems. Second, the results are extended to include DAE embedded hybrid systems. Existence and uniqueness theorems are also presented for the sensitivity functions of hybrid systems. These theorems shed light on the issue of sequencing of state events in hybrid systems. Numerical results are given for the calculation of sensitivity functions for hybrid systems.

3.1 Mathematical Representation of Hybrid Systems

A formalism derived from [6] and [15] is used to model a broad class of hybrid discrete/continuous phenomena. Consider a system described by a state space $S = \bigcup_{k=1}^{n_k} S_k$ where each *mode* S_k which is characterized by:

1. A set of variables $\{\dot{x}^{(k)}, x^{(k)}, y^{(k)}, u^{(k)}, p, t\}$, where $x^{(k)}(p,t) \in \mathbb{R}^{n_x^{(k)}}$ are the differential state variables, $y^{(k)}(p,t) \in \mathbb{R}^{n_y^{(k)}}$ the algebraic state variables and $u^{(k)}(p,t) \in \mathbb{R}^{n_u^{(k)}}$ the controls. The time invariant parameters $p \in \mathbb{R}^{n_p}$ and time t are the independent variables, and the controls u are explicit functions of both the parameters and time.

2. A set of equations $f^{(k)}(\dot{x}^{(k)}, x^{(k)}, y^{(k)}, u^{(k)}, p, t) = 0$, usually a coupled system of differential and algebraic equations, $f^{(k)} : \mathbb{R}^{n_x^{(k)}} \times \mathbb{R}^{n_x^{(k)}} \times \mathbb{R}^{n_y^{(k)}} \times \mathbb{R}^{n_u^{(k)}} \times \mathbb{R}^{n_p} \times \mathbb{R} \rightarrow \mathbb{R}^{n_x^{(k)}+n_y^{(k)}}$. In the mode S_k the specification of the parameters p (and, consequently, the controls) coupled with a consistent initial condition $T_k(\dot{x}^{(k)}, x^{(k)}, y^{(k)}, u^{(k)}, p, t) = 0$ at $t = t_0^{(k)}$ determines the evolution of the system in $[t_0^{(k)}, t_f^{(k)})$.

3. A (possibly empty) set of transitions to other modes. The set of modes S_j where a transition from mode S_k is possible is $J^{(k)}$. These transitions are described by:

   (a) Transition conditions $L_j^{(k)}(\dot{x}^{(k)}, x^{(k)}, y^{(k)}, u^{(k)}, p, t)$, $j \in J^{(k)}$, determining the transition times at which switching from mode k to mode j occurs. The transition conditions are represented by logical propositions that trigger the switching when they become true. They are described in section 3.3. Note that discontinuities in the controls are included here.

   (b) Sets of transition functions:

$$T_j^{(k)}(\dot{x}^{(k)}, x^{(k)}, y^{(k)}, u^{(k)}, \dot{x}^{(k+1)}, x^{(k+1)}, y^{(k+1)}, u^{(k+1)}, p, t)$$

are associated with the transition conditions which relate the variables in the current mode S_k and the variables in the new mode S_j at the transition time $t_f^{(k)}$. A special case of the transition functions is the set of initial conditions for the initial mode S_1. These initial conditions will be designated by $T_0^{(1)}$.

3.2 Sensitivities

The partial derivatives of the variables with respect to the parameters are known as *sensitivity functions*. Before discussing the sensitivities, the solution of the hybrid system in isolation must be examined in more detail.

3.2.1 Consistent initialization

At $t_0^{(1)}$, the following system exists:

$$f^{(1)}(\dot{x}^{(1)}, x^{(1)}, y^{(1)}, u^{(1)}, p, t) = 0 \tag{3.1}$$

$$T_0^{(1)}(\dot{x}^{(1)}, x^{(1)}, y^{(1)}, u^{(1)}, p, t) = 0 \tag{3.2}$$

$$t = t_0^{(1)} \tag{3.3}$$

where $T_0^{(1)} : \mathbb{R}^{n_x^{(1)}} \times \mathbb{R}^{n_x^{(1)}} \times \mathbb{R}^{n_y^{(1)}} \times \mathbb{R}^{n_u^{(1)}} \times \mathbb{R}^{n_p} \times \mathbb{R} \rightarrow \mathbb{R}^{n_x^{(1)}}$. For the purposes of this chapter it is assumed that:

$$\operatorname{rank}\left(\begin{bmatrix} \frac{\partial f^{(k)}}{\partial \dot{x}^{(k)}} & \frac{\partial f^{(k)}}{\partial y^{(k)}} \end{bmatrix}\right) = n_x^{(k)} + n_y^{(k)} \tag{3.4}$$

everywhere. This condition holds for all index 0 systems (ODEs) and most index 1 systems, and implies that $n_x^{(1)}$ side conditions $T_0^{(1)}$ are required to define uniquely a consistent initial condition [15]. Chapter 5 details methods for deriving an equivalent index-1 DAE from a high-index DAE that work for a broad class of high-index DAEs.

Equation (3.3) is usual at initialization, but in general the initial time can be an implicit or explicit function of other parameters. Actually, $t_0^{(1)}$ is a parameter. The more general case is considered below when dealing with the sensitivities at transitions.

The number of equations $n_x^{(1)} + n_x^{(1)} + n_y^{(1)} + 1$ and the number of variables $n_x^{(1)} + n_x^{(1)} + n_y^{(1)} + n_p + 2$ (u are explicit functions of p and t) admit in general $n_p + 1$ degrees of freedom for specification of the independent variables p and $t_0^{(1)}$.

A sufficient (implicit function theorem) and practical (numerical solution by a

Newton method) condition for the solvability of the system (3.1–3.3) is that the matrix:

$$\begin{bmatrix} \frac{\partial f^{(1)}}{\partial \dot{x}^{(1)}} & \frac{\partial f^{(1)}}{\partial x^{(1)}} & \frac{\partial f^{(1)}}{\partial y^{(1)}} & \frac{\partial f^{(1)}}{\partial t} \\ \frac{\partial T_0^{(1)}}{\partial \dot{x}^{(1)}} & \frac{\partial T_0^{(1)}}{\partial x^{(1)}} & \frac{\partial T_0^{(1)}}{\partial y^{(1)}} & \frac{\partial T_0^{(1)}}{\partial t} \\ 0 & 0 & 0 & 1 \end{bmatrix} \Longleftrightarrow \begin{bmatrix} \frac{\partial f^{(1)}}{\partial \dot{x}^{(1)}} & \frac{\partial f^{(1)}}{\partial x^{(1)}} & \frac{\partial f^{(1)}}{\partial y^{(1)}} \\ \frac{\partial T_0^{(1)}}{\partial \dot{x}^{(1)}} & \frac{\partial T_0^{(1)}}{\partial x^{(1)}} & \frac{\partial T_0^{(1)}}{\partial y^{(1)}} \end{bmatrix}$$

is nonsingular.

The solution of (3.1), (3.2) and (3.3) is represented as:

$$x_0^{(1)}(p, t_0^{(1)}), \quad \dot{x}_0^{(1)}(p, t_0^{(1)}), \quad y_0^{(1)}(p, t_0^{(1)}), \quad t_0^{(1)} \tag{3.5}$$

and let:

$$x^{(1)}(p, t_0^{(1)}, t), \quad y^{(1)}(p, t_0^{(1)}, t) \tag{3.6}$$

be the solutions of $f^{(1)}$ that satisfy (3.1), (3.2) and (3.3). These solutions are functions of t that pass through the point (3.5). The following relations can be derived:

$$\dot{x}^{(1)}(p, t_0^{(1)}, t) = \frac{\partial x^{(1)}(p, t_0^{(1)}, t)}{\partial t} \tag{3.7}$$

$$x^{(1)}(p, t_0^{(1)}, t_0^{(1)}) = x_0^{(1)}(p, t_0^{(1)}) \tag{3.8}$$

$$y^{(1)}(p, t_0^{(1)}, t_0^{(1)}) = y_0^{(1)}(p, t_0^{(1)}) \tag{3.9}$$

$$\dot{x}^{(1)}(p, t_0^{(1)}, t_0^{(1)}) = \left.\frac{\partial x^{(1)}(p, t_0^{(1)}, t)}{\partial t}\right|_{t=t_0^{(1)}} = \dot{x}_0^{(1)}(p, t_0^{(1)}) \tag{3.10}$$

3.2.2 Initial sensitivities

Now consider sensitivity functions of the above system. Differentiating the system used for consistent initialization and applying the chain rule yields:

$$\begin{bmatrix} \frac{\partial f^{(1)}}{\partial \dot{x}^{(1)}} & \frac{\partial f^{(1)}}{\partial x^{(1)}} & \frac{\partial f^{(1)}}{\partial y^{(1)}} & \frac{\partial f^{(1)}}{\partial u^{(1)}} & \frac{\partial f^{(1)}}{\partial t} & \frac{\partial f^{(1)}}{\partial p} & \frac{\partial f^{(1)}}{\partial t_0^{(1)}} \\ \frac{\partial T_0^{(1)}}{\partial \dot{x}^{(1)}} & \frac{\partial T_0^{(1)}}{\partial x^{(1)}} & \frac{\partial T_0^{(1)}}{\partial y^{(1)}} & \frac{\partial T_0^{(1)}}{\partial u^{(1)}} & \frac{\partial T_0^{(1)}}{\partial t} & \frac{\partial T_0^{(1)}}{\partial p} & \frac{\partial T_0^{(1)}}{\partial t_0^{(1)}} \\ 0 & 0 & 0 & 0 & 1 & 0 & -1 \end{bmatrix} \begin{bmatrix} \frac{\partial \dot{x}_0^{(1)}}{\partial p} & \frac{\partial \dot{x}_0^{(1)}}{\partial t_0^{(1)}} \\ \frac{\partial x_0^{(1)}}{\partial p} & \frac{\partial x_0^{(1)}}{\partial t_0^{(1)}} \\ \frac{\partial y_0^{(1)}}{\partial p} & \frac{\partial y_0^{(1)}}{\partial t_0^{(1)}} \\ \frac{\partial u^{(1)}}{\partial p} & \frac{\partial u^{(1)}}{\partial t_0^{(1)}} \\ \frac{\partial t}{\partial p} & \frac{\partial t}{\partial t_0^{(1)}} \\ I & 0 \\ 0 & 1 \end{bmatrix} = 0 \tag{3.11}$$

From the last row:

$$\frac{\partial t}{\partial p} = 0 \quad \frac{\partial t}{\partial t_0^{(1)}} = 1 \tag{3.12}$$

Since $\frac{\partial f^{(1)}}{\partial t_0^{(1)}} = 0$ and $\frac{\partial T_0^{(1)}}{\partial t_0^{(1)}} = 0$, the system is:

$$\begin{bmatrix} \frac{\partial f^{(1)}}{\partial \dot{x}^{(1)}} & \frac{\partial f^{(1)}}{\partial x^{(1)}} & \frac{\partial f^{(1)}}{\partial y^{(1)}} \\ \frac{\partial T_0^{(1)}}{\partial \dot{x}^{(1)}} & \frac{\partial T_0^{(1)}}{\partial x^{(1)}} & \frac{\partial T_0^{(1)}}{\partial y^{(1)}} \end{bmatrix} \begin{bmatrix} \frac{\partial \dot{x}_0^{(1)}}{\partial p} & \frac{\partial \dot{x}_0^{(1)}}{\partial t_0^{(1)}} \\ \frac{\partial x_0^{(1)}}{\partial p} & \frac{\partial x_0^{(1)}}{\partial t_0^{(1)}} \\ \frac{\partial y_0^{(1)}}{\partial p} & \frac{\partial y_0^{(1)}}{\partial t_0^{(1)}} \end{bmatrix} = - \begin{bmatrix} \frac{\partial f^{(1)}}{\partial u^{(1)}} \frac{\partial u^{(1)}}{\partial p} + \frac{\partial f^{(1)}}{\partial p} & \frac{\partial f^{(1)}}{\partial u^{(1)}} \frac{\partial u^{(1)}}{\partial t} + \frac{\partial f^{(1)}}{\partial t} \\ \frac{\partial T_0^{(1)}}{\partial u^{(1)}} \frac{\partial u^{(1)}}{\partial p} + \frac{\partial T_0^{(1)}}{\partial p} & \frac{\partial T_0^{(1)}}{\partial u^{(1)}} \frac{\partial u^{(1)}}{\partial t} + \frac{\partial T_0^{(1)}}{\partial t} \end{bmatrix} \tag{3.13}$$

Note that the sensitivities of the solution of the consistent initialization problem (3.5) have been defined. The initial sensitivities of the dynamic problem must also be determined, which may or may not be the same. For the parameters p the initial

sensitivities are:

$$\frac{\partial x_0^{(1)}(p,t_0^{(1)})}{\partial p} = \frac{\partial x^{(1)}(p,t_0^{(1)},t_0^{(1)})}{\partial p} = \left.\frac{\partial x^{(1)}(p,t_0^{(1)},t)}{\partial p}\right|_{t=t_0^{(1)}} \tag{3.14}$$

$$\frac{\partial y_0^{(1)}(p,t_0^{(1)})}{\partial p} = \frac{\partial y^{(1)}(p,t_0^{(1)},t_0^{(1)})}{\partial p} = \left.\frac{\partial y^{(1)}(p,t_0^{(1)},t)}{\partial p}\right|_{t=t_0^{(1)}} \tag{3.15}$$

$$\frac{\partial \dot{x}_0^{(1)}(p,t_0^{(1)})}{\partial p} = \frac{\partial \dot{x}^{(1)}(p,t_0^{(1)},t_0^{(1)})}{\partial p} = \left.\frac{\partial \dot{x}^{(1)}(p,t_0^{(1)},t)}{\partial p}\right|_{t=t_0^{(1)}} \tag{3.16}$$

but for the sensitivities with respect to the initial time:

$$\frac{\partial x_0^{(1)}(p,t_0^{(1)})}{\partial t_0^{(1)}} = \frac{\partial x^{(1)}(p,t_0^{(1)},t_0^{(1)})}{\partial t_0^{(1)}} = \left.\frac{\partial x^{(1)}(p,t_0^{(1)},t)}{\partial t}\right|_{t=t_0^{(1)}} + \left.\frac{\partial x^{(1)}(p,t_0^{(1)},t)}{\partial t_0^{(1)}}\right|_{t=t_0^{(1)}} = \dot{x}^{(1)}(p,t_0^{(1)},t_0^{(1)}) + \left.\frac{\partial x^{(1)}(p,t_0^{(1)},t)}{\partial t_0^{(1)}}\right|_{t=t_0^{(1)}} \tag{3.17}$$

Therefore

$$\left.\frac{\partial x^{(1)}(p,t_0^{(1)},t)}{\partial t_0^{(1)}}\right|_{t=t_0^{(1)}} = \frac{\partial x_0^{(1)}(p,t_0^{(1)})}{\partial t_0^{(1)}} - \dot{x}^{(1)}(p,t_0^{(1)},t_0^{(1)}) \tag{3.18}$$

Similarly,

$$\left.\frac{\partial y^{(1)}(p,t_0^{(1)},t)}{\partial t_0^{(1)}}\right|_{t=t_0^{(1)}} = \frac{\partial y_0^{(1)}(p,t_0^{(1)})}{\partial t_0^{(1)}} - \dot{y}^{(1)}(p,t_0^{(1)},t_0^{(1)}) \tag{3.19}$$

$$\left.\frac{\partial \dot{x}^{(1)}(p,t_0^{(1)},t)}{\partial t_0^{(1)}}\right|_{t=t_0^{(1)}} = \frac{\partial \dot{x}_0^{(1)}(p,t_0^{(1)})}{\partial t_0^{(1)}} - \ddot{x}^{(1)}(p,t_0^{(1)},t_0^{(1)}) \tag{3.20}$$

Hence, $\dot{y}^{(1)}$ and $\ddot{x}^{(1)}$ must be known in order to determine the initial sensitivities. The next section shows how $\dot{y}$ and $\ddot{x}$ can be calculated at any time.

3.2.3 Sensitivity trajectories

From a practical point of view, the trajectory of the variables for $t \in (t_0^{(k)}, t_f^{(k)})$ can be computed by numerical integration of the system $f^{(k)} = 0$ from the initial values. Hereafter p will include *all* the parameters (including $t_0^{(1)}$).

Differentiating the system with respect to the independent variables p and t gives:

$$\begin{bmatrix} \frac{\partial f^{(k)}}{\partial \dot{x}^{(k)}} & \frac{\partial f^{(k)}}{\partial x^{(k)}} & \frac{\partial f^{(k)}}{\partial y^{(k)}} \end{bmatrix} \begin{bmatrix} \frac{\partial \dot{x}^{(k)}}{\partial p} & \frac{\partial \dot{x}^{(k)}}{\partial t} \\ \frac{\partial x^{(k)}}{\partial p} & \frac{\partial x^{(k)}}{\partial t} \\ \frac{\partial y^{(k)}}{\partial p} & \frac{\partial y^{(k)}}{\partial t} \end{bmatrix} = - \begin{bmatrix} \frac{\partial f^{(k)}}{\partial u^{(k)}} \frac{\partial u^{(k)}}{\partial p} + \frac{\partial f^{(k)}}{\partial p} & \frac{\partial f^{(k)}}{\partial u^{(k)}} \frac{\partial u^{(k)}}{\partial t} + \frac{\partial f^{(k)}}{\partial t} \end{bmatrix} \quad (3.21)$$

Then, representing the matrix of sensitivities by s, under conditions of continuity of the time derivative of the state variable sensitivities and independence between t and p, define:

$$s_x^{(k)} = \frac{\partial x^{(k)}}{\partial p} \quad (3.22)$$

$$\dot{s}_x^{(k)} = \frac{\partial s_x^{(k)}}{\partial t} = \frac{\partial}{\partial t}(\frac{\partial x^{(k)}}{\partial p}) = \frac{\partial}{\partial p}(\frac{\partial x^{(k)}}{\partial t}) = \frac{\partial \dot{x}^{(k)}}{\partial p} \quad (3.23)$$

$$s_y^{(k)} = \frac{\partial y^{(k)}}{\partial p} \quad (3.24)$$

Sensitivity trajectories are determined by integrating the differential linear time varying system:

$$\begin{bmatrix} \frac{\partial f^{(k)}}{\partial \dot{x}^{(k)}} & \frac{\partial f^{(k)}}{\partial x^{(k)}} & \frac{\partial f^{(k)}}{\partial y^{(k)}} \end{bmatrix} \begin{bmatrix} \dot{s}_x^{(k)} \\ s_x^{(k)} \\ s_y^{(k)} \end{bmatrix} = - \begin{bmatrix} \frac{\partial f^{(k)}}{\partial u^{(k)}} \frac{\partial u^{(k)}}{\partial p} + \frac{\partial f^{(k)}}{\partial p} \end{bmatrix} \quad (3.25)$$

from the initial sensitivities.

The derivative with respect to time is a linear but not differential system ($i.e. \dot{x}^{(k)}$ is known) that allows us to obtain the the values of $\dot{y}^{(k)}$ and $\ddot{x}^{(k)}$ required in the

previous section and below:

$$\begin{bmatrix} \frac{\partial f^{(k)}}{\partial \dot{x}^{(k)}} & \frac{\partial f^{(k)}}{\partial y^{(k)}} \end{bmatrix} \begin{bmatrix} \ddot{x}^{(k)} \\ \dot{y}^{(k)} \end{bmatrix} = - \left[\frac{\partial f^{(k)}}{\partial x^{(k)}} \dot{x}^{(k)} + \frac{\partial f^{(k)}}{\partial u^{(k)}} \frac{\partial u^{(k)}}{\partial t} + \frac{\partial f^{(k)}}{\partial t} \right] \tag{3.26}$$

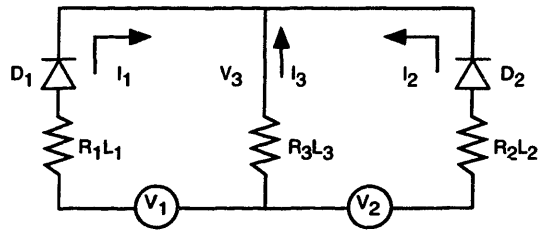

Figure 3-1: *Rectifier circuit*

3.3 Transitions

The transition conditions $L_j^{(k)}(\dot{x}^{(k)}, x^{(k)}, y^{(k)}, u^{(k)}, p, t)$, $j \in J^{(k)}$, are formed by logical propositions that contain logical operators (*e.g.*NOT, AND, OR) connecting atomic propositions (*i.e.*relational expressions) composed of valid real expressions and the relational operators $\{>, <, \leq, \geq\}$. For example, in the rectifier circuit of Figure 3-1 [31], the condition for both diodes D_1 and D_2 to be conductive requires the following logical proposition to be true:

$$[(i_1 > 0) \vee (v_1 > v_3)] \wedge [(i_2 > 0) \vee (v_2 > v_3)] \tag{3.27}$$

Associated with every relational expression is a discontinuity function:

$$g_{ji}^{(k)}(\dot{x}^{(k)}, x^{(k)}, y^{(k)}, u^{(k)}, p, t) \quad i = 1, \ldots, n_j^{(k)}$$

formed by the difference between the two real expressions. For example, in the rectifier system:

$$g_{2,4}^{(1)} = v_2 - v_3 \tag{3.28}$$

Each relational expression changes its value whenever its corresponding discontinuity function crosses zero. The transition conditions neutralize the only degree of freedom (time), determining $t_f^{(k)}$. The set $L^{(k)}$ of all transition conditions for the mode S_k defines the boundary of S_k that, when intercepted by the trajectory, triggers the switching to a new mode. Notice that the system can be in the same mode after the

application of a transition function.

Several issues arise in connection with the boundary created by the transition conditions that are beyond the scope of this chapter. Some of them are discussed in [69]. Here smoothness is assumed in the neighborhood of the transition time and that only one relational expression activates at that moment. Hereafter, $g_j^{(k)}$ will be used to designate the discontinuity function that actually determines the transition from mode S_k to S_j.

3.3.1 Time of the event

In practice, the satisfaction of the transition conditions (events) is monitored during numerical integration so that the smallest $t_f^{(k)}$ for all the possible transitions in the current mode, which establishes the actual transition, is easily found. An extensive treatment of this subject appears in [110].

Figure 3-2 illustrates different classes of events. The simplest condition is that of the controlled transitions (Π_1), where only independent variables (necessarily time, as it is the only degree of freedom) appear. This case is known as a *time event.* Traditionally, if the transition functions involve discontinuities in the state variables, then the transitions are called *controlled jumps*, usually reverting to the same mode. If there is a change of mode this is termed *controlled switching.* Controlled switching is the case of discontinuities in the controls.

Autonomous transitions are those whose conditions include the dependent variables. The satisfaction of these conditions is called a *state event.* A further distinction is whether time appears explicitly (Π_3) or not (Π_2). The same distinction between switchings and jumps can be made.

The evolution of a hybrid system can be viewed as a sequence of subdomains of the time interval, where each subdomain is characterized by a continuous evolution, an event, and a transition to a new mode (which can be the same). For convenience, the use of the superscript (k) and the subscript $(k+1)$ will indicate, respectively, the mode in which the event occurred and the subsequent mode which is determined by the condition.

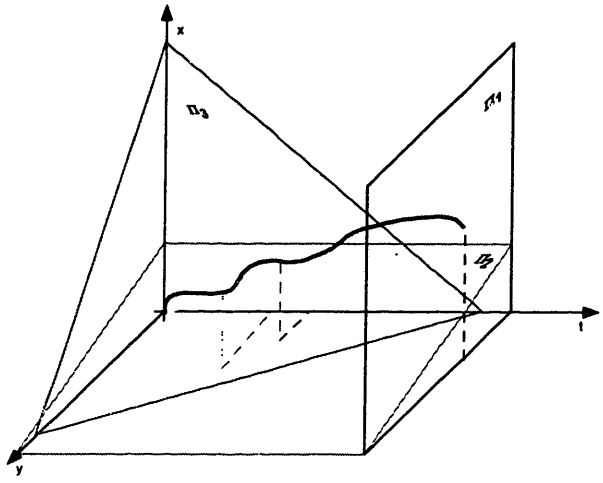

Figure 3-2: *Controlled and autonomous transitions*

The transfer to the new mode $k+1$ is described by the transition function $T_{k+1}^{(k)}$. At the event following system exists:

$$\dot{x}^{(k)} = \dot{x}^{(k)}(p,t) \qquad x^{(k)} = x^{(k)}(p,t) \qquad y^{(k)} = y^{(k)}(p,t) \tag{3.29}$$

$$g_{k+1}^{(k)}(\dot{x}^{(k)}, x^{(k)}, y^{(k)}, u^{(k)}, p, t) = 0 \tag{3.30}$$

$$T_{k+1}^{(k)}(\dot{x}^{(k)}, x^{(k)}, y^{(k)}, u^{(k)}, \dot{x}^{(k+1)}, x^{(k+1)}, y^{(k+1)}, u^{(k+1)}, p, t) = 0 \tag{3.31}$$

$$f^{(k+1)}(\dot{x}^{(k+1)}, x^{(k+1)}, y^{(k+1)}, u^{(k+1)}, p, t) = 0 \tag{3.32}$$

This system may have multiple solutions, but for simplicity it is assumed that criteria are given that define a unique solution (for example, limiting the domain of the transition function).

The equations (3.29) represent the state trajectory resulting from the integration of $f^{(k)}$ in mode k. In practice they are calculated numerically. The system naturally decomposes into two structural blocks that can be solved sequentially. The first two equations determine $t = t_f^{(k)} = t_0^{(k+1)}$ (*i.e.*, time is a dependent variable), and the corresponding values for $\dot{x}^{(k)}$, $x^{(k)}$, $y^{(k)}$ and $u^{(k)}$. The last two equations allows calculation of the initial values of $\dot{x}^{(k+1)}$, $x^{(k+1)}$, $y^{(k+1)}$, and $u^{(k+1)}$ in the new mode.

Again, a sufficient and practical condition for solving the system formed by (3.29)

and (3.30) is the nonsingularity of the Jacobian:

$$\begin{bmatrix} 1 & 0 & 0 & -\frac{\partial \dot{x}^{(k)}}{\partial t} \\ 0 & 1 & 0 & -\frac{\partial x^{(k)}}{\partial t} \\ 0 & 0 & 1 & -\frac{\partial y^{(k)}}{\partial t} \\ \frac{\partial g_{k+1}^{(k)}}{\partial \dot{x}^{(k)}} & \frac{\partial g_{k+1}^{(k)}}{\partial x^{(k)}} & \frac{\partial g_{k+1}^{(k)}}{\partial y^{(k)}} & \frac{\partial g_{k+1}^{(k)}}{\partial u^{(k)}} \frac{\partial u^{(k)}}{\partial t} + \frac{\partial g_{k+1}^{(k)}}{\partial t} \end{bmatrix}$$

That is, the following determinant is not zero:

$$\frac{\partial g_{k+1}^{(k)}}{\partial \dot{x}^{(k)}} \frac{\partial \dot{x}^{(k)}}{\partial t} + \frac{\partial g_{k+1}^{(k)}}{\partial x^{(k)}} \frac{\partial x^{(k)}}{\partial t} + \frac{\partial g_{k+1}^{(k)}}{\partial y^{(k)}} \frac{\partial y^{(k)}}{\partial t} + \frac{\partial g_{k+1}^{(k)}}{\partial u^{(k)}} \frac{\partial u^{(k)}}{\partial t} + \frac{\partial g_{k+1}^{(k)}}{\partial t} \neq 0 \qquad (3.33)$$

The derivatives must exist and the trajectory should not be tangent to $g_{k+1}^{(k)}$ in the neighborhood of $t_f^{(k)}$. If the trajectory is tangent to $g_{k+1}^{(k)}$ in the neighborhood of $t_f^{(k)}$, small variations of the parameters may not lead to a unique solution for the time of the transition. As shown later in this chapter, the points where (3.33) is not satisfied play an important role for hybrid systems.

3.3.2 Reinitialization at the transition

The initialization of the new mode is a problem similar to the one described in Section 3.2.1. The same assumptions for the transition functions T are made, providing a condition for well-posed transitions. Again, the nonsingularity condition applies:

$$\text{rank}\left(\begin{bmatrix} \frac{\partial T_{k+1}^{(k)}}{\partial \dot{x}^{(k+1)}} & \frac{\partial T_{k+1}^{(k)}}{\partial x^{(k+1)}} & \frac{\partial T_{k+1}^{(k)}}{\partial y^{(k+1)}} \\ \frac{\partial f^{(k+1)}}{\partial \dot{x}^{(k+1)}} & \frac{\partial f^{(k+1)}}{\partial x^{(k+1)}} & \frac{\partial f^{(k+1)}}{\partial y^{(k+1)}} \end{bmatrix}\right) = n_x^{(k+1)} + n_x^{(k+1)} + n_y^{(k+1)} \qquad (3.34)$$

Using the same notation applied above, the solution of the reinitialization is expressed as:

$$x_0^{(k+1)}(p), \quad \dot{x}_0^{(k+1)}(p), \quad y_0^{(k+1)}(p), \quad t_0^{(k+1)}(p) \qquad (3.35)$$

3.3.3 Sensitivity transfer at transitions

The discontinuity system is differentiated for evaluating the sensitivities. The only degrees of freedom are the time-invariant parameters. The structural decomposition is used again to solve the system in two steps.

First, (3.29) and (3.30) are differentiated. Differentiation of (3.29) indicates that the sensitivities are the ones calculated in the mode k. Differentiation of (3.30) yields a system that can be solved for the switching time sensitivities:

$$\frac{\partial g_{k+1}^{(k)}}{\partial \dot{x}^{(k)}}\left(\dot{s}_x^{(k)} + \ddot{x}^{(k)}\frac{\partial t}{\partial p}\right) + \frac{\partial g_{k+1}^{(k)}}{\partial x^{(k)}}\left(s_x^{(k)} + \dot{x}^{(k)}\frac{\partial t}{\partial p}\right) + \frac{\partial g_{k+1}^{(k)}}{\partial y^{(k)}}\left(s_y^{(k)} + \dot{y}^{(k)}\frac{\partial t}{\partial p}\right) + \frac{\partial g_{k+1}^{(k)}}{\partial u^{(k)}}\left(\frac{\partial u^{(k)}}{\partial p} + \frac{\partial u^{(k)}}{\partial t}\frac{\partial t}{\partial p}\right) + \frac{\partial g_{k+1}^{(k)}}{\partial p} + \frac{\partial g_{k+1}^{(k)}}{\partial t}\frac{\partial t}{\partial p} = 0 \quad (3.36)$$

Note that in the case of a controlled transition with the logical condition $t \geq t_0^{(k+1)}$, the solution is similar to that obtained in the initialization (3.12).

Second, the system formed by (3.31) and (3.32) is differentiated. Applying the chain rule:

$$\begin{bmatrix} \frac{\partial T_{k+1}^{(k)}}{\partial \dot{x}^{(k)}} & 0 \\ \frac{\partial T_{k+1}^{(k)}}{\partial x^{(k)}} & 0 \\ \frac{\partial T_{k+1}^{(k)}}{\partial y^{(k)}} & 0 \\ \frac{\partial T_{k+1}^{(k)}}{\partial u^{(k)}} & 0 \\ \frac{\partial T_{k+1}^{(k)}}{\partial \dot{x}^{(k+1)}} & \frac{\partial f^{(k+1)}}{\partial \dot{x}^{(k+1)}} \\ \frac{\partial T_{k+1}^{(k)}}{\partial x^{(k+1)}} & \frac{\partial f^{(k+1)}}{\partial x^{(k+1)}} \\ \frac{\partial T_{k+1}^{(k)}}{\partial y^{(k+1)}} & \frac{\partial f^{(k+1)}}{\partial y^{(k+1)}} \\ \frac{\partial T_{k+1}^{(k)}}{\partial u^{(k+1)}} & \frac{\partial f^{(k+1)}}{\partial u^{(k+1)}} \\ \frac{\partial T_{k+1}^{(k)}}{\partial p} & \frac{\partial f^{(k+1)}}{\partial p} \\ \frac{\partial T_{k+1}^{(k)}}{\partial t} & \frac{\partial f^{(k+1)}}{\partial t} \end{bmatrix}^T \begin{bmatrix} \frac{\partial \dot{x}^{(k)}}{\partial p} & \frac{\partial \dot{x}^{(k)}}{\partial t} \\ \frac{\partial x^{(k)}}{\partial p} & \frac{\partial x^{(k)}}{\partial t} \\ \frac{\partial y^{(k)}}{\partial p} & \frac{\partial y^{(k)}}{\partial t} \\ \frac{\partial u^{(k)}}{\partial p} & \frac{\partial u^{(k)}}{\partial t} \\ \frac{\partial \dot{x}^{(k+1)}}{\partial p} & \frac{\partial \dot{x}^{(k+1)}}{\partial t} \\ \frac{\partial x^{(k+1)}}{\partial p} & \frac{\partial x^{(k+1)}}{\partial t} \\ \frac{\partial y^{(k+1)}}{\partial p} & \frac{\partial y^{(k+1)}}{\partial t} \\ \frac{\partial u^{(k+1)}}{\partial p} & \frac{\partial u^{(k+1)}}{\partial t} \\ I & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} I \\ \frac{\partial t}{\partial p} \end{bmatrix} = 0 \quad (3.37)$$

Reordering to separate known and unknown variables yields:

$$\begin{bmatrix} \frac{\partial T_{k+1}^{(k)}}{\partial \dot{x}^{(k+1)}} & \frac{\partial f^{(k+1)}}{\partial \dot{x}^{(k+1)}} \\ \frac{\partial T_{k+1}^{(k)}}{\partial x^{(k+1)}} & \frac{\partial f^{(k+1)}}{\partial x^{(k+1)}} \\ \frac{\partial T_{k+1}^{(k)}}{\partial y^{(k+1)}} & \frac{\partial f^{(k+1)}}{\partial y^{(k+1)}} \end{bmatrix}^T \begin{bmatrix} \frac{\partial \dot{x}^{(k+1)}}{\partial p} \\ \frac{\partial x^{(k+1)}}{\partial p} \\ \frac{\partial y^{(k+1)}}{\partial p} \end{bmatrix} = - \begin{bmatrix} \frac{\partial T_{k+1}^{(k)}}{\partial \dot{x}^{(k+1)}} & \frac{\partial f^{(k+1)}}{\partial \dot{x}^{(k+1)}} \\ \frac{\partial T_{k+1}^{(k)}}{\partial x^{(k+1)}} & \frac{\partial f^{(k+1)}}{\partial x^{(k+1)}} \\ \frac{\partial T_{k+1}^{(k)}}{\partial y^{(k+1)}} & \frac{\partial f^{(k+1)}}{\partial y^{(k+1)}} \end{bmatrix}^T \begin{bmatrix} \frac{\partial \dot{x}^{(k+1)}}{\partial t} \\ \frac{\partial x^{(k+1)}}{\partial t} \\ \frac{\partial y^{(k+1)}}{\partial t} \end{bmatrix} \frac{\partial t}{\partial p}$$

$$- \begin{bmatrix} \frac{\partial T_{k+1}^{(k)}}{\partial \dot{x}^{(k)}} & 0 \\ \frac{\partial T_{k+1}^{(k)}}{\partial x^{(k)}} & 0 \\ \frac{\partial T_{k+1}^{(k)}}{\partial y^{(k)}} & 0 \\ \frac{\partial T_{k+1}^{(k)}}{\partial u^{(k)}} & 0 \\ \frac{\partial T_{k+1}^{(k)}}{\partial u^{(k+1)}} & \frac{\partial f^{(k+1)}}{\partial u^{(k+1)}} \\ \frac{\partial T_{k+1}^{(k)}}{\partial p} & \frac{\partial f^{(k+1)}}{\partial p} \\ \frac{\partial T_{k+1}^{(k)}}{\partial t} & \frac{\partial f^{(k+1)}}{\partial t} \end{bmatrix}^T \begin{bmatrix} \frac{\partial \dot{x}^{(k)}}{\partial p} + \frac{\partial \dot{x}^{(k)}}{\partial t}\frac{\partial t}{\partial p} \\ \frac{\partial x^{(k)}}{\partial p} + \frac{\partial x^{(k)}}{\partial t}\frac{\partial t}{\partial p} \\ \frac{\partial y^{(k)}}{\partial p} + \frac{\partial y^{(k)}}{\partial t}\frac{\partial t}{\partial p} \\ \frac{\partial u^{(k)}}{\partial p} + \frac{\partial u^{(k)}}{\partial t}\frac{\partial t}{\partial p} \\ \frac{\partial u^{(k+1)}}{\partial p} + \frac{\partial u^{(k+1)}}{\partial t}\frac{\partial t}{\partial p} \\ I \\ \frac{\partial t}{\partial p} \end{bmatrix} \tag{3.38}$$

Again, two conditions are required for the existence of the sensitivity function in a hybrid system: differentiability of the transfer functions $T_{k+1}^{(k)}$ and (3.34).

From equation (3.38) is obtained the initial sensitivities for the new mode directly (recall that $\ddot{x}$ and $\dot{y}$ can be calculated at any time with (3.26)). For clarity, the derivation is performed in two steps for the initialization (*i.e.*, sensitivity of the initialization solution and initial sensitivity of the trajectory) but now the differentiation has been applied taking in account that the time of the event is a function of the parameters. Note that the solution obtained in both cases is the same.

A procedure similar to the one in 3.2.1 may be followed to find the sensitivities of the variables at the discontinuities (3.35). The difference is that in this case, when the parameters change, the time of the event also changes, and the sensitivity includes the variation due to this factor.

3.4 Existence and Uniqueness of the Sensitivity Functions for Hybrid Systems

Existence and uniqueness theorems for sensitivity functions are intimately related to theorems governing existence and uniqueness of the embedded differential system [68]. Hence, since no existence and uniqueness theorem exists for general nonlinear DAEs [21], it is not possible to state an existence and uniqueness theorem for sensitivity functions of general nonlinear DAEs. However, existence and uniqueness theorems exist for both nonlinear explicit ODEs and linear time invariant DAEs. Thus, this section establishes sufficient conditions for the existence and uniqueness of sensitivity functions for both nonlinear explicit ODE embedded hybrid systems and linear time-invariant DAE embedded hybrid systems.

3.4.1 Existence and uniqueness of the sensitivity functions for hybrid systems with ODEs

Let $[S_1, S_2, \ldots, S_j, \ldots, S_{n_j}]$ with $n_j \geq 2$ be the ordered succession of modes, (this sequence is a function of the parameter values), describing the evolution of an ODE embedded hybrid system characterized by:

1. A system of differential equations (index 0) for every S_j:

$$\dot{x}^{(j)} = f^{(j)}(x^{(j)}, p, t) \tag{3.39}$$

2. The initial conditions for S_1:

$$T_0^{(1)}(\dot{x}^{(1)}, x^{(1)}, p, t_0^{(1)}) = 0 \tag{3.40}$$

3. The transitions to the next stage determined by the discontinuity function:

$$g_{j+1}^{(j)}(\dot{x}^{(j)}, x^{(j)}(t_f^{(j)}), p, t_f^{(j)}) = 0 \tag{3.41}$$

The transition functions:

$$T_{j+1}^{(j)}(\dot{x}^{(j)}, x^{(j)}(t_f^{(j)}), \dot{x}^{(j+1)}, x^{(j+1)}(t_f^{(j)}), p, t_f^{(j)}) = 0 \tag{3.42}$$

where $[t_f^{(1)}, .., t_f^{(j)}, .., t_f^{(n_j-1)}]$ are the times when the changes of mode happen.

Theorem 3.1. *Suppose that:*

1. *For* $t \in [t_0^{(j)}, t_f^{(j)}]$, $j = 1, ..., n_j$ *the partial derivatives*

$$\frac{\partial f^{(j)}}{\partial x^{(j)}} \quad \text{and} \quad \frac{\partial f^{(j)}}{\partial p}$$

exist and are continuous in a neighborhood of the solution $x^{(j)}(p, t)$.

2. $\forall t_f^{(j)}$, $j = 0...n_j - 1$, *the system* $h^{(j)}(\dot{x}^{(j)}, x^{(j)}, \dot{x}^{(j+1)}, x^{(j+1)}, t_f^{(j)}; p) = 0$:

$$x^{(j)}(t_f^{(j)}) - x^{(j)}(t_0^{(j)}) - \int_{t_0^{(j)}}^{t_f^{(j)}} f^{(j)}(x^{(j)}, p, t)dt = 0 \tag{3.43}$$

$$g_{j+1}^{(j)}(\dot{x}^{(j)}(t_f^{(j)}), x^{(j)}(t_f^{(j)}), p, t_f^{(j)}) = 0 \tag{3.44}$$

$$T_{j+1}^{(j)}(\dot{x}^{(j)}(t_f^{(j)}), x^{(j)}(t_f^{(j)}), \dot{x}^{(j+1)}(t_f^{(j)}), x^{(j+1)}(t_f^{(j)}), p, t_f^{(j)}) = 0 \tag{3.45}$$

$$\dot{x}^{(j+1)}(t_f^{(j)}) - f^{(j+1)}(x^{(j+1)}(t_f^{(j)}), p, t_f^{(j)}) = 0 \tag{3.46}$$

is a continuously differentiable mapping of an open set

$$E \subset \mathbb{R}^{(2n_x^{(j)} + 2n_x^{(j+1)} + 1) + (n_p)}$$

($n_p = 1$*) into* $\mathbb{R}^{(2n_x^{(j)} + 2n_x^{(j+1)} + 1)}$, *such that* $h^{(j)}(\dot{x}^{(j)}, x^{(j)}, \dot{x}^{(j+1)}, x^{(j+1)}, t_f^{(j)}; p) = 0$ *for the point* $(\dot{x}^{(j)}, x^{(j)}, \dot{x}^{(j+1)}, x^{(j+1)}, t_f^{(j)}; p) \in E$. *Assume that the submatrix formed by the columns of the Jacobian matrix corresponding to the variables* $\dot{x}^{(j)}$, $x^{(j)}$, $\dot{x}^{(j+1)}$, $x^{(j+1)}$, *and* $t_f^{(j)}$ *is invertible.*

In the special case of the initialization at S_1, $(j = 0)$ *there is no equation (3.43) and no variables* $x^{(j)}$ *in the system* $h^{(0)}$.

Then,

1. *$\forall j$ the partial derivatives:*

$$\frac{\partial x^{(j)}}{\partial p}$$

exist, are continuous, and satisfy the differential equations:

$$\frac{\partial}{\partial t}\left(\frac{\partial x_i^{(j)}}{\partial p}\right) = \frac{\partial f_i^{(j)}}{\partial x^{(j)}}\frac{\partial x^{(j)}}{\partial p} + \frac{\partial f_i^{(j)}}{\partial p} \tag{3.47}$$

in $(t_0^{(j)}, t_f^{(j)})$

2. *At $t_f^{(j)}$ the relationship between the right-hand sensitivities of the variables in mode S_j and the left-hand sensitivities of the variables in mode S_{j+1} is determined by:*

$$\begin{aligned}\frac{\partial x^{(j+1)}}{\partial p} = &-\left[f^{(j+1)} + \left(\frac{\partial T_{j+1}^{(j)}}{\partial x^{(j+1)}}\right)^{-1}\right. \\ &\left.\left(\frac{\partial T_{j+1}^{(j)}}{\partial \dot{x}^{(j)}}\frac{\partial f^{(j)}}{\partial t} + \frac{\partial T_{j+1}^{(j)}}{\partial x^{(j)}}f^{(j)} + \frac{\partial T_{j+1}^{(j)}}{\partial \dot{x}^{(j+1)}}\frac{\partial f^{(j+1)}}{\partial t} + \frac{\partial T_{j+1}^{(j)}}{\partial t}\right)\right]\frac{dt}{dp} \\ &-\left(\frac{\partial T_{j+1}^{(j)}}{\partial x^{(j+1)}}\right)^{-1}\left(\frac{\partial T_{j+1}^{(j)}}{\partial \dot{x}^{(j)}}\frac{\partial f^{(j)}}{\partial p} + \frac{\partial T_{j+1}^{(j)}}{\partial x^{(j)}}\frac{\partial x^{(j)}}{\partial p} + \frac{\partial T_{j+1}^{(j)}}{\partial \dot{x}^{(j+1)}}\frac{\partial f^{(j+1)}}{\partial p} + \frac{\partial T_{j+1}^{(j)}}{\partial p}\right)\end{aligned} \tag{3.48}$$

where

$$\frac{dt}{dp} = -\frac{\dfrac{\partial g_{j+1}^{(j)}}{\partial \dot{x}^{(j)}}\dfrac{\partial f^{(j)}}{\partial p} + \dfrac{\partial g_{j+1}^{(j)}}{\partial x^{(j)}}\dfrac{\partial x^{(j)}}{\partial p} + \dfrac{\partial g_{j+1}^{(j)}}{\partial p}}{\dfrac{\partial g_{j+1}^{(j)}}{\partial \dot{x}^{(j)}}\dfrac{\partial f^{(j)}}{\partial t} + \dfrac{\partial g_{j+1}^{(j)}}{\partial x^{(j)}}\dfrac{\partial x^{(j)}}{\partial t} + \dfrac{\partial g_{j+1}^{(j)}}{\partial t}} \tag{3.49}$$

Proof. Assumptions 1 and 2 correspond, respectively, to (1) the Gronwall's theorems [68, 71] for existence and uniqueness of the derivatives with respect to parameters and initial conditions, including the initial time, and (2) the implicit function theorem.

For the initialization problem, the implicit function theorem guarantees the existence of open sets $U^{(1)}$ and $W^{(1)}$ such that to every $p \in W^{(1)} \subset \mathbb{R}^{n_p}$ $(n_p = 1)$ there corresponds a unique $(x^{(1)}, \dot{x}^{(1)}, t_0^{(1)}; p) \in U^{(1)} \subset \mathbb{R}^{(n_x^{(1)}+n_x^{(1)}+1)+(n_p)}$ such that $h^{(0)}(\dot{x}^{(1)}, x^{(1)}, t_0^{(1)}; p) = 0$, and there is a continuously differentiable mapping of $W^{(1)}$ into $\mathbb{R}^{(n_x^{(1)}+n_x^{(1)}+1)}$ defining $(x^{(1)},\dot{x}^{(1)},t_0^{(1)})$. Therefore, it is possible to find in a neighborhood of p a point that produces $(x^{(1)}, \dot{x}^{(1)}, t_0^{(1)})$ within that set where the conditions of the Gronwall's theorem apply for system $f^{(1)}$.

Similarly Gronwall's theorems assure that the difference:

$$\left\| x^{(j)}(p+\epsilon, t) - x^{(j)}(p, t) \right\|$$

can be made as small as desired with $t \in [t_0^{(j)}, t_f^{(j)}]$ by selecting a sufficiently small ϵ. This is true for $j = 1$.

In the transition to the mode S_2, or in general from S_j to S_{j+1}, the implicit function theorem is applied again. Therefore, sets $U^{(j)}$ and $W^{(j)}$ can be found such that to every $p \in W^{(j)} \subset W^{(j-1)} \subset \mathbb{R}^{n_p}$ $(n_p = 1)$ there corresponds a unique $(\dot{x}^{(j)}, x^{(j)}, \dot{x}^{(j+1)}, x^{(j+1)}, t_f^{(j)}; p) \in U^{(j)} \subset \mathbb{R}^{(2n_x^{(j)}+2n_x^{(j+1)}+1)+(n_p)}$ such that

$$h^{(j)}(\dot{x}^{(j)}, x^{(j)}, \dot{x}^{(j+1)}, x^{(j+1)}, t_f^{(j)}; p) = 0$$

and there is a continuously differentiable mapping of $W^{(j)}$ into $\mathbb{R}^{(2n_x^{(j)}+2n_x^{(j+1)}+1)}$ defining $(\dot{x}^{(j)}, x^{(j)}, \dot{x}^{(j+1)}, x^{(j+1)}, t_f^{(j)})$.

Consequently, by selecting the smallest neighborhood of p from those that apply for the Gronwall's theorems and the implicit function theorem for all the modes, and applying them in a chain, the existence and uniqueness of the sensitivity functions for the ODE-embedded hybrid system is demonstrated. Expression (3.47) is from Gronwall's theorem. Expressions (3.48) and (3.49) are derived by differentiation of the system $h^{(j)}$ (see Section 3.3.3). □

Remark 3.1. *Since it is desired to obtain partial derivatives, the theorem is applied individually to every parameter while the rest are fixed, so the statement of the theorem*

with only one parameter is applicable to the general multiple parameter case.

Remark 3.2. *The controls $u(p,t)$ have been dropped from the formulation of the theorem, but are implicit since in all the functions p and t appear as variables. If the transition function is explicit in the state variables $x^{(j+1)}$ and does not depend on the time derivatives:*

$$T_{j+1}^{(j)}(\dot{x}^{(j)}(t_f^{(j)}), x^{(j)}(t_f^{(j)}), \dot{x}^{(j+1)}(t_f^{(j)}), x^{(j+1)}(t_f^{(j)}), p, t_f^{(j)}) = x^{(j+1)}(t_f^{(j)}) - \Phi_{j+1}^{(j)}(x^{(j)}(t_f^{(j)}), p, t_f^{(j)}) = 0 \quad (3.50)$$

$$\frac{\partial T_{j+1}^{(j)}}{\partial x^{(j)}} = -\frac{\partial \Phi_{j+1}^{(j)}}{\partial x^{(j)}} \qquad \frac{\partial T_{j+1}^{(j)}}{\partial \dot{x}^{(j)}} = 0 \quad (3.51)$$

$$\frac{\partial T_{j+1}^{(j)}}{\partial x^{(j+1)}} = I \qquad \frac{\partial T_{j+1}^{(j)}}{\partial \dot{x}^{(j+1)}} = 0 \quad (3.52)$$

$$\frac{\partial T_{j+1}^{(j)}}{\partial p} = -\frac{\partial \Phi_{j+1}^{(j)}}{\partial p} \qquad \frac{\partial T_{j+1}^{(j)}}{\partial t} = -\frac{\partial \Phi_{j+1}^{(j)}}{\partial t} \quad (3.53)$$

Substituting and reordering in (3.48) yields the expression derived by Rozenvasser [124]:

$$\frac{\partial x^{(j+1)}}{\partial p} - \frac{\partial x^{(j)}}{\partial p} = \left[-\left(f^{(j+1)} - f^{(j)}\right) + \left(\frac{\partial \Phi_{j+1}^{(j)}}{\partial x^{(j)}} - I\right) f^{(j)} + \frac{\partial \Phi_{j+1}^{(j)}}{\partial t} \right] \frac{dt}{dp} + \left(\frac{\partial \Phi_{j+1}^{(j)}}{\partial x^{(j)}} - I\right) \frac{\partial x^{(j)}}{\partial p} + \frac{\partial \Phi_{j+1}^{(j)}}{\partial p} \quad (3.54)$$

Remark 3.3. *Hybrid systems formulated with this model are a sequence of alternating differential and algebraic systems of equations, and the conditions for solvability, existence and uniqueness are the union of these conditions for all of the systems.*

Remark 3.4. *The sensitivity functions are not defined for solutions passing through points where g or T are not differentiable. In particular, this is true at the points where $g_{j\alpha}^{(k)} = g_{i\beta}^{(k)} = 0$ or $g_{j\alpha}^{(k)} = g_{j\beta}^{(k)} = 0$,which are transitions to a different mode or transitions to the same mode with different discontinuity functions.*

Remark 3.5. *In general sensitivities jump even if the states are continuous. This statement implies:*

$$\frac{\partial T_{j+1}^{(j)}}{\partial x^{(j+1)}} = I \qquad \frac{\partial T_{j+1}^{(j)}}{\partial x^{(j)}} = -I \qquad \frac{\partial T_{j+1}^{(j)}}{\partial \dot{x}^{(j+1)}} = \frac{\partial T_{j+1}^{(j)}}{\partial \dot{x}^{(j)}} = \frac{\partial T_{j+1}^{(j)}}{\partial p} = \frac{\partial T_{j+1}^{(j)}}{\partial t} = 0 \tag{3.55}$$

so equation (3.48) becomes:

$$\frac{\partial x^{(j+1)}}{\partial p} - \frac{\partial x^{(j)}}{\partial p} = -\left[f^{(j+1)} - f^{(j)}\right] \frac{dt}{dp} \quad . \tag{3.56}$$

As Rozenvasser points out, sensitivities are continuous either if the time derivative is continuous or if the event time does not depend on the parameter.

3.4.2 Existence and uniqueness of the sensitivity functions for hybrid systems with linear time-invariant DAEs

In the remaining part of this section define:

$$z = \begin{bmatrix} x \\ y \end{bmatrix}$$

Lemma 3.1. *Suppose that the linear constant-coefficient DAE system with differential index = ν:*

$$F(\dot{z}, z, t; p) = A\dot{z} + Bz - f(p, t) = 0 \tag{3.57}$$

*is solvable (*i.e., *$\lambda A + B$ is a regular pencil) for $t \in [t_0, t_f]$ and the partial derivatives*

$$\frac{\partial}{\partial p}\left(\frac{\partial^i f}{\partial t^i}\right) \qquad i = 0, \ldots, \nu - 1 \tag{3.58}$$

exist and are continuous in a neighborhood of the solution $z(p, t)$.

Then, the partial derivatives:

$$s = \frac{\partial z}{\partial p} \tag{3.59}$$

exist, are continuous and satisfy the differential equations

$$A\dot{s} + Bs = \frac{\partial f}{\partial p} \tag{3.60}$$

Proof. Since the system is solvable, there exist nonsingular matrices p, Q such that:

$$z = Qw \tag{3.61}$$

$$PAQ\dot{w} + PBQw = Pf(p,t) \tag{3.62}$$

$$PAQ = \begin{bmatrix} I & 0 \\ 0 & N \end{bmatrix} \qquad PBQ = \begin{bmatrix} C & 0 \\ 0 & I \end{bmatrix} \tag{3.63}$$

where N is a matrix of nilpotency ν. The resulting system after the change of coordinates is:

$$\dot{w}_1 + Cw_1 = f_1 \tag{3.64}$$

$$N\dot{w}_2 + w_2 = f_2 \tag{3.65}$$

The first equation is an ODE and Gronwall's theorems can be applied. The second equation has only one solution:

$$w_2 = \sum_{i=0}^{\nu-1} (-1)^i N^i \frac{\partial^i f_2}{\partial t^i} \tag{3.66}$$

and the existence and uniqueness of the remaining parametric sensitivities can be deduced directly from (3.58). The system (3.60) obtained by differentiating (3.57), has the same matrix pencil as (3.57), assuming the forcing functions are sufficiently differentiable (3.58). Hence (3.60) is solvable. □

Remark 3.6. *The sufficient differentiability (3.58) is not required for all the compo-*

nents of f but only for the ones appearing in the linear combination f_2. If the exact differentiability required for each component of f is specified that can be deduced from (3.66), then the conditions of the lemma are necessary too.

Theorem 3.2. *Suppose the linear constant coefficient DAE system with differential index= ν:*

$$F(\dot{z}, z, t; p) = A(p)\dot{z} + B(p)z - f(p,t) = 0 \tag{3.67}$$

is solvable for $t \in [t_0, t_f]$, that f is $(2\nu-1)$-times differentiable with respect to time, and that the partial derivatives:

$$\frac{\partial}{\partial p}\left(\frac{\partial^i f}{\partial t^i}\right) \qquad i = 0, \ldots, \nu - 1 \tag{3.68}$$

$$\frac{\partial A(p)}{\partial p} \tag{3.69}$$

$$\frac{\partial B(p)}{\partial p} \tag{3.70}$$

exist and are continuous in a neighborhood of the solution $z(p,t)$.

Then, the partial derivatives:

$$s = \frac{\partial z}{\partial p} \tag{3.71}$$

exist, are continuous, and satisfy the differential equations:

$$A\dot{s} + Bs = \frac{\partial f}{\partial p} - \frac{\partial A(p)}{\partial p}\dot{z} - \frac{\partial B(p)}{\partial p}z \tag{3.72}$$

Proof. The partial differentiation of (3.67) leads to (3.72). This system is solvable if the right hand side is $(\nu-1)$-times differentiable with respect to time, which requires z to be ν-times differentiable with respect to time, (3.68), (3.69) and (3.70). Since the solution z needs f to be $(\nu-1)$-times differentiable with respect to time, f must be $(2\nu-1)$-times differentiable. □

Remark 3.7. *As in the lemma, not all the components of f need to be $(2\nu-1)$-times*

differentiable, since in general not all of them will end up in the rows of the nilpotent matrix that requires this property.

Theorem 3.3. *Given a solution of a hybrid system described by a sequence of modes* $[S_1, S_2, .., S_j, .., S_{n_j}]$, *every mode characterized by a set of variables* $z^{(j)}$ *and a linear time invariant DAE satisfying (3.4), whose coefficients are functions of the parameter* p*:*

$$F^{(j)}(\dot{x}^{(j)}, x^{(j)}, y^{(j)}, u^{(j)}, t; p) = A^{(j)}(p)\dot{z}^{(j)} + B^{(j)}(p)z^{(j)} - f^{(j)}(p, t) = 0 \qquad (3.73)$$

and transitions to the following mode represented by the discontinuity function $g_{j+1}^{(j)}$ *and the transition functions* $T_{j+1}^{(j)}$.

Suppose that:

1. *For* $t \in [t_0^{(j)}, t_f^{(j)}]$, $j = 1, ..., n_j$, *every* $f^{(j)}$ *satisfies the conditions of Theorem 3.2.*

2. $\forall t_f^{(j)}$, $j = 0, ..., n_j - 1$, *the system* $h^{(j)}(\dot{x}^{(j)}, x^{(j)}, y^{(j)}, \dot{x}^{(j+1)}, x^{(j+1)}, y^{(j+1)}, t_f^{(j)}; p) = 0$*:*

$$\dot{x}^{(j)} = \dot{x}^{(j)}(t_f^{(j)}; p) \qquad x^{(j)} = x^{(j)}(t_f^{(j)}; p) \qquad y^{(j)} = y^{(j)}(t_f^{(j)}; p) \qquad (3.74)$$

$$g_{j+1}^{(j)}(\dot{x}^{(j)}, x^{(j)}, y^{(j)}, t_f^{(j)}; p) = 0 \qquad (3.75)$$

$$T_{j+1}^{(j)}(\dot{x}^{(j)}, x^{(j)}, y^{(j)}, \dot{x}^{(j+1)}, x^{(j+1)}, y^{(j+1)}, t_f^{(j)}; p) = 0 \qquad (3.76)$$

$$f^{(j+1)}(\dot{x}^{(j+1)}, x^{(j+1)}, y^{(j+1)}, t_f^{(j)}; p) = 0 \qquad (3.77)$$

is a continuously differentiable mapping of an open set:

$$E \subset \mathbb{R}^{(2n_x^{(j)} + n_y^{(j)} + 2n_x^{(j+1)} + n_y^{(j+1)} + 1) + (n_p)}$$

$(n_p = 1)$ *into* $\mathbb{R}^{(2n_x^{(j)} + n_y^{(j)} + 2n_x^{(j+1)} + n_y^{(j+1)} + 1)}$, *such that* $h^{(j)} = 0$ *for the point:*

$$(\dot{x}^{(j)}, x^{(j)}, y^{(j)}, \dot{x}^{(j+1)}, x^{(j+1)}, y^{(j+1)}, t_f^{(j)}; p) \in E.$$

Assume that the submatrix formed by the columns of the Jacobian matrix corresponding to the variables $\dot{x}^{(j)}, x^{(j)}, y^{(j)}, \dot{x}^{(j+1)}, x^{(j+1)}, y^{(j+1)}$, *and* $t_f^{(j)}$ *is invertible.*

The equations (3.74) represent the state trajectory resulting from the integration of $f^{(j)}$ *in mode* j.

In the initialization at S_1, *there is no equation (3.74), the transition functions are replaced by the initial conditions* $T_0^{(1)}(\dot{x}^{(1)}, x^{(1)}, y^{(1)}, t_0^{(1)}; p) = 0$, *and there are no variables from a previous mode in the system* $h^{(0)}$.

Then:

1. $\forall j$ *the partial derivatives:*

$$s^{(j)} = \frac{\partial z^{(j)}}{\partial p}$$

exist, are continuous, and satisfy the differential equations (3.72) in $(t_0^{(j)}, t_f^{(j)})$.

2. *At* $t_f^{(j)}$, *the relationship between the right-hand sensitivities of the variables in mode* S_j *and the left-hand sensitivities of the variables in mode* S_{j+1} *is determined by:*

$$\frac{\partial g_{j+1}^{(j)}}{\partial \dot{x}^{(j)}}\left(\dot{s}_x^{(j)} + \ddot{x}^{(j)}\frac{dt}{dp}\right) + \frac{\partial g_{j+1}^{(j)}}{\partial x^{(j)}}\left(s_x^{(j)} + \dot{x}^{(j)}\frac{dt}{dp}\right) + \frac{\partial g_{j+1}^{(j)}}{\partial y^{(j)}}\left(s_y^{(j)} + \dot{y}^{(j)}\frac{dt}{dp}\right) + \\ + \frac{\partial g_{j+1}^{(j)}}{\partial p} + \frac{\partial g_{j+1}^{(j)}}{\partial t}\frac{dt}{dp} = 0 \quad (3.78)$$

$$\begin{bmatrix} \frac{\partial T_{j+1}^{(j)}}{\partial \dot{x}^{(j+1)}} & \frac{\partial T_{j+1}^{(j)}}{\partial x^{(j+1)}} & \frac{\partial T_{j+1}^{(j)}}{\partial y^{(j+1)}} \\ \frac{\partial f^{(j+1)}}{\partial \dot{x}^{(j+1)}} & \frac{\partial f^{(j+1)}}{\partial x^{(j+1)}} & \frac{\partial f^{(j+1)}}{\partial y^{(j+1)}} \end{bmatrix} \begin{bmatrix} \dot{s}_x^{(j+1)} + \ddot{x}^{(j+1)} \frac{dt}{dp} \\ s_x^{(j+1)} + \dot{x}^{(j+1)} \frac{dt}{dp} \\ s_y^{(j+1)} + \dot{y}^{(j+1)} \frac{dt}{dp} \end{bmatrix} =$$

$$- \begin{bmatrix} \frac{\partial T_{j+1}^{(j)}}{\partial \dot{x}^{(j)}} & \frac{\partial T_{j+1}^{(j)}}{\partial x^{(j)}} & \frac{\partial T_{j+1}^{(j)}}{\partial y^{(j)}} & \frac{\partial T_{j+1}^{(j)}}{\partial t} & \frac{\partial T_{j+1}^{(j)}}{\partial p} \\ 0 & 0 & 0 & \frac{\partial f^{(j+1)}}{\partial t} & \frac{\partial f^{(j+1)}}{\partial p} \end{bmatrix} \begin{bmatrix} \dot{s}_x^{(j)} + \ddot{x}^{(j)} \frac{dt}{dp} \\ s_x^{(j)} + \dot{x}^{(j)} \frac{dt}{dp} \\ s_y^{(j)} + \dot{y}^{(j)} \frac{dt}{dp} \\ \frac{dt}{dp} \\ 1 \end{bmatrix} \tag{3.79}$$

Proof. Provided Theorem 3.2 that replaces Gronwall's theorems for the existence and uniqueness of the sensitivities of a linear time invariant DAE where the coefficients are functions of the parameters, the same arguments applied in the proof of Theorem 3.1 can be repeated. □

Remark 3.8. *This theorem can be extended to systems not satisfying (3.4), but then the number of equations in the transition functions is smaller, as not all the variables can be specified independently in the new mode.*

3.5 Examples

3.5.1 Implementation

The equations derived in Sections 3.2 and 3.3 lead to a straightforward numerical implementation for the combined state and sensitivity integration in the ABACUSS mathematical modeling environment. The evolution of a hybrid system can be viewed as a sequence of subproblems, each characterized by a continuous evolution in a mode terminated by an event and then a solution of the transition functions to initialize the new mode.

Continuous evolution of the states and sensitivities in a mode is determined by simultaneous numerical solution of $f^{(k)}$ and (3.25) using the staggered corrector method implemented in the code DSL48S (Chapter 4). Correct location of event times is guaranteed by the state event location algorithm of Park and Barton [110]. Solution of the transition conditions (3.31–3.32) is implemented as a sequence of Newton iterations. However, since the transitions may take the system outside the region of convergence for Newton's method, a more robust approach would involve a homotopy continuation procedure in which the system is deformed continuously from conditions of state continuity to the new state. Finally, the switching time sensitivity (3.36) and then transfer of the sensitivities to the new mode (3.38) are linear systems that require the previous solution of another linear system (3.26) for $\ddot{x}$ and $\dot{y}$ at both sides of the transition.

3.5.2 Critical values for the parameters

This example illustrates the practical significance of not satisfying the conditions of the existence and uniqueness theorem.

Consider the following hybrid system with two modes and a reversible transition

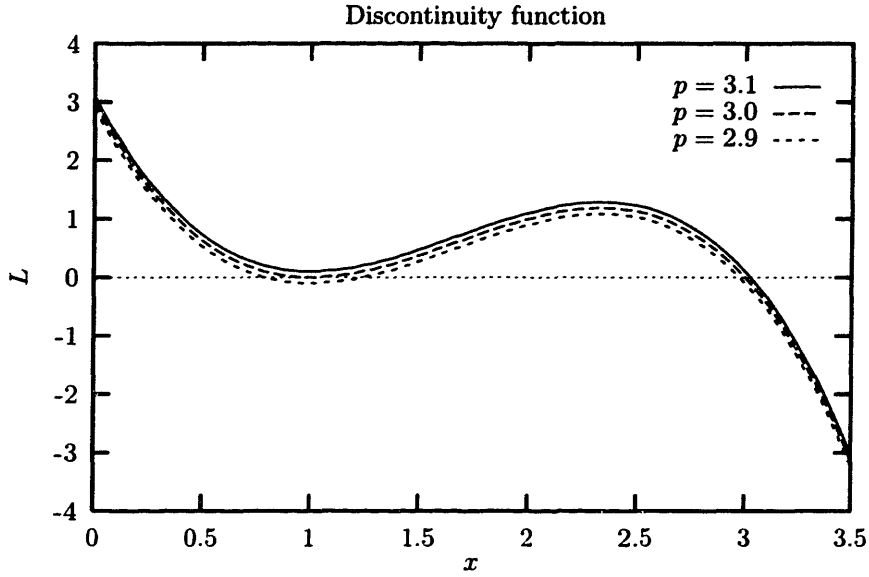

Figure 3-3: *Discontinuity function for the system (3.80–3.81)*

condition:

$$S_1 : \begin{cases} \frac{dx^{(1)}}{dt} = 4 - x^{(1)} \\ L_2^{(1)} : -(x^{(1)})^3 + 5(x^{(1)})^2 - 7x^{(1)} + p \leq 0 \\ T_2^{(1)} = x^{(2)} - x^{(1)} = 0 \end{cases} \tag{3.80}$$

$$S_2 : \begin{cases} \frac{dx^{(2)}}{dt} = 10 - 2x^{(2)} \\ L_1^{(2)} : -(x^{(2)})^3 + 5(x^{(2)})^2 - 7x^{(2)} + p > 0 \\ T_1^{(2)} = x^{(1)} - x^{(2)} = 0 \end{cases} \tag{3.81}$$

The initial mode is S_1 with initial condition $x^{(1)}(0) = 0$. There is only one parameter p that only appears in the transition condition. Figure 3-3 shows the discontinuity function as a function of the state $x^{(k)}$ for three different values of p. When $p = 3.1$ the discontinuity function activates at a value of $x^{(1)}$ close to 3. If $p = 2.9$ then there are:

1. a transition to mode 2 around $x^{(1)} = 0.8$,

2. a transition to mode 1 around $x^{(2)} = 1.2$, and

3. a transition to mode 2 around $x^{(1)} = 3$.

For $p = 3$ the function is tangent at $x^{(1)} = 1$ and crosses zero at $x^{(1)} = 3$. The first point is singular. It touches 0, switches to the second mode, and immediately changes to mode 1 again. At this point the gradient of the discontinuity function equals zero and (3.33) is not satisfied.

If the evolution of the transition point as a function of the parameter p is examined, it is observed that for values less than 3 there are three transition times that vary continuously. But at $p = 3$ there is a nonsmoothness in the event time: now there is only one transition time and the transition time for the first event has jumped from the previous case. Actually, at this point the second condition for the existence and uniqueness of the parametric sensitivity functions is not satisfied. The changes in the sequence of events can be seen to be related to these 'critical' points.

The sensitivity functions for $p = 2.9$ and $p = 3.1$ are plotted in Figures 3-4 and 3-5. In this system, the sensitivity functions are discontinuous at the time of switching. The expression for transfer of the sensitivities is:

$$s^+ = s^- - (\dot{x}^+ - \dot{x}^-)\frac{\left[\frac{1}{-3x^2+10x-7} - s^-\right]}{\dot{x}^-} \tag{3.82}$$

In Figures 3-4 and 3-5 the discontinuous effect of the different sequences is perceptible in the trajectory, but almost negligible in the long term. Consider now a system with a nonreversible transition condition, that is, a system where there is a switch from S_1 to S_2 but no switch back to S_1:

$$S_1 : \begin{cases} \frac{dx^{(1)}}{dt} = 4 - x^{(1)} \\ L_2^{(1)} : -(x^{(1)})^3 + 5(x^{(1)})^2 - 7x^{(1)} + p \leq 0 \\ T_2^{(1)} = x^{(2)} - x^{(1)} = 0 \end{cases} \tag{3.83}$$

$$S_2 : \begin{cases} \frac{dx^{(2)}}{dt} = 0 \end{cases} \tag{3.84}$$

Figures 3-6 and 3-7 show the trajectories and sensitivities for this system. Now the jump in the final values of the state and the sensitivity for the variation of the

parameter p is evident.

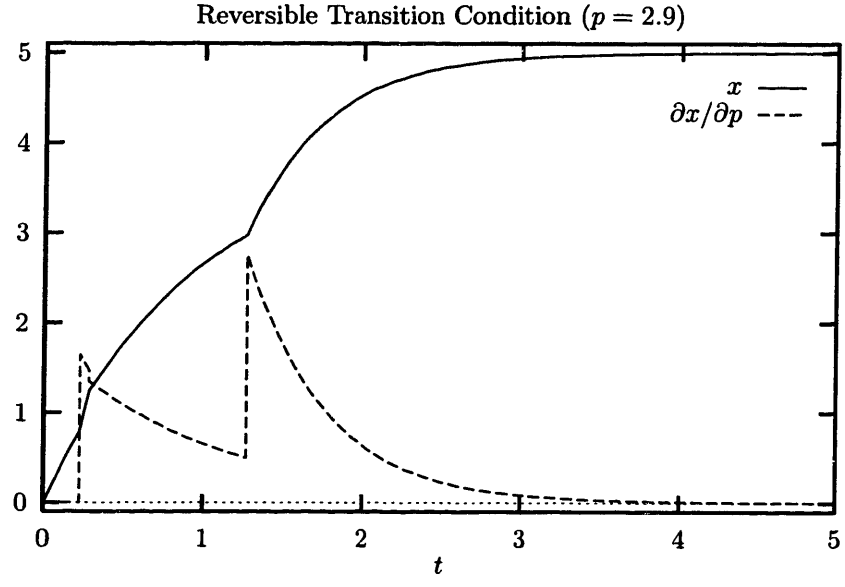

Figure 3-4: *Sensitivity and state trajectory for reversible transition condition when* $p = 2.9$

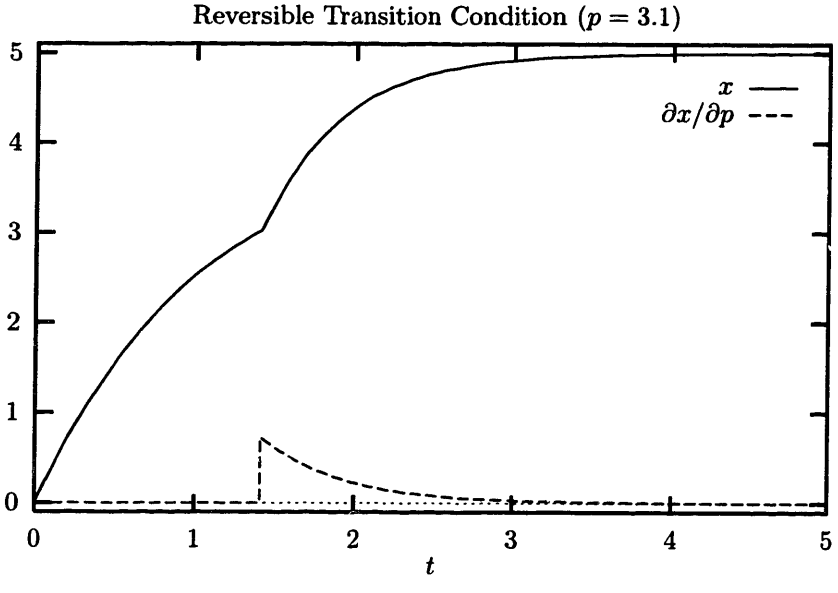

Figure 3-5: *Sensitivity and state trajectory for reversible transition condition when* $p = 3.1$

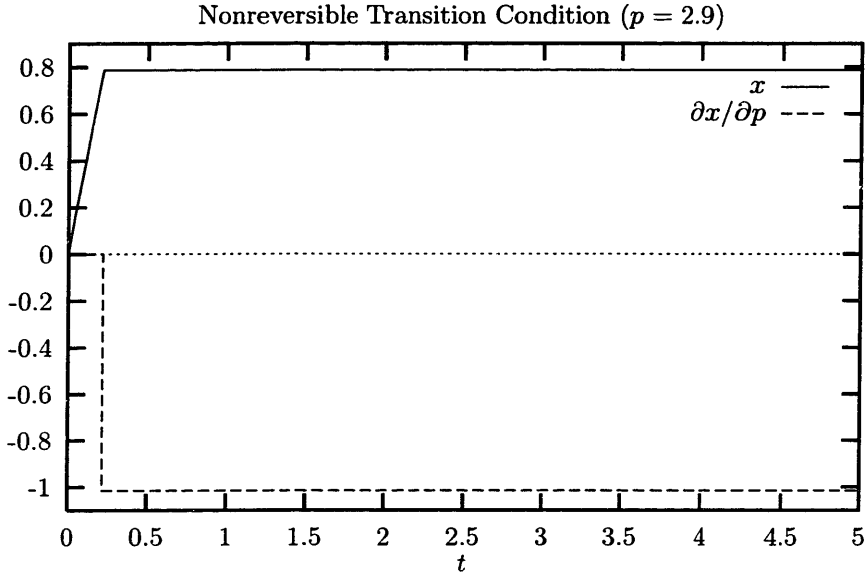

Figure 3-6: *Sensitivity and state trajectory for nonreversible transition condition when* $p = 2.9$

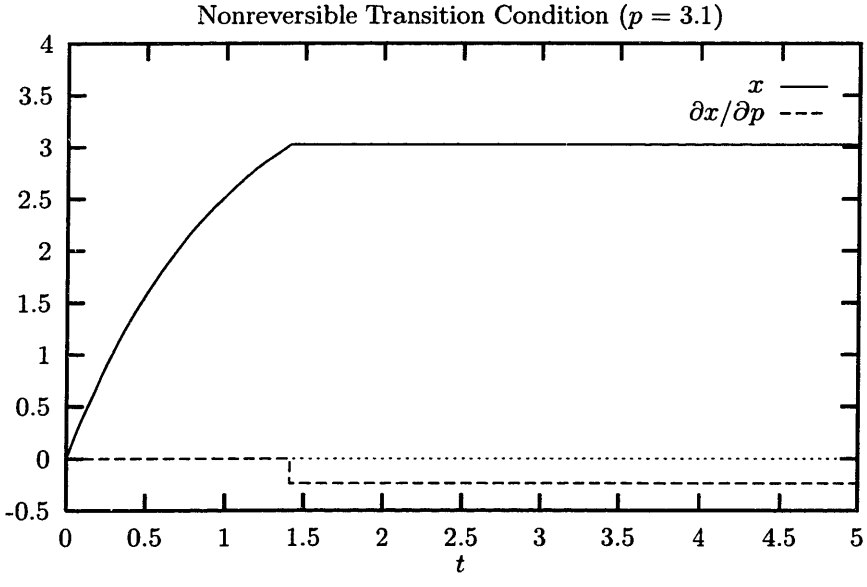

Figure 3-7: *Sensitivity and state trajectory for nonreversible transition condition when* $p = 3.1$

3.5.3 Functions discretized by finite elements

In many problems the control variables are approximated by a finite set of parameters that define some generic functions over finite elements that are different for every element. When the discretization is over time, the problem can be assimilated to a hybrid one where the modes are the elements.

For example, let the control variables be discretized using Lagrange polynomials $\psi_i^{(k)}(t)$ of order I over K finite elements, as described in Chapter 2. The transition conditions and functions assuming the usual continuity of the state variable are:

$$g_{k+1}^{(k)} = t - t_I^{(k)} \tag{3.85}$$

$$T_{k+1}^{(k)} = x^{(k+1)} - x^{(k)} \tag{3.86}$$

Now consider the transfer of the sensitivities with respect to the junction time $t^* = t_I^{(k)}$ at the end of element k. Equation (3.36) gives,

$$\frac{\partial t}{\partial p} = \begin{cases} 1 & \text{if } p = t^*, \\ 0 & \text{if } p \neq t^* \end{cases} \tag{3.87}$$

and from (3.38), for $p = t^*$,

$$\begin{bmatrix} 0 & I & 0 \\ \frac{\partial f^{(k+1)}}{\partial \dot{x}^{(k+1)}} & \frac{\partial f^{(k+1)}}{\partial x^{(k+1)}} & \frac{\partial f^{(k+1)}}{\partial y^{(k+1)}} \end{bmatrix} \begin{bmatrix} \dot{s}_x^{(k+1)} + \ddot{x}^{(k+1)} \\ s_x^{(k+1)} + \dot{x}^{(k+1)} \\ s_y^{(k+1)} + \dot{y}^{(k+1)} \end{bmatrix} = - \begin{bmatrix} -I & 0 & 0 \\ 0 & \frac{\partial f^{(k+1)}}{\partial u^{(k+1)}} & \frac{\partial f^{(k+1)}}{\partial t} \end{bmatrix} \begin{bmatrix} s_x^{(k)} + \dot{x}^{(k)} \\ \frac{\partial u^{(k+1)}}{\partial p} + \frac{\partial u^{(k+1)}}{\partial t} \\ 1 \end{bmatrix} \tag{3.88}$$

Noting that the derivative with respect to time of the DAE is identically zero, gives:

$$s_x^{(k+1)} = s_x^{(k)} - (\dot{x}^{(k+1)} - \dot{x}^{(k)}) \tag{3.89}$$

$$\frac{\partial f^{(k+1)}}{\partial \dot{x}^{(k+1)}} \dot{s}_x^{(k+1)} + \frac{\partial f^{(k+1)}}{\partial y^{(k+1)}} s_y^{(k+1)} = -\frac{\partial f^{(k+1)}}{\partial x^{(k+1)}} s_x^{(k+1)} - \frac{\partial f^{(k+1)}}{\partial u^{(k+1)}} \frac{\partial u^{(k+1)}}{\partial p} \tag{3.90}$$

For another junction time $p \neq t^*$, always supposing that p does not appear explicitly in f, the first equation simplifies to:

$$s_x^{(k+1)} = s_x^{(k)} \tag{3.91}$$

Therefore, for stage $k \neq 1$ the initial sensitivities are all zero for (3.13), (3.14), (3.15) and (3.16). The right hand side of (3.25) for element 1 is zero and the sensitivity functions are all zero in that stage. The system (3.90) and (3.91) transfers zero initial sensitivities to the new stage and this stage continues until stage k is entered. Here the initial values for $s_x^{(k)}$, $s_y^{(k)}$, and $\dot{s}_x^{(k)}$ are zero. However, the RHS of (3.25) is not zero and therefore integration provides the values for the sensitivity functions.

At $t = t_I^{(k)}$, the transfer is governed by (3.89) and (3.90). Therefore, the sensitivities will in general jump even if the states are continuous. If the controls are continuous over the junction of the finite elements, then $\dot{x}^{(k+1)} = \dot{x}^{(k)}$, and the sensitivities of the differential variables will be continuous too.

In the following elements, the integration continues from the initial values at every junction time. The sensitivities are continuous across element boundaries but in general not differentiable at the boundary if the controls are discontinuous. The RHS of (3.25) is zero, which indicates a system without excitation.

Figure 3-8 plots the sensitivities of three variables with respect to the junction time $t_I^{(2)}$, using five finite elements. These sensitivities exhibit the typical behavior described by discontinuous controls: zero sensitivity, excitation, jump and decay with non differentiable junctions.

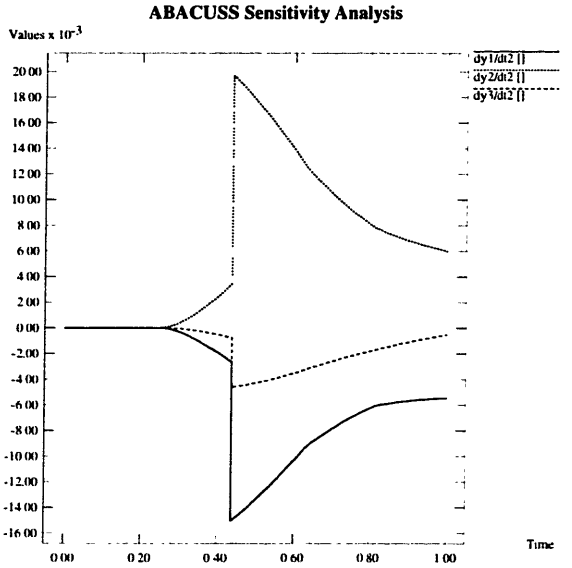

Figure 3-8: *Sensitivities with respect to junction time*

3.5.4 Singular Van der Pol's equation

Dorodnicyn studied the Van der Pol equation:

$$\epsilon\ddot{y} + (y^2 - 1)\dot{y} + y = 0 \tag{3.92}$$

with small ϵ. If $\epsilon = 0$ the problem can be written as a DAE:

$$\dot{x} = -y \tag{3.93}$$

$$x = \frac{y^3}{3} - y \tag{3.94}$$

At $y = \pm 2/3$, $x = \mp 1$ the solutions cease to exist. For small values of ϵ, the solutions jump with almost 'infinite' speed.

This behavior can be approximated with a DAE-embedded hybrid system with three modes where (3.93–3.94) is the continuous part for every mode. The modes are

defined in relation to values of y:

$$S_1 : \begin{cases} y < -1.001 & \\ L_2^{(1)} : y \geq -1.001 & T_2^{(1)} : y = 2 \end{cases} \tag{3.95}$$

$$S_2 : \begin{cases} y > 1.001 & \\ L_1^{(2)} : y \leq 1.001 & T_1^{(2)} : y = -2 \end{cases} \tag{3.96}$$

$$S_3 : \begin{cases} -0.999 < y < 0.999 & \\ L_1^{(3)} : y \geq 0.999 & T_1^{(3)} : y = -2 \\ L_2^{(3)} : y \leq -0.999 & T_2^{(3)} : y = 2 \end{cases} \tag{3.97}$$

The parameter is the initial value of y. At every event the sensitivity transfer is:

$$s_y^+ = \frac{\dot{y}^+}{\dot{y}^-} s_y^- \tag{3.98}$$

Figure 3-9 shows the trajectories of x and y in the time domain and Figure 3-10 shows y versus x for $y(0) = p = -3$. After some initial time the system begins cycling. The sensitivities are plotted in Figures 3-11 and 3-12.

When $p = 0$, the system starts at a fixed point and $y(t) = x(t) = 0$. The sensitivities are defined for t in a bounded interval:

$$s_x = -e^t \tag{3.99}$$

$$s_y = e^t \tag{3.100}$$

For a given bounded time interval, there will exist a neighborhood of parameter values around zero for which the sensitivity functions are qualitatively similar in the sense that they do not experience discontinuities. Outside of this neighborhood, the evolution of the hybrid system is not smooth. On the positive side, the oscillating solution is 'phase shifted' with respect to one starting from the negative side, as shown in Figures 3-13 and 3-14. The sensitivities are the same in this example for a positive

or a negative value of the parameter of the same magnitude, but this is a special case where there is symmetry in the system. In general, the sensitivity functions could be different.

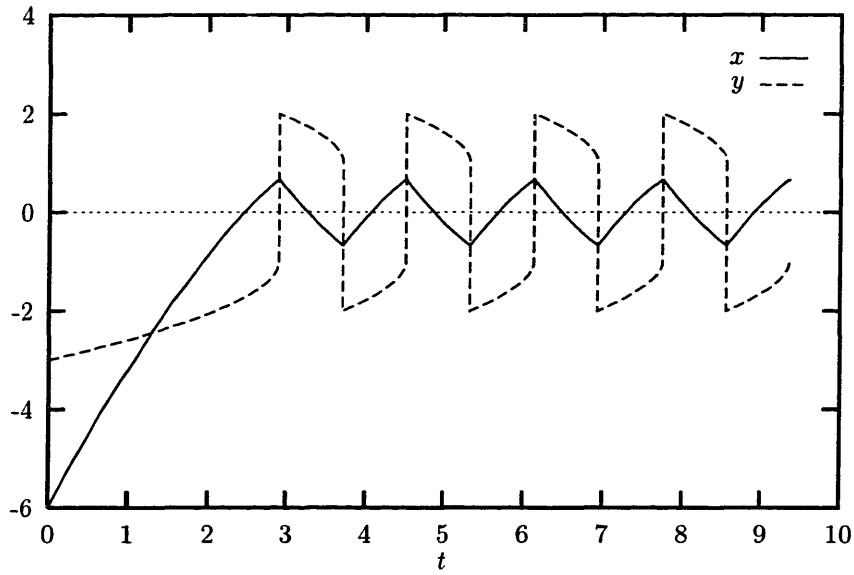

Figure 3-9: *State variable trajectories for* $p = -3$

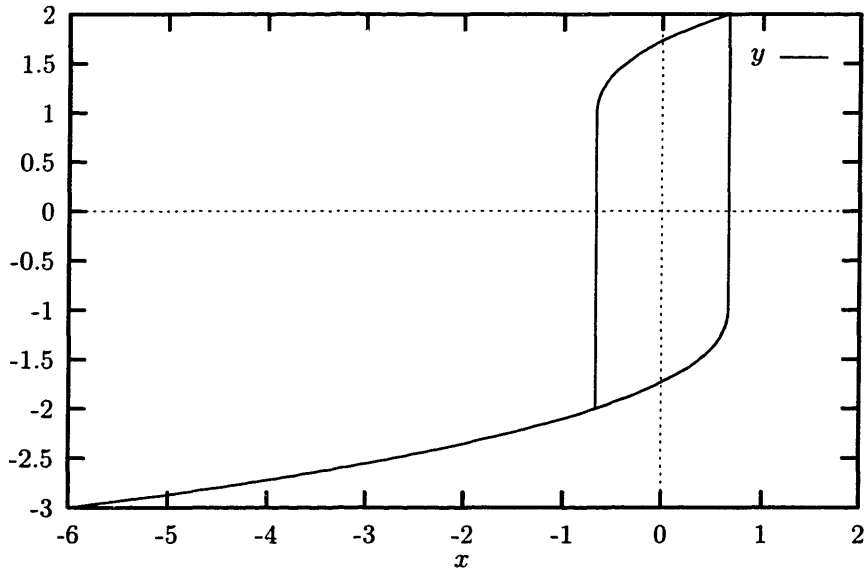

Figure 3-10: *State space plot for* $p = -3$

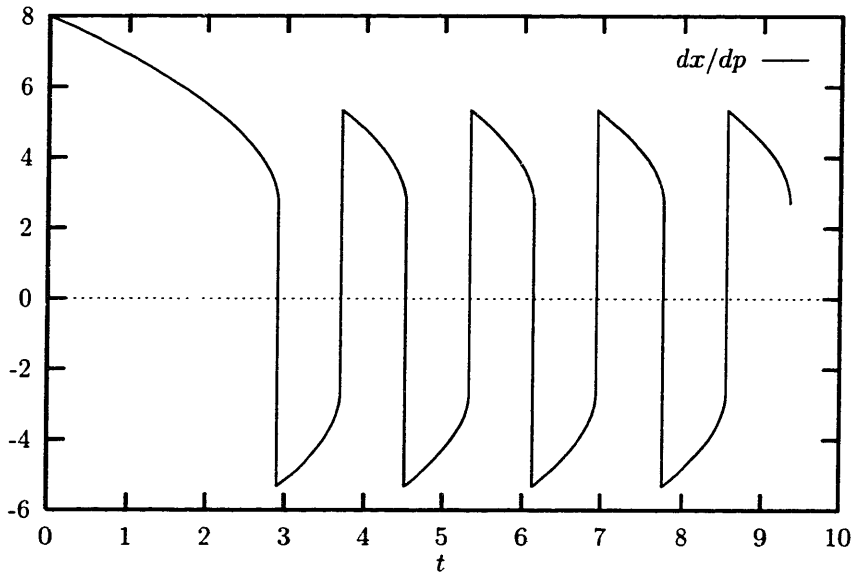

Figure 3-11: *Sensitivity trajectory for* $p = -3$

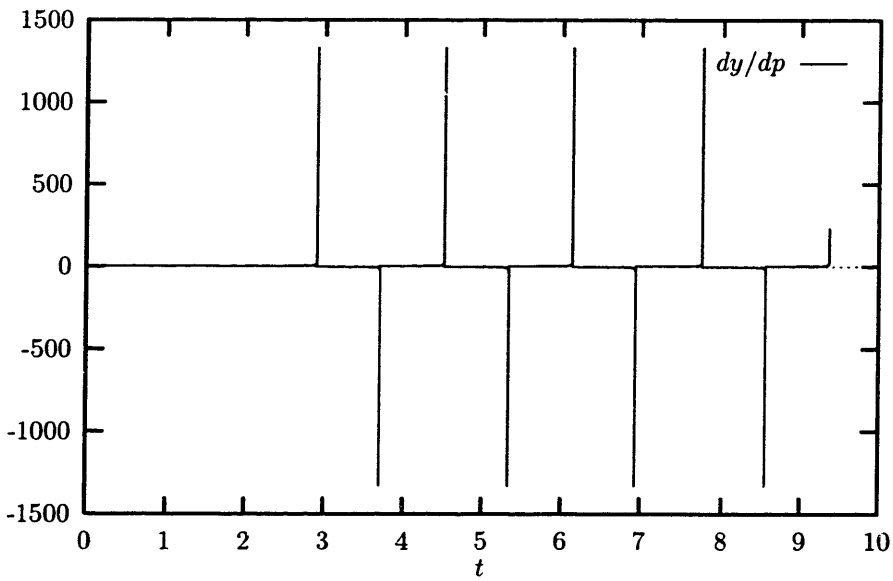

Figure 3-12: *Sensitivity trajectory for* $p = -3$

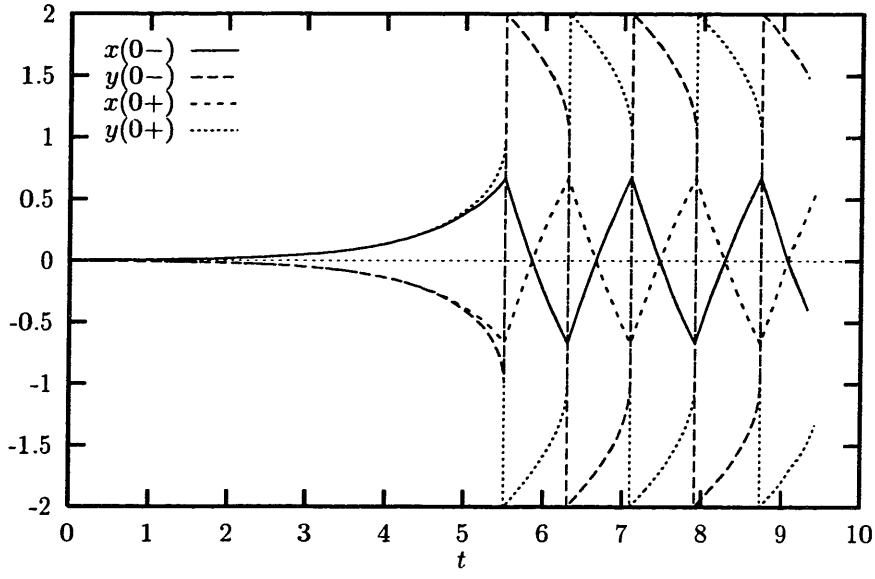

Figure 3-13: *State trajectories for* $p = 0$

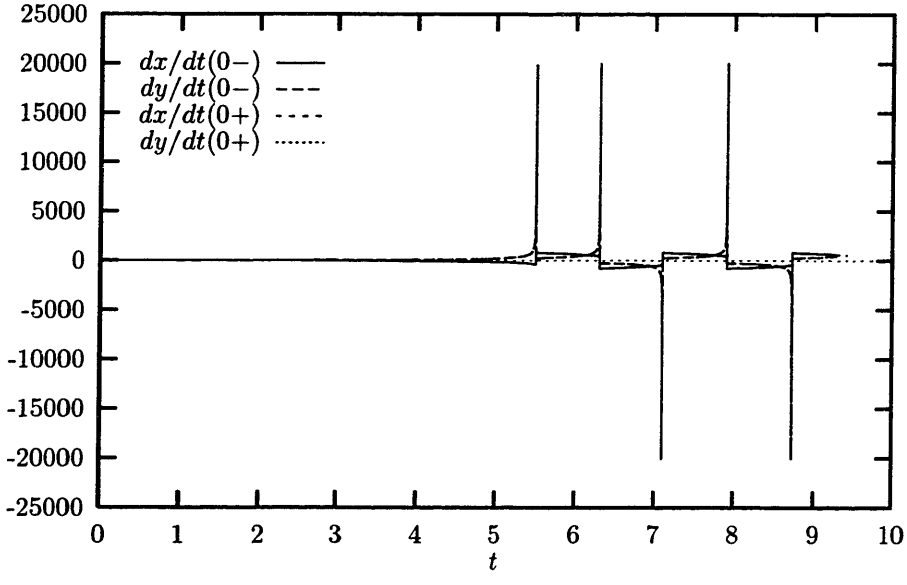

Figure 3-14: *Sensitivity trajectories for* $p = 0$

3.6 Conclusions

The parametric sensitivity functions for a broad class of index ≤ 1 DAE-embedded hybrid systems have been derived and illustrated with examples. Computationally, calculation of the sensitivity functions is inexpensive compared with solution of the original system.

In the cases of index-0 DAEs (ODEs) or linear time invariant DAEs, theorems giving sufficient conditions for existence and uniqueness of sensitivities have been proven. These theorems imply the existence of 'critical' values for the parameters related to qualitative changes in the evolution of the system, specifically bifurcations in the sequence of events.

These results have important implications concerning the application of sensitivity functions for the optimization of hybrid discrete/continuous dynamic systems. Sufficient conditions for smoothness of the master optimization problem require existence and local continuity of the sensitivity functions. Thus, changes in the sequence of events at these critical values introduce nonsmoothness, confounding gradient-based algorithms. The practical issue of the finite nature of numerical methods may aggravate the nonsmoothness.

Chapter 4

Efficient Calculation of Sensitivity Equations

Parametric sensitivity analysis of large-scale differential-algebraic equation systems (DAEs) is important in many engineering and scientific applications. The information contained in the parametric sensitivity trajectories is useful for parameter estimation, optimization, process sensitivity studies, model simplification, and experimental design. For example, the control parameterization approach for numerical solution of optimal control problems can require sensitivity information with respect to hundreds of parameters. Inexpensive calculation of sensitivities is particularly important in many areas of chemical, mechanical and electrical engineering, and economics because the systems of interest are typically modeled with $10^2 - 10^5$ equations.

Traditionally, the combined DAE and sensitivity system has been handled by noting that the sensitivities depend on the solution to the DAE system, and thus may be solved using a *staggered direct* scheme in which the linear systems for the sensitivity corrector steps are solved directly after convergence of the nonlinear corrector step [30, 38, 62, 72, 73, 86, 119]. Recently, a method was proposed that substantially reduces the cost of parametric sensitivity analysis [94] relative to these earlier efforts. This method solves the DAEs and sensitivities simultaneously, obtaining the numerical solution from a corrector iteration on the combined nonlinear system. Within the context of popular DAE solvers such as DASSL [21], this *simultaneous corrector*

method has lower computational cost than the staggered scheme because it minimizes the number of matrix factorizations that need to be performed along the solution trajectory.

In this chapter, a novel *staggered corrector* method for solving DAEs and sensitivities is developed and demonstrated. In particular, the approach exhibits a computational cost that is proven to be a strict lower bound for that of the simultaneous corrector algorithm described in [94]. The staggered corrector method uses two corrector iteration loops at each step, one for the DAE and a second for the sensitivities. The computational savings result from fewer Jacobian updates in order to evaluate the residuals of the sensitivity equations.

Many large DAEs found in engineering applications have Jacobians that are large, sparse, and unstructured. Although the DAE may contain many equations, the average number of variables appearing in each equation is much lower than the number of equations. Standard linear algebra codes exist that exploit this sparsity to reduce dramatically the computational cost and memory resources required for the solution of such systems [44]. The method for solving sensitivities described in this chapter is particularly useful for, but not confined to, large sparse unstructured DAEs, because the computational time spent updating the Jacobian matrix is a significant portion of the total solution time. The reason for this counterintuitive situation is that state-of-the-art BDF integrators factor the corrector matrix infrequently, and the factorization of large sparse matrices is particularly efficient, while on the other hand the simultaneous corrector method [94] requires the Jacobian to be updated frequently in order to calculate sensitivity residuals.

Also present in this chapter is a code for solving large, sparse DAEs and sensitivities. The DSL48S code is based on the DASSL code but contains several novel features, including the use of the highly efficient staggered corrector method. Examples are presented using this code demonstrating that the staggered corrector iteration is a significant improvement over existing methods.

It is assumed that the reader is familiar with the DASSL code and the algorithms used in that code [21, 94].

4.1 The Staggered Corrector Sensitivity Method

Consider the general DAE system with parameters:

$$f(t, y, y', p) = 0 \tag{4.1}$$

$$\phi(t_o, y(t_o), y'(t_o), p) = 0 \tag{4.2}$$

where $y \in \mathbb{R}^{n_y}$ are the state variables and $p \in \mathbb{R}^{n_p}$ are the parameters. Consistent initial conditions are given by (4.2). If (4.1) is a sparse unstructured DAE, each equation in (4.1) involves on average c variables, where $c \ll n_y$.

Sensitivity analysis on (4.1) yields the derivative of the state variables y with respect to each parameter. This analysis adds the following sensitivity system of $n_s = n_p \cdot n_y$ equations to (4.1):

$$\frac{\partial f}{\partial y} s_i + \frac{\partial f}{\partial y'} s_i' + \frac{\partial f}{\partial p_i} = 0 \qquad i = 1, \ldots, n_p \tag{4.3}$$

$$\frac{\partial \phi}{\partial y} s_i(t_o) + \frac{\partial \phi}{\partial y'} s_i'(t_o) + \frac{\partial \phi}{\partial p_i} = 0 \qquad i = 1, \ldots, n_p \tag{4.4}$$

where $s_i = \partial y / \partial p_i$. Note that consistent initial conditions (4.4) for the sensitivity system (4.3) are derived from (4.2).

A numerical solution of the nonlinear DAE system (4.1) is obtained at each time step of the integration by approximating the solution with a suitable discretization, typically the k-step general backward difference formula (BDF), to yield the nonlinear system:

$$g(y_{(n+1)} = f\left(t_{(n+1)}, y_{(n+1)}, \sum_{j=0}^{k} \alpha_j y_{(n+1-j)}, p\right) = 0 \tag{4.5}$$

where α_j are the coefficients of the BDF formula. The solution of (4.5) is performed iteratively by solving the following linear system at each iteration:

$$J\left(y_{(n+1)}^{(m)} - y_{(n+1)}^{(m+1)}\right) = g(y_{(n+1)}^{(m)}) \tag{4.6}$$

where

$$J = \alpha_0 \frac{\partial f}{\partial y'^{(m)}_{(n+1)}} + \frac{\partial f}{\partial y^{(m)}_{(n+1)}} \tag{4.7}$$

Note that the corrector matrix J is calculated using information from the Jacobian of (4.1). DAE codes such as DASSL use an approximation to J in order to minimize the frequency of updates and factorizations, which are typically the most computationally expensive steps of the integration algorithm. Thus, the corrector solution is obtained by solving:

$$\hat{J}\left(y^{(m)}_{(n+1)} - y^{(m+1)}_{(n+1)}\right) = g(y^{(m)}_{(n+1)}) \tag{4.8}$$

at each iteration, where $\hat{J}$ is an approximation to J that is updated and factored infrequently.

The staggered corrector method proposed in this chapter uses a separate but similar Newton iteration to solve the sensitivity system (4.3). Once the solution to the corrector for the DAE (4.1) has been obtained, the sensitivity system becomes:

$$A \begin{bmatrix} s'_i \\ s_i \end{bmatrix} = -\frac{\partial f}{\partial p_i} \tag{4.9}$$

where

$$A = \begin{bmatrix} \frac{\partial f}{\partial y'_{(n+1)}} & \frac{\partial f}{\partial y}_{(n+1)} \end{bmatrix} \tag{4.10}$$

In order to avoid factoring A, (4.9) may be solved as in (4.8) using the iteration:

$$\hat{J}\left(s^{(m)}_{i_{(n+1)}} - s^{(m+1)}_{i_{(n+1)}}\right) = A \begin{bmatrix} \left(\alpha_0 s^{(m)}_{i_{(n+1)}} + \sum_{j=1}^{k} \alpha_j s_{i_{(n+1-j)}}\right) \\ s^{(m)}_{i_{(n+1)}} \end{bmatrix} + \frac{\partial f}{\partial p_i} \tag{4.11}$$

Put simply, a quasi-Newton iteration is used to converge a linear system. Note that the matrix A and the vector $\partial f / \partial p_i$ need be updated only once per step of

N_{iter}	Number of Newton iterations
N_p	Number of parameters
C_{SRES}	Cost of sensitivity residual evaluation (overall)
C_{RES}	Cost of DAE residual evaluation
C_{BS}	Cost of a backsubstitution
C_{MF}	Cost of LU matrix factorization
C_{MV}	Cost of a matrix-vector product
C_{VA}	Cost of a vector-vector addition
C_{VS}	Cost of a scalar operation on a vector
C_{JU}	Cost of a Jacobian update
C_{RHS}	Cost of an evaluation of $\partial f/\partial p_i$

Table 4-1: *Notation used in calculation of computational cost*

the integrator, after the DAE corrector iteration and before the sensitivity corrector iteration, because they are dependent only on information from the DAE solution. This feature is the main reason that the staggered corrector method is attractive compared with other methods, because the cost of updating A and $\partial f/\partial p_i$ can be high. Further, note that solution of the sensitivity corrector iteration does not require additional factorization of a matrix, since $\hat{J}$ is already available as LU factors from the DAE corrector iteration (4.8).

At each time step of the integrator, the additional cost involved in solving the combined DAE and sensitivity system using the staggered corrector method versus solving the DAE without sensitivities is:

$$C_{\text{SRES}} + 2N_p N_{\text{iter}} C_{\text{BS}} \tag{4.12}$$

where the definitions of the symbols in this equation are given in Table 4-1. The cost of sensitivity residual evaluation is discussed in later sections.

The incremental cost presented in (4.12) is for the corrector iteration alone and assumes that $\hat{J}$ is a sufficiently good approximation to J for this sensitivity corrector iteration to converge without additional corrector matrix factorizations. The additional cost differentials that arise outside the corrector iteration are ignored here but

are discussed in a later section.

4.2 Other Sensitivity Methods

The staggered corrector sensitivity method is an improvement on two other methods that have been developed to solve the combined DAE and sensitivity system. In this section, these methods and how they differ are briefly described.

4.2.1 The staggered direct method

A number of codes solve the sensitivity system directly using a staggered scheme [30, 77, 86]. This method involves solving the nonlinear DAE system (4.1) by approximating the solution with a BDF method, as in (4.5–4.8).

Once an acceptable numerical solution has been obtained from the corrector iteration, the solution to the sensitivity system (4.9) is obtained through the direct solution of:

$$A \begin{bmatrix} s_i' \\ s_i \end{bmatrix} = -\frac{\partial f}{\partial p_i} \tag{4.13}$$

where the BDF discretization is used to eliminate s_i'.

This method is termed the staggered direct method because the DAE corrector is first solved using a Newton iteration, and then the linear sensitivity system solution is obtained directly. Although A is based on the same information as the corrector matrix $\hat{J}$, it is necessary to update and factor A at every step of the integrator because $\hat{J}$ is only updated occasionally. As shown in Section 4.4, the computational cost associated with these additional matrix updates and factorizations makes the staggered direct method unattractive compared to other methods.

A variant on this method was proposed in [88], in which the already factored approximation to the corrector matrix $\hat{J}$ was used in the direct solution of (4.13) for the sensitivities. The fact that no additional matrix factorizations are required makes this method highly efficient; however, since $\hat{J}$ is not equal to the A that defines the sensitivity equations locally, the solution to the sensitivity equations thus obtained is

in reality a solution to the unknown perturbed sensitivity system:

$$\hat{J}\begin{bmatrix} s_i' \\ s_i \end{bmatrix} = -\frac{\partial f}{\partial p_i} \tag{4.14}$$

Therefore, no guarantees may be made concerning the distance between the solution to this system and the true sensitivity system. For example, in [77] it is noted that sensitivities obtained using this method are not sufficiently accurate for the master iteration of control parameterization.

At each step of the integrator, the additional cost involved in solving the combined DAE and sensitivity system using the staggered direct method versus solving the DAE without sensitivities is:

$$C_{\mathrm{JU}} + C_{\mathrm{MF}} + N_p\left(2C_{\mathrm{BS}} + C_{\mathrm{RHS}}\right) \tag{4.15}$$

where the symbols are given in Table 4-1.

4.2.2 The simultaneous corrector method

The method described in [94] combines the DAE and sensitivities to form a combined system which is then solved using a BDF method on each step. The combined system is:

$$F = \left[f(t,y,y',p), \frac{\partial f}{\partial y}s_1 + \frac{\partial f}{\partial y'}s_1' + \frac{\partial f}{\partial p_1}, \ldots, \frac{\partial f}{\partial y}s_{n_p} + \frac{\partial f}{\partial y'}s_{n_p}' + \frac{\partial f}{\partial p_{n_p}}\right] \tag{4.16}$$

where $Y = [y, s_1, \ldots, s_{n_p}]$.

This combined nonlinear system can then be discretized as in (4.4–4.8):

$$G(Y_{(n+1)}) = F\left(t_{(n+1)}, Y_{(n+1)}, \sum_{j=0}^{k} \alpha_j Y_{n+1-j}, p\right) = 0 \tag{4.17}$$

which can be solved using Newton's method by solving the corrector equation:

$$M\left(Y_{(n+1)}^{(m)} - Y_{(n+1)}^{(m+1)}\right) = G(Y_{(n+1)}^{(m)}) \tag{4.18}$$

at each iteration, where

$$M = \begin{pmatrix} J & & & & \\ J_1 & J & & & \\ J_2 & 0 & J & & \\ \vdots & \vdots & \ddots & \ddots & \\ J_{n_p} & 0 & \dots & 0 & J \end{pmatrix} \tag{4.19}$$

where J is defined as in (4.7) and

$$J_i = \frac{\partial J}{\partial y} s_i + \frac{\partial J}{\partial p_i} \tag{4.20}$$

In [94] it is shown that if M is approximated by its block diagonal part, (4.19) achieves 2-step quadratic convergence for nonlinear problems. Thus, this method allows the factored corrector matrix to be reused for multiple steps.

This method is a significant improvement over the staggered direct method because the need for additional matrix factorizations in order to solve the sensitivity system has been eliminated. However, the disadvantage of the simultaneous corrector method compared to the staggered corrector method is that the former requires the system Jacobian A be updated at every corrector iteration in order to calculate the portion of G due to sensitivity residuals. Although this cost is minor compared with matrix factorization, it is shown below to be a significant cost for large problems.

At each step of the integrator, the additional cost involved in solving the combined DAE and sensitivity system using the simultaneous corrector method versus solving the DAE without sensitivities is:

$$C_{\text{SRES}} + 2N_{\text{iter}}N_p C_{\text{BS}} \tag{4.21}$$

4.3 Methods for Evaluating Sensitivity Residuals

There are three practical methods for evaluating the sensitivity residuals that may be used with either the staggered corrector or the simultaneous corrector methods. As shown in this section, the choice of method can significantly affect the cost of the staggered corrector method.

In this section, it is assumed that it is possible to evaluate the Jacobian of the DAE either analytically or using finite differences, since this evaluation is necessary to solve the DAE without sensitivities. The concern in this section is with the additional information that is required to calculate sensitivity residuals and how to best obtain it.

4.3.1 Analytic evaluation of $\partial f/\partial p_i$

One way to evaluate sensitivity residuals is using the equation:

$$r_i = A \begin{bmatrix} s_i' \\ s_i \end{bmatrix} + \frac{\partial f}{\partial p_i} \tag{4.22}$$

where r_i is the vector of sensitivity residuals, and A is calculated by updating the Jacobian of the DAE. If $\partial f/\partial p_i$ is somehow available, sensitivity residual evaluation is reduced to two matrix-vector products and two vector-vector additions.

This is the most desirable option because it ensures the accuracy of the sensitivity residuals and experience shows that it is typically less expensive than the other residual methods described below. In practice, if automatic differentiation is used to obtain the DAE Jacobian (for example, see [65]), it is not much trouble to extend the automatic differentiation to produce $\partial f/\partial p_i$.

The cost of an analytic sensitivity residual evaluation is:

$$C_{\text{JU}} + N_p \left(C_{\text{RHS}} + 2C_{\text{MV}} + 2C_{\text{VA}} \right) \tag{4.23}$$

4.3.2 Finite differencing for $\partial f/\partial p_i$

If it is not possible or desirable to evaluate $\partial f/\partial p_i$ analytically, it is still possible to calculate the sensitivity residuals as in (4.22). Doing this requires that $\partial f/\partial p_i$ are evaluated via a finite difference approximation. For example, a first order forward finite difference approximation gives the following equation for the sensitivity residuals:

$$r_i = A \begin{bmatrix} s_i' \\ s_i \end{bmatrix} + \frac{f(t, y, y', p + \delta_i e_i) - f(t, y, y', p)}{\delta_i} \tag{4.24}$$

where δ_i is a small scalar quantity, and e_i is the ith unit vector.

The cost of the finite difference sensitivity residual evaluation given in (4.24) is:

$$C_{\text{JU}} + N_p \left(C_{\text{RES}} + 3C_{\text{VA}} + C_{\text{VS}} + 2C_{\text{MV}} \right) \tag{4.25}$$

4.3.3 Directional derivatives

In [94] it is noted that it is possible to determine sensitivity residuals using a directional derivative finite difference approximation. For example, the sensitivity residuals could be approximated as:

$$r_i = \frac{f(t, y + \delta_i s_i, y' + \delta_i s_i', p + \delta_i e_i) - f(t, y, y', p)}{\delta_i} \tag{4.26}$$

The cost of the directional derivative sensitivity residual evaluation given in (4.26) is:

$$N_p \left(C_{\text{RES}} + C_{\text{VA}} + C_{\text{VS}} \right) \tag{4.27}$$

Note that directional derivative residuals are less costly than finite difference residuals, and therefore directional derivatives are the preferred method if analytical derivatives are unavailable. However, it is demonstrated in Section 4.6 that analytic residuals are preferred over directional derivative residuals from a cost standpoint.

4.3.4 Comparison of methods for evaluating sensitivity residuals

The above expressions for the cost of evaluating sensitivity residuals are not the same as the C_{SRES} term in equations (4.12), (4.15), and (4.21).

For analytic sensitivities, these costs are:

$$C_{\text{SRES}}^{\text{StCorr}} = C_{\text{JU}} + N_p \left(C_{\text{RHS}} + N_{\text{iter}} \left(2C_{\text{MV}} + 2C_{\text{VA}}\right)\right) \tag{4.28}$$

$$C_{\text{SRES}}^{\text{SimCorr}} = N_{\text{iter}} \left(C_{\text{JU}} + N_p \left(C_{\text{RHS}} + 2C_{\text{MV}} + 2C_{\text{VA}}\right)\right) \tag{4.29}$$

The above equations show that the staggered corrector method has an advantage over the simultaneous corrector method in the calculation of sensitivity residuals because it is not necessary to update the Jacobian and the vector $\partial f / \partial p_i$ at each corrector iteration.

For finite difference sensitivities, the sensitivity residual costs are:

$$C_{\text{SRES}}^{\text{StCorr}} = C_{\text{JU}} + N_p \cdot N_{\text{iter}} \left(C_{\text{RES}} + 3C_{\text{VA}} + C_{\text{VS}} + 2C_{\text{MV}}\right) \tag{4.30}$$

$$C_{\text{SRES}}^{\text{SimCorr}} = N_{\text{iter}} \left(C_{\text{JU}} + N_p \left(C_{\text{RES}} + 3C_{\text{VA}} + C_{\text{VS}} + 2C_{\text{MV}}\right)\right) \tag{4.31}$$

Therefore the staggered corrector method has an advantage over the simultaneous corrector method when the sensitivity residuals are calculated with finite differencing, because it needs to update the Jacobian only once.

For directional derivative sensitivities, the sensitivity residual costs are:

$$C_{\text{SRES}}^{\text{StCorr}} = N_p \cdot N_{\text{iter}} \left(C_{\text{RES}} + C_{\text{VA}} + C_{\text{VS}}\right) \tag{4.32}$$

$$C_{\text{SRES}}^{\text{SimCorr}} = N_p \cdot N_{\text{iter}} \left(C_{\text{RES}} + C_{\text{VA}} + C_{\text{VS}}\right) \tag{4.33}$$

There is no cost advantage for the staggered corrector method when directional derivative sensitivity residuals are used. Also, since this method requires fewer operations and no Jacobian update, it is preferred over the finite differencing of $\partial f/\partial p_i$ method for calculating sensitivity residuals for either the staggered or simultaneous corrector method.

In practice, for large problems the Jacobian and residual evaluations are expensive, and therefore the ability to reuse A and $\partial f/\partial p_i$ during the corrector iteration results in significant cost savings.

4.4 Cost Comparison of Sensitivity Methods

This section compares the computational costs of the staggered direct method and the simultaneous corrector method with the staggered corrector method. The cost differences presented are for one step of the integration, assuming that the step passes the corrector and truncation error tests, and that (4.22) is used to calculate the sensitivity residuals.

The cost comparison measures presented in this section have been derived so that they may be tested in numerical experiments. In the numerical experiments, the important statistic to compare is the ratio of the additional cost of integrating state and sensitivity systems simultaneously to the cost of integrating a state system alone. This ratio is:

$$\psi_{n_p} = \frac{\text{Additional time for sens. integration}}{\text{Time for state integration}} = \frac{(\tau_{n_p} - \tau_0)}{\tau_0} \tag{4.34}$$

where τ_{n_p} is the time for the integration of the sensitivity and state system with n_p parameters, and τ_0 is the time for the integration of the DAE without sensitivities.

4.4.1 Staggered direct versus simultaneous corrector

The state integration is dominated by the cost of matrix factorizations, which occur for many DAEs on average every 5-10 steps. For large sparse systems, the cost of matrix factorization has been frequently reported to be approximately 90% of the total solution time, and the numerical results (Section 4.6) indicate that the balance is dominated by Jacobian evaluations. The staggered direct method factors the matrix at every step, so that between 5-10 times more evaluations and factorizations are performed during the integration than would be performed to solve the DAE alone. Therefore a lower bound on ψ_{n_p} for the staggered direct method is:

$$\psi_{n_p} = \frac{N_{\text{steps}} - N_{\text{factorizations}}}{N_{\text{factorizations}}} \tag{4.35}$$

where N_{steps} is the number of BDF steps and $N_{\text{factorizations}}$ is the number of corrector matrix factorizations in the state integration. In the limit as the number of equations becomes large the staggered direct method can do no better than this ratio (typically about 4-9).

The same cost difference estimate can be performed for the simultaneous corrector method with analytic derivatives. The additional cost of sensitivities in this method is dominated by the need to update the system Jacobian at each iteration of the corrector. However, the cost of these extra Jacobian updates is dominated by the cost of matrix factorizations in an asymptotic sense, and a lower bound on ψ_{n_p} for the simultaneous corrector method is therefore zero.

4.4.2 Simultaneous corrector versus staggered corrector

As in the simultaneous corrector method, the asymptotic lower bound on ψ_{n_p} for the staggered corrector method is also zero. However, if the cost of matrix factorizations (which is the same in both methods) is ignored, ψ_{n_p} for both methods is dominated by the cost of Jacobian updates and backsubstitutions. Therefore, the ratio of ψ_{n_p} for the two methods is approximately:

$$\frac{\psi_{n_p}^{SimCorr}}{\psi_{n_p}^{StCorr}} = \frac{N_{iter}(C_{JU} + n_p(C_{BS} + C_{RHS})}{C_{JU} + n_p(C_{RHS} + N_{iter}C_{BS})} \tag{4.36}$$

The two methods have the same cost only if $N_{iter} = 1$, and the cost differential should decrease as n_p increases.

The staggered corrector method is also less expensive than the simultaneous corrector method if finite differencing (4.24) is used to calculate the sensitivity residuals. However, there is little difference in cost if directional derivatives (4.26) are used to calculate the sensitivity residuals.

4.4.3 Other cost considerations

The above analysis considers only the cost differences within the corrector iteration. However, there are several other considerations that affect the overall cost of the integration.

The sequence of step sizes taken by the integrator will vary according to which sensitivity method is used. This observation is due to the fact that each method is using the corrector iteration to solve a different nonlinear system. The step size and the choice of when to factor the corrector are dependent upon the errors in the Newton iteration, which are different in each method. The solution obtained to the DAE is still correct to within the tolerances specified for all the methods, but the number of steps and corrector matrix factorizations may vary. In practice, both the simultaneous corrector and the staggered corrector often factor the corrector matrix more often than would be done if just solving the DAE, but the difference in number of factorizations is not typically large.

When the error control algorithm in the integrator is considered, the differences in cost between the staggered and simultaneous corrector methods is even greater than the above analysis would indicate. The staggered corrector method is able to avoid solving the sensitivity system for steps where the corrector iteration or the truncation error check fail for the DAE, while the simultaneous corrector iteration is not capable of such a discrimination. In practice, the overall convergence of the two corrector iterations in the staggered corrector method appears to be more robust than convergence of the single corrector iteration in the simultaneous corrector method.

4.5 Description of DSL48S

The simultaneous corrector method has been implemented in a FORTRAN code called DSL48S. This code is based on the original DASSL code [21], with modifications to handle large unstructured sparse DAEs and to add the staggered corrector method for sensitivities.

The DSL48S code has the following features:

1. The large-scale unstructured sparse linear algebra package MA48 [45] is embedded in DSL48S for solution of the corrector equation. The MA48 package is especially suitable for the types of problems that arise in chemical engineering, as well as many other applications.
2. The staggered corrector method described above has been implemented, and DSL48S offers options for solving the DAE either alone or with sensitivities.
3. The code has been adapted for use within a larger framework for solving a broad class of high-index nonlinear DAEs [47] and dynamic optimization problems.
4. DSL48S offers the option to identify sensitivities with respect to a subset of the parameters as identically zero. This option improves the efficiency of the control parameterization method for dynamic optimization.
5. There is an option to include/exclude the sensitivity variables from the truncation error test. This is included because it may improve the efficiency of some applications that do not require guaranteed accuracy for the sensitivity variables.
6. DSL48S is capable of evaluating sensitivity residuals using either an analytic expression for $\partial f/\partial p_i$, finite differencing to obtain $\partial f/\partial p_i$, or directional derivatives. When using analytic expressions for $\partial f/\partial p_i$, the user must provide an external subroutine that may be called by DSL48S.

The result is a code that conforms closely with DASSL's interface and uses its excellent heuristics for both the DAE and the sensitivity equations. A diagram detailing the algorithm is shown in Figure 4-1.

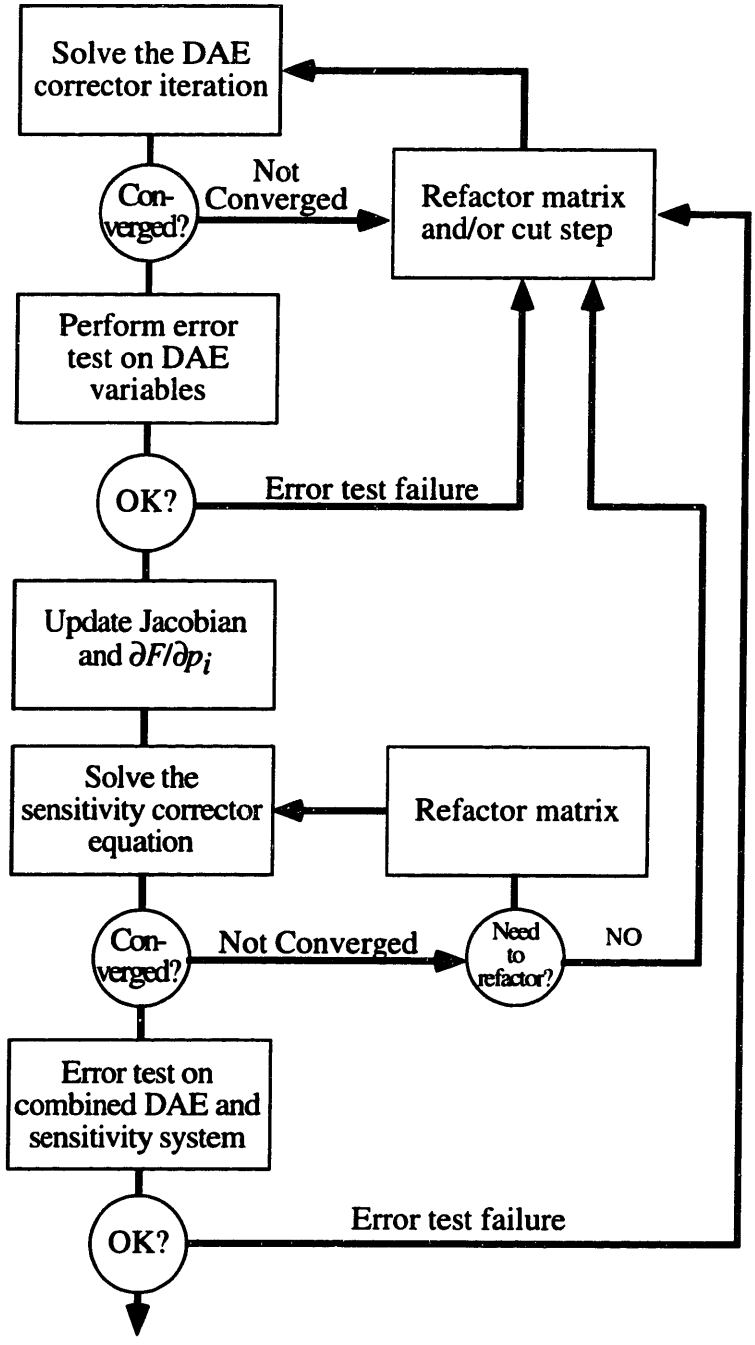

Figure 4-1: *DSL48S algorithm*

At each corrector iteration on the DAE, if the norm of the update is sufficiently small, the corrector iteration is considered to have converged. If not, another iteration is performed, possibly after refactoring the matrix or reducing the step size, as in DASSL. When the corrector has converged, a truncation error test is performed on the state variables. If the truncation error test fails, the corrector matrix is refactored and/or the step size is reduced, and the DAE corrector equation is solved again. If the state variables pass the truncation error test, the Jacobian is updated, and the sensitivity corrector equation is solved in the same manner as the DAE corrector equation. Provision is made to refactor the corrector matrix if it is determined that the corrector iteration is not converging to the predicted values. After the sensitivity variables have passed the corrector convergence test, a truncation error test is performed on the combined state and sensitivity system. If this test fails, the step size is reduced and/or the corrector matrix refactored and the step is attempted again.

The algorithm contains several features designed to minimize wasted computations due to corrector convergence failure or error test failure. An error test is performed on the DAE before the sensitivity corrector iteration is started because an error failure on the DAE will usually cause an error failure on the combined system. The error check on the DAE is inexpensive compared with the wasted work that would be done in solving the sensitivity system if the error failure was not caught at this stage. It was found empirically that an approximation to the corrector matrix that is sufficient to converge the corrector iteration on the DAE on rare occasions may not be sufficient to converge the corrector iteration on the sensitivities. Provision is therefore made to update and re-factor the corrector matrix without reducing the step size if the sensitivity corrector iteration does not converge.

4.6 Numerical Experiments

The DSL48S code was tested on all of the example problems in [94], and produced the same answers as the DASSLSO code presented in that paper. To compare the code and the staggered corrector iteration, the problem was tested on a large-scale pressure-swing adsorption problem [13, 81]. Included in the problem are a number of partial differential equations which are discretized spatially using a backward difference grid. The expressions in the DAEs of this problem are complicated and lead to a sparse unstructured Jacobian. The problem is scalable by the number of spatial mesh points in the adsorbers, and several different problem sizes were chosen. The number of equations in this system is $30 \cdot N + 9$, where N is the number of mesh points in the backward difference grid. The number of nonzero elements in the Jacobian of this system is $180 \cdot N + 11$, and therefore the average number of variables in each equation c is approximately 6.

In order to compare the staggered corrector method with the simultaneous corrector method, an option exists in DSL48S to use the simultaneous corrector method as described in [94]. The code was extensively tested with both the simultaneous and staggered corrector options.

The results of two different numerical experiments are reported. In the first experiment, the solution times were recorded as the size of the DAE was increased. In the second experiment, the size of the DAE was fixed, and the solution times were recorded as the number of parameters was increased.

The performance measures that are reported for the first experiment are ψ_1 and ψ_2/ψ_1 for both the staggered and the simultaneous corrector method, as well as the number of steps (N_{steps}), the number of corrector matrix factorizations (N_{MF}), and the number of Jacobian updates (N_{JU}). In both the simultaneous and the staggered corrector methods, the incremental cost for solving one sensitivity should be much higher than the incremental cost for each additional sensitivity because the same number of Jacobian updates are performed provided that $n_p \geq 1$.

# Equations	τ_0	N_{steps}	N_{MF}
309	2.54s	129	36
3009	44.70s	212	33
6009	107.77s	257	34
9009	179.10s	288	35
12009	265.95s	345	30
15009	351.86s	355	33
18009	456.10s	393	30
21009	574.59s	422	32
30009	921.17s	477	31

Table 4-2: *Results for integration of DAE*

The timing was performed by embedding the DSL48S code within the ABACUSS[1] large-scale equation-oriented modeling system. ABACUSS is an example of high level modeling environment with an interpretative software architecture. Rather than automatically generating FORTRAN or C code which is then compiled and linked to form a simulation executable, ABACUSS creates data structures representing the model equations in machine memory, and during numerical solution these data structures are "interpreted" by numerical algorithms to evaluate residuals, partial derivatives, etc. Details of the ABACUSS implementation are given in [13].

Solving the combined sensitivity and DAE system requires more Jacobian evaluations than solving the DAE alone, and hence the use of automatic differentiation to evaluate the Jacobian has much more impact on the sensitivity and DAE solution time than on the DAE solution time. With automatic differentiation techniques, the cost of a Jacobian evaluation is typically 2-3 times a function evaluation, although rigorous upper bounds on this ratio are dependent upon the particular algorithm employed. For the results reported in this chapter, the modified reverse-mode algorithm described in [134] was used, which is particularly well-suited for large sparse Jacobians.

Timing data comparing the staggered and simultaneous corrector method were

[1]ABACUSS (Advanced Batch And Continuous Unsteady-State Simulator) process modeling software, a derivative work of gPROMS software, ©1992 by Imperial College of Science, Technology, and Medicine.

# Equations	Staggered Corrector				Simultaneous Corrector			
	ψ_1	N_{steps}	N_{MF}	N_{JU}	ψ_1	N_{steps}	N_{MF}	N_{JU}
309	0.79	126	38	126	1.52	130	38	326
3009	0.80	204	37	204	1.45	195	35	453
6009	0.83	246	32	246	1.71	253	31	562
9009	0.86	277	33	277	1.53	286	30	617
12009	0.85	309	31	309	1.77	308	30	682
15009	0.89	337	32	337	1.65	338	29	729
18009	0.89	362	30	362	1.74	363	30	776
21009	0.94	398	29	398	1.79	408	30	873
30009	0.89	455	30	455	1.62	457	28	979

Table 4-3: *Results for one parameter sensitivity and DAE system (analytic sensitivity residuals)*

obtained on an HP C160 single processor workstation. The integration tolerances used were RTOL = ATOL = 10^{-5} for all sensitivity and state variables. The computational times obtained were user times for the integration only, and excluded setup times for the problem. Performance data is reported for a wide range covering the size problems typically of interest, ranging from $3 \cdot 10^2$ to $3 \cdot 10^4$ state equations. The results for the integration of the state variables with no sensitivities are given in Table 4-2. The results for the integration of DAE/sensitivity system with one parameter are given in Table 4-3, and those for two parameters are given in Table 4-4. The results in Tables 4-3-4-4 were obtained with analytic derivatives.

Table 4-3 shows a dramatic difference in the computational cost for the staggered and simultaneous corrector method with one parameter. Over a wide range of problem sizes, an average of 30% savings in the integration time was achieved with the staggered corrector method with one parameter. This is largely due to the empirical observation that matrix factorizations of the corrector matrix for this problem scale less than cubically. The cost of a Jacobian update is a significant portion of the cost of a matrix factorization, and thus the ability of the staggered corrector method to reduce the number of Jacobian updates results in significant cost savings.

The second numerical experiment that was performed compared the solution cost

# Equations	Staggered Corrector				Simultaneous Corrector			
	$\frac{\psi_2}{\psi_1}$	N_{steps}	N_{MF}	N_{JU}	$\frac{\psi_2}{\psi_1}$	N_{steps}	N_{MF}	N_{JU}
309	1.13	124	39	124	1.07	131	37	319
3009	1.06	188	33	188	1.10	190	32	441
6009	1.06	237	35	237	0.99	231	31	522
9009	1.08	274	31	274	1.12	274	29	590
12009	1.07	305	31	305	1.00	313	29	677
15009	1.06	325	32	325	1.06	327	29	708
18009	1.09	362	31	362	1.02	366	28	777
21009	1.07	381	31	381	1.02	374	34	815
30009	1.07	444	30	444	1.07	447	29	946

Table 4-4: *Results for two parameter sensitivity and DAE system (analytic sensitivity residuals)*

as the number of parameters increases. The same 3009 equation model as was used in the previous experiment was solved, using $n_p = 1 \ldots 20$. Figure 4-2 shows the incremental cost of adding additional parameters for both methods. As (4.36) indicates, the cost differential is more significant for fewer parameters. For ease of comparison, these results were obtained with the truncation error control on the sensitivities turned off.

A comparison was also made of the use of analytical residuals and directional derivative residuals within the staggered corrector method. Table 4-5 shows the results of integrating the same 6009 equation pressure swing adsorption system with an increasing number of parameters. The ψ_{n_p} statistic is reported for both the analytical and the directional derivative residual methods. These results show that the analytical method was favored over directional derivatives for all the tested problems, and that the relative difference increases as n_p increases. Note that the results in this table are not consistent with the results in Tables 4-3 and 4-4 because the parameters used in the problem were different.

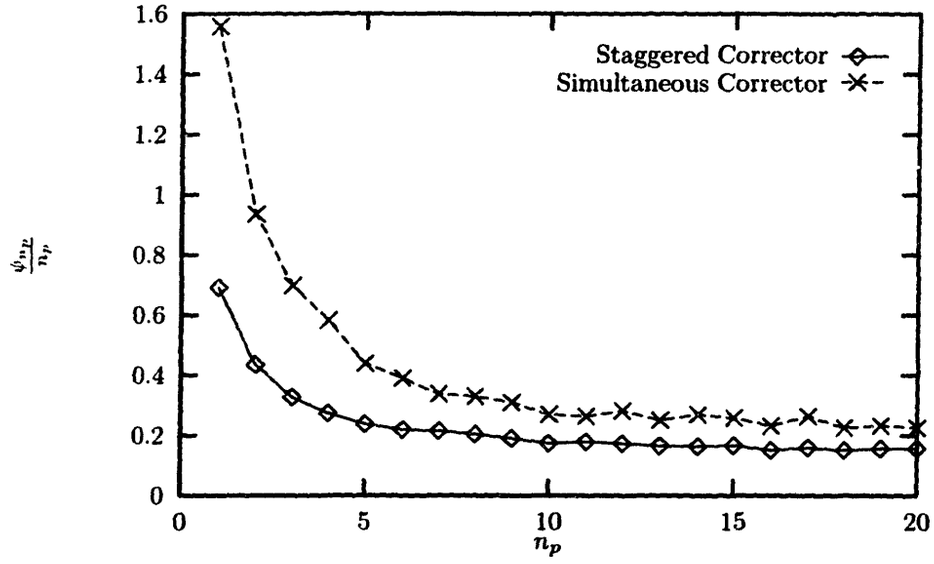

Figure 4-2: *Sensitivity incremental cost per parameter*

n_p	Analytical Residuals				Directional Derivative Residuals			
	ψ_{n_p}	N_{steps}	N_{MF}	N_{RES}	ψ_{n_p}	N_{steps}	N_{MF}	N_{RES}
1	0.85	248	30	568	1.36	248	30	1516
2	0.99	246	35	540	2.03	246	35	1974
3	1.08	240	31	547	2.62	240	31	2379
4	1.33	247	31	664	3.46	247	31	2924

Table 4-5: *Comparison of analytic residuals and directional derivative residuals for staggered corrector*

4.7 Truncation Error Control on Sensitivities

There has been some uncertainty [77, 94] about whether truncation error control must be maintained on the sensitivity variables as well as the state variables. The reason often given for not including the sensitivity variables in the truncation error test is that as long as the state variables are accurate, the sensitivity variables should be fairly accurate. Furthermore, many applications that use sensitivity information could arguably withstand a small amount of inaccuracy in the values of the sensitivities. For example, in dynamic optimization it may not be necessary to have extremely accurate sensitivity information when the optimizer is far from the solution.

There are significant computational advantages for skipping the truncation error test on the sensitivities. The cost of the test itself is a linear function of problem size, but even more significant cost savings may come from the integrator being able to take larger steps on many problems. The larger step size will result even for problems where the sensitivities never fail the truncation error test, since the test is also used to choose the step size.

It was found during the course of this work that when the sensitivities are solved using the staggered corrector scheme, the truncation error test should be performed on the state variables. When the sensitivities are not included in the truncation error test, the integrator is sometimes able to take large enough steps to miss features of the dynamics of the sensitivity system. This effect is not limited to the staggered corrector method, and should be observed in all of the sensitivity methods detailed in this chapter. The staggered direct and the simultaneous direct methods also use the BDF formula to approximate the time derivatives of the sensitivity variables, which is incorrect if the stepsize is too large.

The problem can be seen by looking at the sensitivities of the following problem:

$$\dot{y} = \sqrt{-2gy}\sin(\gamma) \tag{4.37}$$

$$\dot{x} = \sqrt{-2gy}\cos(\gamma) \tag{4.38}$$

$$\gamma = p_1\left(\frac{t-t_f}{t_o-t_f}\right) + p_2\left(\frac{t-t_o}{t_f-t_o}\right) \tag{4.39}$$

where $p_1 = 0.5$, $p_2 = 0.5$, $t_o = 0$, and $t_f = 0.589$.

Figures 4-3 and 4-4 show that the sensitivities can contain significant error when they are excluded from the truncation error test, even for problems such as this one that are not considered particularly stiff or otherwise hard to solve. The 'kinks' in the trajectories without error control are due to the integrator taking very large steps.

It is very tempting to exclude the sensitivities from the truncation error test because this results in substantial computational savings for some problems. However, doing so in this example resulted in the numerical code giving no warning when the sensitivity trajectories became incorrect, even by large amounts. Therefore, it is recommended that the truncation error test should always be performed on the sensitivities in order to avoid this problem.

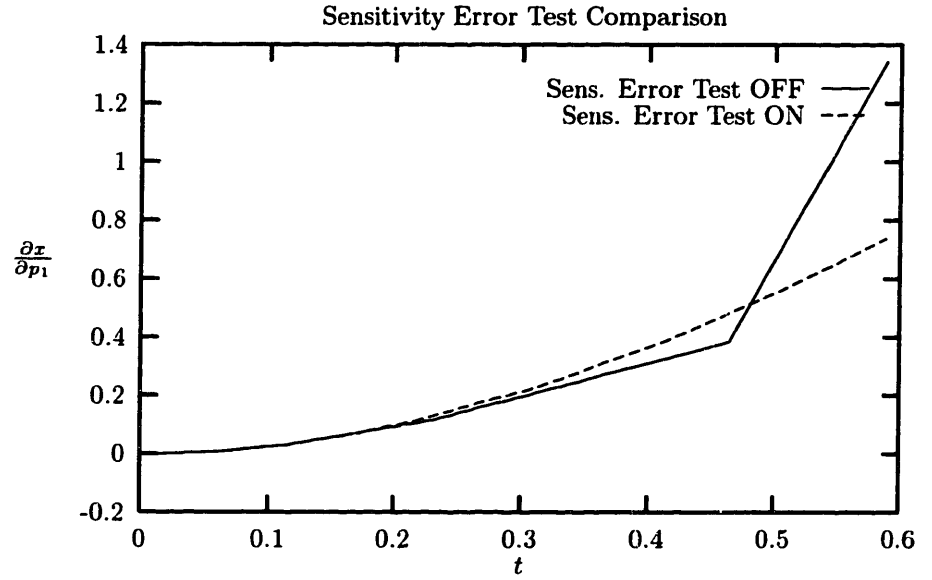

Figure 4-3: *Comparison of x sensitivity of Brachistochrone with and without sensitivity error test*

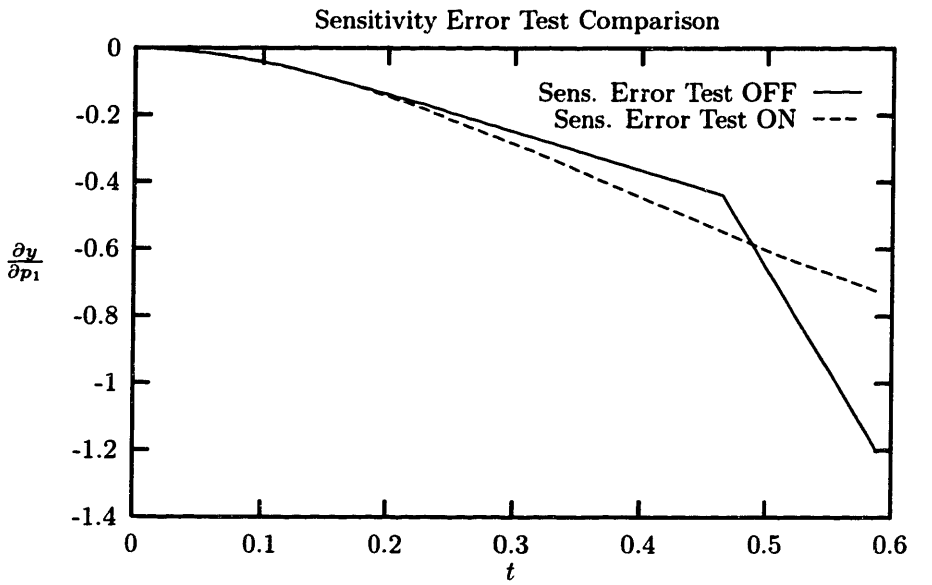

Figure 4-4: *Comparison of y sensitivity of Brachistochrone with and without sensitivity error test*

4.8 Conclusions

The staggered corrector method for numerical sensitivity analysis of DAEs has been shown to exhibit a strict lower bound on the computational cost for the two other methods typically used, the staggered direct and the simultaneous corrector methods. Experience with large sparse problems has shown that the ability of the staggered corrector method to reduce the number of Jacobian updates leads to significant cost savings that are especially apparent with large, sparse DAEs, but also for other types of DAEs.

It is possible to adapt the staggered corrector method for parallel execution. Parallelization may be accomplished by using different processors for each sensitivity calculation and staggering the sensitivity calculation to follow the DAE corrector, similar to the method described in [77].

The code DSL48S is reliable, easy to use, and efficient for sensitivities of large sparse DAEs.

Chapter 5

Numerical Solution of High-Index Differential-Algebraic Equations

This chapter addresses the problem of numerical solution of high-index differential-algebraic equations (DAEs). The need for solving high-index DAEs in practical engineering applications has been somewhat controversial in the literature, since several authors [56, 87, 95, 117, 139] have advocated reformulating process models to avoid high-index DAEs when solving dynamic simulation problems. Whatever the merits of that argument, Chapter 2 has shown that dynamic optimization problems are naturally DAEs, and often high-index DAEs. Furthermore, in Chapter 6 it will be shown that using the control parameterization method to solve dynamic optimization problems efficiently requires a direct method for solving high-index DAEs. In addition to dynamic optimization, the ability to solve high-index DAEs can in fact often be useful for certain types of process simulation, allowing the use of alternate coordinate systems and a broader spectrum of modeling assumptions. Therefore, this chapter is concerned with the development of a numerical method that is capable of solving as broad a class of high-index DAEs as possible. The method described here is called the *dummy derivative* method, which has been developed during the course of this thesis into a reliable method for solving many high-index DAEs.

5.1 Background on DAEs

The focus here is on the numerical solution of general nonlinear high-index differential-algebraic equations (DAEs) of the form:

$$f(\dot{x}, x, y, t) = 0 \tag{5.1}$$

where $x \in \mathbb{R}^{m_x}$ are the differential state variables, $y \in \mathbb{R}^{m_y}$ are the algebraic state variables, $t \in T \equiv (t_o, t_f]$, and $f : \mathbb{R}^{m_x} \times \mathbb{R}^{m_x} \times \mathbb{R}^{m_y} \times \mathbb{R} \rightarrow \mathbb{R}^{m_x+m_y}$. The partitioning of the state variables into algebraic and differential variables does not imply a loss of generality.

There are several basic types of DAEs. The following definitions are from [21]. *Linear constant coefficient DAEs* have the form:

$$A\dot{x}(t) + Bx(t) = f(t) \tag{5.2}$$

where A and B are (possibly singular) square matrices. *Linear time varying DAEs* have the form:

$$A(t)\dot{x}(t) + B(t)x(t) = f(t) \tag{5.3}$$

Although very few systems in practical applications fall into this class, linear time varying DAEs are important because there are proofs that exist only for this class that lead to results and techniques that seem appropriate for more general nonlinear DAEs.

This thesis is concerned mostly with nonlinear DAEs, which typically occur in chemical process modeling. These may be either *fully-implicit*, which have the form:

$$f(\dot{x}, x, t) = 0 \tag{5.4}$$

or *semi-explicit*, which have the form:

$$f_1(\dot{x}, x, t) = 0 \tag{5.5}$$

$$f_2(x, t) = 0 \tag{5.6}$$

The definition of differential index was given in the glossary of DAE terms at the beginning of Chapter 2. Although there are other definitions of indices of a DAE, the differential index is of most relevance in this thesis, and it shall be referred to simply as the index. By this definition, the index of an ODE system is zero. Numerical solution of DAEs is significantly different than solution of ODEs [112, 130], but the solution of DAEs with index ≤ 1 is relatively straightforward using one of several methods. These methods include backwards-difference formula (BDF) methods such as those implemented in DASSL [21, 113] and DASOLV [76], the extrapolation code LIMEX [40], and implicit Runge-Kutta methods such as RADAU [70] and those reviewed by [21] and [70]. Of these, it has generally become accepted that the BDF methods are the most efficient for a broad spectrum of problems [51, 95], and a variant of DASSL called DSL48S (see Chapter 4) was chosen for the numerical integrations performed in this thesis. However, the numerical solution of index > 1 DAEs presents complications [21]. Convergence proofs for the BDF method are given in [21, 78, 96], and generally apply only to problems with index less than two and very restricted forms of higher-index problems. This chapter describes the dummy derivative method for the solution of what are loosely called high-index DAEs, *i.e.*, those with index > 1.

Since it is difficult to solve high-index DAEs numerically, a criterion must be established to distinguish between high-index DAEs and those that can be solved using standard numerical techniques. Based on the definition of the index, it can be shown [21] that a sufficient condition for the index to be at most one is for the Jacobian of the DAE (5.1) to be nonsingular with respect to the highest order derivatives $(\dot{x}, y)$ in the DAE. Note that this is only a sufficient condition, and there are index-1 DAEs for which this criterion does not hold. In [87] a different sufficient condition for (5.1) to be index-1 is presented (although the original presentation in [87] incorrectly claims

this to be a necessary and sufficient condition.) It is interesting to note that while the two sets of index-1 DAEs that satisfy each of these necessary conditions respectively intersect, they do not coincide. In other words, there exist index-1 DAEs that satisfy both, one, or none of these two conditions. So, in principle, both criteria should be applied to cover a broader class of index-1 DAEs.

This criterion is still not of much practical use from a numerical point of view because, due to rounding error, it is very difficult to tell the difference numerically between a singular matrix and one that simply has an extremely high condition number [131]. Also, rank detection computations are very expensive for large-scale systems [9]. Further, singularity of the Jacobian is a local property, but the nonsingularity condition must hold globally. For these reasons, the concept of *structural singularity* is used. The following definition is from [121]:

Definition 5.1 (Structural matrix). *The elements of a structural matrix* $[Q]$ *are either fixed at zero or indeterminate values which are assumed to be independent of one another.*

Therefore, the entries in structural matrices differentiate only between hard zeros (0) and nonzeros ($*$) [139]. A given matrix Q is called an *admissible numerical realization* with respect to $[Q]$ if it can be obtained by fixing all indeterminate entries ($*$) of $[Q]$ at some particular values. Two matrices A and B are said to be *structurally equivalent* if both A and B are numerical realizations of the same structural matrix $[Q]$.

In other words, a property of a matrix is a generic or structural property if it holds in a neighborhood of the nonzero entries of the matrix. A matrix A is *structurally nonsingular* if it is an admissible numerical realization of the structural matrix $[A]$ and there exists a permutation P on the structural matrix such that $P[A]$ has a nonzero diagonal, which is referred to equivalently as a *maximum transversal*, an *output set assignment*, or an *augmenting path*. It can be easily demonstrated that a structurally singular matrix is also singular, but the converse is not necessarily true. The advantage of using structural properties is that evaluating structural properties is much less computationally intensive than numerical evaluation and exact in the

sense that it is not subject to rounding error.

When the sufficient conditions for a DAE to be at most index 1 are modified so that structural singularity of the relevant Jacobian is examined, the criteria are no longer sufficient conditions. That is, there are DAEs which have structurally nonsingular Jacobians which are singular. On the other hand, if the Jacobian is structurally singular, the DAE definitely has an index ≥ 1. It is interesting to note that there are index-1 DAEs for which the Jacobian with respect to the highest order time derivatives is structurally singular, for example:

$$\dot{x}_1 + 2\dot{x}_2 = 0 \tag{5.7}$$

$$x_1 + x_2 = 1 \tag{5.8}$$

Differentiating this class of "special" index-1 DAEs once produces an index-0 DAE (at least in the structural sense, which provides a means to detect them.

Structural nonsingularity can be checked using algorithms for finding a maximum transversal [45]. Even though the index may be underestimated for some DAEs due to numerical singularity, structural nonsingularity of the Jacobians is an attractive approach to detect index ≤ 1 DAEs because of the ease of using the relevant criteria from a numerical standpoint and because it has been found that they work well for the vast majority of DAEs of practical interest.

A particular solution of a DAE is defined by (5.1) and a set of initial conditions that hold only at $t = t_0$:

$$\phi(\dot{x}(t_0), x(t_0), t_0) = 0 \tag{5.9}$$

and also additional hidden constraints that are obtained by differentiating some of the equations of the DAE. The initial condition (5.9) must be *consistent*, which practically means that it must allow us to find *consistent initial values* for the state variables and their time derivatives at t_0 so that numerical integrations may be started smoothly. Consistent initial values are defined as [26]:

Definition 5.2 (Consistent Initial Values). *Initial values x_0 are consistent if they admit a smooth solution in $[0, t]$ for $t \to 0$ and $x(0) = x_0$.*

Unlike the solution of ODEs, not all initial conditions (5.9) of DAEs admit a smooth solution. So, from a practical standpoint solving DAEs requires both specifying a valid (5.9) and then being able to use it to obtain consistent initial values.

In [139], the r components of $x(t_0)$, $\dot{x}(t_0)$ that can be assigned an arbitrary value and still allow a consistent initialization are defined as *dynamic degrees of freedom.* The number of dynamic degrees of freedom is dependent on the index of the DAE, but not related by any explicit formula. The issue of finding a valid set of initial conditions is addressed later in this chapter.

Once a consistent initial condition has been obtained, finding consistent initial values is nontrivial from a practical point of view. The consistent initial values must satisfy the nonlinear algebraic system formed by (5.1), (5.9) and the nonredundant additional constraints formed by differentiating (5.1) one or more times with respect to time; reliable solution of which is difficult with current large-scale root-finding technology. Moreover, consistent initial values are necessary because weak instabilities have been shown to occur when the DAE is solved with a BDF or implicit Runge-Kutta method from inconsistent initial values [96]. There are several methods for finding consistent initial values (see [95] for a review, and also [21, 93]) but this problem is not in the scope of this thesis, and it is assumed to be possible to find a set of consistent initial values for all of the problems discussed.

5.1.1 Reasons for solving high-index DAEs

High-index DAEs arise when the number of truly independent state variables is less than the number of variables that have time derivatives appearing in the DAE, or more simply stated, when there are explicit and/or implicit algebraic relationships among differential state variables. In general, high-index DAEs may result from several sources. They may come from modeling assumptions that were made about the physical system, which in turn may arise from a lack of information about the pa-

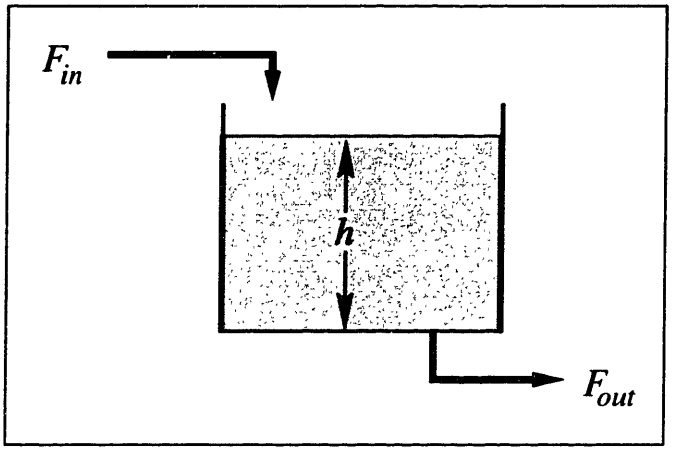

Figure 5-1: *Simple tank model*

rameters or rate processes of a phenomenon, or from a desire to reduce the stiffness of the system to be solved. For example, a simplified model of a total condenser for a pure component vapor is high-index because of the simultaneous specification of both the volume and the vapor fraction of the system [108]. Many high-index DAEs in chemical engineering result from pseudo-steady state, equilibrium, or incompressibility assumptions [56, 87, 108, 117].

More significantly, as shown in Chapter 6, high-index DAEs also arise naturally in the solution of simulation and dynamic optimization problems that include path constraints on the state variables [46, 48], which are common in many engineering applications. In particular, path constraints on ODE embedded systems require either implicit or explicit treatment of high-index DAEs.

One example of high-index DAEs in engineering applications is *prescribed path control.* Prescribed path control problems are those where it is desired to determine the input variable trajectories that will produce a specified output trajectory. For example, consider the simple model of a tank shown in Figure 5-1. A simple model describing the dynamic behavior of such a system is:

$$\frac{dh}{dt} = F_{out} - F_{in} \tag{5.10}$$

$$F_{out} = ah \tag{5.11}$$

where F_{in} is the flow into the valve, F_{out} is the flow out of the tank, and a is a constant determined by the characteristics of the output orifice. This problem is not high-index if the design degree of freedom is satisfied with the constraint:

$$F_{in} = f(t) \tag{5.12}$$

On the other hand, a high-index prescribed path control problem can be created by satisfying the design degree of freedom with the constraint:

$$h = \bar{f}(t) \tag{5.13}$$

Note that the DAE given by (5.10–5.11) and (5.13) is high-index because the differential state variable h is explicitly constrained by (5.13).

In this thesis the interest in high-index DAEs is primarily to create an algorithm that can handle path constrained dynamic optimization problems. However, the above discussion has shown that it is sometimes convenient to deliberately formulate a high-index model for simulation purposes. Therefore, the interest here is on methods that can solve general nonlinear DAEs of arbitrary index.

5.1.2 Methods for solving high-index DAEs

It is possible [21] to solve certain classes of semi-explicit high-index DAEs using BDF and implicit Runge-Kutta methods. However, these methods are currently impractical for general use due to the limitations on the functional form of the DAEs, the need to determine if the DAE has the correct form to be solved using one of these methods, and the fact that modification of numeric codes to solve such systems is not straightforward. In fact, this modification has often been done by reducing the error control tolerances or eliminating the error control entirely on certain variables, which produces solutions of dubious accuracy. Another issue is that the corrector becomes poorly conditioned as $h \to 0$. The conditioning of the corrector scales with $1/h^v$, *i.e.*the algebraic error in the corrector can be so large it will disrupt the truncation

error control [1, 25]. To get around some of these problems, several methods have been proposed that require some form of manipulation of the high-index DAE. Since the objective is to find a method that works for general arbitrary-index DAEs, attention is restricted here for the most part to methods that are not limited to a particular form or index of DAE.

As noted in [21, 60], one method for obtaining numerical solutions to high-index DAEs is to differentiate a subset of the equations in the DAE until an index-1 or 0 DAE is obtained that may be solved using standard numerical techniques. This introduces the concept of *underlying DAEs* (UDAEs):

Definition 5.3 (UDAE). *A DAE $\bar{A}$ is said to be an underlying DAE for the DAE A if for every equation $e \in A$ there is a corresponding $\bar{e} \in \bar{A}$ where $\bar{e} = d^n e/dt^n$, $n \geq 0$.*

For any DAE that is sufficiently differentiable, there are associated UDAEs that are ODEs. These ODEs are termed *underlying ODEs* (UODE). Any solution to the original DAE must also satisfy all associated UDAEs, including UODEs. However, it is possible that some equations in the set of UDAEs may be redundant. For example, consider the index-2 DAE:

$$\dot{x}_1 = x_2 + x_1 \tag{5.14}$$

$$x_1 = f(t) \tag{5.15}$$

One UDAE for this system may be obtained by differentiating (5.15) once:

$$\dot{x}_1 = x_2 + x_1 \tag{5.16}$$

$$\dot{x}_1 = \frac{d}{dt} f(t) \tag{5.17}$$

and another UDAE may be obtained by differentiating both (5.14) and (5.15) once:

$$\ddot{x}_1 = \dot{x}_2 + \dot{x}_1 \tag{5.18}$$

$$\dot{x}_1 = \frac{d}{dt} f(t) \tag{5.19}$$

and a third UDAE may be obtained by differentiating (5.14) once and (5.15) twice:

$$\ddot{x}_1 = \dot{x}_2 + \dot{x}_1 \tag{5.20}$$

$$\ddot{x}_1 = \frac{d^2}{dt^2} f(t) \tag{5.21}$$

Th UDAEs (5.18–5.19) and (5.20–5.21) are also UODEs. Equations (5.16), (5.17), (5.18), (5.19), and (5.21) are nonredundant equations that must be satisfied by any valid numerical solution to (5.16–5.17). In general, the numerical difficulties involved with solving high-index DAEs are associated with finding a solution that satisfies all nonredundant equations in the set of all possible UDAEs.

An example of a high-index DAE that will be used throughout this chapter for illustrative purposes is the following model of a pendulum with a rigid rod derived in Cartesian coordinates (see Figure 5-2):

$$m\ddot{x} + \frac{\lambda}{L} x = 0 \tag{5.22}$$

$$m\ddot{y} + \frac{\lambda}{L} y = -mg \tag{5.23}$$

$$x^2 + y^2 = L^2 \tag{5.24}$$

$$x(0) = x_o \qquad \dot{y}(0) = \dot{y}_o$$

where x and y are the Cartesian coordinates, λ is the tension in the rod, L is the length of the rod, m is the mass of the pendulum bob, and g is the gravitational constant.

The DAE (5.22–5.24) is index-3 because the length constraint (5.24) constrains the x and y differential state variables. An index-1 underlying DAE may be obtained by differentiating the length constraint (5.24) twice in order to obtain a UDAE. When

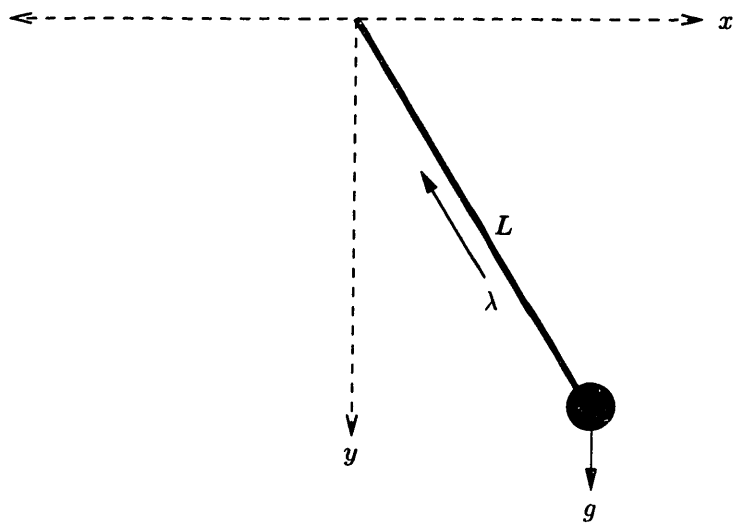

Figure 5-2: *High-index pendulum*

this is done, an index-1 UDAE for the high-index DAE is:

$$m\ddot{x} + \frac{\lambda}{L}x = 0 \tag{5.25}$$

$$m\ddot{y} + \frac{\lambda}{L}y = -mg \tag{5.26}$$

$$2x\ddot{x} + 2\dot{x}^2 + 2y\ddot{y} + 2\dot{y}^2 = 0 \tag{5.27}$$

$$x(0) = x_o \qquad \dot{y}(0) = \dot{y}_o \qquad x(0)^2 + y(0)^2 = L^2 \qquad x(0)\dot{x}(0) + y(0)\dot{y}(0) = 0$$

Note that this UDAE requires the specification of four initial conditions, rather than the two required for the original DAE.

Figure 5-3 shows the numerical results obtained by solving the reduced-index model (5.25–5.27) over a large number of oscillations with the initial conditions $x_0 = 1$ and $\dot{y}_0 = 0$ and the parameters $m = 1$ and $L = 1$ using a standard BDF method integrator. The numerical code produced no error messages or any other sign that numerical difficulties had been encountered, but it is obvious that there is some problem because this is a frictionless system but the bob does not reach the same x point at each oscillation. This can be seen more clearly in Figure 5-4 which shows the length of the pendulum throughout the simulation, which shows that the length invariant is not satisfied. Thus, although the analytic solution of (5.25–5.27) must

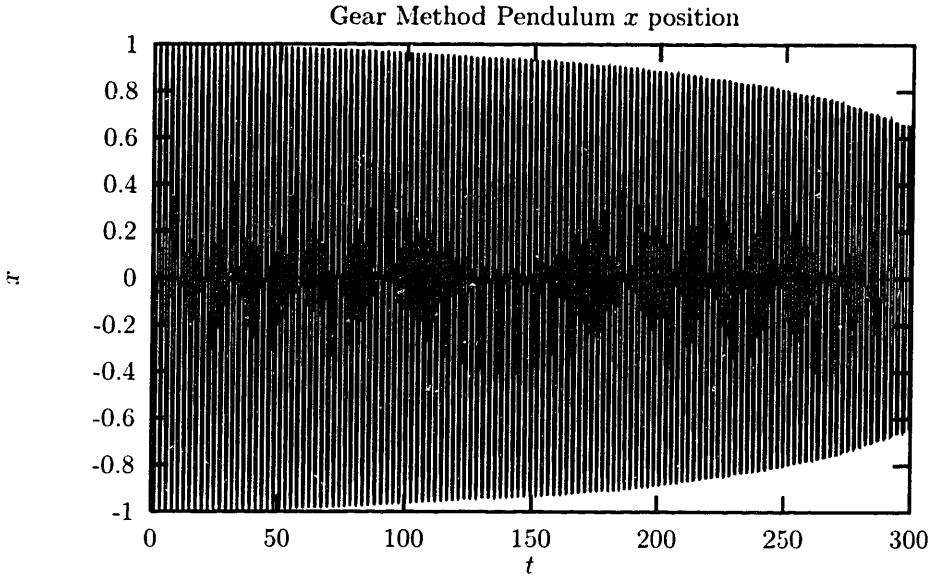

Figure 5-3: *Pendulum x trajectory with Gear method index reduction*

satisfy (5.24), the numeric solution does not.

This phenomenon is called *constraint drift* and was noted in [57, 58, 59]. Several methods have been proposed to handle this problem. In general, nonlinear implicit constraints are not enforced when the problem is discretized. The simplest solution is to integrate a UDAE using step sizes that are hopefully small enough to keep the constraint drift to an acceptable minimum. However, it was shown in [54] that this strategy may not work because differentiating a nonlinear constraint can affect the stability properties of the DAE.

An alternative is the constraint stabilization method proposed in [58]. This method uses the UODE:

$$\dot{x} = f(x, t) \tag{5.28}$$

and the constraints obtained while deriving the UODE from the DAE:

$$g(x, t) = 0 \tag{5.29}$$

where $g : \mathbb{R}^{m_x} \times \mathbb{R} \rightarrow \mathbb{R}^{n_g}$. By introducing new variables $\mu \in \mathbb{R}^{n_g}$ the following system

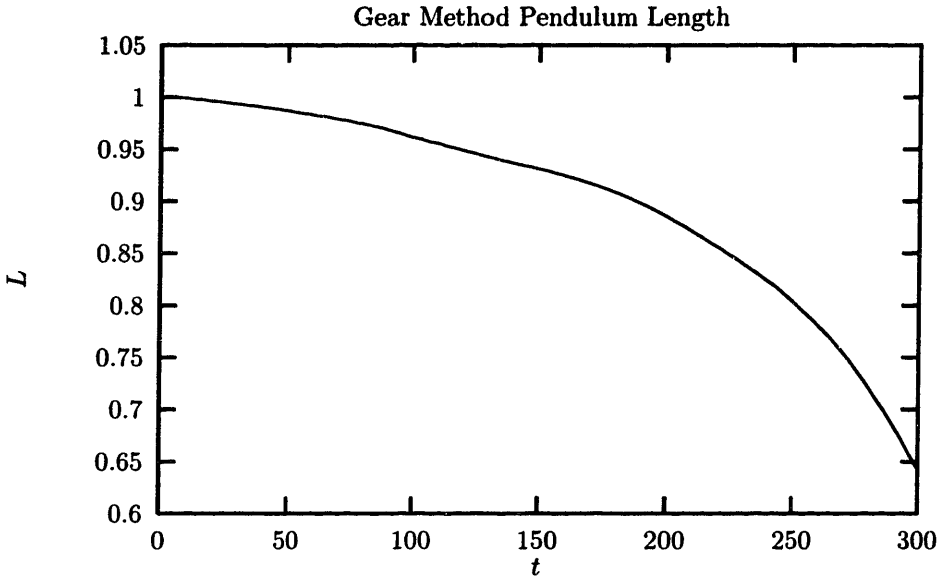

Figure 5-4: *Pendulum length with Gear method index reduction*

may be obtained:

$$\dot{x} = f(x,t) + \left[\frac{\partial g}{\partial x}\right]^T \mu \tag{5.30}$$

$$g(x,t) = 0 \tag{5.31}$$

which is a semi-explicit index-2 DAE. The solution of all the algebraic variables in this problem can be shown to be zero, and the constraints and all of their derivatives are explicitly enforced. Therefore, the numerical solution does not exhibit constraint drift and general multistep methods can be used to solve the problem. In [21] this constraint stabilization technique was used to solve the high-index pendulum (5.22–5.24), and it was shown to eliminate the problem of constraint drift. However, a numerical solution to the resulting semi-explicit index-2 DAE was obtained only by excluding the algebraic variables from the integration error control estimates, and solutions at very tight tolerances could not be obtained because the algebraic variables kept the corrector iteration from converging. This could be due to the fact that the condition number of the corrector matrix of a high-index DAE increases very rapidly as the stepsize tends toward zero [1, 25]. Constraint stabilization methods were also discussed and applied to constrained mechanical systems in [54].

Another method for obtaining numerical solutions to high-index DAEs is through the use of *regularizations* [17, 114]. A regularization of a DAE is the introduction of a small parameter into the DAE in such a way that solution of the regularized DAE approaches the solution of the high-index DAE as the parameter approaches zero. Regularization is a standard method that has been employed by engineers for decades, such as the use of a weir equation for a vessel filled with an incompressible fluid and any "controller" type equation [53]. Essentially regularization creates a very stiff system where the solution decays very quickly to the solution manifold of the high-index DAE. Regularizations have been applied with mixed success to high-index DAEs [21], but their use has not advanced to the point where they may be easily applied to arbitrary-index general DAEs. Moreover, in [2] it is shown that the method of dummy derivatives described later in this chapter is more efficient numerically than a regularization technique, due to the high degree of stiffness of the regularized system.

There have been several attempts to develop numerical methods for direct solution of high-index DAEs. Projected Runge-Kutta collocation methods have been shown to work for some classes of high-index DAEs [4, 90]. Another interesting approach is the least squares projection technique described in [11, 27, 28, 29]. This method works by determining a local set of coordinates for x and then solving the *derivative array* equations numerically using least-squares for $\dot{y}$. Although these approaches seem promising, neither has been developed to the point where it is as reliable and easy to use as BDF methods are for index-1 DAEs.

Finally, a method for finding an index-1 DAE with the same solution set as the high-index DAE was described in [5, 58, 60, 139]. This method is termed the *elimination* method of deriving an equivalent index-1 DAE. Essentially the elimination method takes the nonredundant equations in the UDAEs and substitutes them into the original high-index DAE, eliminating some variables and thus creating an index-1 DAE with a reduced number of degrees of freedom. As an example of this method applied to the pendulum (5.22–5.24) the nonredundant equations from the UDAEs

are the ones created by differentiating the length constraint (5.24) twice:

$$2x\dot{x} + 2y\dot{y} = 0 \tag{5.32}$$

$$2x\ddot{x} + 2\dot{x}^2 + 2y\ddot{y} + 2\dot{y}^2 = 0 \tag{5.33}$$

When (5.32–5.33), and (5.22) are used to eliminate $\ddot{x}$ and $\dot{x}$, an equivalent index-1 DAE is obtained:

$$m\left(-\frac{y^2\dot{y}^2}{x^3} - \frac{\dot{y}^2}{x} - \frac{y\ddot{y}}{x}\right) + \frac{\lambda}{L}x = 0 \tag{5.34}$$

$$m\ddot{y} + \frac{\lambda}{L}y = -mg \tag{5.35}$$

$$x^2 + y^2 = L^2 \tag{5.36}$$

$$x(0) = x_o \qquad \dot{y}(0) = \dot{y}_o$$

This system can be explicitly solved for its highest order time derivatives $(\lambda, \ddot{y}, x)$ in terms of $(\dot{y}, y)$ and the parameters to get:

$$\lambda = -\frac{y^3mg - ymgL^2 + m\dot{y}^2L^2}{L\left(y^2 - L^2\right)} \tag{5.37}$$

$$\ddot{y} = \frac{y^4mg - 2y^2mgL^2 + ym\dot{y}^2L^2 + mgL^4}{L^2\left(y^2 - L^2\right)m} \tag{5.38}$$

$$x = \pm\sqrt{-y^2 + L^2} \tag{5.39}$$

$$x(0) = x_o \qquad \dot{y}(0) = \dot{y}_o$$

These equations no longer uniquely determine the highest order time derivatives when $y \rightarrow \pm L$. Therefore, this system is either locally high-index or locally unsolvable when $y = \pm L$. It can easily be seen that no finite number of differentiations of this DAE will produce a UDAE that uniquely defines the highest order time derivatives when $y = \pm L$, since the denominators of all time derivatives of (5.37–5.38) will contain powers of $(y^2 - L^2)$. It appears that this system is locally unsolvable when $y = \pm L$, since the definition of solvability given in [21] would seem to require that the index be finite. This same solvability phenomena shall be seen again in the discussion of

the dummy derivative method.

However, there is another equivalent index-1 DAE that is formed by eliminating $\ddot{y}$ and $\dot{y}$ instead of $\ddot{x}$ and $\dot{x}$ that is solvable when $x = 0$ but is not solvable when $y = 0$. Theoretically the elimination method is valid for solving high-index DAEs without constraint drift, and it does work for small systems such as the one demonstrated above, but in practice the algebraic elimination required is extremely computationally expensive for large nonlinear DAEs.

A more promising method that is the somewhat related method of *dummy derivatives* [97] which is described in the later sections of this chapter. Although this method was originally described in [97], it required considerable development to create a practical automated algorithm. This method allows for solution of a broad class of nonlinear arbitrary-index DAEs efficiently and with guaranteed accuracy, and it does not require expensive algebraic eliminations. Since the method is based on the ability to find the nonredundant UDAE equations that permit the solution for consistent initial values, the next sections of this chapter are used to describe Pantelides' algorithm for obtaining a consistent initial condition for high-index DAEs.

5.2 Consistent Initialization of High-Index DAEs

In general, all of the UDAEs of a DAE must be satisfied at the initial condition, however, only a subset of the UDAE equations are nonredundant. The set of all UDAEs of a DAE is also called the *derivative array.* All the equations in the derivative array that are nonredundant and constrain the state variables or their time derivatives must be found in order to find a numerical solution using the BDF method. This problem is related to the problem of finding a consistent initial condition, which requires finding which equations are differentiated and how many times to constrain independent initial conditions.

The problem of determining which equations in (5.1) and (5.9) need to be differentiated at the initial condition was addressed in a structural sense in [107]. In this section, Pantelides' structural algorithm is described. Also described are modifications to the algorithm, and an implementation of the modified algorithm. In later sections this modified algorithm is incorporated into a general framework for obtaining numerical solutions to high-index DAEs.

5.2.1 Description of Pantelides' algorithm

An algorithm for finding a consistent initialization of DAEs using structural criteria was proposed in [107]. The algorithm detects the presence of a high-index DAE and indicates which equations must be differentiated to obtain a consistent initial condition, provided that this can be done from purely structural information.

At each iteration, Pantelides' algorithm returns the subset of equations in the DAE that need to be differentiated to produce a UDAE with index one lower than the current index. The algorithm terminates when it returns a UDAE that is structurally nonsingular with respect to the highest order time derivatives in the DAE.

Duff [43] has developed an efficient algorithm for finding an augmenting path for a set of equations and variables. Simply described, the algorithm determines whether it is possible find a unique assignment for each variable in the set of interest to a unique equation in the set. It is equivalent to finding a structural permutation P

that gives a nonzero structural diagonal of maximum size.

Pantelides' algorithm uses Duff's algorithm to find structurally singular subsets of equations, or those that are not part of an assignment. These structurally singular equation subsets are then differentiated and replaced by their differentiated subset, and the algorithm applied again until a complete assignment can be made for the highest order time derivatives of the variables appearing in a mixed set of undifferentiated and differentiated equations. This algorithm uses a bipartite graph of equation and variable vertices. In this type of graph, an edge exists between a variable vertex and an equation vertex when a variable is present in an equation, and an assignment is a set of edges such that only one variable is assigned to each equation.

Pantelides' algorithm proceeds by applying Duff's algorithm to the graph representation of the structural Jacobian of the DAE with respect to the highest order time derivatives. The algorithm terminates if Duff's algorithm finds an augmenting path. If an augmenting path does not exist for the current DAE, the structurally singular subsets of equations are differentiated with respect to time, and the algorithm is then applied to this UDAE. The algorithm returns nonredundant UDAEs, each of which must be explicitly satisfied by a consistent initial condition for the DAE. Since it uses structural criteria, Pantelides' algorithm provides only a lower bound on the index and an upper bound on the dynamic degrees of freedom, and may not return the correct result for systems that arise in certain (mostly pathological) situations.

The advantage of Pantelides' algorithm is that it both detects structurally high-index systems and returns a necessary set of equations that must be satisfied by a consistent set of initial values. If this set of equations is also sufficient, it may then be used to formulate an equivalent index-1 DAE using the method of dummy derivatives.

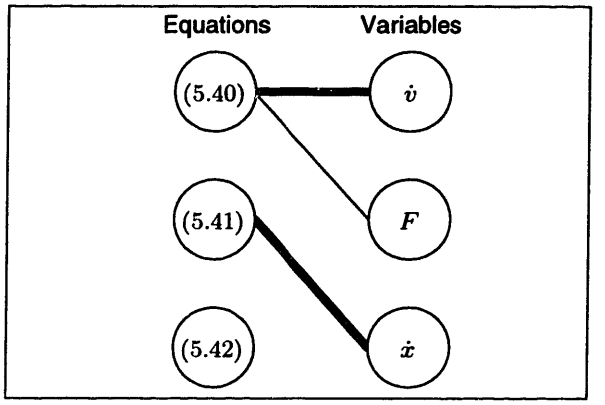

Figure 5-5: *Starting structural graph for index-3 problem*

5.2.2 Example of Pantelides' algorithm

Consider the following very simple system that was given in [97]:

$$m\dot{v} = F \tag{5.40}$$

$$\dot{x} = v \tag{5.41}$$

$$x = \bar{x}(t) \tag{5.42}$$

The solution to this problem may be interpreted as the force F required to make the mass m follow a given trajectory $x(t)$. Note that the selection of x as the input to the system has caused it to be high-index. If instead F had been selected as the input the problem would be an index-0 system of ODEs in state-space form.

Pantelides' algorithm may be applied to the system (5.40–5.42) to determine any additional equations that must be explicitly satisfied by the initial condition. For this example, the structural graph will be shown using circles for the vertices of a graph, and the lines connecting them as the graph edges. The system (5.40–5.42) is structurally represented by Figure 5-5.

The bold edges show one possible attempt to find a matching, but no matching is possible on this graph because (5.42) has no edges connecting it with a variable. Therefore, this graph is structurally singular with respect to (5.42), which is differentiated, giving:

$$\dot{x} = \bar{x}'(t) \tag{5.42'}$$

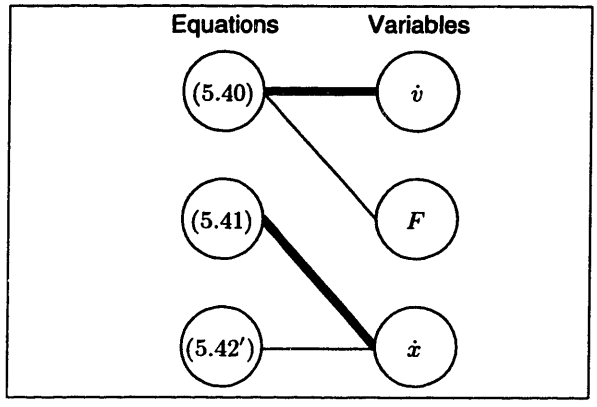

Figure 5-6: *Graph of index-3 system after one step of Pantelides' algorithm*

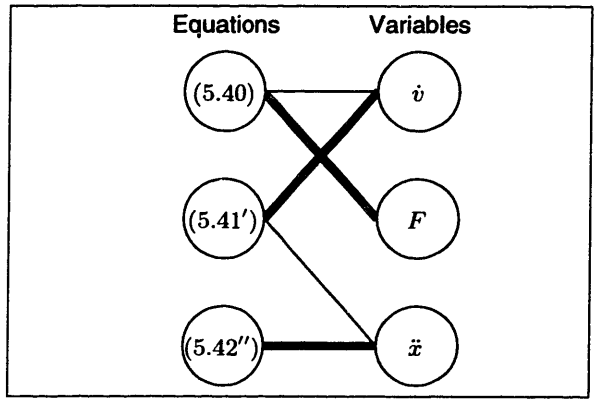

Figure 5-7: *Graph of index-3 system after two steps of Pantelides' algorithm*

which replaces the (5.42) equation node in Figure 5-5 to produce the graph in Figure 5-6 for the next step of the algorithm.

It can be seen from Figure 5-6 that no matching exists because (5.41) and (5.42′) each have only one edge connecting to the same variable, $\dot{x}$. In the scenario depicted in Figure 5-6, the algorithm has assigned $\dot{x}$ to (5.41), and cannot assign (5.42′). Since the two equations cannot both be assigned to $\dot{x}$, they are a structurally singular subset and are differentiated giving:

$$\ddot{x} = \dot{v} \tag{5.41′}$$

$$\ddot{x} = \bar{x}''(t) \tag{5.42″}$$

which results in the graph in Figure 5-7.

The algorithm terminates at this step because a matching is found, shown by the bold edges in Figure 5-7 which connect each equation to a unique variable. The UDAE

returned by the algorithm on the last step is index-1 because it contains an algebraic state variable F. The final UDAE returned by the algorithm must be structurally index-1, since the algorithm is guaranteed to return a UDAE with structural index at most one, and it cannot be index-0 since there is an algebraic variable F in the graph. Therefore, the original DAE (5.40–5.42) was index-3, since (5.42) was differentiated twice, and the final UDAE returned by the algorithm is index-1. The initial conditions must satisfy all of the equations in all of the UDAEs produced by the algorithm, *i.e.*: (5.40-5.42), (5.41′), (5.42′), and (5.42″).

5.2.3 A Modified Pantelides' algorithm

An implementation of Pantelides' algorithm (PALG) and a sparse implementation (SPALG) was reported in [139]. These programs return the structural index and the structural degrees of freedom of a DAE, as well as the numbers of differentiations that have to be applied to each equation in the DAE to obtain the nonredundant equations that must be satisfied by a consistent initialization. Another algorithm (ALGO) was also reported in [139] which is essentially a structural implementation of the elimination index-reduction method of [5, 58]. The combined use of ALGO and PALG returns a list of differentiations and algebraic eliminations required to produce an equivalent index-1 DAE by the elimination method.

A modified version of Pantelides' algorithm has been implemented that is somewhat different from the ALGO/PALG implementation. Since the method of dummy derivatives is being used, rather than the elimination method to derive an equivalent index-1 DAE, the information returned by ALGO is not needed. The implementation takes Pantelides' algorithm and combines it with symbolic differentiation technology. The algorithm takes a symbolic representation of the equations in the DAE, and returns a symbolic representation of the nonredundant equations that must be satisfied by a consistent initial condition, with the caveat that since the algorithm is structural in nature, some required differentiations may not be performed. The modified version of Pantelides' algorithm was implemented in the ABACUSS large-scale equation-oriented modeling system. Section 5.3 discusses the symbolic representation

of equations and the differentiation approach employed.

Modifications to Pantelides' algorithm are also necessary to keep the differentiated DAE from having an order higher than 1, so that the standard DAE solvers may be used. This criteria is enforced simply by introducing new equations and variables as necessary. Thus, when:

$$f_1(\dot{x}, x, y, t) = 0 \tag{5.43}$$

is differentiated:

$$f_1'(\dot{a}, \dot{x}, x, \dot{y}, y, t) = 0 \tag{5.44}$$

$$a = \dot{x} \tag{5.45}$$

is obtained, where (5.45) is a new equation and a is a new variable introduced into the model unless it is already present. This is somewhat more complicated in practice than it would seem because of the desire not to introduce a if it is already in the problem.

A sketch of the index detection and differentiation algorithm REDUCE-INDEX that has been implemented in ABACUSS is given below. It uses the AUGMENTPATH algorithm given in [107]. The inputs to the algorithm are the DAE f, the initial conditions ϕ, the time derivatives of the differential state variables $\dot{x}$, and the algebraic state variables y. Upon successful termination, the algorithm returns the extended DAE $\bar{F}$, its initial conditions D, and the structural index of $\bar{F}$.

REDUCE-INDEX(f,ϕ,$\dot{x}$,y)

1. INITIALIZATION: $F_0 := f$, $z := (\dot{x}, y)$, $D := \phi$, $Index := 0$, $Q := \emptyset$.
2. IF the DAE F_0 is structurally ill-posed, THEN STOP.
3. $\forall \{i\} \in F_{Index}$:
   (a) Apply the AUGMENTPATH algorithm to equation $\{i\}$ and variable set z to obtain a set of structurally singular equations $W \subseteq F_{Index}$.

(b) $\forall \{j\} \in W$:

i. For every time derivative variable $\dot{q} \in \dot{x}$ in equation $\{j\}$ create a new equation $\{k\}$ of the form $a = \dot{x}$, where a is a new variable. Set $z := z \cup \dot{a}$ and $\dot{x} = \dot{x} \cup a$. IF $\{k\} \cap (F_{Index} \cup W) = \emptyset$ THEN set $F_{Index} := F_{Index} \cup \{k\}$ and $Q = Q \cup \{k\}$.

ii. For every algebraic variable $q \in y$ that is in equation $\{j\}$ set $\dot{x} := \dot{x} \cup \dot{q}$, $y := y \setminus q$, and $z := (z \setminus q) \cup \dot{q}$.

iii. Differentiate equation $\{j\}$ to obtain equation $\{l\}$. Set $OldIndex := Index$.

iv. Set $DerivNo$:=the number of times an equation $\{m\} \in F_0$ was differentiated to obtain $\{l\}$. IF $DerivNo > Index$ THEN $Index := DerivNo$.

v. $F_{Index} := (F_{OldIndex} \setminus \{j\}) \cup \{l\}$, $Q := Q \cup \{l\}$.

4. $\forall \{i\} \in F_{Index}$:

(a) Apply the AUGMENTPATH algorithm to equation $\{i\}$ and variable set $\dot{x}$ to obtain a set of structurally singular equations $W \subseteq F_{Index}$.

(b) IF $W \neq \emptyset$ THEN $Index := Index + 1$, GOTO 6.

5. IF $Index > 0$ THEN $D := D \cup (F_{Index} \cap Q)$, $Q := Q \setminus (F_{Index} \cap Q)$.

6. $\bar{F} := F_{Index-1} \cup Q$.

7. $\forall \{i\} \in G$:

(a) Apply the AUGMENTPATH algorithm to equation $\{i\}$ and variable set $\hat{z} = z \cup x$ to obtain a set of structurally singular equations $W \subseteq G$.

(b) IF $W \neq \emptyset$ THEN STOP.

8. RETURN($Index$, $\bar{F}$, D, $F_0, \ldots, F_{Index-1}$).

Like the original algorithm in [107], REDUCE-INDEX will terminate if and only if the corresponding extended system:

$$f(\dot{x}, x, y, t) = 0 \tag{5.46}$$

$$h_i(\dot{x}, x) = 0 \qquad \forall i = 1 \ldots m_x \tag{5.47}$$

is structurally nonsingular with respect to all occurring variables. The exact form of (5.47) is not important since the condition is structural, but in practice it could be thought of as a difference formula relating $\dot{x}$ and x. In [139] the termination condition (5.46–5.47) is noted to be equivalent to nonsingularity of the structural *matrix pencil*:

$$\text{pat}\left(\frac{\partial F}{\partial \dot{z}}\right) + \text{pat}\left(\frac{\partial F}{\partial z}\right) \tag{5.48}$$

where pat(A) is denotes the structural realization of matrix A. Structural nonsingularity of this pencil may be regarded not only as a termination condition for REDUCE-INDEX, but also as a practical solvability condition for the DAE, since most numerical methods (including the BDF methods that used here) require the matrix pencil $\lambda(\partial F/\partial \dot{z}) + (\partial F/\partial z)$ to be regular since it is necessary to invert a matrix of that form to solve the corrector iteration. The REDUCE-INDEX algorithm checks this condition in Step 2.

Upon successful termination, Pantelides' algorithm returns a DAE with structural index at most one [107]. Steps 4-5 check the index of the returned DAE by detecting singularity of the matrix $(\partial F/\partial \dot{x})$. Structural nonsingularity of this matrix is a necessary condition for the DAE to be an (index-0) ODE. If the algorithm returns an ODE, it is preferred to 'backtrack' in the algorithm to find an index-1 DAE since its extended DAE $\bar{F}$ will have fewer equations. Step 5 does this for index-0 DAEs by taking the equations created on the last step and appending them to the initial conditions D, rather than the extended DAE $\bar{F}$. In this way, the equations created on the last step will be enforced in a consistent initialization, but they will not be explicitly enforced during the integration.

Step 7 is a check of the structural nonsingularity of the extended DAE and its initial condition with respect to $(\dot{x}, x, y)$ to ensure that the initial conditions are consistent in a structural sense. This check has proven to be useful in practical modeling activities, although it by no means ensures that it is easy to obtain a set of consistent initial values.

Implementation of this algorithm in ABACUSS has proved to be very useful for several reasons:

- Structurally ill-posed problems are detected before the start of the integration.
- The user is informed that the DAE is high-index, which is important because the formulation of a high-index DAE could be unintentional.
- The correct equations for initializing a high-index DAE are automatically derived and reported.
- The algorithm is necessary for the dummy derivative method.

5.2.4 The dummy derivative algorithm

Although the REDUCE-INDEX algorithm is useful by itself to determine equations that must be satisfied by a consistent initialization, it is not possible to solve directly the DAE $\bar{F}$ that is returned by REDUCE-INDEX if the original DAE was high-index. The reason is that $\bar{F}$ is an overdetermined system; that is, there are more equations than state variables. The elimination method for deriving an equivalent index-1 system from $\bar{F}$ was described above. A somewhat similar but more practically useful method, the method of dummy derivatives was proposed by [97].

The dummy derivative method works by substituting some (or even all) of the time derivatives with "dummy" algebraic variables, effectively removing these time derivatives from the time discretization and resulting in a completely determined system. Unlike the elimination method, which eliminates some variables in the original DAE using the extra constraints of the extended DAE, the dummy derivative method

adds variables so that the entire extended system may be solved. The main advantage of the dummy derivative algorithm is that it does not require computationally expensive algebraic substitutions. The key to the method is to pick a set of time derivatives to replace by dummy algebraic variables.

The method of dummy derivatives is described in detail in [97], but it is summarized here for the purpose of clarity in the rest of this chapter. For convenience, let us rewrite the DAE as an operator equation:

$$\mathcal{F}z = 0 \tag{5.49}$$

where the dependent variables z may appear algebraically or differentiated up to q times. It is assumed that for some $p > q$ that $z(t) : \mathbb{R} \rightarrow \mathbb{R}^n$ and $\mathcal{F}z : \mathbb{R} \rightarrow \mathbb{R}^n$. Let $D = d/dt$ denote the differentiation operator, and for $\nu \in \mathbb{N}^n$ define the operator $\mathcal{D} = \text{diag}(D^{\nu_1} \ldots D^{\nu_n})$. Assign $\mu(\mathcal{F}) \in \mathbb{N}^n$ such that $\mathcal{D}^{\mu(\mathcal{F})}$ are the highest order derivatives appearing in the DAE. REDUCE-INDEX finds the minimal ν such that the differentiated problem $F_{Index} = \mathcal{G}z = \mathcal{D}^{\nu}\mathcal{F}z = 0$ is structurally nonsingular with respect to the highest order derivatives $\mathcal{D}^{\mu(\mathcal{F})}z$.

The dummy derivative algorithm has three steps:

1. Use the REDUCE-INDEX algorithm to differentiate equations in the DAE to reduce the index to one. REDUCE-INDEX returns an overdetermined UDAE $\bar{F}$ which can be used to determine both ν and $\mu(\mathcal{F})$. It also returns a UDAE F_{Index}. It is assumed that F_{Index} is in Block Lower Triangular (BLT) form. Let g_i denote the ith block of of F_{Index} and let z_i denote the vector of highest order derivatives of the unknowns associated with g_i.

2. Sort each g_i in descending order with respect to the number of differentiations required to obtain each equation. If g_i has m differentiated equations, let h_i denote the m first equations of g_i. Define the Jacobian $H_i^1 = \partial h_i / \partial z_i$.

3. Select m columns of H_i^j to make a (square) nonsingular matrix M_i^j. Replace the time derivatives that are associated with the selected columns with dummy

algebraic variables.

4. Increment j and repeat Step 3 using the "predecessor" of h_i, which is $\mathcal{D}^{-(j-1)}h_i = 0$, thus eliminating the last differentiation. The new candidates for replacement with dummy derivatives are time derivatives that are of one order less than those replaced on the previous step. Continue to repeat these two steps until there are no more candidates for replacement.

5. The equivalent index-1 DAE is constructed by including all original and all differentiated equations, with dummy algebraic variables substituted for those variables indicated in Step 3.

5.2.5 Example of dummy derivative algorithm

As an example of the application of the dummy derivative algorithm, consider the pendulum problem (5.22–5.24) in which substitutions have been made to reduce the order of the highest order time derivatives to one. When REDUCE-INDEX is applied to this problem, $\bar{F}$ consists of:

$$a = \dot{x} \tag{5.50}$$

$$b = \dot{z} \tag{5.51}$$

$$m\dot{a} + \frac{\lambda}{L}x = 0 \tag{5.52}$$

$$m\dot{b} + \frac{\lambda}{L}y = -mg \tag{5.53}$$

$$x^2 + y^2 = L^2 \tag{5.54}$$

$$2x\dot{x} + 2y\dot{y} = 0 \tag{5.55}$$

$$2x\dot{a} + 2a^2 + 2y\dot{b} + 2b^2 = 0 \tag{5.56}$$

Block triangularization results in one block g_1, consisting of (5.50–5.53) and (5.56). Since the only differentiated equation in g_1 is (5.56), the vector of highest order

derivatives $z_1 = (\dot{a}, \dot{b}, \dot{x}, \dot{y}, \lambda)$ and the Jacobian H_1^1 is:

$$H_1^1 = \begin{bmatrix} 2x & 2y & 0 & 0 & 0 \end{bmatrix} \tag{5.57}$$

There are two choices of columns that would make M_1^1 nonsingular, because neither x nor y are nonzero for all t. The states x and y are not zero simultancously because of the length constraint. When either x nor y is locally zero, the choice of M_1^1 is clear. When neither x nor y are locally zero, either choice is valid.

If $\dot{a}$ is chosen as the dummy derivative, $z_2 = [\dot{x}]$. Even though the $\dot{a}$ does not appear to have a time derivative of one order less than $\dot{a}$, the substitution $\ddot{x} = \dot{a}$ was made when doing the differentiation to obtain (5.56) and therefore $\dot{x}$ is a time derivative of one order less than $\dot{a}$. The H_2^1 matrix is simply:

$$H_2^1 = \begin{bmatrix} 2x \end{bmatrix} \tag{5.58}$$

and thus the choice of dummy derivatives is clear.

Depending on the choice of M_1^1 matrices, the following *two* equivalent index-1 formulations of (5.22–5.24) can be obtained:

$$a = \bar{x} \tag{5.59}$$

$$b = \dot{y} \tag{5.60}$$

$$m\bar{a} + \frac{\lambda}{L}x = 0 \tag{5.61}$$

$$m\dot{b} + \frac{\lambda}{L}y = -mg \tag{5.62}$$

$$x^2 + y^2 = L^2 \tag{5.63}$$

$$2x\bar{x} + 2y\dot{y} = 0 \tag{5.64}$$

$$2x\bar{a} + 2\bar{x}^2 + 2y\dot{b} + 2\dot{y}^2 = 0 \tag{5.65}$$

and:

$$a = \dot{x} \tag{5.66}$$

$$b = \bar{y} \tag{5.67}$$

$$m\dot{a} + \frac{\lambda}{L}x = 0 \tag{5.68}$$

$$m\bar{b} + \frac{\lambda}{L}y = -mg \tag{5.69}$$

$$x^2 + y^2 = L^2 \tag{5.70}$$

$$2x\dot{x} + 2y\bar{y} = 0 \tag{5.71}$$

$$2x\dot{a} + 2a^2 + 2y\bar{b} + 2b^2 = 0 \tag{5.72}$$

where $\bar{a}, \bar{b}, \bar{x}, \bar{y}$ denote dummy derivatives (that is, they are algebraic state variables).

This example is useful because it illustrates several key points about the dummy derivative algorithm. Since in general the choice of a nonsingular square sub-matrix at each step of the algorithm is not unique, a general high-index DAE will have a "family" of equivalent index-1 DAEs. Furthermore, it is important to recognize that the dummy derivative method relies on numerical nonsingularity of the M matrices, which in general may be a local property of the DAE. Therefore, it may become necessary to perform *dummy derivative pivoting* between the family of equivalent index-1 DAEs, in which different M matrices and therefore a different set of dummy derivatives are selected during the solution of the DAE depending on the local properties of H.

Detecting the need for dummy derivative pivoting would seem to require the rectangular H matrices to be factored at every integration step of the simulation, which is computationally prohibitive. In [97] it is demonstrated that nonsingularity of M_i implies nonsingularity of H_{i-1}, hence only H_{i-1} need be monitored. In general, each matrix M could be as large as the Jacobian matrix of the original high-index system, although in many practical problems it has been found to be much smaller. Described below is a strategy that avoids this expensive factorization at each step, which otherwise would make the dummy derivative method extremely costly in a similar manner

to the staggered direct method for sensitivities described in Chapter 4.

5.3 Differentiation in ABACUSS

Derivation of the family of equivalent index-1 DAEs required by the method of dummy derivatives in an automated fashion requires the application of computational differentiation technology. This section describes the differentiation strategy adopted for implementation of the dummy derivative method in ABACUSS.

Step 3(b)iii of algorithm REDUCE-INDEX above requires derivation of the derivative with respect to the independent variable time of one element of the vector of implicit relationships (5.1). This derivative is defined by the *sparse* inner product of the partial derivative vector of f_i with respect to $\{\dot{x}, x, y, t\}$ and the derivative vector $(\ddot{x}, \dot{x}, \dot{y}, 1)^T$ (note from (5.44–5.45) that $\ddot{x}$ is immediately eliminated). In many engineering applications it can normally be assumed that there are on average only a small number of entries in the partial derivative vector that are not identically zero. The fact that this derivative must also equal zero results in a new equation that is augmented to the original system. Note that subsequent iterations of REDUCE-INDEX may require derivation of the time derivative of this new equation. In addition, differentiation technology is employed to derive the Jacobian of the augmented system (which has similar sparsity) for use by an index-1 solver.

ABACUSS is an example of high level modeling environment with an interpretative software architecture. Rather than automatically generating FORTRAN or C code which is then compiled and linked to form a simulation executable, ABACUSS creates data structures representing the model equations in machine memory, and during numerical solution these data structures are "interpreted" by numerical algorithms to evaluate residuals, partial derivatives, etc. This approach offers significant advantages from the point of view of interactive model development, debugging, and reporting and diagnosis of problems during solution of the model [116]. Further, it allows the functional form of the model to be manipulated efficiently in a general manner throughout the solution process. This latter feature is particularly advantageous for the solution of combined discrete/continuous simulations [15] and inequality path constrained dynamic optimization problems [48]. In both cases, solution of the

IVP is characterized by a sequence of discrete changes to the functional form of the system (5.1) along the trajectory. For example, in the case of the activation of an inequality path constraint at a specific time, the active inequality must be augmented to the DAE and REDUCE-INDEX applied to derive a family of equivalent index-1 models for this new DAE. These changes in the functional form of the DAE poses problems for the code generation approach: either new code must be generated and linked each time the DAE changes, or all possible functional forms must be anticipated *a priori* and appropriate code generated for each one. The former approach is very costly at run time, whereas the latter is in general combinatorial, since, for example, all admissible combinations of inequality constraint activations must be enumerated, even if only a small subset is encountered along the trajectory. On the other hand, the interpretative approach allows for very efficient manipulation of the DAE by merely adjusting the relevant data structures and performing differentiation in an incremental manner as necessary along the trajectory.

Details of the ABACUSS implementation are given in [13]. Sketched here is how a typical problem is processed. The user defines the model equations in natural language using the high level ABACUSS input language. In the first phase, this input is then compiled into an intermediate representation of the model held in appropriate data structures. In the second phase, a particular calculation is selected and the ABACUSS simulation executive transforms this intermediate representation into the run-time data structures for a calculation. In particular, a function f_i in (5.1) is stored in a binary tree using a dynamic data structure. Given values for the unknowns, a subroutine can then "interpret" this binary tree to evaluate the function.

Derivatives of all of the functions are obtained using *automatic differentiation* technology. Automatic differentiation is a method of obtaining derivatives symbolically without resulting in unnecessary and highly inefficient expression "swell" in the resulting expressions. The automatic differentiation in ABACUSS uses an algorithm detailed in [135, 134] that is highly efficient for large sparse equations.

5.4 Dummy Derivative Pivoting

The key to the dummy derivative method is the selection of dummy derivatives. As shown in the Section 5.2.5, the choice is in many cases non-unique, and a practical implementation must select the set of dummy derivatives, monitor the set to see if it is no longer optimal or valid, and switch (pivot) among equivalent index-1 DAEs as necessary.

5.4.1 Selection of an equivalent index-1 model

The problem of selecting an appropriate equivalent index-1 DAE from the family of possible equivalent index-1 DAEs depends on selecting a nonsingular square sub-matrix M_i^1. Since structural algorithms are needed in the REDUCE-INDEX algorithm, it might seem reasonable to choose any structurally nonsingular submatrix of M_i^1. However, as the example in Section 5.2.5 demonstrates, structural criteria are not sufficient to detect the local points at which an equivalent index-1 model is not valid. A reliable method for selecting nonsingular sub-matrices from a rectangular matrix (provided that such a matrix exists) is to use Gaussian elimination with full pivoting. This Gaussian elimination has been implemented using the MA48 code [45], which is capable of LU factoring large sparse rectangular matrices. Although this step is fairly expensive computationally, the overall expense may be reduced by minimizing the number of times that it is done during solution of the DAE, as described below.

5.4.2 Reasons for switching index-1 models

The questions that arise when the example in Section 5.2.5 are examined is what is happening when the M_i^1 matrix of an equivalent index-1 DAE becomes singular, and whether there are reasons to prefer one equivalent index-1 DAEs over another provided their respective M_i^1 matrices are nonsingular.

Dummy derivative pivoting is required because the current matrices M_i^1 may become locally singular at certain points in state space. The consequence of a singular M_i^1 is that the equivalent index-1 DAE may cease to define uniquely all of its highest-

order time derivatives at some points in the solution trajectory. To see this, consider one of the equivalent index-1 models (5.59–5.65) of the pendulum. When this system is solved for explicit expressions for the highest-order time derivatives, the following system is obtained:

$$a = \bar{x} \tag{5.73}$$

$$b = \dot{y} \tag{5.74}$$

$$\lambda = -\frac{L^2 ymg - L^2 mb^2 - y^3 mg}{(-y^2 + L^2)\, L} \tag{5.75}$$

$$\dot{b} = -\frac{yb^2 L^2 + gy^4 - 2gy^2 L^2 + gL^4}{(-y^2 + L^2)\, L^2} \tag{5.76}$$

$$x = \pm\sqrt{-y^2 + L^2} \tag{5.77}$$

$$\bar{x} = \mp\frac{yb}{\sqrt{-y^2 + L^2}} \tag{5.78}$$

$$\bar{a} = \pm\frac{L^2 yg - L^2 b^2 - y^3 g}{L^2\sqrt{-y^2 + L^2}} \tag{5.79}$$

This system does not uniquely define the highest order time derivatives when $y = \pm L$. The situation is similar to that encountered with the elimination method (5.37–5.39), since no finite number of differentiations of (5.73–5.79) will produce a UDAE for which there are not powers of $(-y^2 + L^2)$ in the denominator. In general, all that is known about points where M_i^1 becomes singular is that the highest-order time derivatives of the equivalent index-1 DAE may not be uniquely defined, which can result when the system is locally high-index or locally unsolvable.

Interestingly, the corrector matrix used the BDF method (which is essentially the local matrix pencil $\frac{1}{h}(\partial F/\partial \dot{z}) + (\partial F/\partial z)$ of the system (5.73–5.79)) does not become locally singular when $y = \pm L$. Indeed, high-index models also do not necessarily have a singular corrector matrix. However, standard codes do experience difficulty when solving an equivalent index-1 model in the neighborhood of such points (the chances of the integrator stepping exactly onto one of these points are vanishingly small). The codes can cut the step-size drastically in the vicinity of the singular point, and sometimes fail to integrate past the point. It was observed that the corrector matrix

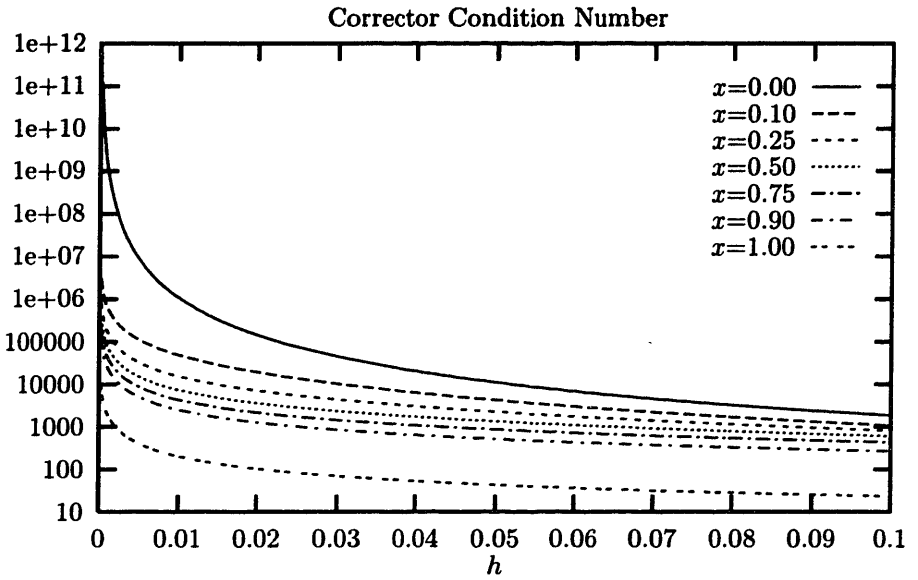

Figure 5-8: *Condition number of the corrector matrix at different points on the solution trajectory of the equivalent index-1 pendulum as a function of the step size* h

becomes ill-conditioned near such points, which can cause inaccuracy in the solution to the corrector and trigger step failures [2].

Figure 5-8 shows the LINPACK estimate of the corrector matrix condition number for (5.59–5.65) as the integrator step size $h \rightarrow 0$. The corrector matrix condition number increases as $x \rightarrow 0$. When the corrector matrix becomes very ill-conditioned (above about $5 \cdot 10^5$ in this case) the corrector matrix may not converge or it may converge to an incorrect point and trigger a truncation error test failure.

5.4.3 Deciding when to switch index-1 models

A previously unanswered problem for the dummy derivative method is how to detect automatically when dummy derivative pivoting is necessary during the solution of the DAE. The brute-force method would be to perform Gaussian elimination on the H_i^1 matrices at every integration step and detect whether a different equivalent index-1 model is chosen than the one most recently solved.

The observation made in the previous section that singular M_i^1 matrices lead to ill-conditioned corrector matrices which can trigger truncation error test failures

provides a convenient heuristic for checking whether dummy derivative pivoting is necessary. The heuristic is to check the factorization of the H_i^1 matrix whenever there are repeated corrector failures or truncation error failures and the integrator calls for a cut in the step size. An example demonstrating the use of this heuristic is given below.

This heuristic is somewhat too conservative. It may call for dummy derivative pivoting at points where none is necessary and the step-size is being reduced for reasons unrelated to the index of the model. On the other hand, the heuristic does not call for pivoting unless the integrator experiences difficulty, which means that some ill-conditioned points may be passed without pivoting. Our experience shows that the heuristic does result in infrequent dummy derivative pivoting checks for most problems. Other situations where there are ill-conditioned corrector matrices are analyzed in [2, 1].

5.4.4 Switching index-1 models during integration

When it has been decided to switch between equivalent index-1 DAEs during the solution, it is unnecessary to restart the integration code. Doing so is undesirable because it requires small step-sizes, many corrector matrix factorizations, and low order polynomial interpolation, increasing the cost of solving the DAE.

One issue with dummy derivative pivoting is that, although it is not necessary theoretically to restart the integration after the pivoting, in practice the DAE integrator may be forced to cut the step size drastically. This happens because the integrator is controlling the truncation error on the differential and algebraic state variables, but not directly on the time derivatives. When one of the time derivatives becomes an algebraic dummy derivative, accurate predictor information does not exist for this new algebraic variable, and the solver is forced to cut the step-size, sometimes drastically. One way to address this is to introduce an extra equation into the system of the form:

$$a = \dot{x} \tag{5.80}$$

where a is a new algebraic variable, for all the time derivatives in the current DAE that have the possibility of becoming dummy derivatives. It is only necessary to include these equations in the truncation error control, since solution of (5.80) is trivial and may be performed after the corrector iteration has finished. The DAE solver is able to take fewer steps after a dummy pivoting operation if it is able to avoid cutting the step by using extrapolation information obtained from equations like (5.80).

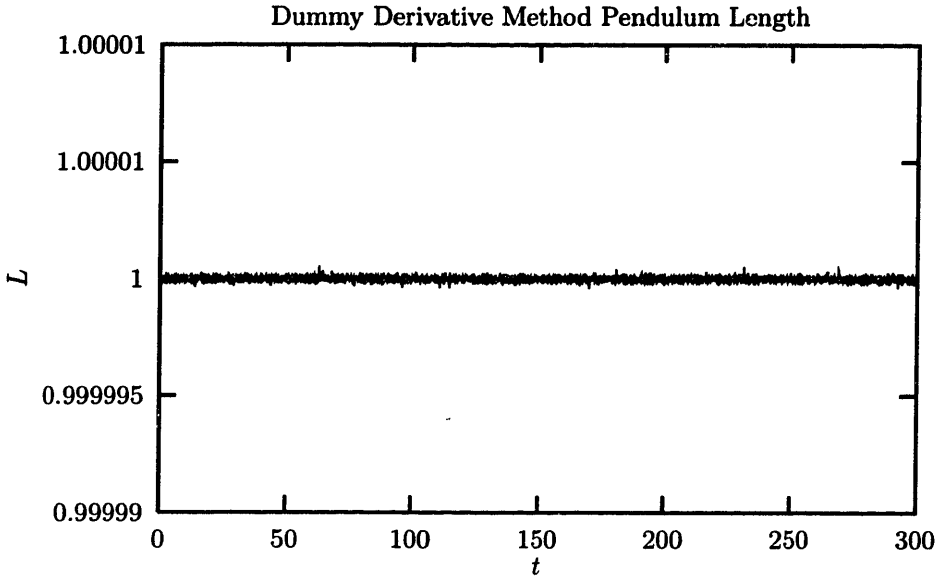

Figure 5-9: *Length constraint in dummy derivative solution of high-index pendulum*

5.5 Pendulum Demonstration and Numerical Results

The use of ABACUSS to solve high-index problems is demonstrated using the pendulum problem (5.22–5.24) with parameters $L = 1$, $g = 1$, and $m = 1$, initial condition $x = 1$ and $\dot{y} = 0$, and integration tolerances $RTOL = ATOL = 10^{-5}$. Figure 5-9 shows that, unlike when this problem was solved using the Gear Method (Figure 5-4), the dummy derivative solution to the problem satisfies the length constraint to within the integration tolerances over many oscillations of the pendulum.

To demonstrate the use of the dummy pivoting heuristic, the index-3 pendulum problem (5.22-5.24) was solved for $t \in (0, 10]$. The different pivoting strategies tested were:

1. Check for pivot when the DAE solver calls for a cut in the step size.
2. Pivot to minimize the condition number of the corrector matrix.
3. Pivot whenever factoring the H matrix at each step leads to a different M matrix.

The heuristic proposed in Section 5.4.3 is used in Strategy 1. Strategy 2 was

Table 5-1: *Different Pivoting Strategies*

Pivot Strat.	Pivot Checks	Pivots	Steps	f-evals	Jac	Trunc. Fail	ΔL	ΔE
1	7	5	256	772	110	7	$1.4 \cdot 10^{-7}$	$4.7 \cdot 10^{-7}$
2	n/a	6	221	656	103	4	$6.3 \cdot 10^{-8}$	$6.2 \cdot 10^{-6}$
3	215	6	221	656	103	0	$6.3 \cdot 10^{-8}$	$6.2 \cdot 10^{-6}$

implemented by monitoring the condition number of the corrector matrix associated with both pendulum equivalent index-1 DAEs. Solution statistics for each of these strategies are given in Table 5-1.

The headings in Table 5-1 refer to the number of pivot checks, the number of pivots, the number of integration steps, the number of function evaluations, the number of evaluations and factorizations of the corrector matrix, and truncation error test failures. There were no corrector convergence failures under any of the strategies. The headings ΔL and ΔE refer to the maximum deviation in the total energy and length constraints during the time interval. The total energy was calculated after the integration using the formula:

$$E = \frac{1}{2}m(\dot{x}^2 + \dot{y}^2) + mgy \tag{5.81}$$

The deviation in the energy was measured relative to the value of the energy on the first step of the integration. The length constraint is present explicitly, and the drift in this constraint was maintained below the integrator tolerances for all of the strategies. The energy constraint is an implicit constraint, which is why it was not enforced as closely as the length constraint, but the drift is still very small.

Strategies 2 and 3 have identical solution statistics because they result in exactly the same pivot times. Strategy 2 checks the pivot selection at every step except immediately following dummy derivative pivots, which accounts for the discrepancy in the number of steps and the number of pivot checks. Table 5-1 shows that the policy of checking the pivots only when the integrator cuts the step (strategy 1) carries a price in terms of the number of steps, the number of residual evaluations, and the

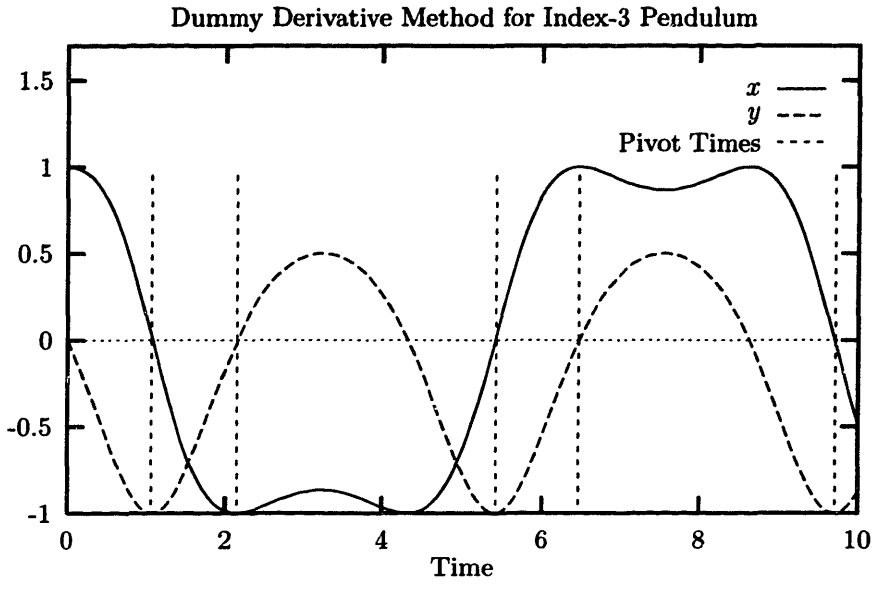

Figure 5-10: *Solution of index-3 pendulum model using ABACUSS*

number of corrector matrix factorizations. However, this price is small compared to the cost of calculating the condition number of the two corrector matrices on every step (strategy 2) or checking the factorization of the H matrix on each step (strategy 3).

Using strategy 1, Figure 5-10 shows the times at which ABACUSS performed dummy derivative pivoting on the pendulum. Figure 5-11 shows that the dummy derivative pivoting was indeed performed at times at which the corrector matrix was ill-conditioned. The "stepwise" appearance of Figure 5-11 is due to the fact that efficient BDF codes do not update the corrector matrix at every solution step.

Table 5-2 gives (strategy 1) results for the solution of the model both with and without the addition of equations (5.80). The integration statistics show a clear benefit from including the extra equations so that the DAE solver is not forced to cut the step after the DAE is changed during dummy derivative pivoting.

Table 5-2: *Results for Solution of Index-3 Pendulum*

	Steps	**f-evals**	**Jac**	**Corr. Fail**	**Trunc. Fail**
Pendulum w/ pivot equations	256	772	110	0	7
Pendulum w/o pivot equations	297	924	165	0	7

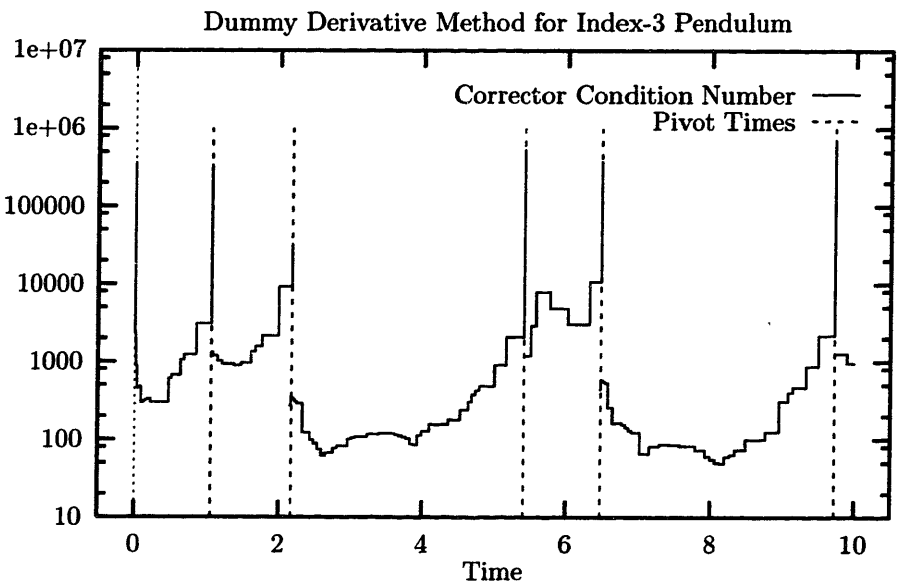

Figure 5-11: *LINPACK estimate of corrector matrix condition number*

Table 5-3: *Example problem solution statistics*

Problem	**Condenser**	**Standard**	**CSTR**	**Batch Column**
Model Equations	5	20	4	297
Additional Eqns. Derived	2	0	5	145
Input Functions	1	4	0	23
Input Funcs. Derived	0	19	0	2
Initial Conditions	1	0	0	9
Integration Steps	363	63	65	96
Residual Evaluations	825	128	144	201
Jacobian Factorizations	62	7	16	28
Corrector Conv. Failures	0	0	0	0
Truncation Test Failures	17	1	2	3
Pivot Checks	17	0	0	3
Pivots	0	0	0	2

5.6 Numerical Examples

This section provides several examples of the use of the dummy derivative method on problems of interest in chemical engineering. The problems used in this chapter have been discussed in the literature under the context of high-index DAEs, but no numerical solutions for most of the problems have been reported. Table 5-3 gives the solution statistics that were obtained by solving these problems using the dummy derivative method implemented in ABACUSS.

5.6.1 Fixed-volume condenser with no liquid holdup

The following simple model of a condenser with fixed volume and negligible liquid holdup was discussed in [108, 139].

$$\dot{N} = F - L \tag{5.82}$$

$$NC_p\dot{T} = FC_p(T_{in} - T) + L\Delta H + kA(T_c - T) \tag{5.83}$$

$$PV = NRT \tag{5.84}$$

$$P = \frac{101325}{760}\exp\left(A - \frac{B}{C - T}\right) \tag{5.85}$$

where (5.82) represents a material balance, (5.83) the energy balance, (5.84) the vapor-liquid equilibrium, and (5.85) an equation of state for the vapor. It was noted in [139] that the assumptions used to derive this model are somewhat questionable, since the vapor phase is modeled as both an ideal gas (5.83) and a saturated vapor (5.85).

The heat capacity was modeled using the relation:

$$C_p = \alpha + \beta(T - 273.15) + \gamma(T - 273.15)^2 + \mu(T - 273.15)^3 \tag{5.86}$$

The feed F is forced with the periodic function:

$$F = 9000 + 1000\sin(\frac{\pi}{2}t) \tag{5.87}$$

The definitions of the variables and parameters are given in Tables 5-4 and 5-5. The parameters for the heat capacity equation, vapor pressure equation, and heat of vaporization are for water and are taken from [50].

The combined model (5.82–5.87) is index-2, and ABACUSS must differentiate (5.84–5.85) to produce an equivalent index-1 DAE. There is one dynamic degree of freedom, and therefore an initial condition must be specified for any one of the state variables (T, L, P, N). Figure 5-12 shows the output that ABACUSS produces for this high-index model. Figures 5-13 and 5-14 show solution trajectories for T and N on the

Table 5-4: *Variables in the high-index condenser model*

Variable	Units	Description
N	$mols$	Molar Holdup in the vessel
T	K	Temperature in the vessel
F	$(mols/hr)$	Feed flow rate
P	Pa	Pressure
L	$(mols/hr)$	Liquid flow rate out of vessel
C_p	$(J/mol \cdot K)$	Vapor heat capacity

Table 5-5: *Parameters in the high-index condenser model*

Parameter	Value	Description
A	7.96681	Antoine Equation Coefficient
B	1668.21	Antoine Equation Coefficient
C	228	Antoine Equation Coefficient
T_{in}	$513\,K$	Feed Temperature
T_c	$280\,K$	Coolant Temperature
V	$100\,m^3$	Vessel volume
U	$10000\,J/(hr \cdot m^2 \cdot K)$	Heat transfer coefficient
A	$50\,m^2$	Heat transfer Area
ΔH	$-40656\,J/mol$	Heat of Condensation
R	$8.3145\,J/(mol \cdot K)$	Universal gas constant
α	33.46	Heat Capacity Correlation Coefficient
β	$0.688 \cdot 10^{-2}$	Heat Capacity Correlation Coefficient
γ	$0.7604 \cdot 10^{-5}$	Heat Capacity Correlation Coefficient
μ	$-3.593 \cdot 10^{-9}$	Heat Capacity Correlation Coefficient

time interval $(0, 10\,hrs]$ that are obtained from an initial condition of $T(0) = 400\text{K}$.

```
The following are the equations in the model:
Input Equation # 1
  CONDENSER.F = 9000 + 1000*SIN(3.14286E+00*TIME/2) ;

Equation # 2
  CONDENSER.$N = CONDENSER.F - CONDENSER.L ;

Equation # 3
  CONDENSER.N*CONDENSER.CP*CONDENSER.$T = CONDENSER.F*CONDENSER.CP*
  (5.13150E+02 - CONDENSER.T) + -4.06560E+04*CONDENSER.L -
  5.00000E+05*(CONDENSER.T - 2.80000E+02) ;

Equation # 4
  CONDENSER.P*1.00000E+02 = CONDENSER.N*8.31450E+00*CONDENSER.T ;

Equation # 5
  CONDENSER.P = 1.33322E+02*EXP(7.96681E+00 - 1.66821E+03/
  (2.28000E+02 + (CONDENSER.T - 2.73150E+02))) ;

Equation # 6
  CONDENSER.CP = 3.34600E+01 + 6.88000E-02*(CONDENSER.T - 2.73150E+02)^2
   + 7.60400E-06*(CONDENSER.T - 2.73150E+02)^3 + -3.59300E-09*
   (CONDENSER.T - 2.73150E+02)^4 ;

Analyzing the structure of the model...

This model is structurally high-index.
 Proceeding with index reduction...

Equation # 7 created by differentiating equation # 5
  -1*(-1*(-1*(1.66821E+03/(2.28000E+02 + (CONDENSER.T - 2.73150E+02))^2))
  *EXP(7.96681E+00 - 1.66821E+03/(2.28000E+02 +
  (CONDENSER.T - 2.73150E+02)))*1.33322E+02)*CONDENSER.$T +
  1*CONDENSER.$P = 0 ;

Equation # 8 created by differentiating equation # 4
  -1*(CONDENSER.N*8.31450E+00)*CONDENSER.$T + -1*(8.31450E+00*
  CONDENSER.T)*CONDENSER.$N + 1.00000E+02*CONDENSER.$P = 0 ;

Before differentiation, this model was index 2.

Equations 1 through 8 form an index=1 model.
Equation 9 is an initial condition.
The following are the specified initial conditions:
Equation # 9
  CONDENSER.T = 400 ;


RESULTS FROM ABACUSS STRUCTURAL ANALYSIS

Number of Variables in model                  : 6
Number of additional variables created        : 0
                                              ----
Total Variables                               : 6

Number of Model Equations                     : 5
Number of Additional Model Equations Derived  : 2
Number of Input Functions Specified           : 1
Number of Additional Input Functions Derived  : 0
Number of Initial Conditions Specified        : 1
Number of Additional Initial Conditions Derived : 0
                                              ----
Total Equations                               : 9

Total Unknowns                                : 9
```

Figure 5-12: *ABACUSS index-reduction output for high-index condenser model*

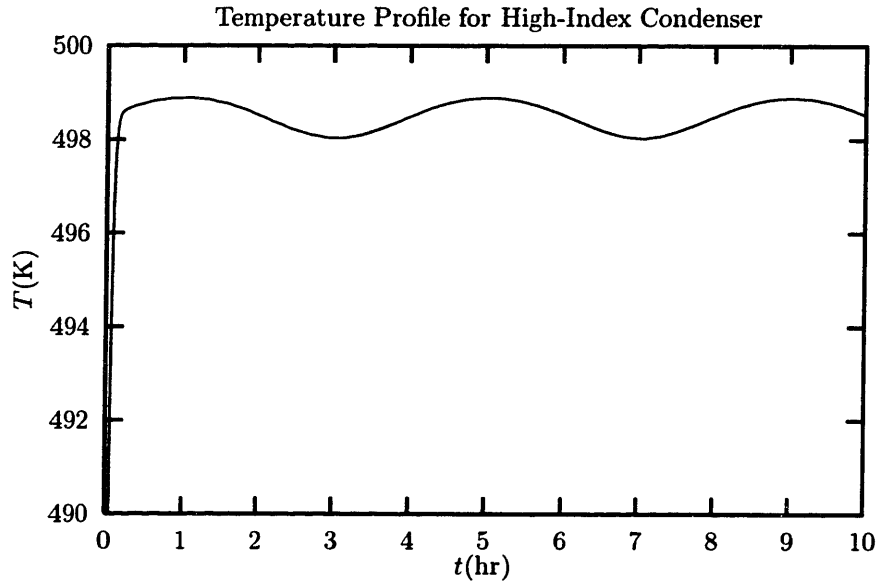

Figure 5-13: *Dummy derivative Temperature profile for high-index condenser*

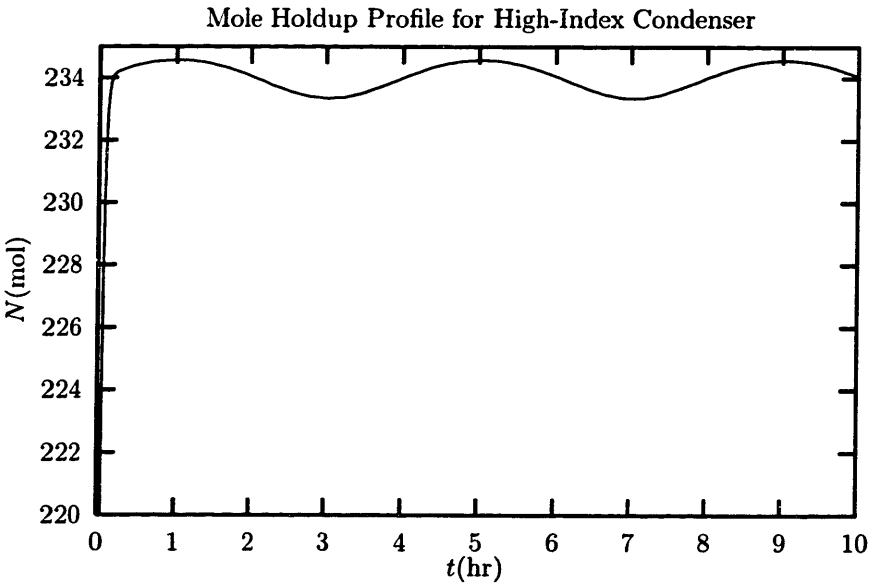

Figure 5-14: *Dummy derivative mole holdup profile for high-index condenser*

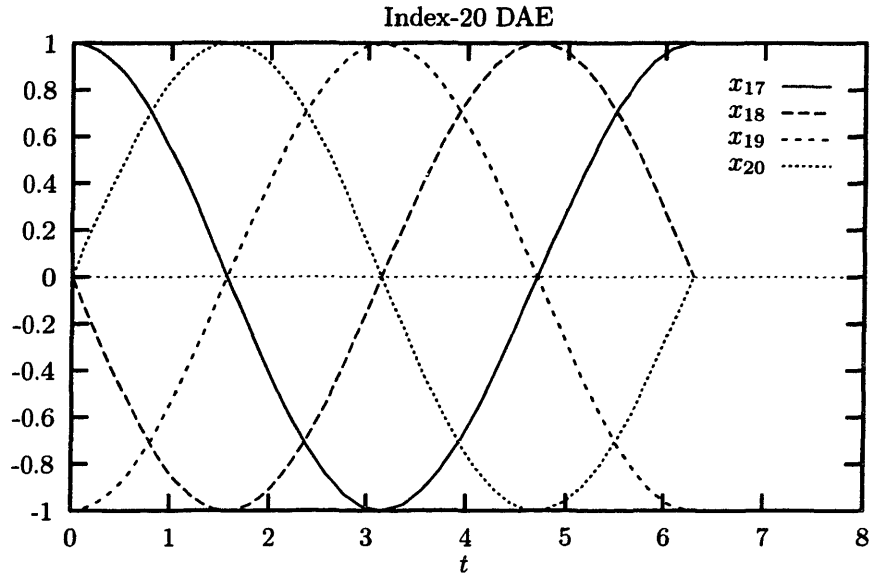

Figure 5-15: *State trajectories for index-20 DAE*

5.6.2 Standard high-index model

A 'standard high-index' model was proposed and solved in [76]. The model is:

$$\dot{x}_i = x_{i+1} \qquad i = 1 \ldots N-1 \tag{5.88}$$

$$x_i = f(t) \tag{5.89}$$

The index of this system is N, and hence the model is primarily interesting because the index is a function of a parameter. An index-20 version of this problem was solved, with $f(t) = sin(t)$, which required ABACUSS to differentiate 19 equations. Some of the state trajectories are shown in Figure 5-15.

Table 5-6: *Parameters for the CSTR example*

Parameter	Value
C_0	$5\,\text{mol/L}$
T_0	$273\,\text{K}$
K_1	$20\,\text{s}^{-1}$
K_2	$3000\,(\text{K}\cdot\text{L})/\text{mol}$
K_3	$1500\,\text{s}^{-1}$
K_4	$3000\,\text{K}$

5.6.3 Continuous stirred-tank reactor

The following model of a well-mixed continuous stirred-tank reactor (CSTR) in which an exothermic reaction takes place:

$$\dot{C} = K_1(C_0 - C) - R \tag{5.90}$$

$$\dot{T} = K_1(T_0 - T) + K_2 R - K_3(T - T_c) \tag{5.91}$$

$$R = K_3 \exp\left(\frac{-K_4}{T}\right) C \tag{5.92}$$

where C is the reactant concentration, T is the temperature in the reactor, C_0 and T_0 are the feed concentration and temperature, R is the rate of reaction, T_c is the coolant temperature, and K_1, K_2, K_3, K_4 are parameters. This model was also discussed in [108], but no numerical solution was given. The values of the parameters used to obtain a numerical solution are given in Table 5-6.

Figures 5-16 and 5-17 show the state variable trajectories obtained from the ABACUSS numerical solution to the problem. An interesting feature of this problem is that it is index-3, and contains no dynamic degrees of freedom, which means that it is not possible to specify any initial conditions. Also, since the H_1^1 matrix for this problem has only one row and one column, ABACUSS is able to determine that pivot checks are unnecessary and thus the numerical solution is very efficient.

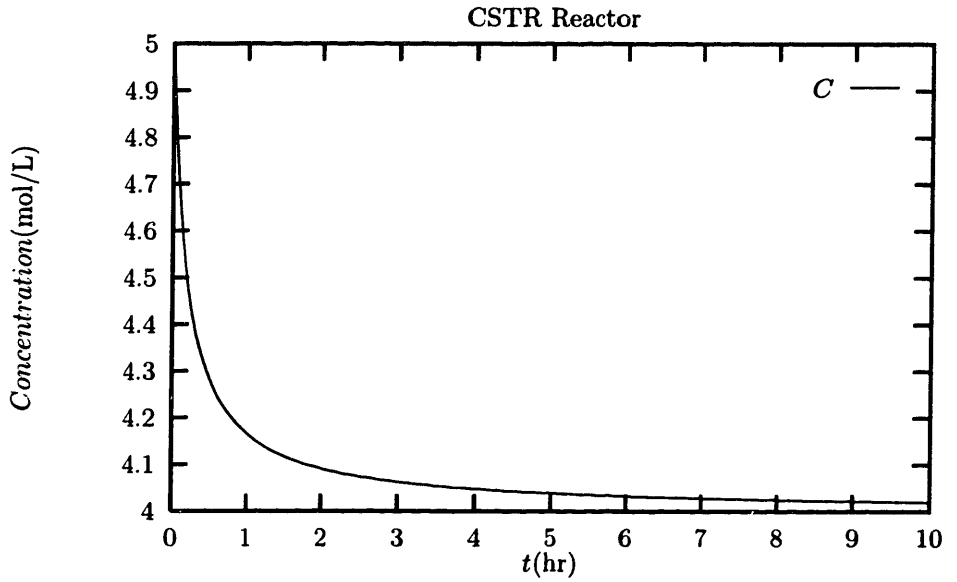

Figure 5-16: *Concentration profile for index-3 CSTR example*

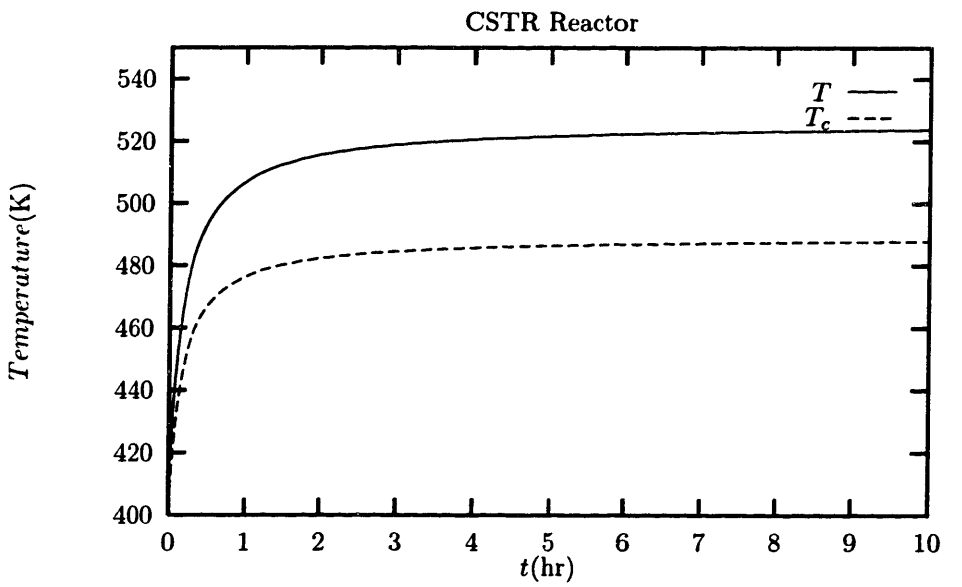

Figure 5-17: *Temperature profiles for index-3 CSTR example*

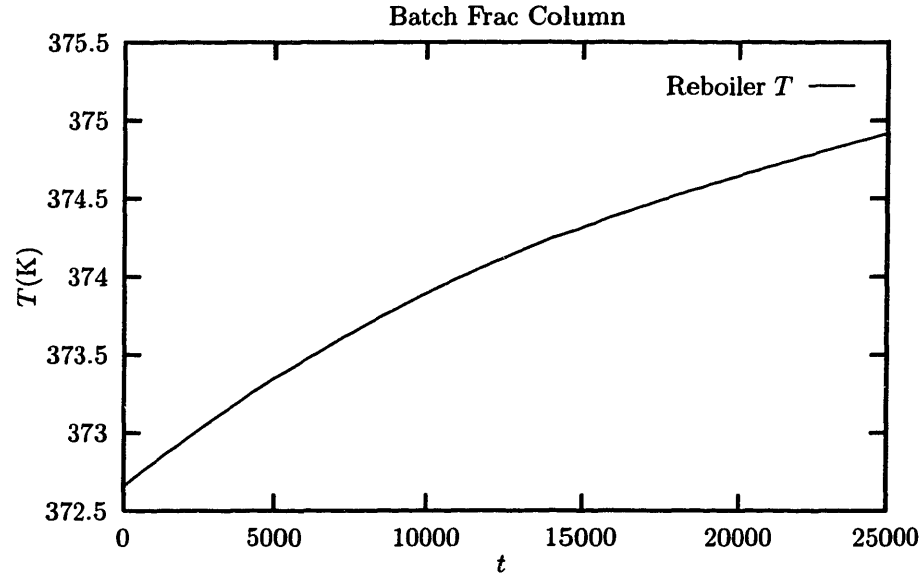

Figure 5-18: *Reboiler temperature profile for BatchFrac Column*

5.6.4 High-Index dynamic distillation column

An interesting high-index example is the BatchFrac model [20]. As written the model is index-2 due to assumptions about the holdup on the trays given in equation (6) of [20]. The fact that the problem is index-2 was not specifically noted in [20], but the problem was solved by replacing the equations for the holdup and enthalpy derivatives with finite difference approximations (see equations (13) and (14) in [20]). The method of dummy derivatives eliminates the need for this approximation. The BatchFrac model was solved in ABACUSS for a five component mixture using ideal thermodynamic assumptions with one tray in the column.

The original index-2 DAE system for this problem consisted of 155 equations, and an additional 75 equations were automatically derived to form the equivalent index-1 system. This model is large because it includes thermodynamic properties as equations, not procedures. The reboiler temperature profile obtained from the solution of the 1-tray system is given in Figure 5-18.

5.7 Conclusions

This chapter has shown that it is possible to reliably solve a large class of arbitrary-index DAEs using the dummy derivative method. The advantages of the dummy derivative method over other methods that have been proposed are that it directly enforces all of the implicit constraints and it may be easily automated and combined with automatic differentiation technology to solve high-index DAEs of practical engineering interest. The dummy derivative method does not work for all high-index DAEs, notably those for which the index and/or dynamic degrees of freedom cannot be determined correctly using structural criteria. However, our experience has shown that numerical solutions to many high-index engineering problems are easily obtained.

Implementation of the dummy derivative algorithm in ABACUSS required significant development of the details of the algorithm, as evidenced in the REDUCE-INDEX algorithm and the dummy derivative pivoting heuristic. This implementation appears to be the first example of a computational environment capable of easily solving high-index DAEs. The ability to solve high-index DAEs directly is necessary for the development of algorithms to solve state-constrained dynamic optimization problems, which are discussed in the next chapters.

Chapter 6

Equality Path Constraints

This chapter is concerned with the solution of equality path-constrained dynamic optimization problems. Equality path constraints (as distinguished from inequality path constraints, which are the subject of the next chapter) constrain the state variables of the DAE in the dynamic optimization problem. To date, this class of problem has not been satisfactorily handled within the control parameterization framework. However, it is shown in this chapter that equality path-constrained problems contain high-index DAEs which may be solved efficiently using the dummy derivative method detailed in Chapter 5.

An equality path-constrained subset of the dynamic optimization formulation given in Chapter 2 is considered in this chapter:

$$\min_{u(t),t_f} J = \psi\left(x\left(t_f\right), y\left(t_f\right), t_f\right) + \int_{t_o}^{t_f} L\left(x, u, t\right) dt \tag{6.1}$$

subject to the DAE:

$$f\left(\dot{x}, x, y, u, t\right) = 0 \tag{6.2}$$

$$h(x, y) = 0 \tag{6.3}$$

$$\phi\left(\dot{x}\left(t_o\right), x\left(t_o\right), y\left(t_o\right), t_o\right) = 0 \tag{6.4}$$

In this formulation, x are differential state variables, y are algebraic state variables

and u are control variables. The path constraint (6.3) is assumed to be a vector function with $h(\cdot) \rightarrow \mathbb{R}^{n_h}$. The DAE (6.2) is assumed to have index ≤ 1, although the results are easily extended if (6.2) is a high-index DAE for which an equivalent index-1 DAE may be derived using the dummy derivative method. Note that (6.1–6.4) does not include point constraints at times other than t_0, which are irrelevant for this chapter, or inequality path constraints on the state variables, which are discussed in Chapter 7.

Our goal is to solve (6.2–6.3) simultaneously as an IVP within the control parameterization method, since the efficiency of the control parameterization method increases as more of the constraints are handled by the IVP solver rather than the NLP solver. Since (6.3) constrains the states of the system (6.2), it can be expected that the combined system may be high-index. However, the combined system (6.2–6.3) is also overspecified, and thus not all of the control variables u are truly independent.

The two problems that must be addressed are:

1. Is the state path-constrained problem feasible?

2. How should the overspecified nature of the problem be handled?

The first issue is related to controllability of nonlinear systems and solvability of nonlinear high-index DAEs. The latter issue leads to the *control matching* problem. Since not all controls are independent, a subset of the controls are allowed to be determined by the solution of the high-index DAE, thus removing the overspecification of the problem. The control matching problem determines which controls are in the subset defined by the high-index DAE, and which ones are free to be determined by the optimization.

This chapter begins with a review of existing methods for solving equality path-constrained dynamic optimization problems and a discussion of the limitations of the current methods. Section 6.2 then presents a series of theorems which describe the circumstances which produce high-index DAEs in the combined system (6.2–6.3). Some issues relating to feasibility of the constrained dynamic optimization problem are discussed in Section 6.3, and the problem of control matching is addressed in

Section 6.4. Finally, examples are presented in Section 6.5.

6.1 Review of Methods for Solving Path-Constrained Dynamic Optimization Problems

Several methods have been developed to date for solving path-constrained dynamic optimization problems within the control parameterization framework. Since it was not possible to solve high-index DAEs directly at the time that these methods were developed, the emphasis was on finding methods that discretize the path constraints so that they could be included either in the objective function or as a set of NLP equality constraints. Therefore, these methods all handle the path constraints indirectly, in the sense that they are included in the master NLP rather than the IVP subproblem.

One method for enforcing path constraints is to modify the objective function to include a measure of the constraint violation [24]:

$$\bar{J} = J + \sum_{i=1}^{n_h} K_i \int_{t_0}^{t_f} h_i^2 dt \tag{6.5}$$

where K is a vector of large numbers. The path constraints are satisfied exactly only if $K \to \infty$. The problem with this approach is that it has been shown to cause numerical difficulties with the NLP master problem because it modifies the shape of the objective function, possibly making it more nonlinear or introducing additional locally optimal points.

Another method [128] is to replace the path constraints with a single end-point constraint:

$$\varphi = \int_{t_0}^{t_f} h^T h dt = 0 \tag{6.6}$$

or by a set of end-point constraints:

$$\varphi_i = \int_{t_0}^{t_f} h_i^2 dt = 0 \qquad i = 1 \ldots n_h \tag{6.7}$$

Since more information is provided to the optimizer by (6.7) than by (6.6), it is generally preferred. However, even (6.6) lumps the violation of the constraint over

the entire state trajectory into a single measure, and thus it provides only limited information about how to modify the input functions to achieve feasibility. Also, both (6.6) and (6.7) have gradients with respect to the optimization parameters that are zero at the optimum, which reduces the efficiency of gradient-based optimization methods.

A somewhat more sophisticated method was proposed in [142, 143], where the relationship between path-constrained dynamic optimization problems and high-index DAEs was noted. It was still not possible at the time of that research to solve most classes of high-index DAEs, so a method was proposed to append directly to the DAE any state variable constraints that did not cause the resulting system to be high-index. The other state variable constraints were transformed into a set of NLP equality constraints. To form these NLP equality constraints, a hybrid method was used which combines both global constraints like (6.7) and point constraints like:

$$\rho_i(t_j) = h_i(x(t_j), y(t_j)) = 0 \tag{6.8}$$
$$i = 1 \ldots n_h \qquad j = 1 \ldots n_{pt} \qquad t_j \in (t_0, t_f]$$

These point constraints may be evaluated either by sampling the state variables as the simulation progresses, or by interpolating the state trajectories after the simulation finishes. The point constraints provide local information along the trajectory to the optimizer, while the global constraint attempts to prevent the constraint from being violated at times other than the points where (6.8) were evaluated. Similar methods were proposed in [63], although the relationship between the path constraints and high-index DAEs was not noted in that work.

In [76] a similar method was described in which high-index DAEs were solved via a transformation to a path-constrained dynamic optimization problem. In this method, a high-index DAE:

$$\bar{f}(\dot{x}, x, y, t) = 0 \tag{6.9}$$

is partitioned into two sets of equations:

$$\bar{f}^{(1)}(\dot{x}, x, y, t) = 0 \tag{6.10}$$

$$\bar{f}^{(2)}(\dot{x}, x, y, t) = 0 \tag{6.11}$$

such that the matrix $\left[(\partial f^{(1)}/\dot{x}^{(1)})\ (\partial f^{(1)}/y^{(1)})\right]$ is nonsingular, and the state variables are partitioned into $x = \left[x^{(1)} : x^{(2)}\right]^T$ and $y = \left[y^{(1)} : y^{(2)}\right]^T$. The solution to the high-index DAE may be obtained by solving the following dynamic optimization problem:

$$\min_{x^{(2)}, y^{(2)}, t \in [t_0, t_f]} \left\| \bar{f}^{(2)}(\dot{x}, x, y, t) \right\| \tag{6.12}$$

subject to:

$$\bar{f}^{1}(\dot{x}, x, y, t) = 0 \tag{6.13}$$

where $\|(\cdot)\|$ is an appropriate norm on the domain $[t_0, t_f]$. The dynamic optimization method for solving high-index DAEs requires the following steps:

1. Solve the IVP numerically using the given control profiles.

2. Evaluate the integral constraint violation (objective function). If the constraint violation is acceptably small, then stop.

3. Adjust the control profiles $x^{(2)}, y^{(2)}$ to try to minimize the constraint violation on the next step. Goto Step 1.

This method is interesting here because it demonstrates the problems with all methods that solve path-constrained dynamic optimization problems by including the path constraints in the master NLP.

There are problems with this method include:

- Since the most expensive step is Step 1, this method is much less efficient than direct methods for solving high-index DAEs (such as the dummy derivative

method described in Chapter 5), which requires the solution of only one (albeit larger) IVP.

- Any method for choosing a time discretization of the controls will be somewhat ad hoc compared to the step size selection algorithm built into the numerical DAE solver that handles the IVP. Time discretization of the controls is arguably not a problem for unconstrained dynamic optimization problems, where the space of implementable control functions is often limited by physical considerations. However, it is a problem here because state trajectories are being discretized for which the functional form is not constrained by implementable control functions.

- There is no guarantee on the accuracy of the control trajectories, since the discretization chosen may not be sufficient to capture all the features of those trajectories. Furthermore, there is no measure of the error that would indicate when a solution is unacceptable.

- Independent of the discretization, the NLP has more equations and decision variables than necessary. Some of the decision variables in an equality path-constrained optimization are actually completely determined by the solution to the path constrained DAE. Therefore, it is more efficient to find the solution of these variables directly, using the IVP solver, rather than indirectly, using the NLP solver.

- Any control discretization as complex as the discretization typically chosen by a BDF integrator would create large-scale dense NLPs, which are difficult to solve using current NLP solution methods.

In short, the combination of inefficiency and uncontrollable accuracy makes the use of control parameterization unattractive to solve equality path-constrained DAEs with all the above methods.

Although the problems described above have not been explicitly recognized by other authors, there have been some methods proposed for solving high-index dy-

namic optimization problems directly. In [66, 67] dynamic optimization of index-2 DAEs was described, and in [109] a method similar to the dummy derivative method was proposed to derive an equivalent index-1 DAE for a given arbitrary-index DAE. However, neither implementation nor numerical results were reported in [109], and the method described requires detection of numerical singularity of matrices, which is problematic both practically and theoretically in the nonlinear sense. In both of these works it was assumed that (6.2) was high-index, and neither proposed a method for directly appending (6.3) to (6.2) to form a high-index system.

6.2 Equality Path Constraints and High-Index DAEs

The method proposed in this chapter for solving equality path-constrained dynamic optimization problems is to append the path constraint (6.3) directly to the DAE (6.2), allowing a subset of the control variables to be determined by the solution of the resulting combined IVP. Then the dummy derivative method is used to derive an equivalent index-1 DAE for the high-index DAE. For the purposes of this section, it is assumed that a control matching can be found, and concentrate instead on the properties of the resulting combined system.

Reference to the following definition which was presented in [21] was made in Chapter 2:

Definition 6.1 (Differential Index of a DAE). *The minimum number of times that all or part of a DAE must be differentiated with respect to time to determine $\dot{x}(t)$ and $\dot{y}(t)$ as continuous functions of $x(t)$, $y(t)$, $u(t)$, and t is the differential index i of a DAE.*

By this definition, ordinary differential equations (ODEs) have $i = 0$. The term *high-index DAE* is commonly used to refer to systems with $i \geq 2$. As mentioned in Chapter 5, standard numerical integration codes experience difficulty when applied to systems with $i \geq 2$, and only limited classes of high-index problems may be solved directly at present.

Definition 6.2 (Structurally Singular DAEs). *A DAE is structurally singular if the Jacobian of the DAE with respect to the highest-order time derivatives in the DAE (*e.g.*, $\dot{x}$ and y in (6.1)) is structurally singular.*

Definition 6.3 (Structural Index of a DAE). *The structural index i_s of a DAE is the maximum number of times that any subset of equations in the DAE is differentiated using Pantelides' algorithm.*

Theorem 6.1. *The index and the structural index of a DAE are related by $i \geq i_s$.*

Proof. Consider a modification to Pantelides' algorithm described in Chapter 5, in which the variables a and equations:

$$a = \dot{x} \tag{6.14}$$

are appended to the underlying DAE as necessary at each step of the algorithm so that time derivatives of order greater than one do not appear in the underlying DAE obtained through the next step. Since the new variable a is uniquely determined by (6.14), the augmented system has the same index as the original DAE.

Define $\hat{z}_{i_s}$ as the set of highest order time derivatives in the final underlying DAE obtained with this modified Pantelides' algorithm, $f^{(i_s)}$. Define $\hat{J}_{i_s}$ as the Jacobian of $f^{(i_s)}$ with respect to $\hat{z}_{i_s}$.

There are four cases:

$i = i_s$: If $\hat{J}_{i_s}$ is nonsingular and $\hat{z}_{i_s}$ does not contain any algebraic variables, then $f^{(i_s)}$ has $i = i_s = 0$, and according to the Implicit Function Theorem, all time derivatives are uniquely determined given the state variables.

$i = i_s + 1$: If $\hat{J}_{i_s}$ is nonsingular and $\hat{z}_{i_s}$ does contain some algebraic variables, then all time derivatives are not uniquely determined by this final underlying DAE. Since by assumption:

$$\hat{J}_{i_s} = \begin{bmatrix} \frac{\partial f^{(i_s)}}{\partial \dot{x}} & \frac{\partial f^{(i_s)}}{\partial y} \end{bmatrix} \tag{6.15}$$

is nonsingular, where $\dot{x}$ and y are respectively the time derivatives and algebraic variables in $\hat{z}_{i_s}$, one further differentiation of $f^{(i_s)}$ produces:

$$f^{(i_s+1)} = \frac{\partial f^{(i_s)}}{\partial \dot{a}}\dot{a} + \frac{\partial f^{(i_s)}}{\partial x}\dot{x} + \frac{\partial f^{(i_s)}}{\partial y}\dot{y} + \frac{\partial f^{(i_s)}}{\partial \hat{u}}\dot{\hat{u}} + \frac{\partial f^{(i_s)}}{\partial t} = 0 \tag{6.16}$$

$$\dot{a} = \ddot{x} \tag{6.17}$$

where $\hat{u} = \{u, (du/dt), \ldots, (d^{(i_s)}u/dt^{(i_s)})\}$. Because $\hat{J}_{i_s}$ is nonsingular, the time derivatives $\dot{a}$ and $\dot{y}$ are uniquely determined by $f^{(i_s)}$.

$i > i_s$: If $\hat{J}_{i_s}$ is singular, then by the properties of Pantelides' algorithm $\hat{J}_{i_s}$ must be numerically but not structurally singular. Since the general DAE is nonlinear, no information about whether $\hat{z}_{i_s}$ is uniquely determined or not is conveyed by the numerical singularity of $\hat{J}_{i_s}$. It is possible that additional differentiations must be performed to uniquely determine $\hat{z}_{i_s}$, and therefore the differential index may be larger than the structural index.

$i \not< i_s$: Since Pantelides' algorithm terminates with the first underlying DAE that possesses a structurally nonsingular $\hat{J}_{i_s}$, and since structural nonsingularity of $\hat{J}_{i_s}$ is a necessary condition for $f^{(i_s)}$ to uniquely define the highest order time derivatives $\hat{z}_{i_s}$, it is not possible for any previous $f^{(i_s-j)}$, $j = 1, \ldots, i_s$ to uniquely determine $\hat{z}_{i_s-j}$. Therefore, the structural index is a lower bound on the differential index. □

Definition 6.4 (Structurally Well-Behaved DAEs). *A structurally well-behaved DAE is one for which $\hat{J}_{i_s}$ is nonsingular for the entire time horizon of interest.*

Note that for structurally well-behaved DAEs, $i = i_s$ or $i = i_s + 1$. Although the structural index is a global property of DAEs, the differential index is only a local property. DAEs that are neither structurally high-index nor structurally well-behaved may or may not be high-index at the points where $\hat{J}_{i_s}$ is singular. That is, even though a nonsingular $\hat{J}_{i_s}$ indicates that the DAE is not high-index, a singular $\hat{J}_{i_s}$ does not necessarily indicate that the DAE is high-index.

The method for solving equality path-constrained dynamic optimization problems described in this chapter applies only if the resulting high-index DAE is structurally well-behaved. The focus is limited to this class of DAEs because of the practical difficulties in detecting numerical singularity of matrices, and the theoretical difficulties of using numerical singularity in the nonlinear or linear time-varying classes of DAEs to determine the true differential index of the DAE. Furthermore, almost all problems of practical interest are structurally well-behaved. For the remainder of this chapter, the DAEs are assumed to be structurally well-behaved.

In general, the strongest statement that can be made is that appending (6.3) to

an index-one DAE *may* result in a high-index DAE. There are cases in which the index of the augmented DAE remains unchanged. For example, consider the index-1 DAE:

$$\dot{x} + y + u = 0 \tag{6.18}$$

$$y - u - x = 0 \tag{6.19}$$

where u is a control variable, and the equality path constraint:

$$2x + y - 5 = 0 \tag{6.20}$$

where x and y are state variables and u is a control variable. When u is treated as an algebraic variable, the augmented DAE (6.18–6.20) also has index = 1.

However, there are some important classes of DAE for which appending equality path constraints to the DAE in the manner described above will result in high-index DAEs.

Theorem 6.2. *Appending $n_g \leq n_u$ state constraints of the form:*

$$g(x) = 0 \tag{6.21}$$

where there is some $\chi \subseteq x$, $\chi \in \mathbb{R}^{n_g}$ such that $(\partial g/\partial \chi)$ is nonsingular, to an explicit ODE:

$$\dot{x} - \phi(x, u, t) = 0 \tag{6.22}$$

yields a DAE with $i \geq 2$, assuming the DAE is solvable.

Proof. Appending the path constraints requires n_g controls to become algebraic state variables denoted by $y \subseteq u$ in the augmented DAE. Define $\bar{\chi} = x \setminus \chi$ and $\bar{u} = u \setminus y$.

A suitable partitioning of (6.22) yields:

$$\dot{\bar{\chi}} - \phi_1(\chi, \bar{\chi}, y, \bar{u}, t) = 0 \tag{6.23}$$

$$\dot{\chi} - \phi_2(\chi, \bar{\chi}, y, \bar{u}, t) = 0 \tag{6.24}$$

By assumption, χ is uniquely determined by (6.21) so it is not available to determine $\dot{x}$ and $\dot{u}$. Differentiate (6.23),(6.24), and (6.21) with respect to time to yield:

$$\ddot{\bar{\chi}} - \frac{\partial \phi_1}{\partial \chi}\dot{\chi} - \frac{\partial \phi_1}{\partial \bar{\chi}}\dot{\bar{\chi}} - \frac{\partial \phi_1}{\partial y}\dot{y} - \frac{\partial \phi_1}{\partial \bar{u}}\dot{\bar{u}} - \frac{\partial \phi_1}{\partial t} = 0 \tag{6.25}$$

$$\ddot{\chi} - \frac{\partial \phi_2}{\partial \chi}\dot{\chi} - \frac{\partial \phi_2}{\partial \bar{\chi}}\dot{\bar{\chi}} - \frac{\partial \phi_2}{\partial y}\dot{y} - \frac{\partial \phi_2}{\partial \bar{u}}\dot{\bar{u}} - \frac{\partial \phi_2}{\partial t} = 0 \tag{6.26}$$

$$\frac{\partial g}{\partial \chi}\dot{\chi} + \frac{\partial g}{\partial \bar{\chi}}\dot{\bar{\chi}} = 0 \tag{6.27}$$

$\dot{\bar{\chi}}$ is uniquely determined by (6.23) and $\dot{\chi}$ by (6.27), but (6.24) is unavailable to determine $\dot{y}$. Differentiating (6.27) with respect to time yields:

$$\left[\frac{\partial g_j}{\partial \chi}\right]^T \ddot{\chi} + \dot{\chi}^T \frac{\partial^2 g_j}{\partial \chi^2}\dot{\chi} + 2\dot{\bar{\chi}}^T \frac{\partial^2 g_j}{\partial \bar{\chi}\partial \chi}\dot{\chi} + \dot{\bar{\chi}}^T \frac{\partial^2 g_j}{\partial \bar{\chi}^2}\dot{\bar{\chi}} + \left[\frac{\partial g_j}{\partial \bar{\chi}}\right]^T \ddot{\bar{\chi}} = 0 \tag{6.28}$$

for all $j = 1, \ldots n_g$.

Nonsingularity of the matrix:

$$\begin{bmatrix} I & 0 & \frac{\partial \phi_1}{\partial y} \\ 0 & I & \frac{\partial \phi_2}{\partial y} \\ \frac{\partial g}{\partial \bar{\chi}} & \frac{\partial g}{\partial \chi} & 0 \end{bmatrix} \tag{6.29}$$

is a sufficient condition for (6.25–6.26) to determine $\ddot{\bar{\chi}}$, $\ddot{\chi}$, and $\dot{y}$ uniquely. Since (6.21) has been differentiated twice, according to Definition 6.1 the index is at least two. Since there is no guarantee that (6.29) has full rank, the index may be greater than two. □

Theorem 6.3. *Appending $n_g \leq n_u$ state constraints of the form:*

$$g(x, u) = 0 \tag{6.30}$$

to an explicit ODE of the form (6.22) yields a DAE with $i \geq 1$, assuming the DAE is solvable.

Proof. Since (6.30) does not contain any time derivatives, the augmented system is structurally rank-deficient with respect to the time derivatives and hence must be at least index-1. □

A consequence of Theorems 6.2 and 6.3 is that optimal control of ODE systems subject to state path constraints requires either implicit or explicit treatment of DAEs.

For cases where the DAE is not an explicit ODE, the index may stay the same or increase when the DAE is augmented with state path constraints. In problems where the index increases, as the theorems indicate, the new index may indeed rise by more than one. An example of the index rising by more than one is the index-0 DAE:

$$\dot{x}_1 = u_0 - 5x_1 \tag{6.31}$$

$$\dot{x}_2 = 5x_1 - 3x_2 \tag{6.32}$$

$$\dot{x}_3 = 3x_2 - 4x_3 \tag{6.33}$$

where u_0 is a control variable. When this system is augmented with:

$$x_3 = 10 \tag{6.34}$$

u_0 becomes an algebraic variable and the augmented problem is index-4.

The case where $(\partial g/\partial x)$ is rank-deficient yields a useful insight for diagnosing poorly posed problems. Rank deficiency of $(\partial g/\partial x)$ implies that the state path constraints are at least locally either redundant or inconsistent. The inconsistent case can be excluded because inconsistent path constraints cannot be satisfied simultaneously. Hence, only the redundant case is possible. Therefore, if $(\partial g/\partial x)$ becomes

nk deficient, a subset of the state path constraints can be ignored at least locally.

6.3 Dynamic Optimization Feasibility

An important consideration when specifying path-constrained dynamic optimization problems is to ensure that the problem is feasible with respect to the path constraints. In this section, it is assumed that (6.2) is a solvable, structurally well-behaved DAE. The purpose of this section is to determine conditions that (6.3) must satisfy for the dynamic optimization problem to be feasible.

When equality path constraints are appended to the DAE, one control variable must become a state variable in the resulting augmented DAE to prevent it from being overdetermined. The rationale is simply that the number of state variables in a properly specified DAE must equal the number of equations in the DAE. One of the implications is that if the number of control variables is less than the number of constraints, the problem is either over-constrained or some of the constraints are redundant. Therefore, the number of control variables must be greater than or equal to the number of equality path constraints for the dynamic optimization problem to be solvable.

If the DAE has an equal number of control variables and equality path constraints, it is still possible that the resulting system is unsolvable, which is to say, there is no state trajectory for a given set of input trajectories. Assuming that the original DAE is well-posed and solvable, the augmented system could be unsolvable because the augmented DAE is uncontrollable, or because the constraints are inconsistent or redundant. For example, no control trajectory u for the system:

$$\dot{x} = \frac{1}{x} + u^2 \tag{6.35}$$

can match the equality path constraint:

$$x = 0 \tag{6.36}$$

Likewise, any dynamic optimization problem that includes the equality path con-

straints:

$$x_1 + x_2 = 5 \tag{6.37}$$

$$x_1 - x_2 = 1 \tag{6.38}$$

$$x_1 + 5x_2 = 100 \tag{6.39}$$

is unsolvable since the constraints are inconsistent with each other.

Three necessary requirements that must be met for the constrained dynamic optimization problem to be feasible are that the equality path constraints must be consistent and nonredundant, and the original DAE must be controllable with respect to the equality path constraints. A system of equations is *consistent* if there exists at least one solution to the system. The requirement that the constraints be *nonredundant* stems from the fact that one control variable is turned into an algebraic variable for every equality path constraint, and the presence of redundant equations would cause the loss of too many degrees of freedom. *Controllability* is defined according to [121]:

> "Roughly speaking, a dynamic system is 'controllable' if its state vector can be caused, by an appropriate manipulation of system inputs, to behave in a desirable manner."

Note that consistency, nonredundancy, and controllability are all local properties of nonlinear systems, and therefore it is difficult in practice to detect the presence or absence of these conditions.

However, as in the solution of nonlinear high-index DAEs, some information can be gained from the structural properties of the augmented system. A necessary condition for a set of nonredundant constraints to be consistent is that an assignment exists for every equation to a unique variable in the system. If this condition is not met, the equations must be either inconsistent or redundant. In the example (6.37–6.39) no assignment exists for each equation to a nonredundant variable because there are three equations but only two variables.

A precondition for controllability is *input connectability*, which exists for a system "if the system inputs are able to influence all the state variables" [121]. Input connectability is defined in a graph-theoretic sense if a path exists for each state variable to at least one control variable in the digraph of the system. The digraph of a DAE was defined in Section 5.2.1.

To see an example of input connectability, consider the system:

$$\dot{x}_1 = x_2 \tag{6.40}$$

$$\dot{x}_2 = x_2 + u \tag{6.41}$$

$$\dot{x}_3 = 4 \tag{6.42}$$

where u is a control variable, with the path constraints:

$$x_1^2 = 3 \tag{6.43}$$

$$x_3 = 9 \tag{6.44}$$

The input variable u directly influences (6.40), which in turn influences x_2 through (6.41). Therefore, the state variable x_2 in the path constraint (6.43) is input connectable with u. However, the variable x_3 in the state path constraint (6.44) is not influenced by the control variable u; therefore x_3 is not input connectable and no feasible solution exists to a dynamic optimization problem that contains (6.40–6.44) which satisfies (6.44).

Input connectability of all the state variables is not a condition for solvability of the DAE. For example, the DAE (6.40–6.43) is solvable (and a dynamic optimization problem would be feasible) even though the state variable x_3 is not input connectable. However, at least one state variable in each of the state variable path constraints must have the ability to be influenced by an input variable in the unconstrained DAE or else the constraint cannot be satisfied. Therefore, the entire augmented DAE does not have to be input connectable, but the state variable constraints do.

Also, there exist input-connectable systems which are still infeasible. For example,

consider the system:

$$\dot{x}_1 = x_2 \tag{6.45}$$

$$\dot{x}_2 = x_2 + u_1 \tag{6.46}$$

$$\dot{x}_3 = 4 + u_2 \tag{6.47}$$

where u_1 and u_2 are control variables, with the path constraints:

$$x_1^2 = 3 \tag{6.48}$$

$$x_2 = \sin(t) \tag{6.49}$$

Although both (6.48) and (6.49) are input connectable to u_1, the control variable u_1 cannot satisfy them both simultaneously. However, if (6.48) is replaced with:

$$x_1 = \cos(t) \tag{6.50}$$

the control u_1 could simultaneously satisfy the state path constraints. Even though the constraints (6.49–6.50) do not appear to be redundant when examined in isolation, they are in fact redundant when coupled with the DAE (6.45–6.47). Therefore, the nonredundancy consideration requires each state path constraint to be input connectable to a unique control variable in the unconstrained DAE.

The result of the discussion in this section is the following theorem:

Theorem 6.4. *Assuming that (6.3) is nonredundant, if an augmented DAE formed by appending (6.3) to (6.2) and allowing n_h control variables to become algebraic state variables does not satisfy all of the following conditions, the associated path-constrained dynamic optimization problem (6.1–6.4) is infeasible. The conditions are:*

1. *The number of control variables must be greater than or equal to n_h.*
2. *The equations (6.3) must be structurally nonsingular with respect to a nonempty subset of the state variables in (6.3).*

3. *There exists at least one transversal of the Jacobian of the constraints (6.3) with respect to the state variables such that a path exists in the graph of (6.2–6.3) from every member of the set of n_h control variables $\hat{u} \subseteq u$ that have become algebraic state variables in (6.2–6.3), to a unique member of the set of state variables in the transversal.*

Proof. All of the following statements are based on the assumption that the constraints are nonredundant. If Condition 1 is not true, then the dynamic optimization problem does not have enough degrees of freedom to satisfy its constraints. If Condition 2 is not true, then (6.3) must be inconsistent, and therefore no state trajectories exist that will satisfy all of the constraints simultaneously. If Condition 3 is not true, then the combined system is either not input connectable, or it is structurally singular with respect to its state variables. □

The assumption that the constraints are nonredundant is difficult to check, but is a reasonable assumption since it is usually obvious to the person specifying the problem as to which nonredundant constraints may be imposed. Theorem 6.4 is useful in practice, since all of the conditions may be easily checked using structural criteria. However, the theorem is, like the other structural criteria used in this thesis, a set of necessary conditions, and there are sets of state path constraints for which this theorem is true which are not feasible because they do not satisfy consistency or input connectability requirements at local points.

6.4 Control Matching

The results presented in this chapter thus far have assumed that it is possible to find a valid subset of the control variables of the unconstrained problem to make into algebraic variables in the constrained problem. This section assumes that a feasible path constrained dynamic optimization problem has been posed as described in the previous section, and describes how to determine which control variables become algebraic variables in the path constraint augmented DAE.

The equality path-constrained dynamic optimization problem may have more controls than state variables. In this case, a valid subset of the control variables must be selected to become algebraic state variables in the augmented DAE. Such a subset is termed a *control variable matching.* If Theorem 6.4 is satisfied and there are exactly the same number of controls as state path constraints, there is no uncertainty about which controls are in the control matching, but it is still necessary to determine whether the resulting augmented DAE is solvable.

The most general statement that may be made about a control variable matching is that it must result in a solvable DAE. Solvability criteria for nonlinear high-index DAEs are discussed in [21, 29], but are of little practical use. Rather, since the dummy derivative method is dependent upon Pantelides' algorithm, a related but more relevant question is to find a control variable matching that will lead to a DAE for which Pantelides' algorithm will terminate. If Pantelides' algorithm terminates, the DAE is solvable in a structural sense, and the corrector iteration of the BDF method will be structurally nonsingular.

The following definitions and theorem were presented in [107]:

Definition 6.5 (DAE Extended System). *Given the DAE system (6.51), the corresponding extended system is:*

$$f(\dot{x}, x, y, u) = 0 \tag{6.51}$$

$$v_i(x_i, \dot{x}_i) = 0 \qquad i = 1 \ldots n_x \tag{6.52}$$

Definition 6.6 (Structurally inconsistent DAE). *The DAE (6.51) is said to be structurally inconsistent if it can become structurally singular with respect to all occurring variables state variables (x, y) by the addition of the time differentials of a (possibly empty) subset of its equations.*

Theorem 6.5. *Pantelides' algorithm terminates if and only if the extended system (6.51–6.52) is structurally nonsingular. Furthermore, if (6.51–6.52) is structurally singular, then the DAE system (6.51) is structurally inconsistent.*

A proof of Theorem 6.5 is given in [107]. As an example, consider the system:

$$f_1(x, y_1, y_2) = 0 \tag{6.53}$$

$$f_2(x, \dot{x}, u_1) = 0 \tag{6.54}$$

$$f_3(x, u_2) = 0 \tag{6.55}$$

with control variables u_1 and u_2, which was given in [107]. The extended system is formed by appending:

$$v(\dot{x}, x) = 0 \tag{6.56}$$

to (6.53–6.55). The extended system is structurally singular with respect to $\dot{x}$, x, y_1, and y_2, and therefore Pantelides' algorithm cannot terminate. This system is not solvable because (6.53) is not sufficient to define both y_1 and y_2, which do not appear elsewhere in the system. Theorem 6.4 is actually a requirement for the extended system to be structurally nonsingular, but Theorem 6.5 is not as useful as Theorem 6.4 when attempting to specify a feasible path-constrained dynamic optimization problem.

However, Theorem 6.5 can be used to find control matchings that lead to structurally consistent DAEs. Essentially, Duff's algorithm is used to extend the augmenting path of a rectangular structural matrix formed by the addition of n_h rows and n_u columns to the structural Jacobian of the unconstrained DAE.

It is assumed that (6.2) is structurally consistent, and that application of the

AUGMENTPATH algorithm (see [43, 44, 107] and Chapter 5) to the extended system of (6.2) has resulted in a vector ASSIGN of length $2n_x + n_y$. The equality path constraints and the control variables are then appended to the extended system. The AUGMENTPATH algorithm is then applied to each equality path constraint, hopefully resulting in a new vector ASSIGN of length $2n_x + n_y + n_h$. Any control variable u_i for which ASSIGN$(2n_x + n_y + i)$ is nonzero is defined to be part of a control matching. If AUGMENTPATH returns PATHFOUND=FALSE for any of the equality path constraints, no control matching is possible for this system.

For example, the structural Jacobian with respect to the state variables and time derivatives for the following extended system:

$$f_1(\dot{x}_1, x_2, u_1) = 0 \tag{6.57}$$

$$f_2(\dot{x}_1, \dot{x}_2, x_1, u_1, u_2) = 0 \tag{6.58}$$

$$f_3(x_1, y, u_3) = 0 \tag{6.59}$$

$$v_1(\dot{x}_1, x_1) = 0 \tag{6.60}$$

$$v_2(\dot{x}_2, x_2) = 0 \tag{6.61}$$

where the x variables are states and the u variables are controls is:

$$\begin{array}{c|ccccc} & \dot{x}_1 & \dot{x}_2 & x_1 & x_2 & y \\ \hline f_1 & \times & & & \otimes & \\ f_2 & \otimes & \times & \times & & \\ f_3 & & & \times & & \otimes \\ v_1 & \times & & \otimes & & \\ v_2 & & \otimes & & \times & \end{array} \tag{6.62}$$

where the symbols '$\times$' and '$\otimes$' denote structural nonzeros, and the set of '$\otimes$' denotes an augmenting path found by successive application of the AUGMENTPATH

algorithm to each equation. When the following equality path constraints are added:

$$h_1(x_1, x_2) = 0 \tag{6.63}$$

$$h_2(y) = 0 \tag{6.64}$$

to form an augmented DAE, the constraints and the u variables are added to the structural Jacobian, giving:

	$\dot{x}_1$	$\dot{x}_2$	x_1	x_2	y	u_1	u_2	u_3
f_1	×			⊗		×		
f_2	×	×	×			×	⊗	
f_3			×		×		×	⊗
v_1	⊗		×					
v_2		⊗		×				
h_1			⊗	×				
h_2					⊗			

(6.65)

when the AUGMENTPATH algorithm is applied to the new equations. The control matching consists of the variables u_2 and u_3, which become algebraic state variables in the augmented DAE. Note that since the structural singularity of (6.62) is typically checked when solving a DAE in a modeling environment such as ABACUSS, the additional computational effort required to obtain the control matching is minimal.

It is possible that the control matching found with this method is nonunique. In the previous example, the augmenting path for the extended Jacobian could have

been:

	$\dot{x}_1$	$\dot{x}_2$	x_1	x_2	y	u_1	u_2	u_3
f_1	×			⊗		×		
f_2	×	×	×			⊗	×	
f_3			×		×		×	⊗
v_1	⊗		×					
v_2		⊗		×				
h_1			⊗	×				
h_2					⊗			

(6.66)

and therefore the control matching could be the variables u_1 and u_3. There is no structural information that would indicate that one choice is preferred over the other, and in many cases it may be that all possible control matchings are acceptable. One consideration may be that the controls that are not in the control matching will be subject to the control parameterization, while the functional form of those in the control matching is unrestricted. If, for example, the implementable function space was limited for some controls but not others, the control matching choice could be guided by the desire to limit the functional form of some of the controls. Also, since structural criteria were used to determine the possible control matchings, it may be that some choices are in fact not permissible because they lead to DAEs which are numerically unsolvable.

The control matching algorithm presented in this section produces an augmented DAE that is solvable in the structural sense. Like all of the structural criteria that have been used in this thesis, the requirement that the extended augmented DAE be structurally nonsingular is a necessary condition for the DAE to be solvable, but it is not a sufficient condition. However, it is a sufficient condition if the augmented system is structurally well-behaved.

6.5 Examples

This section gives numerical results for several equality path-constrained dynamic optimization problems. For other examples of path constrained dynamic optimization problems in this thesis, see Chapter 7 and Chapter 8. There are very few examples of equality path-constrained dynamic optimization problems in the literature (there are examples of inequality path constrained problems, but those are discussed in the next chapter). One of the few problems for which a numerical solution has been reported is the high-index pendulum, which was solved in [76] and [142] as an equality path-constrained dynamic optimization problem. However, the high-index pendulum was solved in Chapter 5 using a single IVP integration using the dummy derivative method, and so it has not been included here.

6.5.1 Two-dimensional car problem

In this problem a car which is able to move in two dimensions (x and y) under the influence of two controls, the acceleration angle θ and magnitude a. The objective is to find the control profiles that cause the car to move from point A to point B in minimum time. An equality path constraint is imposed by requiring the car to remain on a road of a given shape. Mathematically, this dynamic optimization problem is:

Integration Tolerance	Optimization Tolerance	IVP Solutions	Objective Function	CPU
10^{-5}	10^{-3}	23	26.85	3.88s

Table 6-1: *Statistics for the two-dimensional car problem*

$$\min_{a(t),\theta(t),t_f} t_f \tag{6.67}$$

subject to:

$$\dot{v}_x = a_x \tag{6.68}$$

$$\dot{x} = v_x \tag{6.69}$$

$$\dot{v}_y = a_y \tag{6.70}$$

$$\dot{y} = v_y \tag{6.71}$$

$$a_x = a\cos(\theta) \tag{6.72}$$

$$a_y = a\sin(\theta) \tag{6.73}$$

$$y = x^2 \tag{6.74}$$

$$x(0) = 0 \qquad x(t_f) = 300$$

$$v_x(0) = 0 \qquad v_x(t_f) = 0$$

The equality path constraint (6.74) can be matched to θ. The resulting augmented DAE is index-3, and the dummy derivative algorithm differentiates (6.74) twice to obtain an equivalent index-1 DAE. This problem was solved using two constant finite elements to approximate a. The control was bounded by $-500 \leq a \leq 500$, and the initial guess was $a = 10$. Solution statistics are given in Table 6-1 and Figures 6-1 to 6-4 show the solution trajectories. The solution found satisfies the path constraint to within the integrator tolerances over the entire state trajectory.

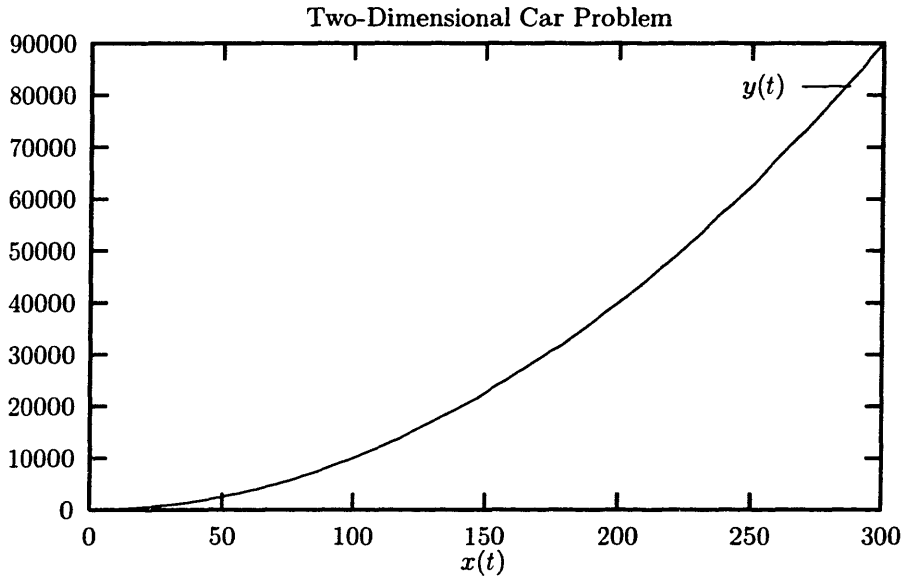

Figure 6-1: *State space plot showing the optimal trajectory of the two-dimensional car problem*

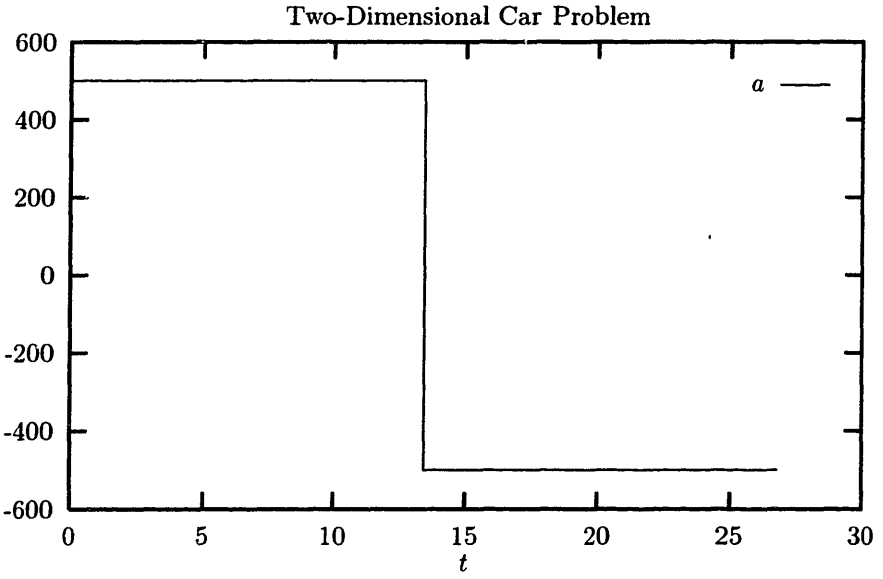

Figure 6-2: *Optimal acceleration trajectory for two-dimensional car problem*

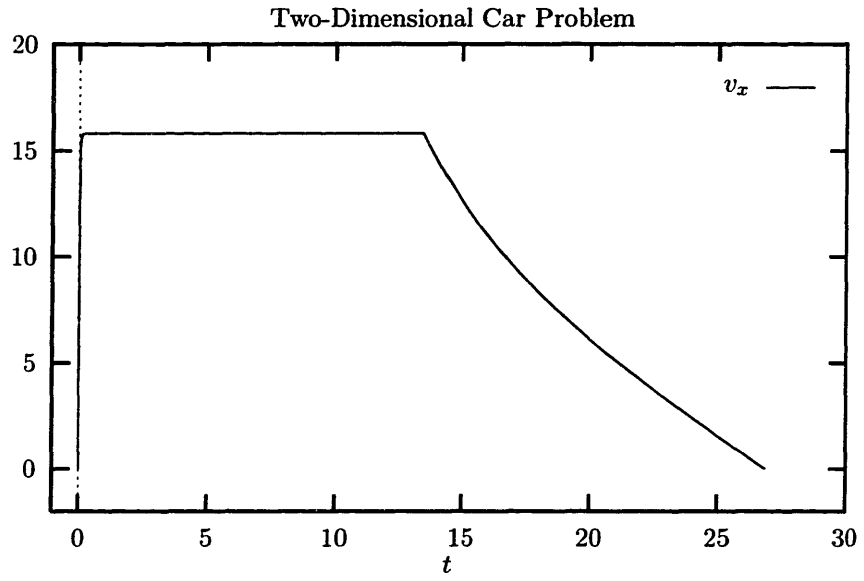

Figure 6-3: *The optimal velocity in the x direction for the two-dimensional car problem*

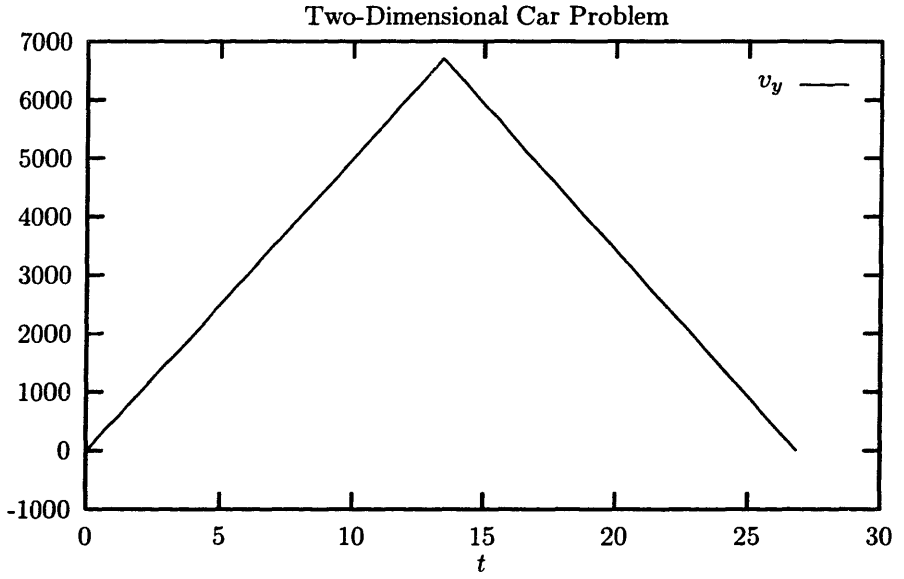

Figure 6-4: *The optimal velocity in the y direction for the two-dimensional car problem*

6.5.2 Brachistochrone

For a description of the brachistochrone problem, see Section 8.1, where it is solved with an inequality path constraint. There are several possible formulations of the brachistochrone problem [24, 84], including the following one [14]:

$$\min_{\theta(t),F(t),t_f} t_f \tag{6.75}$$

subject to:

$$\dot{x} = u \tag{6.76}$$

$$\dot{y} = v \tag{6.77}$$

$$\dot{u} = F\sin(\theta) \tag{6.78}$$

$$\dot{v} = g - F\cos(\theta) \tag{6.79}$$

$$x(t_f) = 1$$

where u and v are respectively the horizontal and vertical velocities, θ is the angle at which the bead is currently heading, and F is the normal contact force. Equations (6.76–6.79) describe the forces acting on the bead, so a path constraint defining the shape of the wire must be added:

$$\tan(\theta) = \frac{v}{u} \tag{6.80}$$

Interestingly, in this problem the index is dependent on the control matching. Either F or θ can be matched to the constraint (6.80). If F is matched to (6.80), the problem is index-2, while if θ is matched to (6.80) the problem is index-1. However, in the latter case $\dot{x}(0) = 0$, $\dot{y}(0) = 0$ is not a valid initial condition because (6.80) is not capable of uniquely defining θ at that point. Therefore, the index-2 formulation was

Integration Tolerance	Optimization Tolerance	IVP Solutions	Objective Function	CPU
10^{-5}	10^{-3}	10	1.772	1.33s

Table 6-2: *Statistics for the brachistochrone problem*

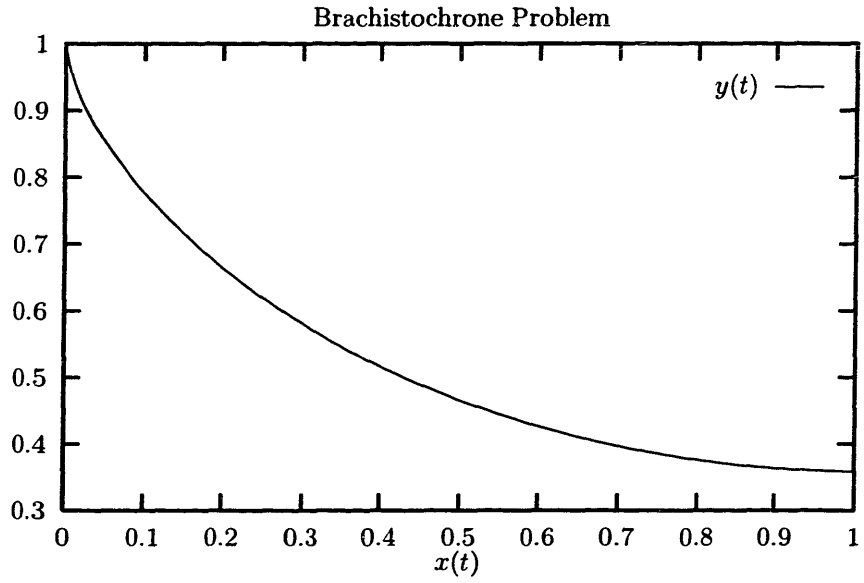

Figure 6-5: *State-space plot for the brachistochrone problem*

specified, and the initial condition:

$$x(0) = 0 \tag{6.81}$$

$$y(0) = 1 \tag{6.82}$$

$$\dot{x}(0) = 0 \tag{6.83}$$

was imposed.

The dummy derivative algorithm calls for differentiating (6.80) once. This dynamic optimization problem was solved using a single linear element to approximate θ. Solution statistics are given in Table 6-2, a state-space plot is shown in Figures 6-5, and the F and θ trajectories are given in Figures 6-6 and 6-7. The solution agrees with numerical solutions obtained for other (lower-index) formulations of the brachistochrone problem.

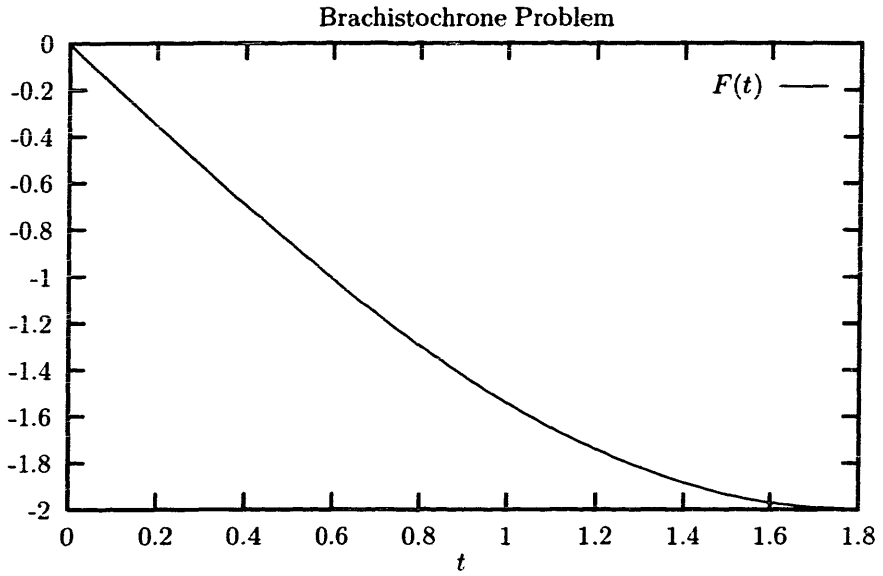

Figure 6-6: *Optimal force trajectory for the brachistochrone problem*

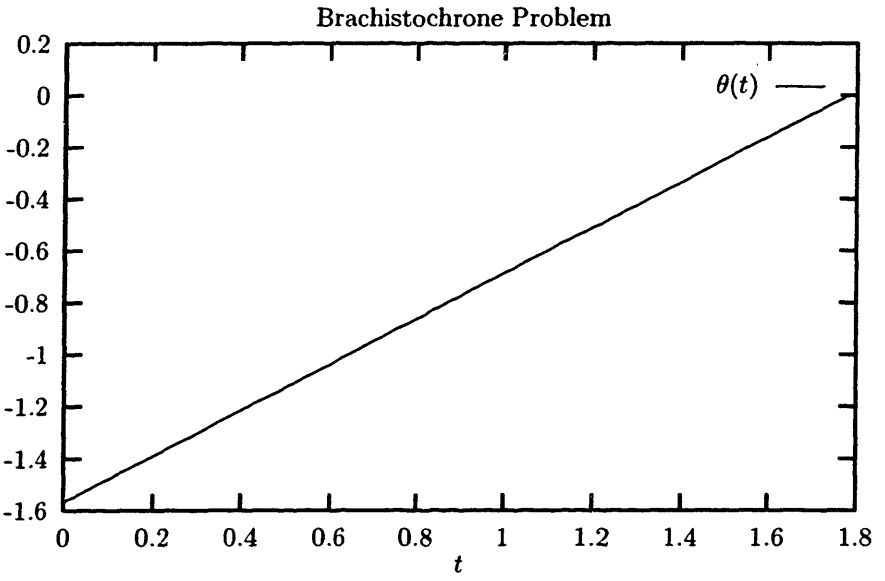

Figure 6-7: *Optimal θ trajectory for the brachistochrone problem*

6.6 Conclusions

This chapter has demonstrated that equality path-constrained dynamic optimization problems require the solution of high-index DAEs. Our method of appending the equality path constraints to the DAE and solving the resulting augmented system directly is superior to indirect methods for handling equality path constraints which rely on including constraint violation measures in the NLP. Not only is our method more efficient, but it eliminates problems that the indirect methods have associated with the error control of the equality path constraints. Also, it sheds light on the difficulties that are involved in specifying an feasible equality path-constrained dynamic optimization problem. Since the DAE is nonlinear, very few general statements can be made about whether it is solvable, and, by extension, whether the dynamic optimization problem is feasible.

The problem of control matching is also very difficult for nonlinear DAEs. However, the structural control matching algorithm is efficient and consistent with the structural criteria that we use to detect and solve high-index DAEs. In the next chapter, we extend these results for inequality path-constrained dynamic optimization problems, where the state variables may not be path-constrained at all points on the state trajectories.

Chapter 7

Inequality Path-Constrained Dynamic Optimization

An inequality-constrained subset of the dynamic optimization problem given in Chapter 2 is considered in this chapter:

$$\min_{u(t),t_f} J = \psi\left(x\left(t_f\right), t_f\right) + \int_{t_o}^{t_f} L\left(x, u, t\right) dt \tag{7.1}$$

subject to the DAE:

$$f\left(\dot{x}, x, u, t\right) = 0 \tag{7.2}$$

$$g(x) \leq 0 \tag{7.3}$$

$$\phi\left(\dot{x}\left(t_o\right), x\left(t_o\right), t_o\right) = 0 \tag{7.4}$$

In this formulation, x are state variables and u are control variables. The DAE (7.2) is assumed to have index ≤ 1, although the results of this chapter are easily extended for equality path-constrained or high-index DAEs for which an equivalent index-1 DAE may be derived using the dummy derivative method.

As in Chapter 6, the focus of this chapter is restricted to direct dynamic optimization methods that use control parameterization to approximate the control variable trajectories. Chapter 6 showed how to include equality path-constraints in

the dynamic optimization problem, but did not address how to include inequality path-constraints, which add an additional layer of complexity to the problem.

Two substantial issues must be addressed to solve problems of the form (7.1–7.4) using control parameterization. First, the DAE (7.2) can become high-index along any trajectory segments on which (7.3) is active, and the trajectories of some subset of u are prescribed during those segments. Therefore, the index of the DAE and the dynamic and design degrees of freedom can fluctuate between segments where the constraints are not active and segments where constraints become active. Second, the sequence of constraint activations and deactivations is unknown *a priori* and thus inequality state-constrained dynamic optimization problems can be viewed from the perspective of *hybrid discrete-continuous dynamic optimization problems.* This fact has not been previously recognized, but it is a very significant insight because it means that there are potentially combinatorial aspects to the solution of inequality constrained dynamic optimization problems.

The first part of this chapter demonstrates the connection between hybrid discrete-continuous and inequality path-constrained dynamic optimization problems. Existing methods for handling these problems using control parameterization are then reviewed. The later sections of the chapter describe three new methods developed in the course of this thesis which take advantage of the high-index nature of state constrained dynamic optimization problems. Of these, the *fluctuating index infeasible path method* has proven the most robust method at present, and examples of its use are provided in Chapter 8. The other two methods, the *modified slack variable method* and the *fluctuating index feasible path method* are not useful for as broad a class of problems, but are described since they are instructive in illustrating the issues involved in solving state inequality constrained dynamic optimization problems. Further, if the nonsmoothness of the resulting master NLP can be dealt with effectively, the fluctuating index feasible path method may be useful in the future because it can handle the sequencing aspect of the problem without introducing integer variables.

7.1 Inequality Path Constraints and the Hybrid Dynamic Optimization Problem

To illustrate the hybrid discrete/continuous formulation for path-constrained dynamic optimization, consider the set of feasible trajectories between two points in state space, subject to inequality constraints on the path between these points. Two such trajectories are illustrated for a two-dimensional state space in Figure 7-1. Each feasible trajectory connecting the two points is composed of a series of constrained and unconstrained segments, where a constrained segment tracks one or more active path constraints.

Trajectories are characterized by a sequence of constraint activations and deactivations at boundaries between constrained and unconstrained segments. A trajectory may be discontinuous at such points (often referred to as *corners* in the optimal control literature [24]). When an unconstrained trajectory intersects one of the path constraints, feasibility may be enforced by augmenting the DAE (7.2) that describes the unconstrained dynamics with the set of active constraints:

$$g_j(x) = 0 \tag{7.5}$$

where g_j is the set of all locally active constraints in (7.3). Each equation in g_j takes up one degree of freedom in the augmented DAE, and therefore for every g_i a control is released and permitted to be implicitly determined by the active inequality. However, an IVP in which the degrees of freedom are fluctuating along the solution trajectory is problematic from the point of view of analysis and solution.

To avoid problems with fluctuating degrees of freedom, the inequality path-constrained problem can be reformulated in a higher dimensional space by relabeling the controls u in (7.2–7.4) as u_c, and introducing a new set of variables $u_s \in \mathbb{R}^{m_u}$. For example, consider a problem with a single control and a single path constraint. In unconstrained portions of the trajectory (*i.e.*, when $g(x) < 0$), the dynamics of the

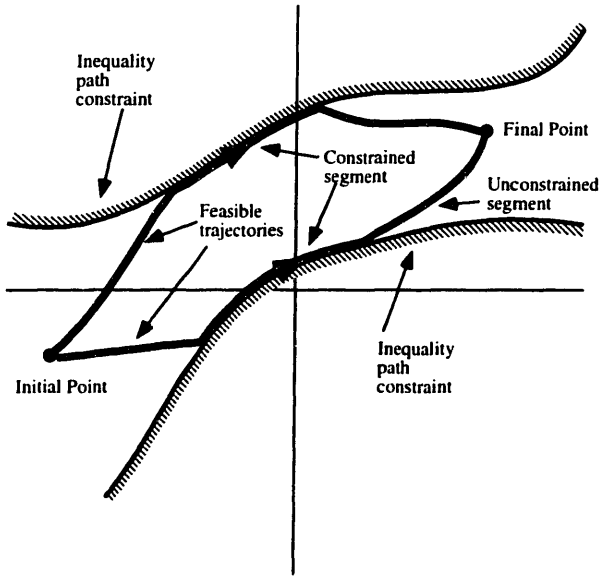

Figure 7-1: *Feasible trajectories in constrained state space*

system are described by:

$$f(\dot{x}, x, u^s, t) = 0 \tag{7.6}$$

$$u^s = u^c \tag{7.7}$$

$$u^c = u^c(t) \tag{7.8}$$

and the constrained portions (*i.e.*, when $g(x) = 0$) as:

$$f(\dot{x}, x, u^s, t) = 0 \tag{7.9}$$

$$g(x) = 0 \tag{7.10}$$

$$u^c = u^c(t) \tag{7.11}$$

Note that (7.7) has been replaced with (7.10), but the degrees of freedom for the overall dynamic system remain unchanged. Thus, during unconstrained portions of the trajectory the control that actually influences the state (u_s) is equivalent to the forcing function for the system (u_c), whereas in constrained portions of the trajectory u_s is determined implicitly by the active path constraint (7.10) and is unrelated to u_c. In a control parameterization context, u_c would be prescribed by the control param-

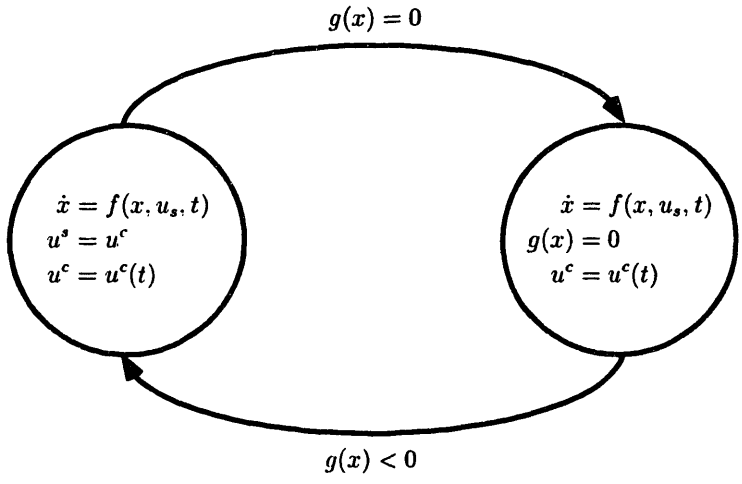

Figure 7-2: *Autonomous switching of constrained dynamic simulation*

eters over the entire time horizon, but during constrained portions of the trajectory it would not influence the solution trajectory.

Equations (7.6–7.11) correspond to a hybrid dynamic system that experiences autonomous switching in response to state (or implicit) events [110]. This behavior is illustrated in Figure 7-2 using the finite automaton representation introduced in [15]. Further, as shown below, the differential index of the constrained problem will in general differ from that of the unconstrained problem.

Given these preliminaries, it is now evident that a path-constrained dynamic optimization with a single control and a single inequality is equivalent to the following hybrid discrete/continuous dynamic optimization problem:

$$\min_{u_c(t),t_f} J = \psi(x(t_f), t_f) + \int_{t_0}^{t_f} L(x, u^s, t)dt \tag{7.12}$$

subject to:

$$f(\dot{x}, x, u^s, t) = 0 \tag{7.13}$$

$$\left\{ \begin{array}{ll} u_s = u_c & \forall t \in T \ : \ g(x) < 0 \\ g(x) = 0 & \forall t \in T \ : \ g(x) = 0 \end{array} \right\} \tag{7.14}$$

Provided each inequality is matched with a unique control, this problem formulation can be extended to multiple controls and inequalities.

The extension to multiple controls and inequalities is evident, provided each inequality is matched with a unique control. Note that this hybrid dynamic optimization problem exhibits the following properties:

- Autonomous switching is defined by implicit events.
- The number and order of implicit events at the optimum is unknown *a priori.*
- The index of the system can fluctuate at the events.

The inequality path-constrained dynamic optimization is but one form of a hybrid discrete/continuous dynamic optimization problem. To illustrate the issues involved in solving such problems, consider the problem, posed by Santos Galán, of a SCUBA diver who would like to come up to the surface in the minimum safe time.

One of the major hazards of diving is decompression sickness, which is caused when the nitrogen (N_2) in the diver's blood and body tissues (which has been dissolved at high pressure during the dive) becomes supersaturated and forms bubbles as the diver ascends from the dive. To avoid decompression sickness, the diver must make "decompression stops" during the ascent to allow the N_2 dissolved in the blood and body tissues to equilibrate with the ambient pressure before the bubbling occurs. At the beginning of this century, J. S. Haldane was appointed by the British Admiralty to develop safer decompression tables for Navy divers, which give the number, depth, and length of the decompression stops required. Haldane experimented with goats and proposed a model describing the uptake and elimination of N_2 as a first order system where the time constants for different tissues are represented by 'half times'. Overall, this yields the formulation:

$$\min_{u(t),t_f,t_b} P_t(t_f) \tag{7.15}$$

subject to:

$$\dot{P}_5 = \frac{\ln(2)}{5}(0.79P - P_5) \tag{7.16}$$

$$\dot{P}_t = -\frac{q}{V_t}P \tag{7.17}$$

$$\dot{P} = u \tag{7.18}$$

$$\begin{cases} \text{State A} : -1 \leq u(t) \leq 3 & \text{Switch to B if } P_5 \geq 1.58P \\ \text{State B} : u(t) = 0 & \text{Wait for 4 min then switch to A} \end{cases} \tag{7.19}$$

$$P_t \geq P \qquad \forall t \in [0, t_f] \tag{7.20}$$

$$P_t(0) = 200 \tag{7.21}$$

$$P(0) = 1 \tag{7.22}$$

$$P(t_b) = 6 \tag{7.23}$$

$$P(t_f) = 1 \tag{7.24}$$

$$0 \leq t_b \leq t_f \tag{7.25}$$

In this example, only a tissue with a five minute half-time is modeled, yielding equation (7.16), where P_5 is the partial pressure of N_2 in the tissue, and P is the surrounding pressure.

The control for this problem is the rate of ascent/descent u, and since pressure P is equivalent to depth, u is equivalent to the rate of change of P (7.18). Diving tables establish decompression stops at different depths depending on the depth-time profile of the dive. These tables follow conceptually the theoretical principles developed by Haldane combined with more recent developments and data. In this simplified example, it is assumed that when the partial pressure in the tissue becomes twice the environmental pressure, the diver must make a decompression stop of four minutes, which effectively constrains the admissible control profile (7.19). Note that discontinuities in the control profile occur at points in time determined by the state trajectory $P_5(t)$ crossing the state trajectory $1.58P(t)$, and therefore they are not known *a priori* but rather are determined implicitly by the solution of the model equations (this type of implicit discontinuity is commonly called a *state event* in the

simulation literature [110]). Hence, the dynamic optimization has to determine the number and ordering of these implicit discontinuities along the optimal trajectory. In addition, the descent and ascent rates are constrained (at 30 m/min and 10 m/min respectively).

Imagine a scenario in which the diver wishes to collect an item from the ocean floor at 50 m with minimum total consumption of air. The volume of a tank of air is $V_t = 15$ L and it is charged to $P_t(0) = 200$ bar. Average air consumption is $q = 8$ L/min, but the mass consumed varies with the depth (pressure), yielding (7.17). A parameter t_b is introduced to model the time at which the diver reaches the bottom, and point constraints (7.23–7.24) force the diver to travel to the bottom and then surface.

In the first approach, two admissible control profiles are examined:

1. Dive as quickly as possible to the bottom, then ascend at 5 m/min, making decompression stops as necessary (Figures 7-3 and 7-4).

2. Dive as quickly as possible to the bottom, then ascend at 10 m/min, making decompression stops as necessary (Figures 7-5 and 7-6).

Observe that the different admissible controls yield different sequences of decompression stops. The consequences of these control policies are that in the first case the diver is forced to make two decompression stops, augmenting the time and air consumption of the dive, whereas in the second case a more rapid ascent rate only requires one decompression stop. Further, if only constant ascent rates are admitted, then if the ascent rate is dropped further, only one stop becomes necessary again. In general, this need to determine the number and ordering of implicit discontinuities along the optimum trajectory appears to be the most difficult issue with the optimization of discontinuous dynamic systems.

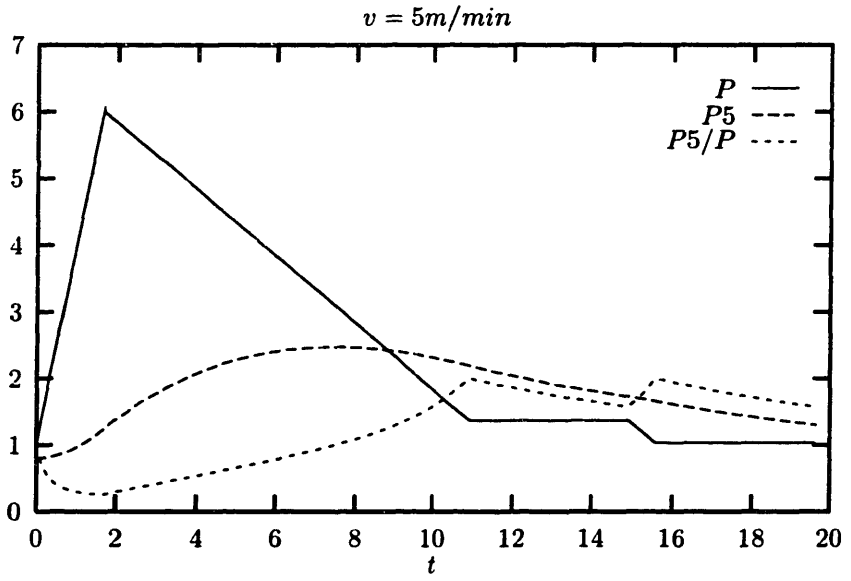

Figure 7-3: *State trajectories for admissible control profile 1*

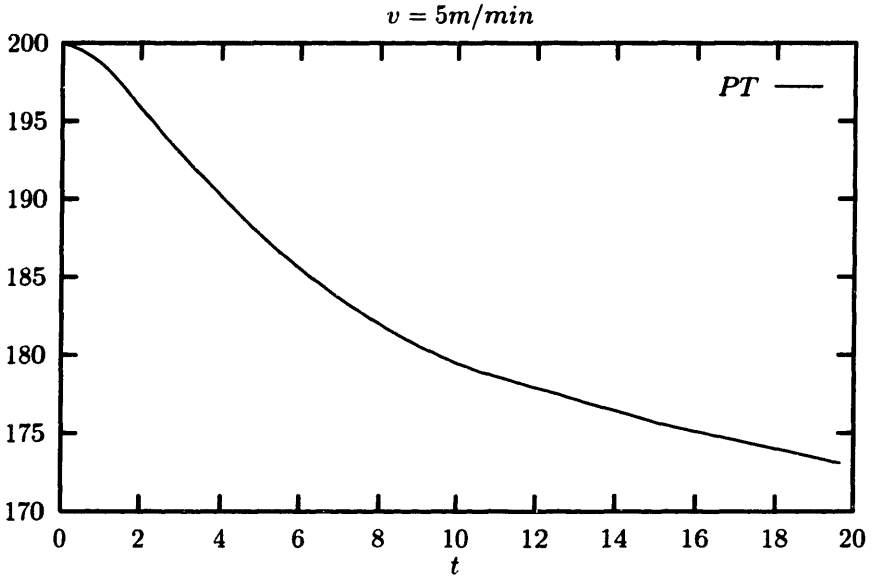

Figure 7-4: *State trajectory for admissible control profile 1*

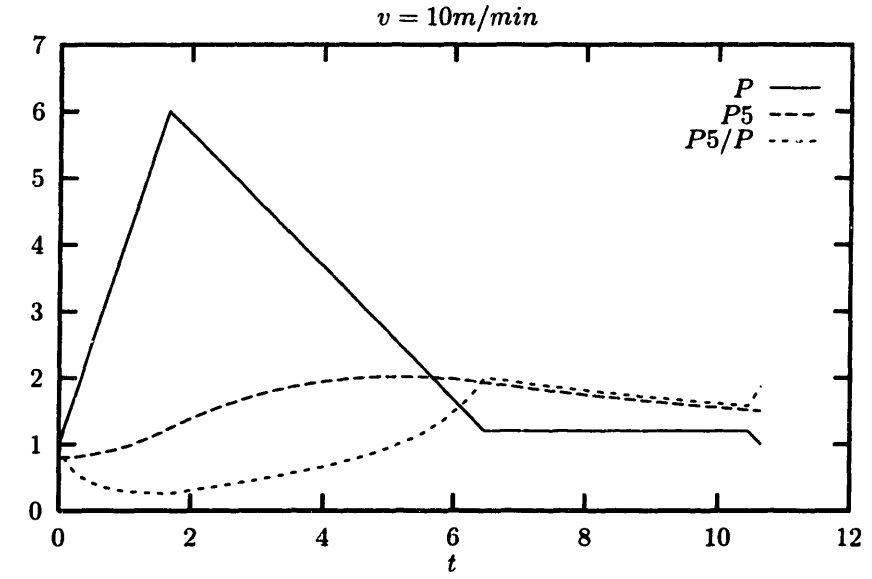

Figure 7-5: *State trajectories for admissible control profile 2*

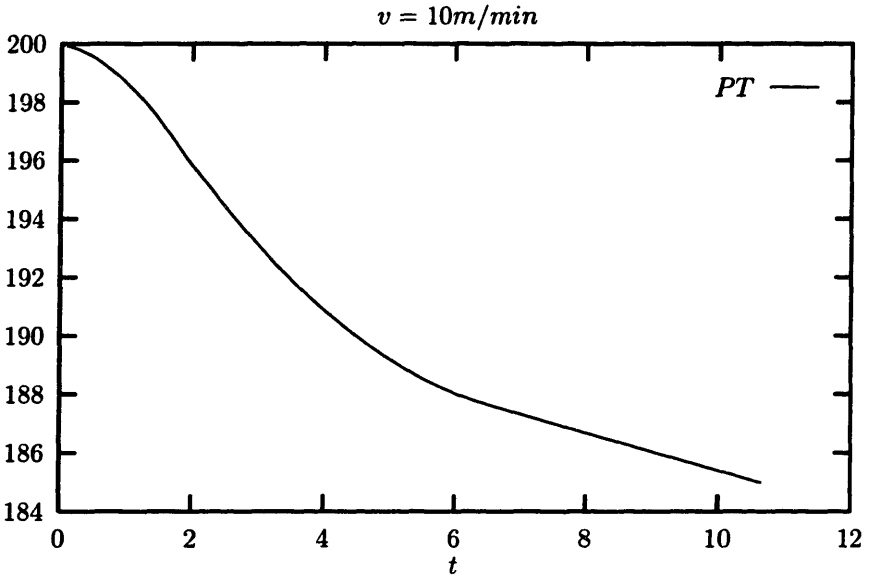

Figure 7-6: *State trajectory for admissible control profile 2*

7.2 Review of Methods for Handling Inequality Path Constraints in Control Parameterization

Several methods have been proposed for handling state variable path constraints within the control parameterization framework. Vassiliadis [142] also reviews such methods. There are four classes of methods: slack variables, random search, penalty functions, and interior-point constraints.

7.2.1 Slack variable approach

Although the slack variable approach [74, 75, 140] (often known as Valentine's method) was not developed for use with control parameterization, its extension to this framework is straightforward. The terminology and approach detailed in [74] is somewhat *ad hoc* because the paper predates all of the work on DAEs. However, this work can be put in the context of modern DAE theory.

The slack variable approach described in [74] is valid when the DAE (7.2) is an ODE:

$$\dot{x} = f(x, u, t) \tag{7.26}$$

The method proceeds by appending the state variable inequality (7.3) to (7.26) using a slack variable $a(t)$:

$$g(x) - \frac{1}{2}a^2 = 0 \tag{7.27}$$

where it is assumed here that $g(\cdot) \to \mathbb{R}$. Squaring the slack variable has the effect of enforcing the inequality constraint for all admissible trajectories of a.

Equation (7.27) is then differentiated:

$$g^{(1)}(x) - aa^{(1)} = 0 \tag{7.28}$$

$$g^{(2)}(x) - (a^{(1)})^2 - aa^{(2)} = 0 \tag{7.29}$$

$$\vdots$$

$$g^{(k)}(x) - \frac{1}{2}\frac{d^k}{dt^k}a^2(t) = 0 \tag{7.30}$$

where $g^{(i)} = \frac{d^i g}{dt^i}$ and $a^{(i)} = \frac{d^i a}{dt^i}$. The differentiation is carried out p times until the system formed by (7.26–7.30) may be solved for some $u_i \in u$. The control u_i is then eliminated from the problem, $a^{(k)}$ becomes a control variable, and the time derivatives $a \ldots a^{(k-1)}$ become differential state variables. Note that the $a^{(k-1)}$ are essentially being enforced as dummy derivatives, and in fact this method is very similar to the elimination index-reduction method of [22].

There are several significant problems with this method:

- As noted by [74], the dynamic optimization problem contains a singular arc whenever $a(t)$=0. In control parameterization this singular arc has the effect of providing no gradient information to the optimizer whenever $a(t)$=0, and thus significantly slowing down or preventing convergence of gradient-based NLP methods.

- Also observed in [74] is that there is no obvious extension of this method for problems that contain more state path inequality constraints than control variables.

- A fundamental characteristic of this method is that stationarity of the slack variable underlying index-1 problem is only a necessary condition for stationarity of the original problem. Thus it is possible that a stationary solution of the slack variable problem is not a stationary solution of the original problem.

- It can be difficult to solve (7.26–7.30) to eliminate u_i, particularly if (7.26) is large and nonlinear.

7.2.2 Random search technique

A random search dynamic optimization solution technique was proposed in [7, 8, 32]. Although this method uses control parameterization, it does not make use of gradient information in the NLP optimizer. Instead, normal probability distributions are used to generate different decision parameter vectors, which are then used to solve the DAE and evaluate the objective function.

Random search techniques handle state variable path constraints by discarding any iteration for which the path constraint is violated. The obvious disadvantage of this method is the large cost associated with solving numerous infeasible subproblems. However, the method has been used with some success on extremely nonconvex and nonsmooth dynamic optimization problems for which gradient-based optimization techniques experience difficulties.

7.2.3 Penalty functions

Another method that has been used to enforce inequality path constraints is the use of a penalty function (for example [24, 146]). One way to use penalty functions is to augment the objective function (7.1) as:

$$\tilde{J} = J + K \int_{t_0}^{t_f} r^T(t) r(t) dt \qquad r \in \mathbb{R}^{n_g} \tag{7.31}$$

or as:

$$\hat{J} = J + \sum_{i=1}^{n_g} K_i \int_{t_0}^{t_f} r_i(x(t)) dt \tag{7.32}$$

where $K \in \mathbb{R}_+$ is a large positive number and $r(t) \in \mathbb{R}^{n_g}$ is a measure of the constraint violation, *e.g.*:

$$r_i(x(t)) = \max\left(g_i(x(t)), 0\right) \qquad i = 1 \ldots n_g \tag{7.33}$$

This approach can cause numerical difficulties because it requires $K \to \infty$ to satisfy the constraint exactly. In addition, many authors that have used operators such as max in (7.33) have not recognized that such operators introduce implicit discontinuities that require special treatment during integration [110] and calculation of sensitivities [55].

7.2.4 End-point constraints

An inequality path constraint can be transformed into an end-point constraint $\Psi(t) \in \mathbb{R}^{n_g}$ (for example [128]):

$$\Psi_i(x(t)) = \int_{t_0}^{t_f} r_i(x(t))dt = 0 \qquad i = 1 \ldots n_g \tag{7.34}$$

This approach also causes numerical difficulties because the gradients of the end-point constraints are zero at the optimum, which reduces the rate of convergence near the solution.

Penalty function methods and end-point constraint methods suffer from problems due to the selection of a suitable $r(t)$. In [133] computational experience was reported which showed that using $r(t)$ of the form (7.33) rarely converges for nontrivial dynamic optimization problems. The primary reason for these difficulties is that (7.33) is nondifferentiable when $g_i(t) = 0$. Such constraints also introduce implicit discontinuities in the simulation subproblem, which must also be handled carefully to avoid numerical difficulties [110]. In [133], a different $r(t)$ was proposed of the form:

$$r_i(t) = \begin{cases} g_i(t) & \text{if } g_i(t) > \epsilon \\ (-g_i(t) - \epsilon)^2/4\epsilon & \text{if } -\epsilon \leq g_i(t) \leq \epsilon \\ 0 & \text{if } g_i(t) \leq -\epsilon \end{cases} \qquad i = 1 \ldots n_g \tag{7.35}$$

This smooth function is differentiable when $r_i(t) = 0$ and contains no implicit discontinuities. However, (7.35) does reduce the feasible state space region since the true inequality path constraints can only be active in the limiting case when $\epsilon = 0$.

7.2.5 Interior-point constraints

Another method proposed to deal with inequality path constraints is to discretize them via interior-point constraints (for example [142, 143]):

$$\rho(g(t_i)) \leq \epsilon_i \qquad i = 0 \ldots n_{pc} \tag{7.36}$$

where $\rho(\cdot) : \mathbb{R}^{n_g} \rightarrow \mathbb{R}$ is a function that provides a measure of the violation of the constraints at a given t_i, the parameter $\epsilon_i \in \mathbb{R}_+$ is a small positive number, and n_{pc} is the number of point constraints.

The problem with this method is that $n_{pc} \rightarrow \infty$ is necessary to guarantee that the state path constraint is not violated during any portion of the optimal trajectory. Since each point constraint adds a dense row to the Jacobian of the constraints of the NLP, large numbers of point constraints can create NLPs that cannot be solved using currently available numerical algorithms. If only a few point constraints are used (one per finite element is typical) there is no guarantee that state trajectories will satisfy the state constraint to within the tolerances that were used to obtain a numerical solution of the IVP.

7.2.6 Hybrid interior-point constraint and penalty function approach

A hybrid approach was used in [33, 142] that combines both the penalty function and interior point constraint methods. This approach transforms the state variable inequality path constraint into an end point inequality constraint:

$$\chi_i(t_f) \leq \epsilon_i \tag{7.37}$$

where ϵ_i is a small positive number, which is evaluated by appending the following equation to: (7.2)

$$\dot{\chi}_i = [\max(0, g_i(x))]^2 \tag{7.38}$$

$$\chi(0) = 0 \tag{7.39}$$

and a set of interior point inequality constraints:

$$g_i(x(t_k)) \leq 0 \qquad k = 1 \ldots N_{FE} \tag{7.40}$$

where t_k are the boundaries of the control finite elements. Although the constraint (7.37) is in principle sufficient to ensure that the original inequality constraint is not violated, it provides no information to the NLP solver when $g_i(x) = 0$. The point constraints (7.40) are included to provide some information to the optimizer when $g_i(x) = 0$.

The disadvantages of this hybrid method are:

- The state variable path constraints are satisfied to within known tolerances only at the points during the state trajectory where NLP point constraints have been imposed. The state variable path constraints may not be satisfied to within acceptable tolerances at other points along the trajectory.

- The value of the control that causes the constraint to be active is implicitly defined, and therefore the NLP solver must iterate to find the control whenever an inequality path constraint is active. Such iteration decreases the computational efficiency of the method. This method can be viewed in some sense as solving a high-index DAE by iterating on an index-1 DAE until all of the equations of the high-index DAE are satisfied.

- The use of the max operator in (7.38), coupled with the discrete nature of (7.40) can cause the method to converge slowly if the constraint violation is occurring at points along the solution trajectory other than at the end of the control finite elements.

7.3 A Modified Slack Variable Approach

This section describes a modification to the slack variable approach that is a significant improvement over the original slack variable method. As mentioned in the previous section, there are several difficulties with the original slack variable approach that have kept it from being widely used. An example of these difficulties can be seen by considering the following problem which was discussed in [66, 74, 90, 142]:

$$\min_{u(t)} V = \int_0^1 \left(x_1^2 + x_2^2 + 0.005u^2\right) dt \tag{7.41}$$

subject to:

$$\dot{x}_1 = x_2 \tag{7.42}$$

$$\dot{x}_2 = -x_2 + u \tag{7.43}$$

$$x_2 - 8(t - 0.5)^2 + 0.5 \leq 0 \tag{7.44}$$

$$x_1(0) = 0 \quad x_2(0) = -1$$

$$t \in [0, 1]$$

A slightly different form was used in [66, 142] where (7.43) was replaced by $\dot{x}_2 = x_2 + u$.

When the method of [74] is applied to this problem (7.41–7.44), (7.44) is replaced by:

$$x_2 - 8(t - 0.5)^2 + 0.5 + \frac{1}{2}a^2 = 0 \tag{7.45}$$

which is then differentiated, producing:

$$\dot{x}_2 - 8(t - 0.5)^2 + 0.5 + aa_1 = 0 \tag{7.46}$$

where $a_1 = (da/dt)$. Equation (7.46) is used to eliminate u, producing the following

unconstrained problem:

$$\min_{a_1(t)} V = \int_0^1 \left(x_1^2 + x_2^2 + 0.005(x_2 + 8(t-0.5) - aa_1)^2\right) dt \tag{7.47}$$

subject to:

$$\dot{x}_1 = x_2 \tag{7.48}$$

$$\dot{x}_2 = -aa_1 + 8(t-0.5) \tag{7.49}$$

$$\dot{a} = a_1 \tag{7.50}$$

$$x_1(0) = 0 \quad x_2(0) = -1 \quad a(0) = \sqrt{5}$$

$$t \in [0,1]$$

The results of a numerical solution of this problem using the ABACUSS control parameterization algorithm are shown in the first row of Table 7-1.

Table 7-1: *Dynamic optimization results for solution of equations (7.41–7.44)*

Solution Method	Objective Function	Major Iter.	Total IVP	Opt. Tol.	IVP Tol.	# Finite Elements	Lagrange Approx.
Slack Variable	0.171042	89	100	10^{-5}	10^{-7}	10	linear
Mod. Slack Var.	0.170586	40	46	10^{-5}	10^{-7}	10	linear
Mod. Slack Var.	0.170243	31	38	10^{-5}	10^{-7}	10	quadratic
Mod. Slack Var.	0.169875	74	81	10^{-7}	10^{-9}	10	quadratic

Although a solution to the dynamic optimization was found, convergence of the optimizer was very slow due to the presence of the singular arc. In addition, the algebraic manipulation required to find (7.48–7.50) is not general enough to be applicable to all problems, and it is very difficult to do automatically for large problems.

The original motivation for performing these algebraic manipulations was to eliminate u from (7.46) so that the resulting problem (7.48–7.50) is an ODE. Since it is now possible to solve high-index DAEs directly using the dummy derivative method described in Chapter 5, this limitation no longer applies, and the algebraic manipulations can be avoided by allowing the DAE integrator to solve for u. Thus, using the

dummy derivative method on (7.41–7.44) gives:

$$\min_{a_1(t)} V = \int_0^1 \left(x_1^2 + x_2^2 + 0.005u^2\right) dt \tag{7.51}$$

subject to:

$$\dot{x}_1 = x_2 \tag{7.52}$$

$$\dot{x}_2 = -x_2 + u \tag{7.53}$$

$$x_2 - 8(t - 0.5)^2 + 0.5 + \frac{1}{2}a^2 = 0 \tag{7.54}$$

$$\dot{x}_2 - 16(t - 0.5) + aa_1 = 0 \tag{7.55}$$

$$x_1(0) = 0 \quad x_2(0) = -1$$

$$t \in [0, 1]$$

However, H_1^1 matrix for this problem that is used in the dummy derivative method is:

$$H_1^1 = [a] \tag{7.56}$$

and therefore no equivalent index-1 DAE can be selected when $a = 0$. As described in Chapter 5, the BDF corrector matrix for (7.52–7.55) becomes highly ill-conditioned when $a = 0$, and the numerical solver is incapable of integrating past this point.

A modified slack variable method is proposed here to handle both the problems with algebraic manipulations and problems with singular arcs. The modification is to make a the control variable, and to apply the method of dummy derivatives to the resulting high-index DAE.

Thus, the state inequality-constrained problem given by (7.1), (7.2), and (7.3) may be transformed into a state *equality*-constrained problem using a slack variable

a:

$$\min_{a(t),t_f} J = \psi\left(x\left(t_f\right), t_f\right) + \int_{t_o}^{t_f} L\left(x, u, t\right) dt \tag{7.57}$$

subject to the DAE:

$$f\left(\dot{x}, x, u, t\right) = 0 \tag{7.58}$$

$$g(x) = -\frac{1}{2}a^2 \tag{7.59}$$

$$\phi\left(\dot{x}\left(t_o\right), x\left(t_o\right), t_o\right) = 0 \tag{7.60}$$

provided that the dimension of g does not exceed that of u, and that the resulting (possibly high-index) DAE is solvable using the dummy derivative method.

Using this modified slack variable method on the problem (7.41–7.44) gives:

$$\min_{a(t)} V = \int_0^1 \left(x_1^2 + x_2^2 + 0.005u^2\right) dt \tag{7.61}$$

subject to:

$$\dot{x}_1 = x_2 \tag{7.62}$$

$$\dot{x}_2 = x_2 + u \tag{7.63}$$

$$x_2 - 8(t - 0.5)^2 + 0.5 = -\frac{1}{2}a^2 \tag{7.64}$$

$$x_1(0) = 0 \quad a(0) = \sqrt{5}$$

$$t \in [0, 1]$$

Then, the dummy derivative method can be applied to (7.62–7.64) to obtain an

equivalent index-1 DAE:

$$\dot{x}_1 = x_2 \tag{7.65}$$

$$\bar{x}_2 = x_2 + u \tag{7.66}$$

$$x_2 - 8(t - 0.5)^2 + 0.5 = -\frac{1}{2}a^2 \tag{7.67}$$

$$\bar{x}_2 = 16(t - 0.5) = -a\bar{a} \tag{7.68}$$

$$\bar{a} = \frac{d}{dt}a(t) \tag{7.69}$$

where $\bar{x}$ and $\bar{a}$ are (dummy derivative) algebraic state variables.

This problem does not have a singular arc when $a = 0$, and the corrector matrix for the equivalent index-1 formulation (7.65–7.69) does not become ill-conditioned when $a = 0$. Solution statistics for this problem are also shown in Table 7-1. They show that the modified slack variable approach required significantly fewer iterations to find the optimum, and actually found a slightly better value for the optimal objective function.

Figure 7-7 shows the optimal trajectory for the control variable u. The u trajectory contains discontinuities at the control finite element boundaries, which is due to the choice of control parameterization of a. Since a controls an equation that involves only a state variable, u is related to the derivative of a. Since a has been control parameterized with linear finite elements with C^0 continuity, a variable such as u that is algebraically dependent upon the derivative of a can not be C^0 continuous at the finite element boundaries. The discontinuities in u can be eliminated if higher-order polynomials are used in the control parameterization of a. Figure 7-8 corresponds to the last row of Table 7-1, and shows the trajectory of u when quadratic finite elements are used to approximate a.

The ability to solve high-index DAEs has allowed the creation of a modified slack variable method with performance characteristics that are superior to the original method. The modified version is interesting because it clearly shows the link between state inequality constrained dynamic optimization problems and high-index DAEs.

However, there are a number of problems with this method that keep it from becoming generally useful. They are:

- It does not appear to be possible to extend the method for problems that contain more inequalities than control variables. In general, it is not possible to have more independent active state variable constraints at any point in time than there are control variables. However, there exist many problems that have more state inequality constraints than control variables, but that never have more *active* constraints than they have control variables, which could not be solved using this method.

- The problem becomes high-index for its entire trajectory, even if a state inequality constraint never becomes active. This phenomenon requires us to solve a equivalent index-1 DAE that may be much larger than the DAE of the problem without constraints, thus decreasing the efficiency of the overall optimization.

- The choice of control parameterization can be very tricky. Since a is being parameterized, this essentially forms an approximation for a state variable trajectory. There may be no information *a priori* that allows a suitable choice for this approximation. Also, there is no way to simultaneously restrict the trajectory of u, either to impose path constraints or to restrict the allowable function space.

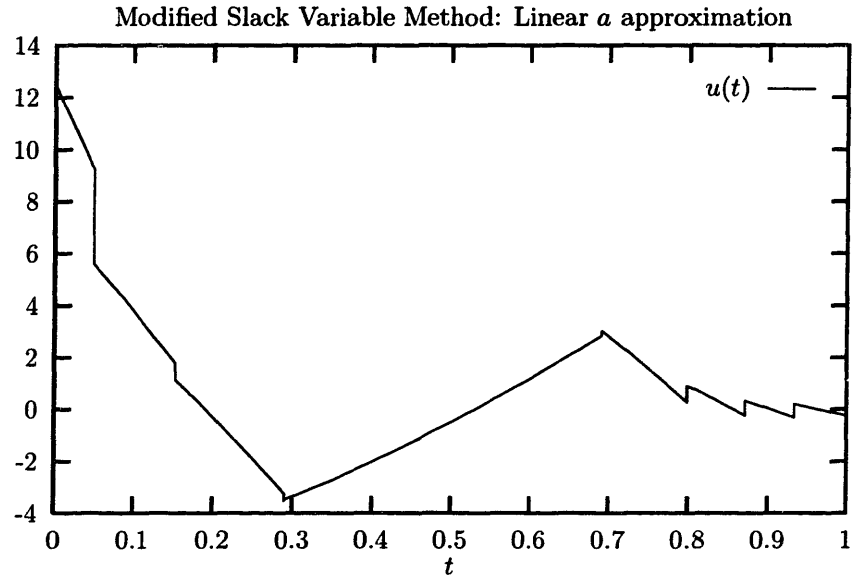

Figure 7-7: *Control variable in modified slack variable method example with 'a' approximated by linear finite elements*

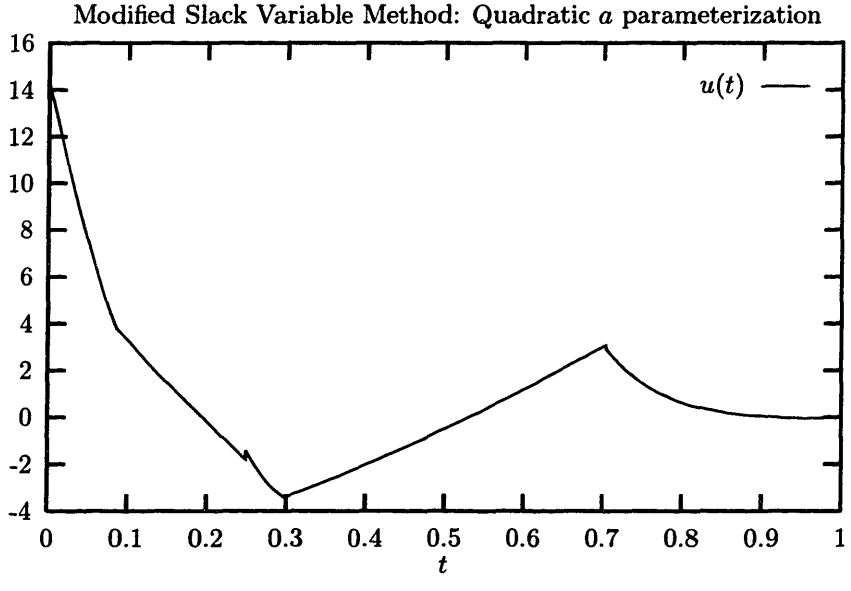

Figure 7-8: *Control variable in modified slack variable method example with 'a' approximated by quadratic finite elements*

7.4 Fluctuating-Index Feasible-Path Method for Dynamic Optimization with Inequality Path Constraints

A novel method for solving inequality path-constrained dynamic optimization problems within the control parameterization framework is described in this section. The *fluctuating-index feasible path* (FIFP) method enforces state variable path constraints through direct solution of high-index DAEs during the state-constrained portions of the solution trajectory. Thus, this method is a *feasible path* method from the standpoint of the inequality path constraints, which are never violated during intermediate iterations of the NLP solver.

The inequality path constraints are not violated during intermediate iterations because the FIFP method uses an *implicit event constrained dynamic simulation algorithm.* That is, the hybrid discrete/continuous problem given by (7.13–7.14) is solved directly. The activation and deactivation of the inequality path constraints are implicit state events that are detected during the integration. Inequality path constraint activation and deactivation events are termed *constraint events.* Implicit event constrained dynamic simulation requires detecting the constraint events, determining which controls cease to be degrees of freedom in the constrained portions of the trajectory, and integrating numerically the resulting (possibly) fluctuating-index DAE.

The implicit event constrained dynamic simulation problem is:

$$f(\dot{x}, x, u^s, t) = 0 \tag{7.70}$$

$$u^c = u^c(t) \tag{7.71}$$

$$\alpha_i(u_i^s - u_i^c) + \sum_{j=1}^{n_g} \beta_{ij} g_j(x) = 0 \quad \forall i = 1 \ldots n_u\,,\ \forall t \in T \tag{7.72}$$

$$\alpha_i + \sum_{j=1}^{n_g} \beta_{ij} = 1 \quad \forall i = 1 \ldots n_u\,,\ \forall t \in T \tag{7.73}$$

$$\sum_{i=1}^{n_u} \beta_{ij} \leq 1 \quad \forall j = 1 \ldots n_g\,,\ \forall t \in T \tag{7.74}$$

$$\alpha_i = \alpha_i(t) \in \{0, 1\} \qquad \beta_{ij} = \beta_{ij}(t) \in \{0, 1\}$$

This formulation shows how the equations in the active DAE change upon activation and deactivation of path constraints. The variables α and β are determined by the solution of the implicit event constrained dynamic simulation problem. When a constraint g_j becomes active, some $\beta_{ij} = 1$ and $\alpha_i = 0$. The β_{ij} and α_i is determined by matching controls u_i to constraints g_j, which is obtained using the method described in Chapter 6. It is assumed that all path constraints are inactive at the initial condition, and therefore that $\alpha(t_0) = 1$ and $\beta(t_0) = 0$.

Assuming that a standard BDF integration method is used to solve the DAE, the algorithm performed at each step of the integration is:

1. Determine whether any of the inequality constraints $g(x)$ activated or deactivated in the last time step. If no, go to Step 5.

2. Detect the earliest constraint activation or deactivation event that occurred during the last time step:

   (a) For every active $g_j(x)$, set one $\beta_{ij} = 1$ and $\alpha_i = 0$. Set all other $\beta \backslash \{\beta_{ij}\} = 0$ and $\alpha \setminus \{\alpha_i\} = 1$.

   (b) Reset the simulation clock to the time of the event.

3. Apply the method of dummy derivatives to derive an equivalent index-1 formulation for the current problem.

4. Consistently initialize the equivalent index-1 formulation.

5. Take one integration step with the current index-1 formulation of the problem, performing dummy pivoting as necessary. If not at the end of the simulation, go to Step 1.

From a practical standpoint, there are several aspects of the above algorithm that need to be described in detail. Among these are detecting of activation and deactivation of constraints and transferring initial conditions between constrained and unconstrained portions of the trajectory. These details are discussed in the following subsections.

7.4.1 Constraint activation

An inequality constraint activates at the earliest point in time at which the previously inactive constraint becomes satisfied with equality. These points in time are often termed *state events*, because they are defined implicitly by the evolution of the system state and are not known *a priori*. The state events marking the constraint activation may be identified algorithmically by introducing a new algebraic variable z_j for each currently inactive constraint:

$$z_j = g_j(x) \tag{7.75}$$

These *discontinuity functions* are appended to the current equivalent index-1 formulation of the problem, and constraint activations are detected by advancing the simulation speculatively until a zero crossing occurs in $z_j(t)$. This situation corresponds to a special case of the state events handled by the algorithm described in [110], which handles equations of the form (7.75) efficiently, and guarantees that all state events are processed in strict time order.

7.4.2 Constraint deactivation

Detecting deactivation of inequality constraints during a simulation is somewhat more complicated than detecting activation because any u_i^c which has a corresponding $\alpha_i = 0$ does not influence the current state variable trajectories. During solution of the equivalent index-1 system using the method of dummy derivatives, the corrector iteration assures that any active constraints remain satisfied within the relevant numerical tolerance, and therefore an additional condition must be considered to detect deactivation of the inequality.

An inequality state constraint is considered to have deactivated when switching some α_i from 0 to 1 would cause the state trajectories to move into the feasible region. This situation is detected by introducing a new set of equalities:

$$\alpha_i(u_i^s - u_i^c) + \sum_{j=1}^{n_g} \beta_{ij}(g_j(x) - \epsilon_j) = 0 \qquad i = 1, \ldots n_u \tag{7.76}$$

Equation (7.76) contains a set of parameters $\{\epsilon_j\}$ such that when $\sum_j \beta_{ij} = 1$ and $\epsilon_j = 0$, the corresponding constraint g_j is active; and if $\epsilon_j \leq 0$, the corresponding constraint has been "tightened" and the feasible state space decreased. As shown below, (7.76) is not actually solved; rather it allows the calculation of the sensitivity of the system to $\{\epsilon_j\}$, which is used to detect inequality deactivation.

Associated with each new equality (7.76) is the sensitivity system:

$$\frac{\partial f}{\partial \dot{x}}\frac{\partial \dot{x}}{\partial \epsilon_j} + \frac{\partial f}{\partial x}\frac{\partial x}{\partial \epsilon_j} + \frac{\partial f}{\partial u^s}\frac{\partial u^s}{\partial \epsilon_j} = 0 \quad \forall j \tag{7.77}$$

$$\alpha_i \frac{\partial u_i^s}{\partial \epsilon_j} + \sum_{k=1}^{n_g} \beta_{ik} \left[\frac{\partial g_k}{\partial x}\right]^T \frac{\partial x}{\partial \epsilon_j} = \sum_{k=1}^{n_g} \beta_{ik}\delta_{jk} \quad \forall i, j \tag{7.78}$$

where δ_{jk} is the Krönecker delta function. Note that, although a large number of sensitivity equations are defined by (7.77–7.78), all sensitivities with respect to ϵ_j are identically zero if $g_j(x)$ is inactive. Provided that all $\beta_{ij} = 0$ at t_0, all sensitivities with respect to ϵ may be initialized to zero. Transfer conditions for these sensitivities when constraints become active are the same as the general sensitivity transfer conditions

given in Chapter 3.

Deactivation of the inequality is detected by solving this sensitivity system to find $(\partial u_i^s/\partial \epsilon_j)$ and detecting the simulation times that are implicitly defined by zero crossings of the discontinuity functions:

$$q_i^a = u_i^c - u_i^s \tag{7.79}$$

$$q_{ij}^b = \frac{\partial u_i^s}{\partial \epsilon_j} \tag{7.80}$$

$$\forall (i,j) : \beta_{ij} = 1$$

These algebraic equations are appended to the DAE and solved as detailed in [110].

Constraint deactivation is detected using the following theorem:

Theorem 7.1. *If at any state event time defined by a zero crossing of (7.80), the condition:*

$$\left(\left(\frac{\partial u_i^s}{\partial \epsilon_j} < 0\right) \wedge (u_i^c > u_i^s)\right) \vee \left(\left(\frac{\partial u_i^s}{\partial \epsilon_j} > 0\right) \wedge (u_i^c < u_i^s)\right) \tag{7.81}$$

is true, the constraint $g_j(x)$ is considered to have deactivated.

Proof. A negative value of ϵ_j corresponds to tightening the constraint, and thus moving the solution trajectory into the feasible region. If the sensitivity $(\partial u_i^s/\partial \epsilon_j)$ is negative, then u_i^c must increase for the solution trajectory to move away from the inequality and into the feasible region. Therefore, if u_i^c becomes greater than u_i^s at some point in the time domain, the specified control will inactivate the inequality. The reasoning for the case in which $(\partial u_i^s/\partial \epsilon_j) > 0$ follows the same logic. □

7.4.3 Transfer conditions for state variables

The constraint events are transition conditions as defined in Chapter 3. Transition functions are also needed that relate the state variables before the constraint event

to the state variables after the constraint event. The transition function is:

$$T(\dot{x}^{(-)}, x^{(-)}, u^{(-)}, \dot{x}^{(+)}, x^{(+)}, u^{(+)}, p, t) = 0 \tag{7.82}$$

where the "−" denotes the values before the constraint activation, and the "+" denotes the values after activation. In some cases, (7.82) could describe a loss function upon activation of the constraint, *i.e.*, frictional losses due to an object interacting with a surface. However, the usual method for determining this transition function is to specify some subset of the state variables x and y which remain continuous across the constraint event. The dimension of the vector function T is equal to the dynamic degrees of freedom of the DAE that is active after the constraint event.

In discrete/continuous dynamic simulation it is often assumed that all differential state variables x are continuous across a discontinuity in the input functions u [15]. This assumption has been proven to be valid for ODEs and for most physically meaningful index-1 DAEs (in fact, all DAEs with $i_s = 0$)[22], but several authors [22, 64, 93] have noted that the assumption is not true for some index-1 and higher-index DAEs.

Continuity conditions for arbitrary-index DAEs are stated in [64], assuming that the index and the vector field is the same on both sides of the discontinuity. The condition is that state variables are continuous across a discontinuity in the controls provided that the corresponding underlying ODE does not depend on any derivatives of the controls that are causing the discontinuity. A successive linear programming (SLP) algorithm was proposed in [64] to find the dependencies of the variable derivatives on the derivatives of the controls that are causing the discontinuity.

The continuity conditions given in [64] are not directly applicable to the FIFP method because the FIFP method constraint events cause the DAE after the constraint event to have a different vector field and possibly a different index. It is possible, however, to define transition functions for the FIFP method by noting that there must be a corresponding explicit input-jump simulation for any valid implicit event simulation. That is, there must be some input function $\bar{u}(t)$ for which the

simulation:

$$f(\dot{x}, x, u^s, t) = 0 \tag{7.83}$$

$$u^s = \bar{u}(t) \tag{7.84}$$

produces the same state variable trajectories as the implicit-event simulation (7.70–7.74). Note that $\bar{u}(t)$ is permitted to contain discontinuities. The existence of a corresponding explicit input-jump simulation is guaranteed because a valid implicit event simulation must define a unique trajectory u^s which can then be used to set $\bar{u}(t)$ in the explicit input-jump simulation. The corresponding explicit input-jump simulation is not necessarily unique since there could be more than one $\bar{u}(t)$ which produce the same state trajectories.

The transition conditions in [64] are valid for the corresponding explicit input-jump simulation. Since the implicit event and explicit input-jump simulations produce the same state trajectories, the transition conditions for the state variables must also be the same for the two DAEs. In practice, defining the transition conditions for the implicit event simulation is even easier than the SLP algorithm in [64] would imply when the explicit input-jump simulation has index ≤ 1, since then it is possible to assume continuity of the variables that are differential state variables in the explicit input-jump simulation.

Since the index of the DAE in the implicit event simulation may be greater than the index of the DAE in the explicit input-jump simulation, there may be more continuity assumptions for state variables than there are dynamic degrees of freedom for the implicit event simulation. In general, the number of degrees of freedom available will be less than or equal to the number of continuity conditions required for (7.2). For example, if (7.2) has $i_s = 0$, then the number of conditions on the continuity of x must be equal to the dimensionality of x, whereas the number of differential state variables in the augmented equivalent index-1 system derived by the method of dummy derivatives will be equal to or less than the dimensionality of x. Hence, in general only that subset of the continuity conditions corresponding to the differential

state variables remaining in the equivalent index-1 DAE describing the new segment can be satisfied by the implicit event constrained dynamic simulation algorithm. The additional continuity assumptions may or may not be automatically enforced by the DAE itself.

To see an example of continuity assumptions for the FIFP method, consider the DAE:

$$\dot{x}_1 = x_1 + x_2 \tag{7.85}$$

$$\dot{x}_2 = u + \sin(t) \tag{7.86}$$

$$u = u(t) \tag{7.87}$$

$$x_1 + x_2 \leq 10 \tag{7.88}$$

The continuity assumptions for (7.85–7.87) upon a jump in $u(t)$ are that x_1 and x_2 are continuous. However, an equivalent index-1 system for the implicit event simulation after activation of (7.88) is:

$$\bar{x}_1 = x_1 + x_2 \tag{7.89}$$

$$\dot{x}_2 = u + \sin(t) \tag{7.90}$$

$$x_1 + x_2 = 10 \tag{7.91}$$

$$\bar{x}_1 + \dot{x}_2 = 0 \tag{7.92}$$

which has only one dynamic degree of freedom. However, either of the transition functions $x_1^{(+)} = x_1^{(-)}$ or $x_2^{(+)} = x_2^{(-)}$ may be chosen, since whichever continuity assumption is not explicitly defined will in this case be implicitly enforced by (7.91).

Trajectories generated by the implicit event constrained dynamic simulation algorithm will potentially not satisfy all the continuity conditions required for an optimal trajectory. However, this complication can be handled by allowing the continuity conditions to be violated by the IVP subproblems at intermediate iterations, and then adding the continuity conditions at constraint activation/deactivation through the use of point constraints in the master NLP. Thus, the master NLP will adjust the

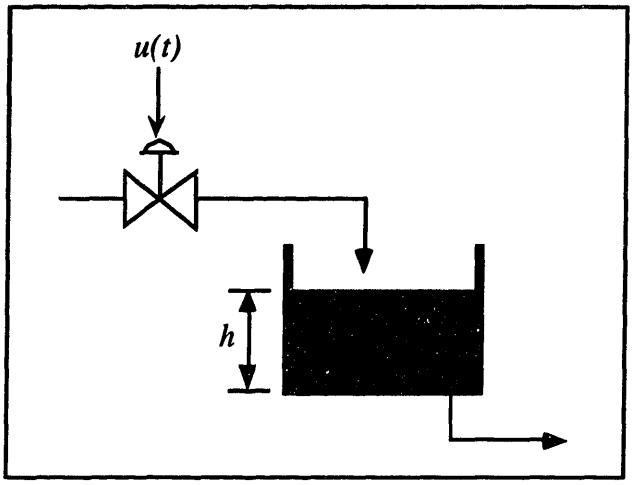

Figure 7-9: *Tank and valve*

control profiles in a manner such that at the optimum, the extra continuity conditions that cannot be enforced by the implicit event constrained dynamic simulation algorithm are satisfied by the NLP. Thus, while the method is always feasible with respect to path constraints, it is not necessarily feasible with respect to continuity conditions at intermediate iterations.

As a simple physical example, consider a constrained dynamic optimization in which the liquid level in the tank shown in Figure 7-9 may not exceed a given threshold. The valve stem position controlling the flow into the tank is a differential state in the unconstrained DAE and exhibits a first order response to the control signal.

$$\dot{h} = (F_{in} - F_{out})/A \tag{7.93}$$

$$F_{out} = f(h) \tag{7.94}$$

$$\dot{F}_{in} = g(u - u_0) \tag{7.95}$$

where F_{in} and F_{out} are the flowrates in and out of the tank, A is the area of the tank, h is the liquid height in the tank, and f and g are functions that describe, respectively, the hydraulics of the flow out of the tank and the flow response to the control signal.

Since the liquid height is constrained to be lower than h_{max}, there is an inequality

constraint on h:

$$h \leq h_{max} \tag{7.96}$$

When this inequality is active the control u becomes an algebraic state variable in the resulting high-index augmented DAE. Applying the dummy derivative method to the augmented DAE results in the following equivalent index-1 DAE:

$$\bar{h} = (F_{in} - F_{out})/A \tag{7.97}$$

$$\tilde{h} = (\bar{F}_{in} - \bar{F}_{out})/A \tag{7.98}$$

$$F_{out} = f(h) \tag{7.99}$$

$$\bar{F}_{out} = f'(h) \tag{7.100}$$

$$\bar{F}_{in} = g(u - u_0) \tag{7.101}$$

$$h = h_{max} \tag{7.102}$$

$$\bar{h} = 0 \tag{7.103}$$

$$\tilde{h} = 0 \tag{7.104}$$

where $\bar{h}$, $\tilde{h}$, $\bar{F}_{in}$, and $\bar{F}_{out}$ are dummy algebraic variables ($\tilde{h}$ eliminates $\ddot{h}$).

Equations (7.97) and (7.103) indicate that the flow into the tank must equal the flow out to maintain the level at its threshold, and hence the stem position u required to achieve this is also prescribed algebraically. Since for an arbitrary control signal profile there is no guarantee that the stem position immediately before the constraint activation will equal that prescribed at the beginning of the constrained segment of the trajectory, it must be allowed to jump, which is clearly non-physical and forbidden by the continuity conditions for the unconstrained DAE. In fact, in this case the equivalent index-1 DAE has no dynamic degrees of freedom to define the initial condition, so both the stem position and the level could potentially jump. However, continuity of the liquid level will always be redundant with the liquid level inequality, so only the stem position can actually jump. Although the constrained dynamic simulation subproblems allow jumps, the point constraints in the master NLP will

force the optimal control profile to be one in which the stem position does not jump. If the unconstrained DAE (7.2) is already high index (*e.g.*, if it includes equality path constraints), then the continuity conditions stated in [64] must be enforced by point constraints in the master NLP.

7.4.4 Sensitivity transfer conditions at inequality deactivation

The points along the solution trajectory where inequality deactivations occur are defined by implicit events. Therefore, the sensitivity variable transfer conditions are defined by the relations given in Chapter 3. However, the deactivation event (7.81) requires special attention because it contains a sensitivity variable.

For the deactivation condition (7.81) to be true, one of (7.79–7.80) must hold at the constraint activation time t^*. The transfer conditions for the sensitivity functions follow the derivation given in Chapter 3. Obtaining the sensitivity transfer functions requires differentiating the discontinuity function, which if (7.79) is currently true gives:

$$\frac{\partial u_i^{c-}}{\partial p} + \dot{u}_i^{c-}\frac{\partial t^*}{\partial p} = \frac{\partial u_i^{s-}}{\partial p} + \dot{u}_i^{s-}\frac{\partial t^*}{\partial p} \tag{7.105}$$

for all $\alpha_i = 0$.

If the discontinuity function is instead (7.80), the sensitivity transfer function is:

$$\frac{\partial^2 u_i^{s-}}{\partial \epsilon_j \partial p} + \frac{\partial \dot{u}_i^{s-}}{\partial \epsilon_j}\frac{\partial t^*}{\partial p} = 0 \tag{7.106}$$

This condition has the added complication that it requires second-order sensitivity information, namely $(\partial^2 u_i^s / \partial \epsilon_j \partial p)$. Obtaining this second order sensitivity information requires the solution of the second-order analog to (7.77–7.78). It is possible to extend the algorithm described in Chapter 4 to handle these second order sensitivities. Although the second order sensitivities are more costly to calculate than first-order sensitivities, only a few are required and hence the additional cost should

not be significant.

There are two possible behaviors for the control variable upon deactivation of an inequality. The state and parameterized trajectories of the control may cross, or else the special sensitivity may cross zero, making the control jump. The latter case is more costly to handle in a numerical algorithm because of the need for second order sensitivities.

7.4.5 Example

An example demonstrating the use of the fluctuating-index feasible-path method for dynamic optimization is the *constrained brachistochrone problem* which was presented both in [24] and in a slightly different form in [84]. The problem is to find the shape of a wire on which a bead will slide between two points under the force of gravity in minimum time.

The formulation of the constrained brachistochrone problem considered here is [84]:

$$\min_{\theta(t),t_f} t_f \tag{7.107}$$

subject to:

$$\dot{x} = v\cos\theta \tag{7.108}$$

$$\dot{y} = v\sin\theta \tag{7.109}$$

$$\dot{v} = g\sin\theta \tag{7.110}$$

$$y \geq ax - b \tag{7.111}$$

$$x(t_f) = x_f \tag{7.112}$$

$$[x(t_0), y(t_0), v(t_0)] = [0, 0, 0] \tag{7.113}$$

where t_f is the final time, g is the gravitational constant, and γ and b are given. Equation (7.111) defines an inequality constraint on the state variables x and y, (7.112) defines a final-point constraint, and (7.113) gives the initial conditions.

Table 7-2: *Solution statistics for constrained brachistochrone*

Initial Guess	QP	LS	CPU
$\theta(t) = -1.558 + 2.645t$	2	3	0.38s
$\theta(t) = -1.6 + 2.2857t$	3	3	0.47s
$\theta(t) = 0.0$	3	5	0.52s
$\theta(t) = -1.6$	5	5	0.67s
$\theta(t) = -1.0$	6	6	0.75s

This problem was solved using the feasible-path control parameterization method described in this section. The discretization chosen for the control $\theta(t)$ was a single linear element. There were therefore three parameters in the NLP (the two for the control variable and t_f). The parameters g, a, b, and x_f were set to -1.0, -0.4, 0.3, and 1.1 respectively. The integrator relative and absolute tolerances (RTOL and ATOL) were both set to 10^{-7} and the optimization tolerance was 10^{-5}. The optimal objective function value was 1.86475. The ABACUSS input file is shown in Figure 2-2, and Table 7-2 gives solution statistics from different initial guesses for the control variable. In this table, "QP" refers to the number of master iterations of the NLP solver, "LS" refers to the number of line searches performed, and "CPU" gives the total CPU time for the solution on an HP C160 workstation. These statistics compare very favorably with those reported for other similarly-sized problems [142].

Figure 7-10 shows the state and control variable solution of the constrained brachistochrone problem described above using the constrained dynamic optimization algorithm. The 'kink' in the control variable profile is due to the control becoming determined by the high-index DAE during the constrained portions of the trajectory.

7.4.6 Critique of the FIFP method

The FIFP method is interesting because it is the first method proposed that explicitly addresses the two issues that have been shown to be characteristic of inequality state-constrained dynamic optimization: the high-index segments of the state trajectory and the hybrid discrete/continuous nature of the problem. FIFP appears to solve the

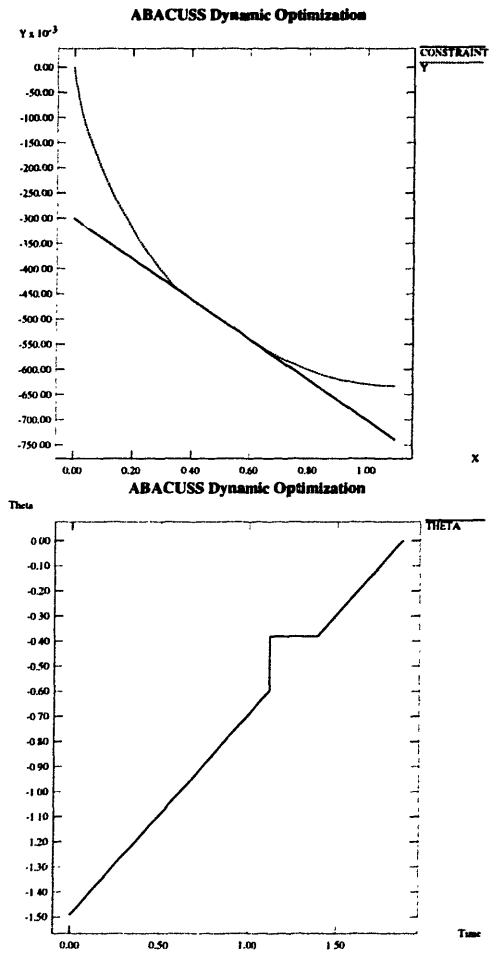

Figure 7-10: *Solution of constrained brachistochrone problem*

problems associated with the modified slack variable approach, since it has the ability to handle more inequality constraints than controls (provided that there is no point in the problem where there are more *active* inequality constraints than controls), and very little extra work is performed when constraints are inactive.

However, experience with the FIFP method has shown that it suffers from a pathological problem. Recall from Chapter 3 that part of the *sufficient* conditions for existence and uniqueness of sensitivity functions was that the sequence of state events remains qualitatively the same: the location of the state events in state space may change, but existence and uniqueness cannot be guaranteed if the ordering of the events changes. This problem was demonstrated in Figures 3-3, 3-4, and 3-5, where a differential change in a parameter produced a discrete jump in the values of both

the state and sensitivity functions.

The fact that an infinitely small movement in the value of a parameter can produce a finite jump in the value of a state variable implies that the dynamic optimization master NLP is not smooth. Nonsmooth NLPs often prevent convergence of gradient-based NLP solution techniques (such as SQP algorithms). Because such methods are generally based on assumptions of smoothness, nonsmooth problems can prevent convergence or slow it to an unacceptably slow speed.

The FIFP method does work for some problems like the simple constrained brachistochrone problem shown above because the FIFP problems are "smooth" in some neighborhood of the optimum. That is, small movements of the parameter values away from their optimal values generally do not change the sequence of events and thus the objective function does not contain discontinuities. However, larger movements of the parameters from their optimal values do cause the state variables to jump. For example, in the constrained brachistochrone problem, some values of the constraints produce state trajectories for which the inequality path constraint does not become active. Fortunately for the constrained brachistochrone problem presented above, the region around the optimum for which the sequence of events remains constant is fairly large and convergence is fairly robust.

In a nutshell, the problem with the FIFP method is that it has an *unpredictable* and potentially very small convergence region. That is, gradient based NLP algorithms can be expected to converge to a local optimum (if one exists) from a starting guess in the neighborhood of the optimum. This convergence neighborhood is defined as the set of parameter values that produce the same constraint event sequence as the locally optimal parameter values do. Since the constraints are defined implicitly, there is no general method at present for defining the convergence neighborhood without actually solving hybrid IVPs.

Note that the convergence problem with the FIFP algorithm exists only when FIFP is used with gradient-based NLP algorithms. Nonsmoothness of the NLP may not be a problem for non-gradient based (*e.g.*, stochastic) optimization algorithms (see, for example [8]). Since the FIFP method generates trajectories that are always

feasible with respect to the path constraints, stochastic methods can then avoid the need to discard many infeasible trajectories, and will be significantly more efficient.

7.5 Fluctuating-Index Infeasible-Path Method for Dynamic Optimization with Inequality Path Constraints

The *fluctuating-index infeasible path* (FIIP) method for inequality constrained dynamic optimization avoids the convergence difficulties that the FIFP method experiences when the constraint event ordering changes during intermediate iterations by requiring that each dynamic simulation subproblem follows the same sequence of constraint activations and deactivations. This requirement is imposed by explicitly defining the constraint activation and deactivation times. The method is an *infeasible* path method from the standpoint of the state variable inequality path constraints because these constraints are not required to be satisfied for all time at intermediate iterations of the FIIP NLP.

Since the inequality path constraints may be violated during intermediate iterations, an *explicit event constrained dynamic simulation algorithm* can be used to solve the IVP subproblem. The activations and deactivations of the state variable inequality path constraints occur at explicitly defined points in the solution trajectory that are determined by the master NLP solver. During segments of the state trajectory where an inequality constraint $g_k(x)$ is defined to be active, the equality:

$$g_k(x) = \epsilon_k \qquad g_k \subseteq g \tag{7.114}$$

is added to the DAE using the method described in Chapter 6. The parameter ϵ_k is used because at the time when the constraint is defined explicitly to be active, the values of the state variables may not satisfy the constraint. Therefore, the constraint is offset by ϵ_k such that it is active at the current time, which is shown in Figure 7-11. The overall NLP contains a constraint that forces $\epsilon_k = 0$ at the optimum. Note that the overall state variable inequality path constraint $g_k(x) = 0$ is active when $\epsilon_k = 0$.

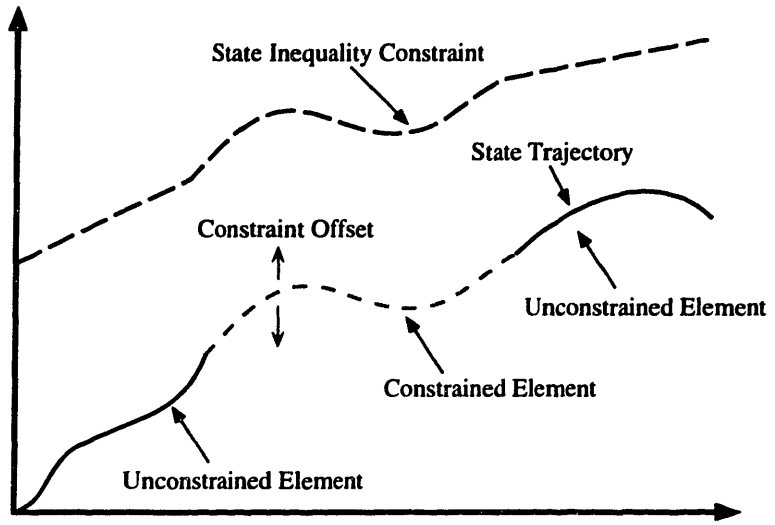

Figure 7-11: *The FIIP method works by offsetting the constraint to the state variable trajectory at points where the constraint activates*

The explicit event constrained dynamic simulation algorithm is:

$$f(\dot{x}, x, u^s, t) = 0 \tag{7.115}$$

$$u^c = u^c(t) \tag{7.116}$$

$$\alpha_i^{(k)}(u_i^s - u_i^c) + \sum_{j=1}^{n_g} \beta_{ij}^{(k)} \left(g_j(x) - \epsilon_j^{(k)} \right) = 0 \tag{7.117}$$

$$\epsilon_j^{(k)} = g_j\left(x(t_k), y(t_k)\right) \tag{7.118}$$

$$\alpha_i^{(k)} + \sum_{j=1}^{n_g} \beta_{ij}^{(k)} = 1 \tag{7.119}$$

$$\sum_{j=1}^{n_g} \beta_{ij}^{(k)} \leq 1 \tag{7.120}$$

$$\alpha_i^{(k)}, \beta_{ij}^{(k)} \in \{0, 1\}$$

$$i = 1 \ldots n_u \qquad k = 1 \ldots N_{FE}$$

As in the implicit event dynamic simulation method (7.70–7.74), the α and β variables are used to express the matching between active state path constraints and the control variables that are not free to be independently specified in the resulting high-index DAE. However, in this case, the α and β variables are not determined by

the occurrence of an implicitly defined state event, but instead are defined explicitly for each control variable finite element. Therefore, the α and β are functions of the finite element index, rather than the simulation time. The start and end times of the finite elements are in general optimization parameters that are determined by the NLP solver.

The ϵ_j variable in (7.117) permits continuity assumptions for the state variables when the constraint activates by allowing the inequality path constraint to be violated. Essentially, ϵ_j is used to offset the constraint so that the state variables do not jump when it activates. The amount of constraint violation is measured at the point where the constraint activates, and remains constant throughout the entire constrained segment. The inequality constraint is satisfied at the optimum by imposing a constraint in the master NLP such that $\epsilon_j = 0$.

Assuming a standard BDF method is used to solve the DAE, and that a set of α and β variables have been defined, the steps of the explicit event dynamic simulation method are:

1. Solve the IVP until the end of the next control finite element is encountered at $t = t_k$.

2. If $t_k = t_f$ then stop.

3. For every j where $\sum_{i=1}^{m_u} \beta_{ij} = 1$, determine $\epsilon_j^{(k)}$ by solving:

$$g_j(x) = \epsilon_j^{(k)} \tag{7.121}$$

   at $t = t_k$.

4. Apply the method of dummy derivatives to derive an equivalent index-1 formulation of the current problem.

5. Consistently initialize the equivalent index-1 formulation.

6. Set $k = k + 1$. Go to Step 1.

The FIIP method is less complicated than the FIFP method because there are no implicit constraint activations and deactivations. However, there are still two aspects of the algorithm which require further explanation: how to specify the transfer conditions for the state variables when constraints activate, and how to determine the sequence of activations and deactivations.

7.5.1 Transfer conditions for state variables

As with the FIFP method, transition conditions must be defined for the state variables when a constraint becomes active. Even though the constraint event time is explicitly defined in the FIIP method, special consideration is required to define the FIIP transition conditions because the dynamic degrees of freedom and possibly the index of the DAE will change after a constraint event.

The principle that was used to define the state transition conditions in Section 7.4.3 for the FIIP method is useful for the FIFP method as well. There is an explicit input-jump simulation that corresponds to the explicit event constrained dynamic simulation, and the continuity conditions for the explicit input-jump system can be used to derive continuity assumptions for the explicit event simulation. The use of the offset parameter ϵ with the active constraints removes the ambiguity that would be associated with constraint activation. For example, suppose that the constraint:

$$x_1 + x_2 + x_3 \leq 0 \tag{7.122}$$

became active when the values of the state variables were $x_1 = -1$, $x_2 = -2$, and $x_3 = -1$, the corresponding explicit input jump simulation indicated that all of the state variables in the constraint remain continuous, but the dynamic degrees of freedom for the DAE after the constraint activation indicated that any two of the three state variables can be specified to remain continuous. Then, one of the state variables would have to experience a jump. However, allowing one of the state variables to jump would violate the corresponding explicit input-jump simulation

continuity assumptions. Also, it is ambiguous as to which state variable would jump. If however the active constraint is defined as:

$$x_1 + x_2 + x_3 = \epsilon \tag{7.123}$$

where $\epsilon = -4$, then no state variable will jump, even though continuity is enforced on only two of the three state variables.

To illustrate the state variable transfer conditions under the FIIP method, consider the DAE:

$$\dot{x} = x + y \tag{7.124}$$

$$y = 3x + u \tag{7.125}$$

$$u = u(t) \tag{7.126}$$

The usual way to reinitialize this DAE at a time t^* when a discontinuity in u is encountered is to assume that the differential state variable x is continuous across the discontinuity. Therefore, a consistent initial condition for the system that is active after the discontinuity would be obtained by solving the algebraic system consisting of (7.124–7.125) and

$$x^{(+)} = x^{(-)} \tag{7.127}$$

$$u = u(t^*) \tag{7.128}$$

to find values for $x^{(+)}$, $y^{(+)}$, and $\dot{x}^{(+)}$. Now suppose that (7.124–7.125) is the DAE part of a dynamic optimization problem that is being solved using the FIIP method and the constraint:

$$x \leq 5 \tag{7.129}$$

becomes active at $t^{(*)}$. After application of the dummy derivative method, the system

solved to find a consistent initialization at $t^{(*)}$ is:

$$\bar{x}^{(+)} = x^{(+)} + y^{(+)} \tag{7.130}$$

$$y^{(+)} = 3x^{(+)} + u^{(+)} \tag{7.131}$$

$$x^{(+)} = 5 + (x^{(-)} - 5) \tag{7.132}$$

$$\bar{x}^{(+)} = 0 \tag{7.133}$$

where $\bar{x}$ is a dummy derivative and $\epsilon = (x^{(-)} - 5)$ in this case. Note that even though this system does not require specification of additional transfer conditions, the differential state variable x remained continuous when the constraint activated. The same state trajectories could have been obtained with the appropriate jump in u in the system (7.124–7.125).

7.5.2 Point constraints

Theoretically, there is no need for additional constraints in the NLP to handle inequality path constraints in the FIIP method. However, including a point constraint representation of the inequality path constraints improves the robustness of the FIFP method. If the point constraints are not included, the FIFP method sometimes finds locally optimal solutions for which the inequality path constraints are satisfied during finite elements where they are defined to be active, but violated during finite elements where the constraint is not supposed to be active. Including a measure of the inequality constraint violation at a discrete number of points as an inequality constraints in the NLP tends to solve this problem. However, since these NLP inequality constraints increase the size of the NLP, it is desirous not to include too many. In practice, one NLP inequality constraint is included per inequality path constraint per finite element.

7.5.3 Specifying a sequence of constraint events

Specifying the sequence of constraint events can be difficult in the FIIP method. The FIIP method decouples the inequality path-constrained dynamic optimization problem into discrete and continuous subproblems. The discrete subproblem is to find the optimal sequence of constraint activations and deactivations. The continuous subproblem is, given a fixed sequence of constraint activations and deactivations, to find the optimal control trajectories. The use of the explicit event constrained dynamic simulation algorithm allows us to solve the continuous subproblem.

Solution of the discrete subproblem is combinatorial in nature, yielding a mixed-integer dynamic optimization subproblem (MIDO) [1, 3]. Putting the FIIP NLP in the form of the MIDO formulation given in [3] yields:

$$\min_{u(t),v,\alpha,\beta,t_f} \phi(x(t_f),u(t_f),v,t_f) + \int_0^{t_f} L(x(t),u(t),v,t)dt \tag{7.134}$$

subject to:

$$f(\dot{x},x,u^s,t) = 0 \tag{7.135}$$

$$u^c = u^c(t) \tag{7.136}$$

$$\alpha_i^{(k)}(u_i^s - u_i^c) + \sum_{j=1}^{n_g} \beta_{ij}^{(k)} \left(g_j(x) - \epsilon_j^{(k)}\right) = 0 \tag{7.137}$$

$$\epsilon_j^{(k)} = g_j\left(x(t_k), y(t_k)\right) \tag{7.138}$$

$$\epsilon_j^{(k)} = 0 \tag{7.139}$$

$$\alpha_i^{(k)} + \sum_{j=1}^{n_g} \beta_{ij}^{(k)} = 1 \tag{7.140}$$

$$\sum_{j=1}^{n_g} \beta_{ij}^{(k)} \leq 1$$

$$\alpha_i^{(k)}, \beta_{ij}^{(k)} \in \{0,1\}$$

$$i = 1 \ldots n_u \qquad k = 1 \ldots N_{FE}$$

At present, there exist no rigorous strategies for solution of general MIDOs. Further,

in the worst case where there are many controls, inequality constraints, and control finite elements, the only rigorous strategy at present that will guarantee an optimal answer is to enumerate explicitly all of the possible constraint event sequences and compare the optimal values obtained by solving the associated continuous subproblems.

Most dynamic optimization problems with state inequality constraints that have been solved in the optimal control literature to date contain only one inequality constraint and one control variable. These problems may be easily solved using the FIIP method. The best way is to solve the unconstrained problem and determine the finite elements during which the constraint is violated. Then, use the FIIP method to solve the constrained problem where those same finite elements are the ones where the constraint is defined to be active. An alternate method for single-control, single-inequality problems is to specify alternating constrained and unconstrained finite elements. Since the end points of the finite elements are optimization parameters, the optimization problem can determine where the constraint activation takes place by shrinking the length of some elements to zero if necessary.

For problems with multiple inequality constraints, one *ad hoc* strategy is to solve the unconstrained problem and determine the sequence of constraint violations, and then solve the constrained problem with that sequence. This strategy does not guarantee an optimal solution and some enumeration of constraint event orderings is usually required. On the other hand, other methods for handling inequality path constraints generally have problems with multiple constraints, and the FIIP method does provide an extremely efficient method (see Chapter 8) to obtain the optimal solution when the constraint activation order is known.

7.6 Conclusions

The advantage of the three methods described in this chapter over other methods for solving inequality path constrained dynamic optimization problems is that these guarantee that the inequality path constraints are not violated during an optimal

solution. In addition, these methods explicitly recognize the high-index nature of path constrained problems, and employ the dummy derivative method to solve them efficiently.

This chapter has clearly shown the hybrid discrete/continuous nature of inequality constrained dynamic optimization problems that arises because of the need to make decisions concerning the ordering of constraint events. Although the modified slack variable method is capable of transforming some inequality path-constrained problems into completely continuous optimization problems, the range of problems that it can handle is limited.

The advantage of the FIFP method over other solution methods is that the discrete nature of the inequality constrained problem is explicitly recognized, and solution is possible if the constraint activation sequence is known. Although it is difficult at this time to use the FIFP method to solve dynamic optimization problems with many constraints because of the combinatorial nature of these problems, dynamic optimization problems with multiple inequality constraints are in fact very difficult to solve by any currently existing method. Furthermore, as shown in the next chapter, the method works well for a broad range of constrained dynamic optimization problems.

Chapter 8

Numerical examples

In this chapter, the FIIP algorithm is applied to several interesting examples in order to demonstrate the ease with which it can be used to solve state-constrained dynamic optimization problems that are considered difficult to solve using other methods. In particular, the abilities of FIIP to track nonlinear constraints, to guarantee that constraints are not violated at the optimum, and to solve problems without the use of complex control basis functions and many finite elements are demonstrated.

An implementation of FIIP in ABACUSS was used to solve these problems. The Harwell code VF13 is used to solve the master NLPs. VF13 uses a sequential quadratic programming method with a gradient-based line search technique, and hence all of the IVP subproblems are required to solve the combined DAE and sensitivity system. This combined system is solved using the DSL48S code detailed in Chapter 4. ABACUSS uses the efficient automatic differentiation algorithm of [135, 134] to obtain the Jacobian of the DAE and right hand sides of the sensitivity equations. ABACUSS does sacrifice some computational efficiency compared to methods where the equations and derivatives are hard-coded and compiled, because it creates and stores the equations and derivatives as memory structures. However, this computational inefficiency is more than compensated by the modeling efficiency that the ABACUSS input language and architecture provide.

There are several solution statistics reported for each problem. The *integration tolerance* was used for both the relative and absolute tolerance for the numerical IVP

solver. The *optimization tolerance* was used for the NLP solver. The number of *IVP solutions* is a measure of how difficult the NLP was to solve, since most of the solution time is spent in numerical integration of the DAE and sensitivity equations. The *CPU* times reported were obtained on an HP 9000 C160 single-processor workstation. The number of IVP solutions and the CPU time statistics are of only limited usefulness since they are highly dependent on the initial guess employed for the control variables, however they are included for use in comparing some of the examples with results obtained in other works that used the same initial guesses.

For most of the problems, results for several tolerance levels are shown. It is common with control parameterization for the number of IVP subproblems required to solve the problem to increase significantly as the tolerances are tightened. This effect occurs because the objective function is often relatively flat near the optimum with respect to the exact positions of the finite element boundaries. When the tolerances are tightened, the optimizer continues to move the finite element boundaries, resulting in small improvements in the objective function value. The solution trajectories shown for each example correspond to the results for the tightest tolerances that were tried. However, the solution statistics for most of the problems show that the work done increases immensely at tight tolerance levels, and the improvements to the solution are very small. Reasonably accurate solutions may be obtained without resorting to the strictest tolerances that were used here.

8.1 Line-Constrained Brachistochrone

The constrained brachistochrone problem was presented both in [24] and in a slightly different form in [84]. The formulation used here is that of [84], who solved the problem using a discretized constraint control parameterization algorithm. The problem is to find the shape of a wire on which a frictionless bead will slide between two points under the force of gravity in minimum time. The brachistochrone is constrained by requiring it to lie on or above a line drawn in state space.

The formulation of the constrained brachistochrone problem that was considered is [84]:

$$\min_{\theta(t),t_f} t_f \tag{8.1}$$

subject to:

$$\dot{x} = v\cos\theta \tag{8.2}$$

$$\dot{y} = v\sin\theta \tag{8.3}$$

$$\dot{v} = g\sin\theta \tag{8.4}$$

$$y \geq ax + b \tag{8.5}$$

$$x(t_f) = x_f \tag{8.6}$$

$$[x(t_0), y(t_0), v(t_0)] = [0, 0, 0] \tag{8.7}$$

where t_f is the final time, g is the gravitational constant, and a and b are given constants. Equation (8.5) defines an inequality constraint on the state variables x and y, (8.6) defines a final-point constraint, and (8.7) gives the initial conditions.

Following [84], the parameters used were $g = -1.0$, $a = -0.4$, $b = 0.2$, and $x_f = 1.0$. The starting guess for the control was $\theta(t) = -\pi/2$ and for the final time was $t_f = 2.0$. The control was discretized using three linear finite elements, and the inequality constraint was active during the second finite element. The optimal trajectories for the states and controls are shown in Figure 8-1 and Figure 8-2. Solution statistics

Integration Tolerance	Optimization Tolerance	IVP Solutions	Objective Function	CPU
10^{-7}	10^{-5}	8	1.8046	0.59s
10^{-9}	10^{-7}	18	1.795220	2.06s
10^{-11}	10^{-9}	22	1.79522045	3.98s

Table 8-1: *Statistics for the line-constrained brachistochrone problem*

are given in Table 8-1.

Our results compare favorably with the results reported in [84], which required 26 IVP solutions to solve the problem from the same initial guess using an optimization tolerance of 10^{-6}, even though only three first-order control finite elements were used for a total of 7 parameters, compared to the ten first-order control finite elements for a total of 31 parameters used in [84]. The method used in [84] discretizes the inequality constraint using NLP point constraints, and hence must use a large number of finite elements to ensure that one lies near the point on the state trajectory at which the inequality constraint activates. Since the FIIP method does not have to have a point constraint near where the constraint activates, the method is more efficient because it requires fewer sensitivity equations and smaller NLPs.

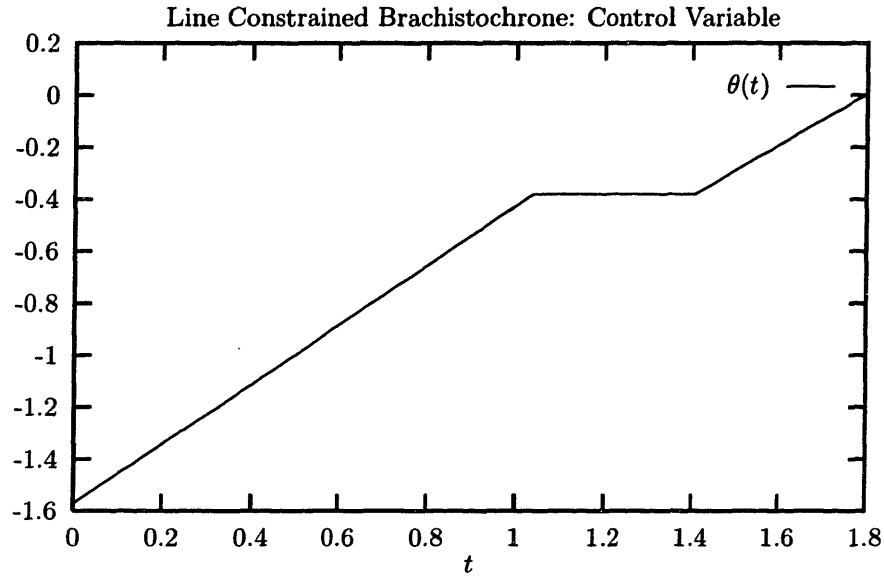

Figure 8-1: *Line-constrained brachistochrone control variable trajectory*

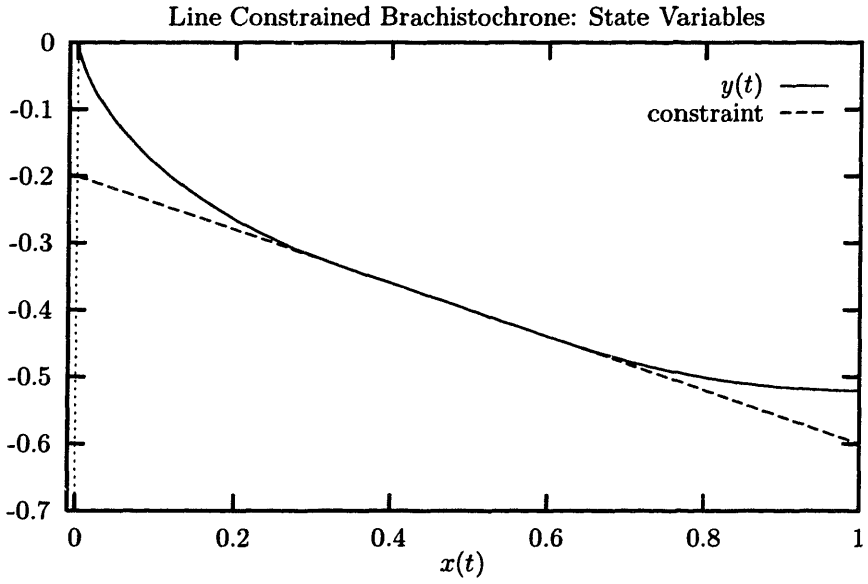

Figure 8-2: *Line-constrained brachistochrone state variable trajectory*

Integration Tolerance	Optimization Tolerance	IVP Solutions	Objective Function	CPU
10^{-7}	10^{-5}	10	1.8230	0.96s
10^{-9}	10^{-7}	34	1.812723	5.38s
10^{-11}	10^{-9}	35	1.81272311	23.55s

Table 8-2: *Statistics for the curve-constrained brachistochrone problem*

8.2 Curve-Constrained Brachistochrone

An interesting variation on the previous problem is to constrain the brachistochrone with a curve. This may be done by replacing (8.5) with

$$y \geq ax^2 + bx + c \tag{8.8}$$

This problem was solved with the parameters in (8.8) set to $a = -0.375$, $b = -0.245$, and $c = -0.4$. The initial guesses, parameter values, and control discretization were the same as those used for the line-constrained brachistochrone. The optimal trajectories for the states and controls are shown in Figure 8-3 and Figure 8-4. The solution statistics are given in Table 8-2. The statistics show that the CPU time grows at a faster rate than the number of IVP solutions as the tolerances are tightened. This effect occurs because the number of integration steps and Jacobian factorizations taken during the solution of each IVP increases as the tolerances are tightened.

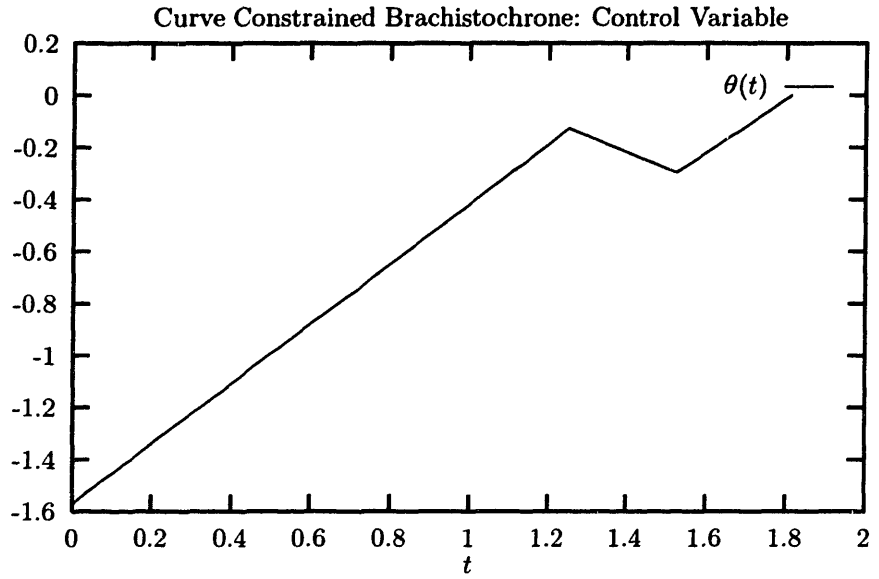

Figure 8-3: *Curve-constrained brachistochrone control variable trajectory*

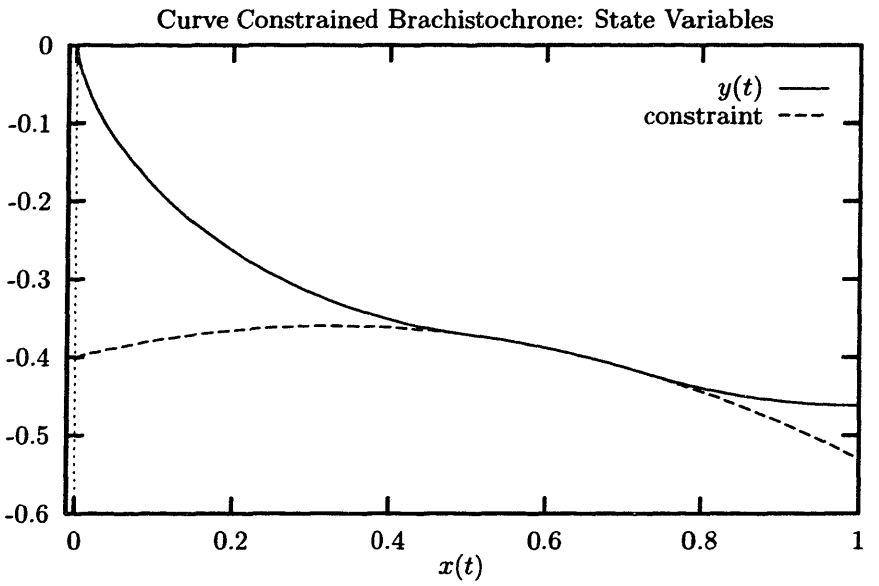

Figure 8-4: *Curve-constrained brachistochrone state variable trajectory*

8.3 Constrained Van Der Pol Oscillator

This problem was taken from [142]. The problem is:

$$\min_{u(t)} y_3(t_f) \tag{8.9}$$

subject to:

$$\dot{y_1} = (1 - y_2^2)y_1 - y_2 + u \tag{8.10}$$

$$\dot{y_2} = y_1 \tag{8.11}$$

$$\dot{y_3} = y_1^2 + y_2^2 + u^2 \tag{8.12}$$

$$-0.3 \leq u \leq 1.0 \tag{8.13}$$

$$y_1 \geq -0.4 \tag{8.14}$$

$$[y_1(t_0), y_2(t_0), y_3(t_0)] = [0, 1, 0] \tag{8.15}$$

The problem was solved using ten linear finite elements to discretize the control, and the inequality constraint was active during the third finite element. The initial guess for the control variable was $u(t) = 0.70$. Solution statistics are given in Table 8-3 and the optimal trajectories for the states and controls are shown in Figure 8-5 and Figure 8-6. The results shown are those corresponding to the tighter tolerance levels.

It may be noted that the CPU times are much longer for this problem than for the brachistochrone examples, even though all of the models contain similar numbers of equations. The reason for the increased solution time for the Van der Pol problem is that it contains ten finite elements, and the integrator must restart at the beginning of each finite element. The BDF method can be inefficient while it is starting because it takes small steps, and therefore a large number of integration restarts can exact a significant toll on the overall computational efficiency. A modification to the BDF method was proposed in [2] which allows the integrator to take larger steps as it is starting.

Several slightly different answers to this problem were presented in [142]. The ob-

Integration Tolerance	Optimization Tolerance	IVP Solutions	Objective Function	CPU
10^{-7}	10^{-5}	25	2.9548	11.37s
10^{-9}	10^{-7}	65	2.954756	44.88s
10^{-11}	10^{-9}	78	2.9547566	80.39s

Table 8-3: *Statistics for the constrained Van der Pol oscillator problem*

jective function values reported in that work vary from 2.95678 to 2.95421, depending on the scheme used to measure the violation of the state path constraint. The FIIP method converged significantly faster, although the best objective function value the algorithm found was slightly larger than the lowest reported in [142]. One explanation of this very small difference is that the results presented in [142] may have actually allowed small violations of the state path constraint, whereas our method guarantees that the solution does not violate this constraint, and hence solves a more constrained problem. Alternatively, this problem, like many control parameterization problems, is multimodal, and the algorithm may not have converged to the same local optimum that was found in [142]. However, the FIIP solution is guaranteed not to violate the inequality path constraint.

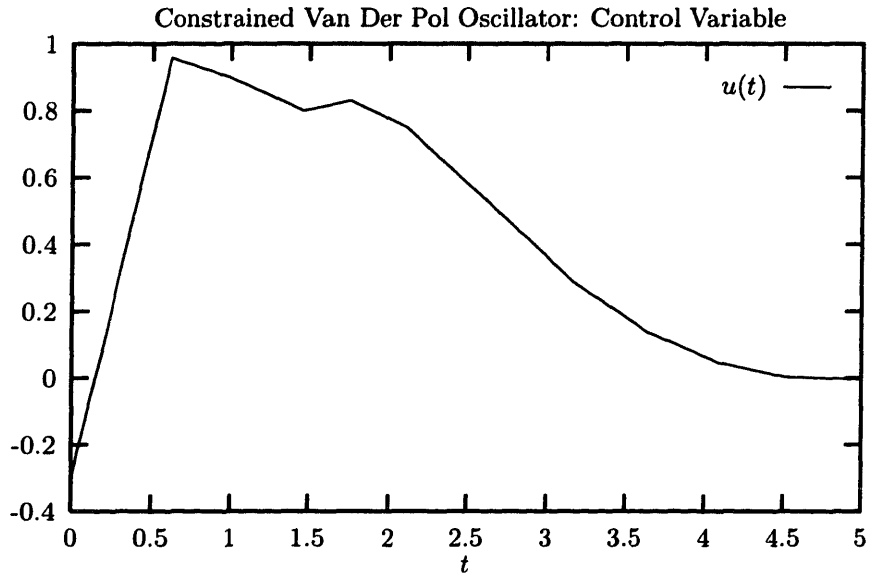

Figure 8-5: *Constrained Van Der Pol control variable trajectory*

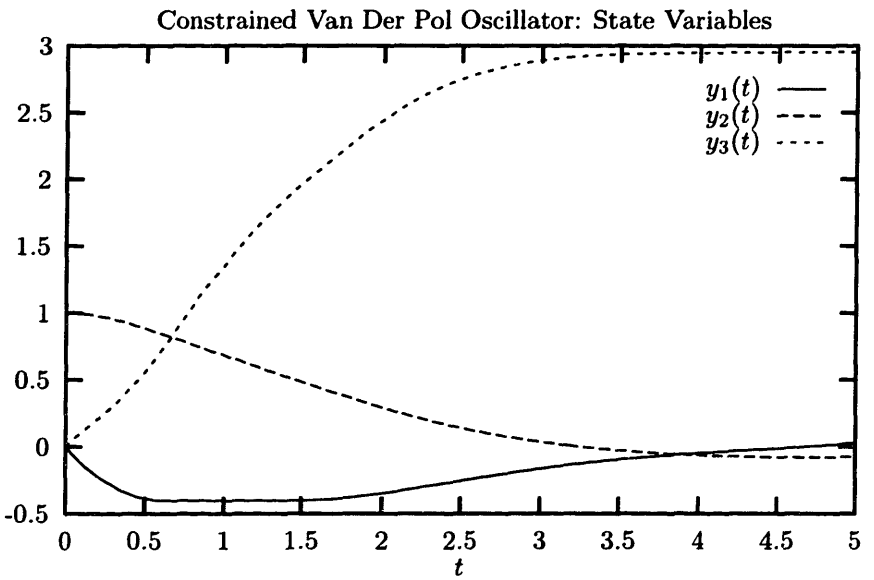

Figure 8-6: *Constrained Van Der Pol state variable trajectories*

Integration Tolerance	Optimization Tolerance	IVP Solutions	Objective Function	CPU
10^{-5}	10^{-3}	17	35.000	0.36s

Table 8-4: *Statistics for the constrained car problem*

8.4 Constrained Car Problem

The constrained car problem is a classic optimal control problem (see for example, [90], [142]). The problem is to find the minimum time for a car to travel between two points, subject to constraints on the velocity and acceleration. The model used was:

$$\min_{a(t),t_f} t_f \tag{8.16}$$

subject to:

$$\dot{v} = a \tag{8.17}$$

$$\dot{x} = v \tag{8.18}$$

$$v \leq 10 \tag{8.19}$$

$$-2 \leq a \leq 1 \tag{8.20}$$

$$[x(t_0), v(t_0)] = [0, 0] \tag{8.21}$$

$$[x(t_f), v(t_f)] = [300, 0] \tag{8.22}$$

where x is the position of the car, v is the velocity, and a is the acceleration. The initial and final point constraints require the optimal solution to be one where the car starts and finishes with zero velocity. There is a speed limit that constrains the velocity to be less than 10.

The problem was solved using three constant finite elements, and the inequality constraint was active during the second element. The optimal trajectories for the acceleration and velocity are shown in Figure 8-7 and Figure 8-8. They match the numerical solution given in [142] and the analytical solution given in [90]. Solution statistics are given in Table 8-4. Statistics for only one tolerance level are reported

because no improvement is possible on the solution found, and tighter tolerances do not increase the number of IVPs or improve the objective function.

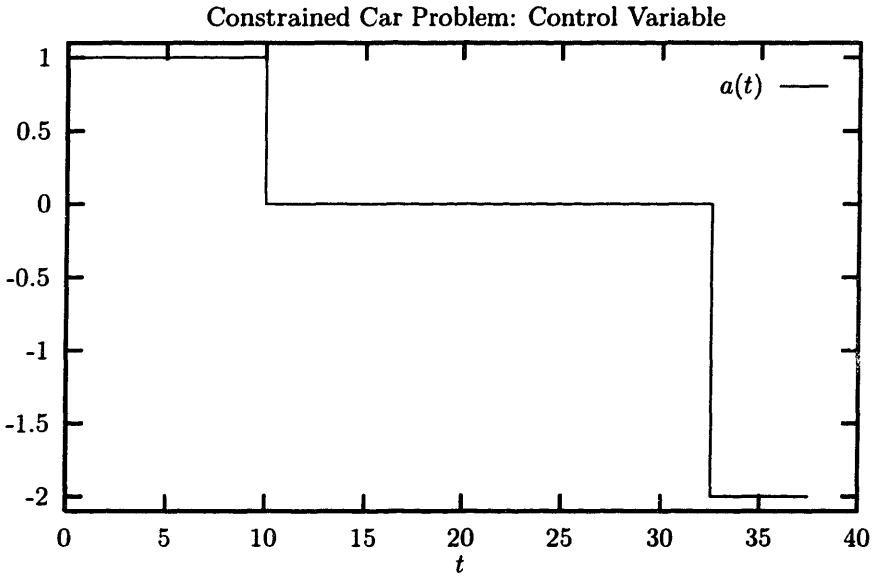

Figure 8-7: *Constrained car problem acceleration profile*

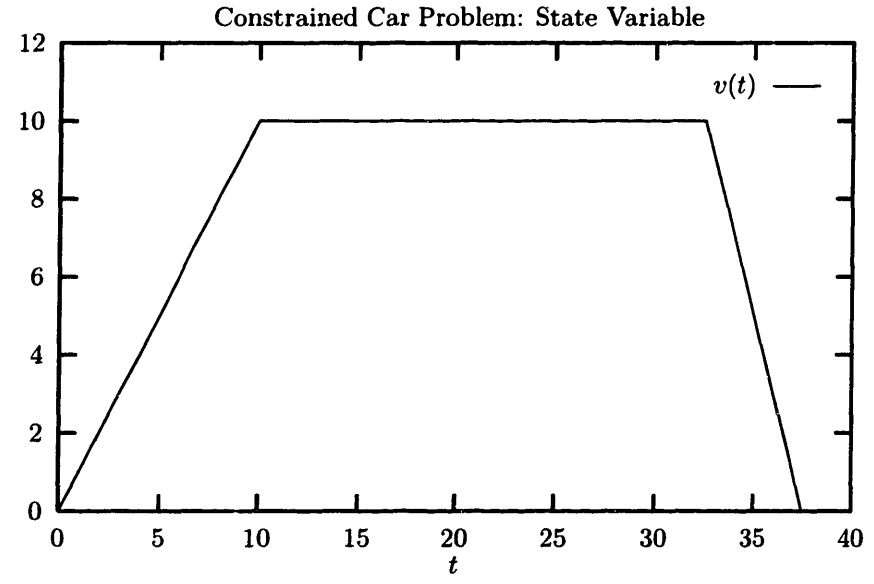

Figure 8-8: *Constrained car problem velocity profile*

8.5 Index-2 Jacobson and Lele Problem

This problem was originally presented in [74], and was later solved using different methods in [66, 90, 98, 106, 142]. The problem is:

$$\min_{u(t)} y_3(t_f) \tag{8.23}$$

subject to:

$$\dot{y_1} = y_2 \tag{8.24}$$

$$\dot{y_2} = -y_2 + u \tag{8.25}$$

$$\dot{y_3} = y_1^2 + y_2^2 + 0.005u^2 \tag{8.26}$$

$$y_2 \leq 8(t - 0.5)^2 - 0.5 \tag{8.27}$$

$$-3.0 \leq u \leq 15.0 \tag{8.28}$$

$$[y_1(t), y_2(t), y_3(t)] = [0, -1, 0] \tag{8.29}$$

There are two versions of this problem. The original form of the problem presented in [74] and solved in [34, 106, 98] is shown above. A modified version was used in [66] and [142], where (8.25) was changed to:

$$\dot{y_2} = y_2 + u \tag{8.30}$$

The latter problem is unstable. Hence, the application of numerical integration algorithms to this problem should be treated with suspicion.

The original form of the problem was solved using twelve finite elements, with the inequality constraint active on the seventh element. The initial guess for the control profile was $u(t) = 6.0$. The optimal trajectories for the control and constrained state variable are shown in Figure 8-9 and Figure 8-10. Solution statistics are given in Table 8-5. The value that was obtained for the objective function is very close (within 0.1%) to the value reported in [90]).

A *sequenced initial guess method* was used in this problem. That is, the solution

Integration Tolerance	Optimization Tolerance	IVP Solutions	Objective Function	CPU
10^{-5}	10^{-3}	11	0.217	2.9s
10^{-7}	10^{-4}	13	0.17982	6.0s
10^{-9}	10^{-7}	61	0.1698402	39.8s
10^{-11}	10^{-9}	54	0.169832150	49.9s

Table 8-5: *Solution statistics for original index-2 Jacobson and Lele problem*

Integration Tolerance	Optimization Tolerance	IVP Solutions	Objective Function	CPU
10^{-7}	10^{-5}	26	0.18421	7.6s
10^{-9}	10^{-7}	73	0.1800561	27.5s
10^{-11}	10^{-9}	60	0.180021318	31.5s

Table 8-6: *Solution statistics for the modified index-2 Jacobson and Lele problem*

was obtained for the tighter tolerances by using as the initial guess the answer from the problem with the next loosest tolerances. This method often decreases the time to solve control parameterization problems because tight integration tolerances do not need to be enforced when the NLP is far from the optimal solution.

The modified version of the problem was also solved using a control parameterization of seven finite elements with the constraint enforced during the fourth finite element. The solution statistics for the modified problem are given in Table 8-6, and the control and constrained state variable trajectory are shown in Figure 8-11 and Figure 8-12. The optimal value for the objective function are slightly better than the one reported in [142].

The optimal trajectories for the control and constrained state variable for the original form are shown in Figure 8-9 and Figure 8-10. The same variables are shown for the modified form in Figure 8-11 and Figure 8-12.

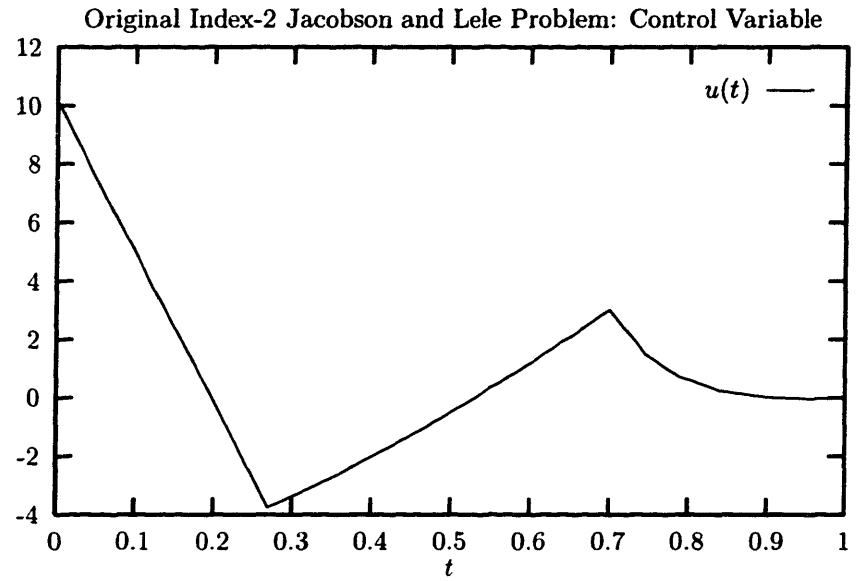

Figure 8-9: *Original Index-2 Jacobson and Lele control trajectory*

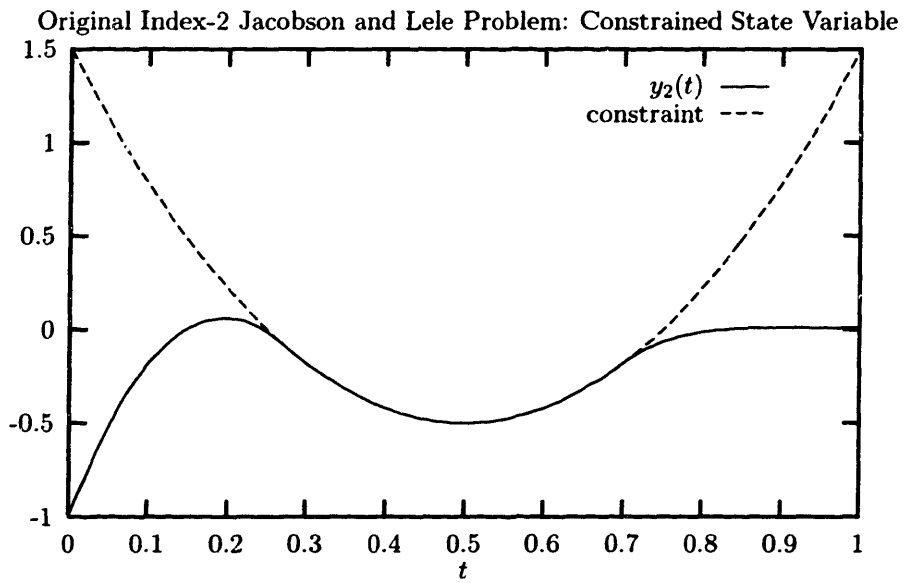

Figure 8-10: *Original Index-2 Jacobson and Lele y_2 trajectory*

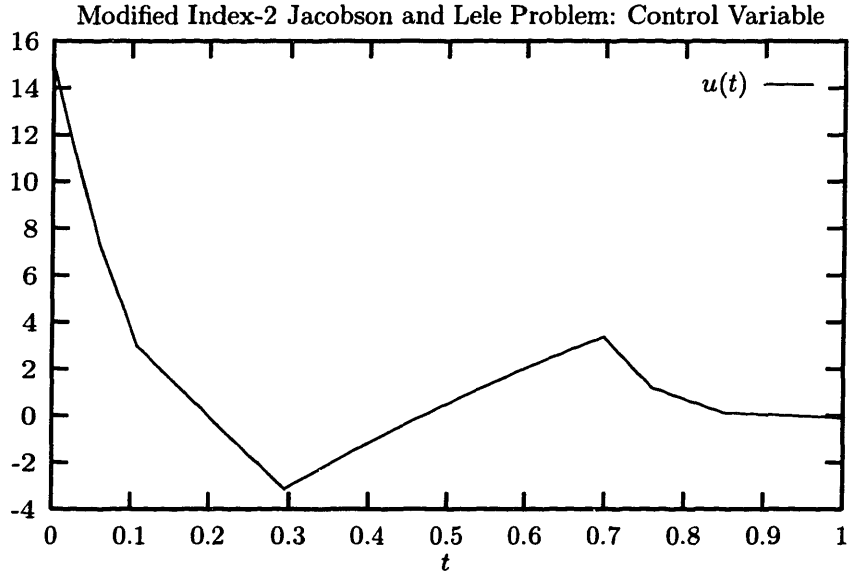

Figure 8-11: *Modified Index-2 Jacobson and Lele control trajectory*

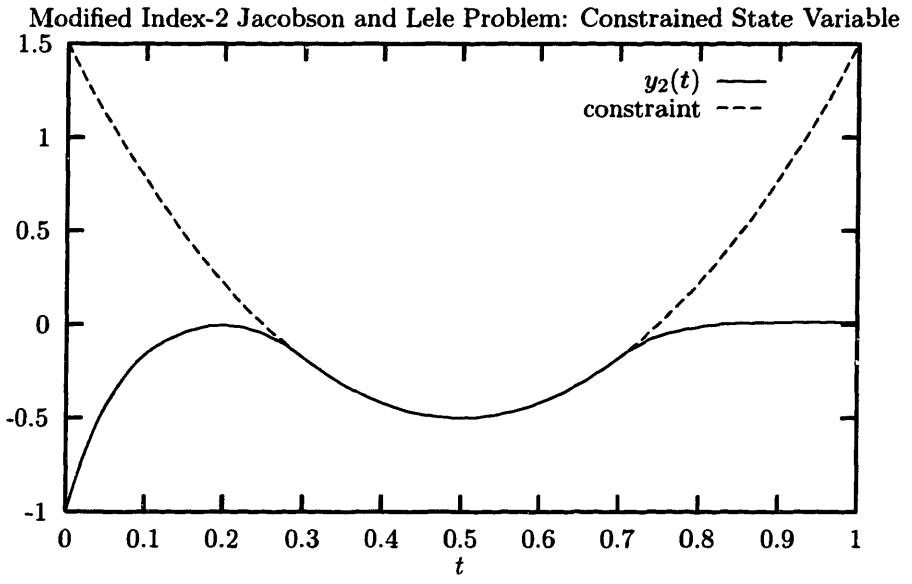

Figure 8-12: *Modified Index-2 Jacobson and Lele y_2 trajectory*

Integration Tolerance	Optimization Tolerance	IVP Solutions	Objective Function	CPU
10^{-7}	10^{-5}	71	0.75156	35.4s
10^{-9}	10^{-7}	37	0.7514907	25.2s
10^{-11}	10^{-9}	251	0.75144685	210.9s

Table 8-7: *Solution statistics for the index-3 Jacobson and Lele Problem*

8.6 Index-3 Jacobson and Lele Problem

Another interesting variation on the Jacobson and Lele problem is to replace (8.27) with

$$y_1 \leq 8(t - 0.5)^2 - 0.5 \tag{8.31}$$

This change has the effect of making the problem index-3 when the constraint is active, rather than index-2 when the problem is constrained as above with (8.27).

This problem was solved using twelve linear finite elements, with the constraint enforced on the third element. A discontinuity in the control was permitted after the second element. The finite element size of all the finite elements except the constrained element was bounded from below by 0.05. The statistics shown in Table 8-7 use the sequenced initial guess method. The optimal trajectories for the control and the constrained state variable in Figure 8-13 and Figure 8-14 show that the constraint is active only at one point. The objective function values reported here are very close to those reported in [90], even though a much simpler control discretization than the one used in that work was used.

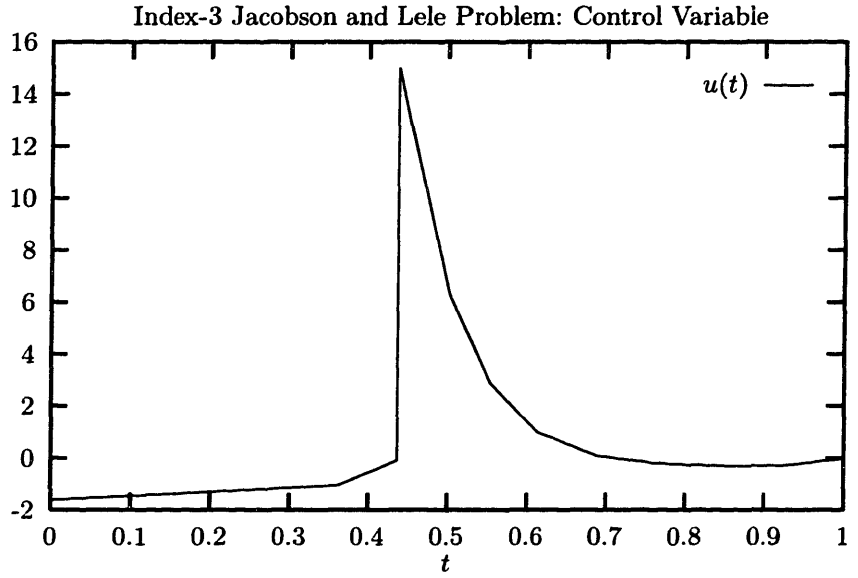

Figure 8-13: *Index-3 Jacobson and Lele control trajectory*

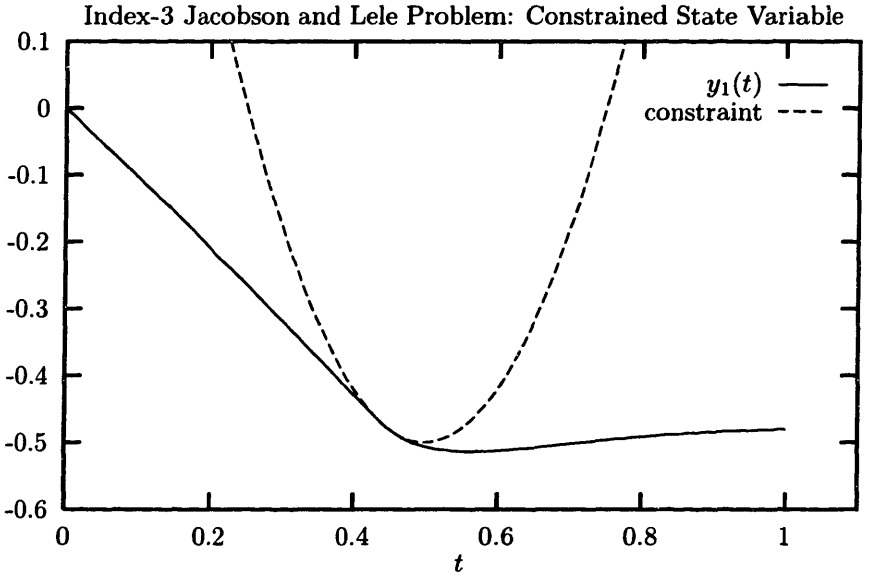

Figure 8-14: *Index-3 Jacobson and Lele y_1 trajectory*

8.7 Pressure-constrained Batch Reactor

The following gaseous reactions take place isothermally and simultaneously in a pressure-constrained batch reactor:

$$A \underset{k_2}{\overset{k_1}{\rightleftharpoons}} 2B \tag{8.32}$$

$$A + B \xrightarrow{k_3} D \tag{8.33}$$

This simple problem is useful for showing the ability of the high-index DAE to find a highly nonlinear control that tracks a constraint.

The dynamic optimization problem is:

$$\max_F C_D(t_f) \tag{8.34}$$

subject to:

$$\dot{C}_A = -k_1 C_A + k_2 C_B C_B + F/V - k_3 C_A C_B \tag{8.35}$$

$$\dot{C}_B = k_1 C_A - k_2 C_B C_B - k_3 C_A C_B \tag{8.36}$$

$$\dot{C}_D = k_3 C_A C_B \tag{8.37}$$

$$N = V(C_A + C_B + C_D) \tag{8.38}$$

$$PV = NRT \tag{8.39}$$

$$P \leq 340000 \tag{8.40}$$

$$0 \leq F \leq 8.5 \tag{8.41}$$

$$[C_A(0), C_B(0), C_D(0)] = [100, 0, 0] \tag{8.42}$$

where C_i is the concentration of species i, P is the pressure in the reactor, N is the total number of moles in the reactor, V is the reactor volume, R is the gas constant, F is the feed rate of pure A, and k_1 and k_2 are the rate constants for the two reactions. The rate constants were set to $k_1 = 0.8\,\mathrm{hr}^{-1}$, $k_2 = 0.02\,\mathrm{m}^3/(\mathrm{mol}\cdot\mathrm{hr})$, $k_3 = 0.003\,\mathrm{m}^3/(\mathrm{mol}\cdot\mathrm{hr})$, the volume was set to $V = 1.0\,\mathrm{m}^3$, the temperature was

Integration Tolerance	Optimization Tolerance	IVP Solutions	Objective Function	CPU
10^{-7}	10^{-5}	10	11.7284	0.99s

Table 8-8: *Solution statistics for the pressure-constrained batch reactor problem*

$T = 400\,\mathrm{K}$, the gas constant was $R = 8.314\,\mathrm{J/(mol \cdot K)}$. The initial guess for the feed rate of species A was $F(t) = 0.5\,\mathrm{mol/hr}$.

This problem was solved using two constant finite elements to approximate the control, and the constraint was active during the second element. The optimal trajectories for the control, the pressure, and the concentrations are shown in Figures 8-15–8-17. Solution statistics are given in Table 8-8. During the constrained portion of the trajectory, the control is highly nonlinear. If this problem were solved using a penalty function approach, the control would require a complex discretization to approximate this nonlinear optimal control trajectory. However, the FIIP method is able to exploit the fact that the control is not independent when the constraint is active, and a simple control discretization may be used.

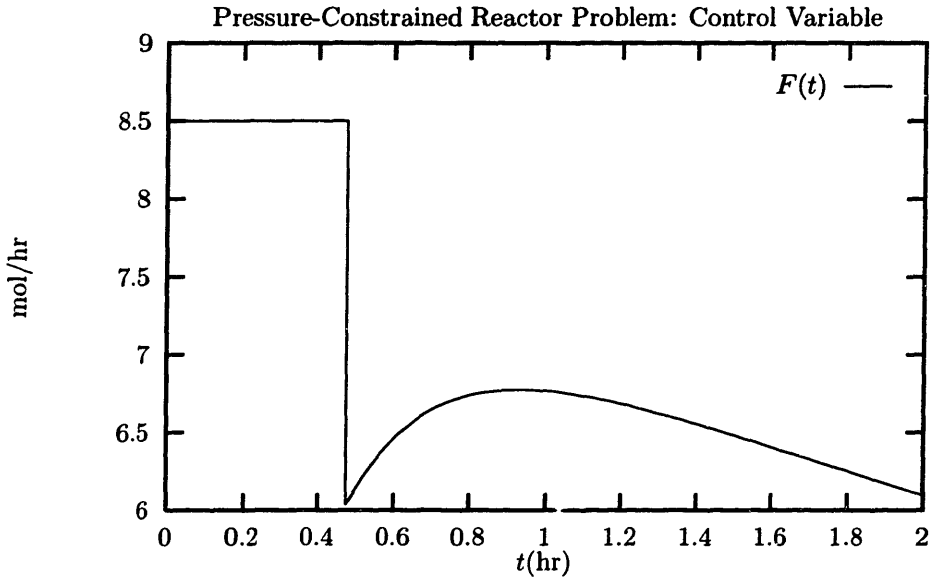

Figure 8-15: *Pressure-constrained reactor control variable trajectory*

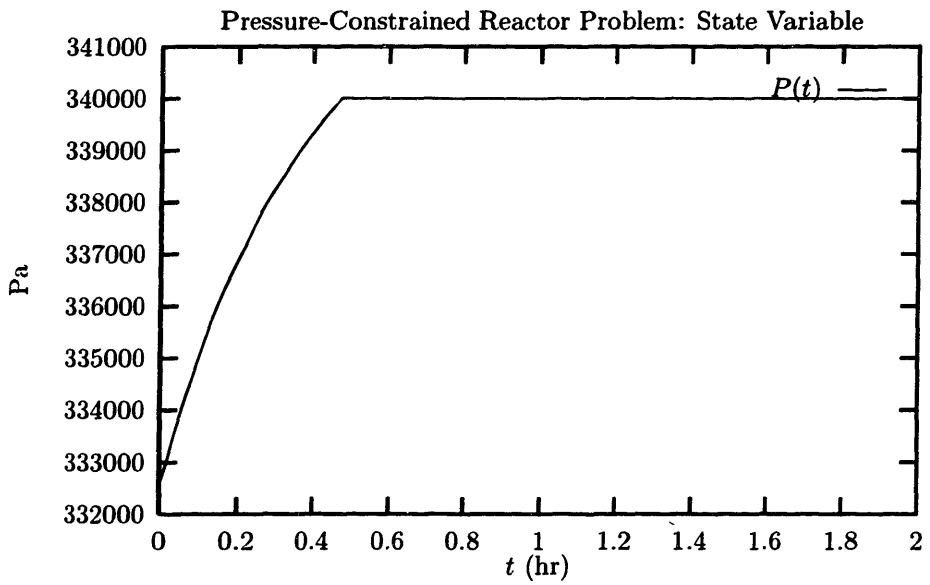

Figure 8-16: *Pressure-constrained reactor pressure trajectory*

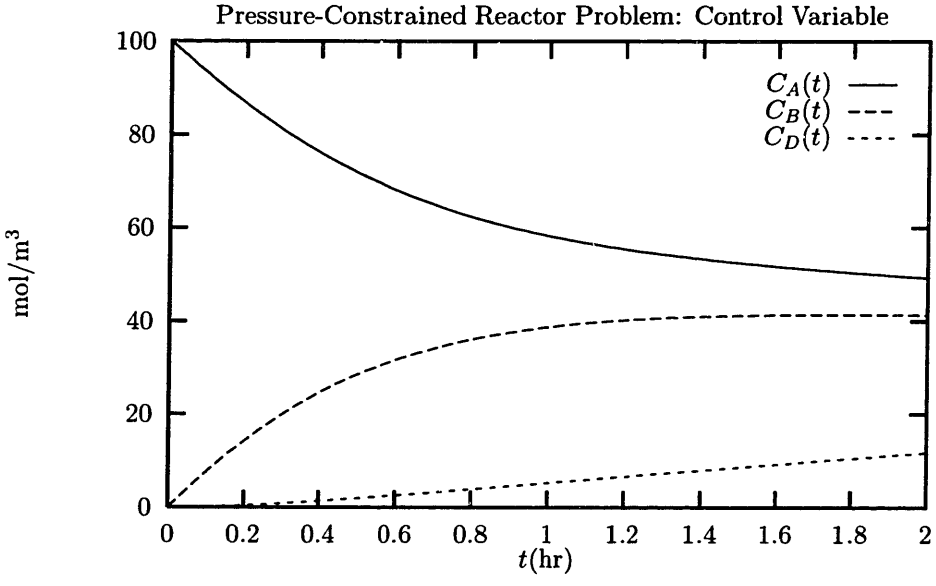

Figure 8-17: *Pressure-constrained reactor concentration trajectories*

8.8 Fed-Batch Penicillin Fermentation

The determination of optimal control profiles for fed-batch bioreactors is an interesting chemical engineering problem that has received significant attention in recent years. These problems are considered difficult because of the nonlinear system dynamics and the existence of constraints on both the state and control variables.

A model of a fed-batch penicillin fermentation reactor was described and solved in [127]. The problem is:

$$\min_{S(t)} J = \int_0^{150} -\theta XV + 0.0103PV + 0.0744\mu X + 0.00102XV + 0.00691358 \quad (8.43)$$

subject to:

$$\dot{X} = \mu X - \frac{FX}{V} \quad (8.44)$$

$$\dot{P} = \theta X - 0.01P - \frac{FP}{V} \quad (8.45)$$

$$\mu = \frac{0.11S}{S + 0.1X} \quad (8.46)$$

$$\theta = \frac{0.004}{1 + 0.0001/S + S/0.1} \quad (8.47)$$

$$\dot{V} = F \quad (8.48)$$

$$0.001 \leq S \leq 0.5 \quad (8.49)$$

$$X \leq 41 \quad (8.50)$$

$$[X(0), P(0), V(0)] = [1, 0, 2.5 \cdot 10^5] \quad (8.51)$$

The objective function in this problem takes into account the revenue from the product, the cost due to product deactivation, the cost of the substrate, and the daily cost. The state variables in the problem are the biomass concentration X (g/L), the amount of penicillin product present P (activity/L), the substrate concentration S (g/L), the reactor volume V (L), the specific product formation rate θ, and the specific growth rate μ. The feed rate parameter F was set to 1666.67 L/h.

This problem is somewhat sensitive to the initial guess for the control trajectory.

Integration Tolerance	Optimization Tolerance	IVP Solutions	Objective Function	CPU
10^{-7}	10^{-5}	21	$-1.1230 \cdot 10^{6}$	8.94s

Table 8-9: *Solution statistics for the CSTR and column startup problem*

The initial guess used in this work was a step function starting out at $S(t) = 0.4$ and falling after the first finite element to $S(t) = 0.01$ at $t = 40.0$.

The solution statistics for this problem are shown in Table 8-9. The optimal trajectories for the control and selected state variables are shown in Figure 8-18 and Figure 8-19. The trajectories and the objective function value agree closely with those reported in [127].

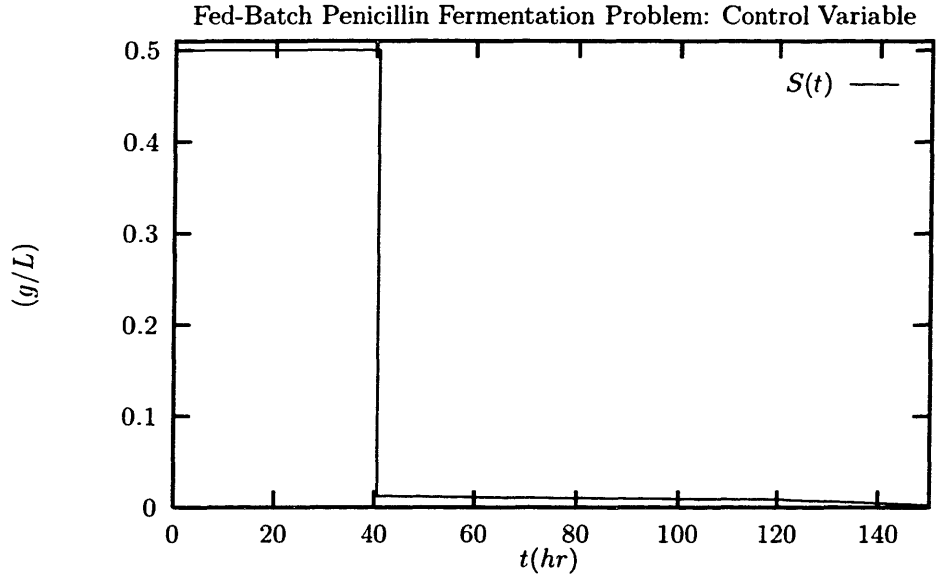

Figure 8-18: *Fed-batch penicillin fermentation control variable trajectory*

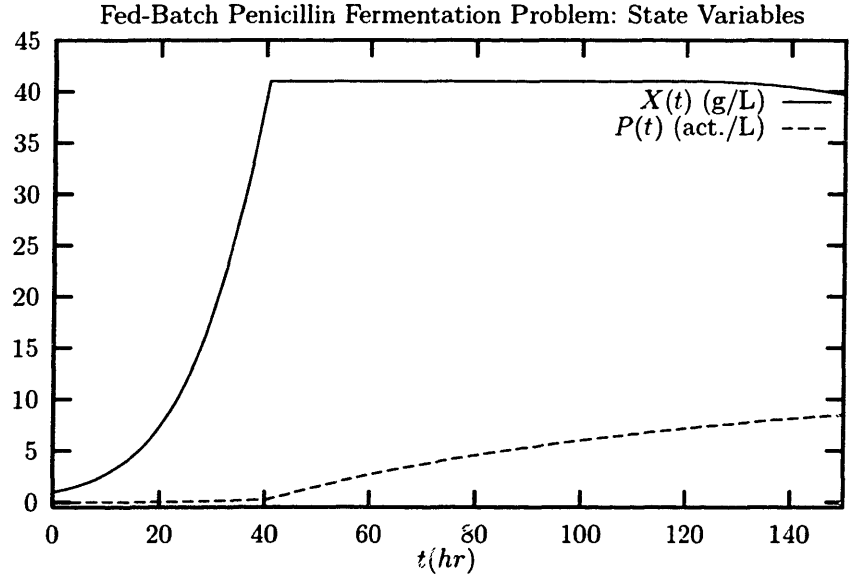

Figure 8-19: *Fed-batch penicillin fermentation state variable trajectories*

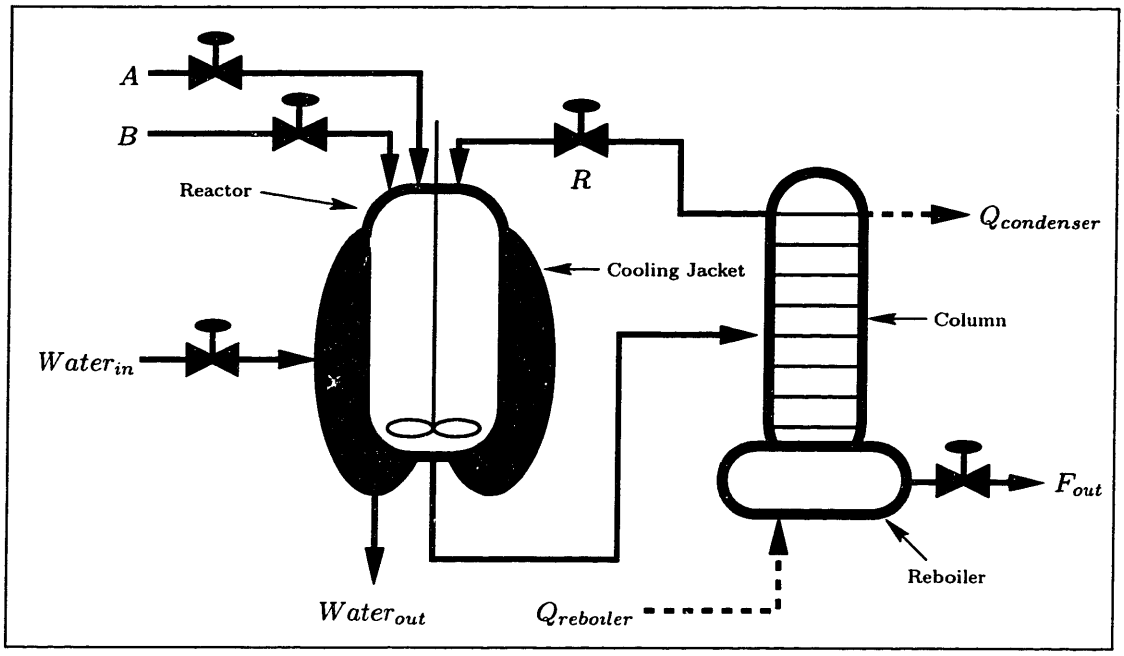

Figure 8-20: *CSTR and column flowsheet*

8.9 Startup of CSTR and Distillation Column

This example demonstrates how constraining the dynamic optimization problem can reduce the number of controls in the dynamic optimization problem, thus making it easier to solve. It also demonstrates the ability of FIIP to handle problems that are modeled by large-scale DAEs.

Consider the flowsheet containing a continuous stirred tank reactor (CSTR) and distillation column shown in Figure 8-20. There are two simultaneous reactions which take place in the liquid phase in the CSTR:

$$A + B \xrightarrow{k_1} C \tag{8.52}$$

$$B + C \xrightarrow{k_2} D \tag{8.53}$$

The desired product is C, but there is a side reaction which is favored at higher temperatures which consumes the desired product and produces the undesired product D.

The reactor contents must be kept below 400K at all times to avoid vaporization

of the reacting mixture. The reactor temperature may be controlled by adjusting the flowrate of water to a cooling jacket which surrounds the CSTR.

The reaction mixture leaves the reactor at a prescribed flowrate and flows into a stripper column. The column is intended to strip off the lighter reactants A and B, and the top product from the column is recycled back to the reactor.

The dynamic model for this example is detailed below. The model and its parameters are loosely based on one given in [33].

8.9.1 CSTR model

The reactor material balance on each species i is:

$$\dot{N}_i = x_i^{in} F^{in} + V \sum_{j=1}^{NR} \nu_{ji} r_j - x_i^r F^{out} \qquad \forall i = 1 \ldots NC \tag{8.54}$$

where N_i is the molar holdup of species i, V is the of the reactor contents, r_j is the rate of reaction of reaction j, ν_{ji} is the stoichiometric coefficient of species i in reaction j, x_i^{in} is the mole fraction of species i in the feed, x_i^r is the mole fraction of species i in the reactor, F^{in} and F^{out} are respectively the mole flows into and out of the reactor, NR is the number of independent reactions, and NC is the number of components.

The reaction rate is modeled by:

$$r_j = k_j \prod_{i=1}^{NC} C_i^{\beta_{ij}} \qquad \forall j = 1 \ldots NR \tag{8.55}$$

where C_i is the molar concentration of species i, β_{ij} is the order of reaction j with respect to species i, and k_j is the rate constant of reaction j given by the Arrhenius equation:

$$k_j = A_j \exp\left(-\frac{E_j}{RT}\right) \qquad \forall j = 1 \ldots NR \tag{8.56}$$

The total enthalpy H^r of the mixture in the reactor is:

$$H^r = \sum_{i=1}^{NC} N_i \hat{H}_i^r \tag{8.57}$$

where $\hat{H}_i^r$ is the specific enthalpy of species i in the reactor, and an energy balance on the entire reactor gives:

$$\begin{aligned} \dot{H}^r = F^{in} \sum_{i=1}^{NC} x_i^{in} \hat{H}_i^{in} \\ + F^{out} \sum_{i=1}^{NC} x_i^{out} \hat{H}_i^r - \dot{Q}_{jacket} \end{aligned} \tag{8.58}$$

where $\hat{H}^{in}$ is the specific enthalpy of species i in the reactor feed and $\dot{Q}_{jacket}$ is the heat removed by the reactor cooling jacket.

The volume of the reactor contents is:

$$V = \sum_{i=1}^{NC} \frac{N_i}{\rho_i} \tag{8.59}$$

where ρ_i is the molar density of species i.

The reactor concentration and mole fraction are respectively given by:

$$C_i = \frac{N_i}{V} \qquad \forall i = 1 \ldots NC \tag{8.60}$$

$$x_i^r = \frac{N_i}{N_{tot}} \qquad \forall i = 1 \ldots NC \tag{8.61}$$

The total reactor molar holdup is:

$$N_{tot} = \sum_{i=1}^{NC} N_i \tag{8.62}$$

The parameters used in this model are given in Table 8-10. Both reactions are first order (therefore $\beta_{ij} = 1 \; \forall (i,j)$), and the gas constant is 8.314 (kJ/kmol $\cdot$ K).

Parameter	Value
NR	2
NC	4
ν_{A1}	-1
ν_{A2}	0
ν_{B1}	-1
ν_{B2}	-1
ν_{C1}	1
ν_{C2}	-1
ν_{D1}	0
ν_{D2}	-1

Parameter	Value
A_1	$100.0\,\mathrm{m^3/(hr \cdot kmol)}$
A_2	$120.0\,\mathrm{m^3/(hr \cdot kmol)}$
E_1	$17000\,\mathrm{kJ/kmol}$
E_2	$20000\,\mathrm{kJ/kmol}$
ρ_A	$11.0\,\mathrm{kmol/m^3}$
ρ_B	$16.0\,\mathrm{kmol/m^3}$
ρ_C	$10.0\,\mathrm{kmol/m^3}$
ρ_D	$10.4\,\mathrm{kmol/m^3}$

Table 8-10: *Parameters used in the CSTR model*

8.9.2 Reactor cooling jacket

The heat removed by the reactor cooling jacket Q_{jacket} is:

$$\dot{Q}_{jacket} = UA\,(T_r - T^w_{out}) \tag{8.63}$$

where T_r is the reactor temperature, T^w_{out} is the outlet temperature of the cooling water, U is the heat transfer coefficient of the jacket, and A is the heat transfer surface area.

The heat is transferred to the cooling water, which gives the temperature relationship:

$$\dot{Q}_{jacket} = C^w_p\,(T^w_{out} - T^w_{in})\,F^w \tag{8.64}$$

where T^w_{out} is the inlet temperature of the cooling water and C^w_p is the heat capacity of the cooling water.

The values of the parameters used for the cooling jacket were $U = 3000\,(\mathrm{kW/m^2 \cdot K})$, $A = 30\,\mathrm{m^2}$, and $T^w_{in} = 288\,\mathrm{K}$. The heat capacity of water was assumed to be independent of temperature at $4200\,(\mathrm{kJ/tonne \cdot K})$.

8.9.3 Column tray model

The column has NS stages, but the first stage is the condenser, and the last is the reboiler, so there are $NS-2$ equilibrium trays. The overall material balance on each tray is:

$$\dot{M}_k = L_{k-1} - L_k + V_{k+1} - V_k + F_k \qquad \forall k = 2 \ldots NS-1 \tag{8.65}$$

and the material balance for each species is:

$$\begin{aligned} \dot{M}_k x_k^i + M_k \dot{x}_k^i =& L_{k-1} x_{k-1}^i - L_k x_k^i + \\ & V_{k+1} y_{k+1}^i - V_k y_k^i + F_k z_k^i \end{aligned} \qquad \forall k = 2 \ldots NS-1 \tag{8.66}$$

where M_k is the molar holdup on stage k, x_k^i and y_k^i are respectively the liquid and vapor mole fractions of species i on stage k, F_k is the feed flow rate to stage k, and z_k^i is the composition of the feed on stage k (assumed to be liquid).

Assuming fast energy dynamics and adiabatic operation, the energy balance on each tray is:

$$\begin{aligned} 0 =& L_{k-1} \hat{H}_{k-1}^l - L_k \hat{H}_k^l + \\ & V_{k+1} \hat{H}_{k+1}^v - V_k \hat{H}_k^v + F_k \hat{H}_k^f \end{aligned} \qquad \forall k = 2 \ldots NS-1 \tag{8.67}$$

where $\hat{H}_k^l$, $\hat{H}_k^v$, and $\hat{H}_k^f$ are respectively the molar specific enthalpies of the liquid, vapor, and feed on stage k.

The vapor mole fractions on each tray are normalized:

$$\sum_{i=1}^{NC} y_k^i = 1 \qquad k = 2 \ldots NS \tag{8.68}$$

and, since thermodynamic equilibrium is assumed on each tray, the liquid phase composition must satisfy:

$$y_k^i = K_k^i x_k^i \qquad \forall k = 2 \ldots NS \tag{8.69}$$

where K_k^i is the vapor/liquid equilibrium distribution coefficient for species i on tray k.

The liquid overflow on each tray is related to the tray holdup by the Francis weir equation:

$$L_k \sum_{i=1}^{NC} x_k^i Z^i = \bar{\rho}_{l_k} W_L \left(\frac{h_k}{750} \right)^{1.5} \tag{8.70}$$

where Z^i is the molecular weight of species i, W_L is the weir length in meters, h_k is the height of the liquid above the weir in millimeters, and $\bar{\rho}_{l_k}$ is the mass density of the liquid. The molecular weights of species A, B, C, and D are 76, 52, 128, and 180 respectively. The mass densities of the liquid and vapor is given by:

$$\bar{\rho}_{l_k} = \left(\sum_{i=1}^{NC} \frac{x_k^i}{Z^i \rho_i} \right)^{-1} \tag{8.71}$$

$$\bar{\rho}_{v_k} = \frac{P_k}{RT_k} \sum_{i=1}^{NC} (y_k^i Z^i) \tag{8.72}$$

The volume of the liquid on the tray is:

$$V_k = \frac{M_k}{\bar{\rho}_{l_k}} \sum_{i=1}^{NC} x_i^k Z_{l_k} \tag{8.73}$$

and therefore h_k is:

$$h_k = \frac{V_k}{A_{tray}} - W_H \tag{8.74}$$

where A_{tray} is the area of the tray, and W_H is the height of the weir.

The pressure drop on each tray is related to the vapor flow by:

$$P_k = P_{k-1} + \Delta P_{k-1}^{tray} \tag{8.75}$$

$$\Delta P_k^{tray} = \Delta P_k^{static} + \Delta P_k^{dry} \tag{8.76}$$

$$\Delta P_k^{static} = \frac{V_k}{A_{tray}} \bar{\rho}_{l_k} g \tag{8.77}$$

$$\Delta P_k^{dry} = \bar{\rho}_{v_k} \left(\frac{V_{k+1} \sum_{i=1}^{NC} y_k^{i+1} Z^i}{\bar{\rho}_{v_k}} A_{free} \omega \right)^2 \tag{8.78}$$

where g is the gravitational constant, and ω is a constant dependent on the tray geometry.

The column model has $NS = 10$ and the feed is on stage 5 (therefore, $F_k = 0; \forall i \neq 5$).

8.9.4 Column reboiler model

The overall mass balance on the reboiler is:

$$\dot{M}_{NS} = L_{NS-1} - B - V_{NS} \tag{8.79}$$

where B is the bottoms flowrate. The material balance for each species is:

$$\dot{M}_{NS} x_{NS}^i + M_{NS} \dot{x}_{NS}^i = L_{NS-1} x_{NS-1}^i - B x_{NS}^i - V_{NS} y_{NS}^i \tag{8.80}$$

Assuming fast energy dynamics, the energy balance is:

$$0 = L_{NS-1} \hat{H}_{NS-1}^l - B \hat{H}_{NS}^l - V_{NS} \hat{H}_{NS}^v + \dot{Q}_R \tag{8.81}$$

where $\dot{Q}_R$ is the reboiler heat duty.

The vapor mole fractions are normalized:

$$\sum_{i=1}^{NC} y_{NS}^i = 1 \tag{8.82}$$

and the liquid phase composition must satisfy:

$$y^i_{NS} = K^i_{NS} x^i_{NS} \tag{8.83}$$

The flow out of the reboiler is related to the reboiler holdup by the proportional control law:

$$B = 0.1(M_{NS} - M^*_{NS}) \tag{8.84}$$

where M^*_{NS} is the set-point level.

8.9.5 Column condenser model

The column uses a total condenser with a constant holdup, therefore:

$$V_2 = D + L_1 \tag{8.85}$$

$$x^i_1 = y^i_2 \tag{8.86}$$

$$V_1 = 0 \tag{8.87}$$

where D is the molar distillate flow.

The amount refluxed back to the top tray obeys the relation:

$$V_2 r = L_1 \tag{8.88}$$

where r is the *normalized* reflux ratio.

The heat required to condense the vapor into the condenser $\dot{Q}_C$ is given by:

$$\dot{Q}_C = V_2 \hat{H}^v_2 - (L_1 + D)\hat{H}^l_1 \tag{8.89}$$

The liquid mole fractions are normalized:

$$\sum_{i=1}^{NC} y_1^i = 1 \tag{8.90}$$

8.9.6 Enthalpy model

The specific enthalpy of a pure species in the vapor phase is assumed to be a function of temperature:

$$\hat{H}_i^v = \Delta H_i^f(T_{ref}) + \int_{T_{ref}}^{T} a_i dT \tag{8.91}$$

where T_{ref} is a reference temperature.

The specific enthalpy of a pure species in the liquid phase is assumed to obey the relationship:

$$\hat{H}_i^l = \hat{H}_i^v - \Delta H_i^{vap} \tag{8.92}$$

The mixing of species is ideal, therefore:

$$\hat{H}^l = \sum_{i=1}^{NS} x_i \hat{H}_i^l \tag{8.93}$$

$$\hat{H}^l = \sum_{i=1}^{NS} y_i \hat{H}_i^v \tag{8.94}$$

The values of the parameters for the enthalpy model are given in Table 8-11. The reference temperature was set to $T_{ref} = 298K$.

8.9.7 Vapor-liquid equilibrium

Raoult's law is assumed, and therefore the vapor/liquid equilibrium distribution coefficients are independent of composition, and are given by:

$$K_i = \frac{P_i^{vap}}{P} \tag{8.95}$$

Parameter	Value
a_A	172.3 kJ/(kmol · K)
a_B	200.0 kJ/(kmol · K)
a_C	160.0 kJ/(kmol · K)
a_D	155.0 kJ/(kmol · K)
ΔH_A^{vap}	31000 kJ/kmol
ΔH_B^{vap}	26000 kJ/kmol
ΔH_C^{vap}	28000 kJ/kmol
ΔH_D^{vap}	34000 kJ/kmol
ΔH_A^f	−30000 kJ/kmol
ΔH_B^f	−50000 kJ/kmol
ΔH_C^f	−20000 kJ/kmol
ΔH_D^f	−20000 kJ/kmol

Table 8-11: *Parameters for the enthalpy model*

where P is the pressure, and P_i^{vap} is the vapor pressure of species i.

The vapor pressure is assumed to obey the following function of temperature:

$$\ln P_i^{vap} = -\frac{\alpha_i}{T} + b_i \tag{8.96}$$

The parameters for the vapor pressure model are given in Table 8-12. The vapor pressure is given in bar with these parameters. The pressure in the condenser was 1 atm.

Parameter	Value
α_A	4142.75 K
α_B	3474.56 K
α_C	3500.00 K
α_D	4543.71 K
b_A	11.7158
b_B	9.9404
b_C	8.9000
b_D	11.2599

Table 8-12: *Parameters for the vapor pressure model (vapor pressure in bar)*

8.9.8 Path constraints

The reactor and column model that has been specified thus far has five controls (corresponding to the valves shown in Figure 8-20). Dynamic optimization problems with many control variables are difficult to solve using any method. In control parameterization, the size of the NLP increases with the number of controls, and in practice, the degree of nonlinearity of the NLP increases. However, there are some additional constraints on the state variables that may be specified which, due to the results of Chapter 6, reduce the number of independent control variables. Note that the reboiler and condenser duties, listed as controls in Figure 8-20, are in fact completely determined by the model that has been specified.

The path constraints that have been explicitly imposed on this model are:

1. Constant Reactor Holdup.

2. Constant ratio between A and B entering the reactor: The flowrate of B was assumed to be 15% of the flowrate of A.

3. Constant flowrate of material out of the reactor. The flowrate was set to 15 kmols/hr.

The only independent control variables remaining in this problem are the column reflux ratio and the flow rate of the cooling water to the reactor jacket. The reactor temperature constraint causes the flow rate of the cooling water to be specified whenever the constraint is active. The flow rate of the cooling water was approximated using a piecewise continuous linear control discretization. The column reflux ratio was made a time-invariant optimization parameter. Hence, the complexity of the dynamic optimization problem has been reduced to a manageable level using path constraints which prescribe the values of some of the controls. The index of the DAE is two when the inequality path constraint is inactive. The index is also two when the inequality path constraint is active, but a larger set of equations must be differentiated in order to derive the equivalent index-1 DAE. Figures 8-21-8-22 show the ABACUSS

```
Before differentiation, this model was index 2.

Equations 1 through 689 form an index=1 model.

RESULTS FROM ABACUSS STRUCTURAL ANALYSIS

Number of Variables in model                    : 680
Number of additional variables created          : 0
                                                  ----
Total Variables                                 : 680

Number of Model Equations                       : 676
Number of Additional Model Equations Derived    : 8
Number of Input Functions Specified             : 5
Number of Additional Input Functions Derived    : 0
Number of Initial Conditions Specified          : 0
Number of Additional Initial Conditions Derived : 0
                                                  ----
Total Equations                                 : 689

Total Unknowns                                  : 689
```

Figure 8-21: *ABACUSS index-reduction output for reactor and column startup model when the constraint is inactive*

index reduction output for the model both without and with the inequality constraint active.

8.9.9 Solution of the problem

The objective function for the dynamic optimization problem was:

$$\min_{F^w,R} \bar{C}(t_f) - 0.1\bar{D}(t_f) \tag{8.97}$$

```
Before differentiation, this model was index 2.

Equations 1 through 705 form an index=1 model.

RESULTS FROM ABACUSS STRUCTURAL ANALYSIS

Number of Variables in model                    : 680
Number of additional variables created          : 0
                                                  ----
Total Variables                                 : 680

Number of Model Equations                       : 677
Number of Additional Model Equations Derived    : 24
Number of Input Functions Specified             : 4
Number of Additional Input Functions Derived    : 0
Number of Initial Conditions Specified          : 0
Number of Additional Initial Conditions Derived : 0
                                                  ----
Total Equations                                 : 705

Total Unknowns                                  : 705
```

Figure 8-22: *ABACUSS index-reduction output for reactor and column startup model when the constraint is active*

where:

$$\dot{\bar{C}} = Bx_{NS}^{C} \tag{8.98}$$

$$\dot{\bar{D}} = Bx_{NS}^{D} \tag{8.99}$$

$$\bar{C}(0) = 0 \tag{8.100}$$

$$\bar{D}(0) = 0 \tag{8.101}$$

$$t_f = 100hr \tag{8.102}$$

This objective function measures the relative cost associated with producing C and D during the startup (*i.e.*, producing an extra kmol of C offsets the disadvantage associated with producing 10 kmols of the undesired species D). The final time was chosen to be long enough so that the system will be close to steady state at the end of the simulation.

The condenser pressure was set to 1 atmosphere for the entire startup procedure. The feed makeup temperature is 300K. The setpoint for the control equation in the column reboiler was set to $M_{NS}^{*} = 22$ kmol At the initial time, the reactor contains 50 kmols of A at 300K, the total holdup in the column is 46.586382 kmols of A, and each of the trays is filled to the weir height with A at its bubble point. Note that since one of the path constraints fixes the ratio between A and B being fed to the reactor, the flow of B into the reactor undergoes a step function at t_0.

The control variable $F^{water}(t)$ was approximated by four linear finite elements, and the inequality path constraint was active during the second element. The initial guess for the control was $F^{water}(t) = 3$tonne/hr, and for the time invariant parameter was $R = 0.5$. The control was bounded by $0 \leq F^{water} \leq 5$, and the reflux ratio was bounded by $0.4 \leq R \leq 0.83$. The size of the DAE was 741 equations, and there were 13 optimization parameters, which brought the size of the combined DAE and sensitivity system to 10374 equations. Solution statistics for this problem are given in Table 8-13. Optimal solution profiles are shown in Figures 8-23–8-28.

Integration Tolerance	Optimization Tolerance	IVP Solutions	Objective Function	CPU
10^{-5}	10^{-3}	91	153.9297	2455s

Table 8-13: *Solution statistics for the CSTR and column startup problem*

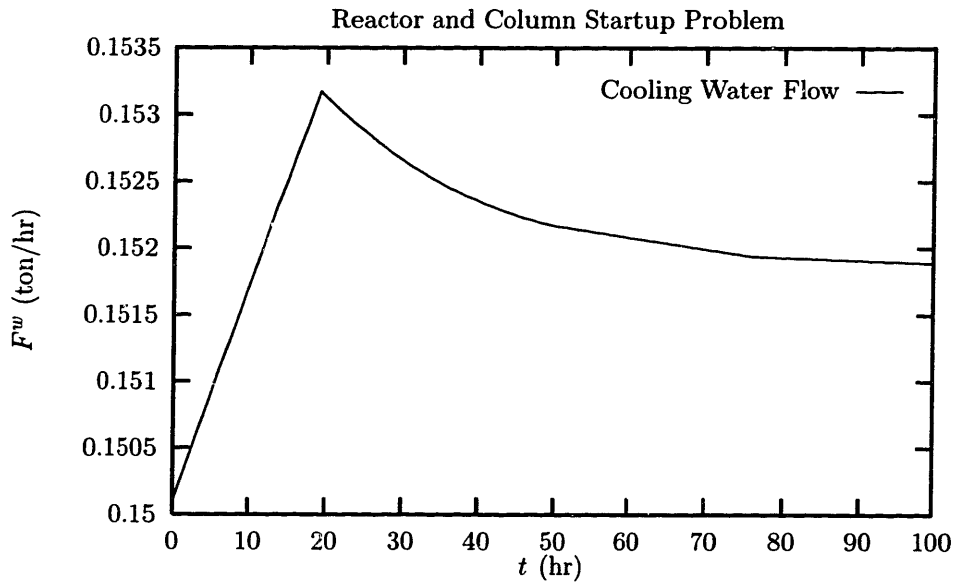

Figure 8-23: *The optimal trajectory for the cooling water flowrate*

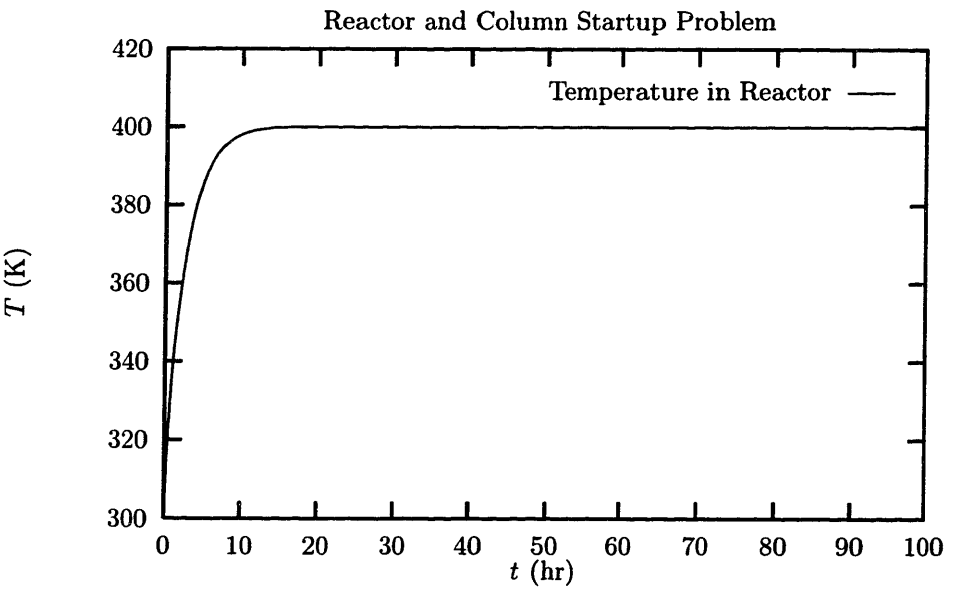

Figure 8-24: *The temperature profile in the reactor*

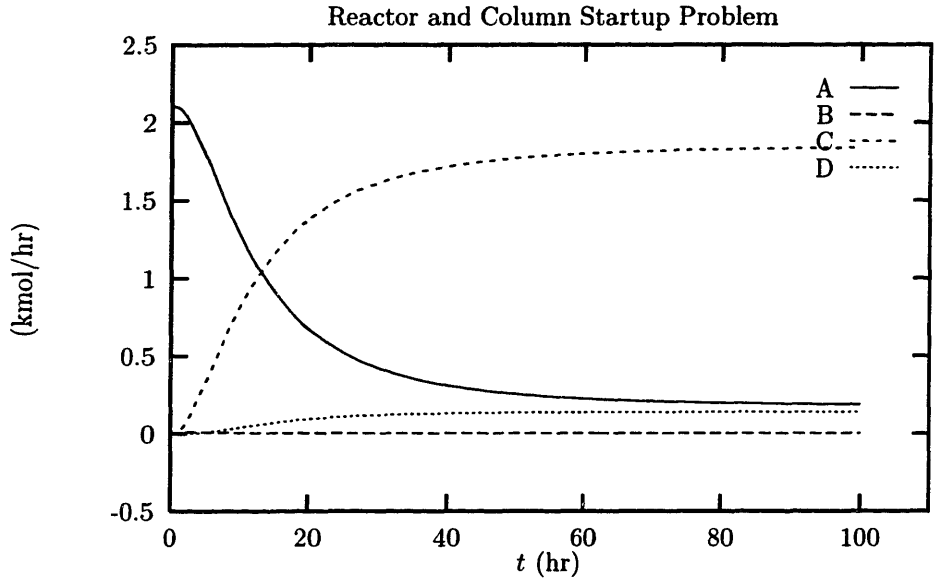

Figure 8-25: *The molar flowrates of the species leaving the system*

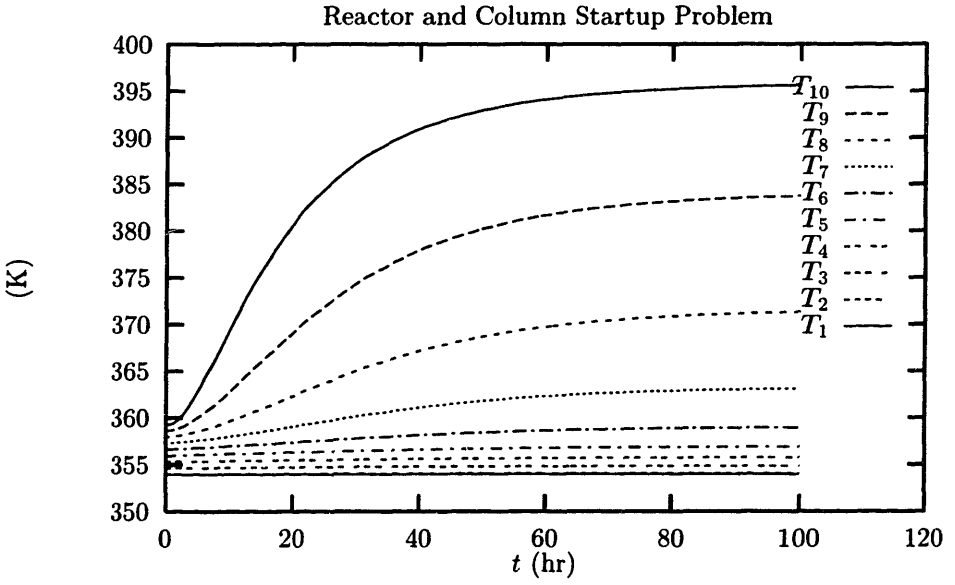

Figure 8-26: *The temperature profile in the column*

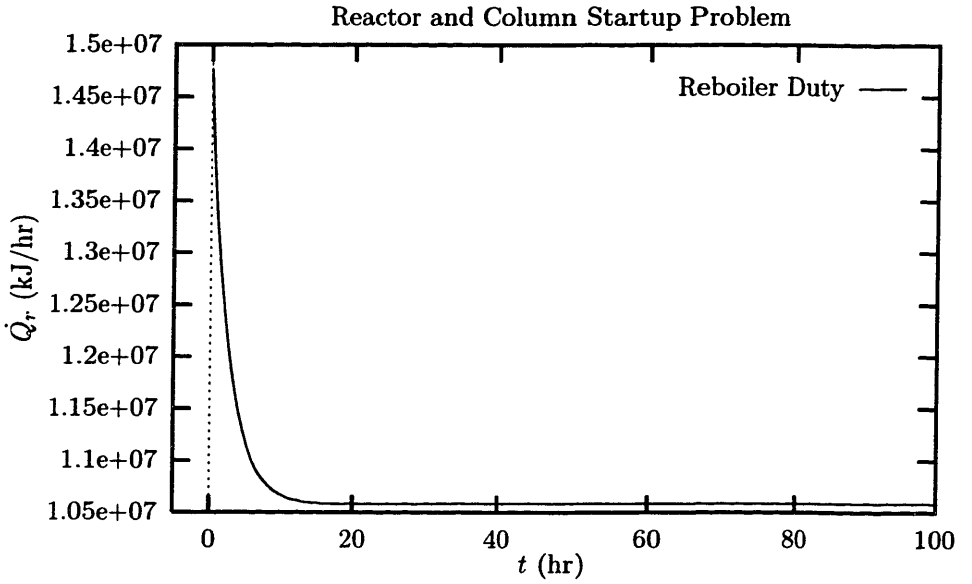

Figure 8-27: *The reboiler duty*

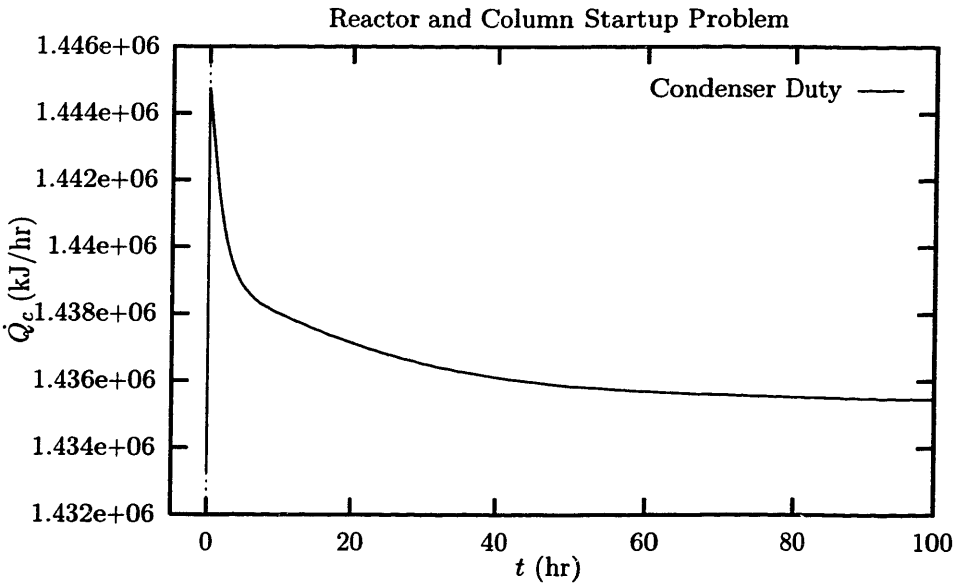

Figure 8-28: *The condenser duty*

Chapter 9

Conclusions

This thesis has demonstrated several developments which facilitate reliable numerical solution of the types of dynamic optimization problems found in process engineering. Specifically, advances were made in three interrelated areas: sensitivity analysis, numerical solution of high-index DAEs, and solution of dynamic optimization problems with state variable path constraints.

The staggered corrector algorithm for numerical sensitivity analysis of DAEs has been demonstrated to be significantly more efficient than other existing methods. Also, it has been implemented in the DSL48S numerical solver for large-scale sparse DAEs which is well-implemented, extensively tested, and stable. Therefore, efficient numerical sensitivity analysis is now an easily available tool in process engineering, and can be used, for example, in designing control strategies, finding the response of a system to disturbances, and determining the design variables to which a system is most sensitive.

The use of efficient sensitivity analysis makes the control parameterization method more attractive relative to indirect methods for solving dynamic optimization problems. Use of indirect methods typically requires forward-integration of the DAE and then backward-integration of the adjoint equations, and it does not appear possible to take advantage of the fact that the Jacobians of both the adjoint and DAE systems are based on similar information. By using sensitivities instead of adjoints, the staggered corrector method is able to minimize both the number of Jacobian factor-

izations and the number of Jacobian updates by taking advantage of the similarities between the DAE and its sensitivity system.

Control parameterization is also more attractive than the simultaneous direct methods that have been proposed for solving dynamic optimization problems. From a practical standpoint, control parameterization is able to take advantage of highly developed and efficient numerical solvers for the IVP and sensitivity subproblem, and the master NLPs are kept to a size easily manageable with current technology. The one supposed advantage of simultaneous methods, the ability to handle path constraints directly, is negated by methods like FIIP which handle the constraints directly within the control parameterization framework. For these reasons, we believe that the control parameterization method is the best currently available method for solving the types of dynamic optimization problems encountered in process engineering.

An important result of this thesis is the clarification of the fact that path-constrained dynamic optimization problems are naturally high-index DAEs, and therefore all methods that solve path constrained dynamic optimization problems are capable of solving high-index DAEs. However, most dynamic optimization methods handle these high-index DAEs indirectly. This thesis shows that there are advantages to be gained by directly solving high-index DAEs, and vice-versa. This observation led to the development in this thesis of the dummy derivative method as a practical technique for solving a broad class of arbitrary-index DAEs. When used with control parameterization, the dummy derivative method allows the path constraints to be included in the IVP subproblem, rather than in the Master NLP. The result is that fewer IVP subproblems are required to solve many dynamic optimization problems, and the accuracy of the path constraints is guaranteed over the entire solution trajectory to within the integration tolerances.

Including path constraints directly in the IVP raises a set of issues concerning dynamic degrees of freedom and feasibility of the dynamic optimization problem. Whenever a path constraint is appended to the system, a control variable becomes determined by the solution to the augmented DAE, and thus is not free to be independently specified. A control matching algorithm was proposed that is able to

determine in many problems which, if any, controls cease to be design degrees of freedom in the augmented DAE. Furthermore, it was shown that the constrained dynamic optimization problem can be feasible only if the augmented DAE is well-posed.

Inequality path constraints on dynamic optimization problems raise an additional set of issues. The index of the DAE can change between segments of the trajectory where the constraint becomes active or inactive. The transfer conditions for state and sensitivity variables at these points require special attention. In Chapter 3 the transfer conditions for the sensitivity variables of hybrid systems (which include systems where inequalities activate and deactivate) were derived. A key result is that sensitivities of hybrid systems exist and are unique provided that the sequence of state events does not change. This result is important for the development of methods for solving inequality path-constrained dynamic optimization and hybrid discrete/continuous dynamic optimization problems.

One of the most interesting results of this work is that there are several strategies which allow inequality path constraints on dynamic optimization problems to be enforced by the IVP solver within the control parameterization framework. The slack variable method transforms the inequality constraints into equality constraints. The FIFP method transforms the inequality constrained problem into a hybrid discrete/continuous optimization problem. The FIIP method transforms the inequality constrained problem into a mixed-integer dynamic optimization problem (MIDO).

Each of these methods has its own set of advantages and disadvantages. The slack variable method avoids problems with sequencing decisions for constraint activations and deactivations. However, the slack variable method is not capable of handling dynamic optimization problems with more inequality constraints than control variables, and it creates a high-index DAE whether or not the inequality constraint is active. The FIFP method does not have restrictions on the relative number of control variables and inequality constraints, and it handles the constraints efficiently, but it creates nonsmooth NLPs with small regions of convergence. Such problems cannot be easily solved with standard gradient-based optimization methods. The FIIP method has all the advantages of the FIFP method and it creates smooth NLPs, but it requires

that the sequence of constraint activation and deactivation events be determined *a priori*.

Of these methods, the FIIP method proved to be the most successful. To use the method, a sequence of constraint events must be specified, and the index of the DAE is permitted to fluctuate between unconstrained and constrained segments. The path constraints are permitted to be violated during intermediate iterations in order to provide transition conditions for the state variables between unconstrained and constrained segments. The FIIP method was successfully demonstrated on several examples. FIIP is efficient and it guarantees that the path constraints are satisfied over the optimal solution trajectory.

Finding the optimal sequence of constraint activations and deactivations for the FIIP method can be difficult. If the dynamic optimization problem contains few constraints, as is the case with almost all problems that have been solved in the literature, it is usually possible to find the sequence by examining the solution of the unconstrained dynamic optimization problem. In the case where the dynamic optimization problem contains many constraints, finding the optimal sequence becomes a combinatorial problem.

The methods developed in this thesis are capable of solving inequality constrained dynamic optimization problems that cannot be solved using other methods, and problems that can be solved using other methods with greater accuracy and efficiency.

9.1 Directions for Future Work

An interesting area for future research is to find methods to solve the classes of high-index DAEs that cannot be solved using the dummy derivative algorithm. The dummy derivative method is not currently capable of solving problems for which the index cannot be detected structurally. Detection of the index using numerical, rather than structural methods is problematic from both theoretical and practical standpoints. Theoretically, singularity of the Jacobian of a nonlinear DAE with respect to the highest order time derivatives does not necessarily indicate that the DAE is high-index. Practically, it is difficult to differentiate between a singular matrix and one that is highly ill-conditioned.

The computationally expensive part of the dummy derivative method is dummy derivative pivoting. Although the pivoting operation is not a significant expense for most problems, detecting the need to pivot can be expensive. The heuristics for minimizing the number of pivot checks proposed in this thesis perform well for most problems. However, the heuristics could be improved if there were a method for differentiating between truncation error failures that occur because of a need for dummy pivoting, and those that occur for other reasons.

Structural criteria are also used in the methods developed in this thesis to determine feasibility and find control matchings for path-constrained dynamic optimization problems. Since the structural criteria do not work for all problems, there are dynamic optimization problems which according to these structural criteria are feasible and/or have a valid control matching which are in fact unsolvable. Also, structural criteria do not indicate the 'best' control matching for those problems where there are several possible control matchings, and solution of such problems requires further research. However, these difficulties are related to the problem of nonlinear controllability, which has been an outstanding problem in the literature for many years.

The algorithm for numerical sensitivity analysis described in this thesis has significantly improved the computational efficiency of numerical solution of dynamic optimization problems. However, a number of interesting research issues remain.

The size of the dynamic optimization problem solvable with the control parameterization method is limited by the number of parameters in the master NLP, rather than by the number of state variables in the DAE. Numerical solution and sensitivity analysis of sparse DAEs containing 10,000-100,000 variables is possible with current technology, but solution of relatively large (1,000-10,000 parameters) dense master NLPs is problematic. Such master NLPs arise in dynamic optimization problems with large numbers of controls in the plant-wide model, higher-order basis functions, or many finite elements. Furthermore, in such problems partial derivatives of the point constraints with respect to all of the parameters can exist, which means that strategies such as range and null space decomposition SQP will not work. Therefore, methods are needed which can solve large dense NLPs for which the explicit functional form of the constraints and objective is not known. A further major challenge is that these large-scale master NLPs are often multi-modal [8].

Another interesting avenue of research is developing parallel algorithms for solution of dynamic optimization problems. At the top level, careful problem formulation (*e.g.*, decoupling of batch units through intermediate storage) can lead to a number of independent IVP subproblems that can be solved in parallel. Within each IVP subproblem, the sensitivity equations can be solved in parallel [77, 94]. Finally, at the lowest level parallel algorithms could be used for the linear algebra and parallel automatic differentiation techniques could be used for the function and derivative evaluations.

This thesis has shown that inequality path-constrained dynamic optimization problems may be posed as either MIDOs or as hybrid dynamic optimization problems. Solution of both types of problems is currently an active area of research, and both have applications other than inequality path-constrained dynamic optimization problems. For example, operating procedure synthesis problems can also be posed as either MIDOs or hybrid dynamic optimization problems. In operating procedure synthesis, it is desired to find the optimal controls that transform a process from one state to another subject to economic, environmental, and safety constraints. The decision variables are typically both discrete (*e.g.*, open or close valves) and continuous

(*e.g.*, controller setpoints).

Hybrid dynamic optimization problems are difficult to solve because they often cause the master NLP to be nonsmooth, which creates problems with gradient-based optimization methods. Sub-gradient optimization techniques are capable of handling some classes of nonsmooth optimization problems, and it may be possible to find a method that transforms the inequality path-constrained dynamic optimization problem into a hybrid problem that has an NLP of the appropriate form. On the other hand, stochastic or random search methods are relatively insensitive to nonsmoothness, so these methods offer an immediate and practical way to make sequencing decisions in dynamic optimization problems. However, a more sophisticated approach that exploits the evident structure of such problems (see Chapter 4) would ultimately be more desirable.

The difficulty with solution of MIDO problems is finding a 'link' between the continuous and discrete aspects of the problem. That is, there is no satisfactory method for determining how to vary the discrete variables based on information obtained from the solution to the continuous problem. Several approaches have been attempted thus far. In [102, 101], an attempt was made to solve MIDOs by using orthogonal collocation on finite elements to transform the MIDO into a finite-dimensional mixed integer nonlinear program (MINLP). More recently, in [103] an analog of the control parameterization method for ordinary dynamic optimization problems was proposed. In this scheme, numerical integration is used to obtain gradient and objective function information for a master MINLP. It is shown that the adjoint variables of the dynamic optimization problem can be used to find the dual information necessary for MINLP algorithms. However, the problem with these approaches is that the NLPs are nonconvex, and therefore general methods for solving MINLPs (for a review, see [52]) do not work [3] (in fact, the methods may converge to points that are not even local extrema).

Appendix A

ABACUSS input file for reactor and column example

```
DECLARE

  TYPE                    # Default   Min      Max    UNITS
      Concentration         = 0     : -1e20 : 1E20     unit = "mol/m^3"
      Energy                = 0     : -1E20 : 1E20    unit = "KJ"
      EnergyFlowRate        = 0     : -1E20 : 1E20    unit = "KJ/hr"
      MolarEnthalpy         = 1     : -1E20 : 1E20    unit = "KJ/mol"
      LiquidMolarVolume     = 20    :  -1E20 : 1E20    unit = "cm3/mol"
      VaporMolarVolume      = .0224: 0   : 1E20      unit = "m3/mol"
      MassFlowRate          = 1     : 0      : 1E20     unit = "g/s"
      KgMassFlowRate        = 1     : 0      : 1E20     unit = "kg/s"
      MolarVolume           = .01   : 0   : 1E20      unit = "m^3/mol"
      Fraction              = .5    : -1E20      : 1E20     unit = "dimensionless"
      MolarFlowRate         = 0     : -1e20 : 1E20    unit = "mol/s"
      Length      = .5    : -1E20 : 1E20    unit = "m"
      MolarHoldup           = 1     : -1e20 : 1E20     unit = "mol"
      PositiveValue         = .75   : 0      : 1E20    unit = "dimensionless"
      Pressure              = 1.013: 0      : 1E20    unit = "bar"
      Power                 = 0     : -1E20 : 1E20    unit = "GJ/s"
      ReactionRate          = 0     : -1e20 : 1E20     unit = "mol/m3/s"
      ReducedQuantity       = .5    :  0     : 1E20      unit = "dimensionless"
      SmallValue            = .02   : -10    : 1E20      unit = "dimensionless"
      Temperature           = 298   : 0    : 1E20    unit = "K"
      Value                 = 4     : -1e20 : 1e20    unit = "dimensionless"
      Velocity              = 2     : -1e20  : 1e20     unit = "m/s"
      Volume                = 1     : 0   : 1E20   unit = "m^3"
      VolumeFlowRate        = 1e-9  : 0   : 1E20      unit = "m^3/hr"
      PosValue          = 1     : 1e-2 : 1e40    unit = "dimensionless"
      Molar_Concentration   = 1.0   :  -1E-2: 1.0e20  unit = "mol/m3"
      Reaction_Rate         = 0.0   :  -1e-2:  1e20    unit = "mol/(m3 s)"
```

```
      WaterFlowRate          = 0     : -1e-2 : 1E20    unit = "ton/hr"

END # Declarations
#=============================================================================

MODEL LiquidEnthalpy

  PARAMETER
    NC                              AS INTEGER # number of components
    RGAS                            AS REAL
    TREF                            AS REAL
    ENTHA                           AS ARRAY(NC) OF REAL
    ENTHB                           AS ARRAY(NC) OF REAL
    ENTHC                           AS ARRAY(NC) OF REAL
    ENTHD                           AS ARRAY(NC) OF REAL
    HVAP                            AS ARRAY(NC) OF REAL
    Heat_Formation                  AS ARRAY(NC) OF REAL

  VARIABLE
    Specific_Enthalpy_Liquid        AS ARRAY(NC) OF MolarEnthalpy
    Specific_Enthalpy_Vapor         AS ARRAY(NC) OF MolarEnthalpy
    Temp                            AS Temperature

  EQUATION
    Specific_Enthalpy_Liquid=
          Specific_Enthalpy_Vapor-HVAP ;

    Specific_Enthalpy_Vapor=Heat_Formation+
       ENTHA*(Temp-TREF)+ENTHB*(Temp-TREF)^2+
       (1/2)*ENTHC*(Temp-TREF)^3+(1/3)*ENTHD*(Temp-TREF)^4 ;

END  #LiquidEnthalpy

#=============================================================================

MODEL RD1_Reactions

  PARAMETER
    A, B, C, D                  AS INTEGER # identifiers for the components
    NR                          AS INTEGER # number of reactions
    NC                          AS INTEGER # number of components
    MOLAR_VOLUME                AS ARRAY(NC) of REAL
    PRE_EXP_FACTOR              AS ARRAY(NR) of REAL
    RGAS                        AS REAL
    ACTIVATION_ENERGY           AS ARRAY(NR) of REAL
    STOICH_COEFF                AS ARRAY(NC,NR) of INTEGER
    #Enthalpy_Reaction           AS ARRAY(NR) OF REAL
    HVAP                           AS ARRAY(NC) OF REAL

  VARIABLE
    Concentration           AS ARRAY(NC) of Molar_Concentration
    no_Mols                      AS ARRAY(NC) of MolarHoldup
    ReactionRate                 AS ARRAY(NR) of Reaction_Rate
    temp                         AS Temperature
```

```
    feed_A                      AS MolarFlowRate
    feed_B                      AS MolarFlowRate
    feed_C                      AS MolarFlowRate
    feed_D                      AS MolarFlowRate
    TotalFeed                   AS MolarFlowRate
    FlowOut                     AS MolarFlowRate

    volume                      AS Volume

    QJacket                     AS EnergyFlowRate

    TotalMols                   AS MolarHoldup
    X                           AS ARRAY(NC) OF Fraction
    Enthalpy                    AS Energy
    Specific_Enthalpy           AS ARRAY(NC) OF Energy
    FeedEnthalpy                AS ARRAY(NC) OF Energy
    FeedTemp                    AS Temperature

EQUATION

  # Material Balances for feed species [kmols]
  $no_Mols(A) = feed_A  +  volume * SIGMA(STOICH_COEFF(A,) * ReactionRate)
                - FlowOut*X(A);
  $no_Mols(B) = feed_B  +  volume *SIGMA(STOICH_COEFF(B,) * ReactionRate)
                - FlowOut*X(B);
  $no_Mols(C) = feed_C  +  volume *SIGMA(STOICH_COEFF(C,) * ReactionRate)
                - FlowOut*X(C);
  $no_Mols(D) = feed_D  +  volume *SIGMA(STOICH_COEFF(D,) * ReactionRate)
                - FlowOut*X(D);

  TotalFeed = feed_A + feed_B + feed_C + feed_D ;

  # Define the reaction rates for each reaction
  ReactionRate(1)
     = PRE_EXP_FACTOR(1)*EXP(-ACTIVATION_ENERGY(1)/(RGAS*temp))
       *Concentration(A)*Concentration(B) ;

  ReactionRate(2)
     = PRE_EXP_FACTOR(2)*EXP(-ACTIVATION_ENERGY(2)/(RGAS*temp))
       *Concentration(B)*Concentration(C) ;

  # Define the the volume and the concentrations
  volume = SIGMA(MOLAR_VOLUME*no_mols) ;
  Concentration*volume = no_mols ;

  TotalMols=SIGMA(no_Mols) ;

  X=no_Mols/TotalMols ;

  Enthalpy=SIGMA(No_Mols*Specific_Enthalpy);

  $enthalpy = (FeedEnthalpy(A)+FeedEnthalpy(B)+FeedEnthalpy(C)+FeedEnthalpy(D)

              -FlowOut*SIGMA(X*Specific_Enthalpy)
```

```
                -QJacket);

END # RD1_Reactions

#=============================================================================

MODEL Reactor_Jacket

PARAMETER
  Heat_Transfer_Coeff              AS REAL
  Jacket_Area                      AS REAL
  CP_water                         AS REAL
  Twater_in                        AS REAL

VARIABLE
  QJacket                          AS EnergyFlowRate
  TWater_out                       AS Temperature
  Temp_Reactor                     AS Temperature
  Flow_Water                       AS WaterFlowRate

EQUATION
  #Heat removed by jacket
  QJacket=Heat_Transfer_Coeff*Jacket_Area*(Temp_Reactor-Twater_out) ;

  #Outlet temperature of water
  QJacket=CP_water*(Twater_out-Twater_in)*Flow_Water ;
END

MODEL ColumnVLE

  PARAMETER
    NSTAGE          AS INTEGER
    NC              AS INTEGER
    VPA             AS ARRAY(NC) OF REAL
    VPB             AS ARRAY(NC) OF REAL
    A, B, C, D      AS INTEGER # identifiers for the components
    TREF            AS REAL
    ENTHA           AS ARRAY(NC) OF REAL
    ENTHB           AS ARRAY(NC) OF REAL
    ENTHC           AS ARRAY(NC) OF REAL
    ENTHD           AS ARRAY(NC) OF REAL
    HVAP            AS ARRAY(NC) OF REAL

  VARIABLE
    X                          AS ARRAY(NSTAGE,NC) OF Fraction
    Y                          AS ARRAY(NSTAGE,NC) OF Fraction
    K                          AS ARRAY(NSTAGE,NC) OF Value
    VAPORENTHALPY              AS ARRAY(NSTAGE) OF MolarEnthalpy
    LIQUIDENTHALPY             AS ARRAY(NSTAGE) OF MolarEnthalpy
    VAPORPRESSURE              AS ARRAY(NSTAGE,NC) OF Pressure
    P                          AS ARRAY(NSTAGE) OF Pressure
    Temp                       AS ARRAY(NSTAGE) OF Temperature
    Specific_Enthalpy_Liquid   AS ARRAY(NSTAGE,NC) OF MolarEnthalpy
    Specific_Enthalpy_Vapor    AS ARRAY(NSTAGE,NC) OF MolarEnthalpy
```

```
  EQUATION
    # Vapor Pressure
    FOR I:=1 TO NSTAGE DO
      VAPORPRESSURE(I,)=EXP(-VPA/Temp(I)+VPB) ;
    END

    FOR I:=1 TO NSTAGE DO
      Specific_Enthalpy_Liquid(I,)=
          Specific_Enthalpy_Vapor(I,)-HVAP ;

      Specific_Enthalpy_Vapor(I,)=
         ENTHA*(Temp(I)-TREF)+ENTHB*(Temp(I)-TREF)^2+
         (1/2)*ENTHC*(Temp(I)-TREF)^3+(1/3)*ENTHD*(Temp(I)-TREF)^4 ;

      LIQUIDENTHALPY(I) = SIGMA(X(I,)*Specific_Enthalpy_Liquid(I,));
      VAPORENTHALPY(I) = SIGMA(Y(I,)*Specific_Enthalpy_Vapor(I,));
    END

    # Ideal Phase equilibrium constant
    FOR I:=1 TO NSTAGE DO
      K(I,)=VAPORPRESSURE(I,)/P(I) ;
    END

END

MODEL Column

  PARAMETER
    NSTAGE          AS INTEGER
    FEEDSTAGE       AS INTEGER
    NC              AS INTEGER
    A, B, C, D      AS INTEGER # identifiers for the components
    MOLAR_VOLUME                AS ARRAY(NC) of REAL
    MW              AS ARRAY(NC) of REAL
    STAGE_AREA      AS REAL
    WEIR_LENGTH     AS REAL
    WEIR_HEIGHT     AS REAL
    ORIFCON         AS REAL
    RGAS            AS REAL
    AFREE           AS REAL
    ALINE           AS REAL
    KLOSS           AS REAL

  VARIABLE
    M                           AS ARRAY(NSTAGE) OF MolarHoldup
    L                           AS ARRAY(NSTAGE) OF MolarFlowRate
    V                           AS ARRAY(NSTAGE) OF MolarFlowRate
    TotalHoldup                 AS MolarHoldup
    VAPORENTHALPY               AS ARRAY(NSTAGE) OF MolarEnthalpy
    LIQUIDENTHALPY              AS ARRAY(NSTAGE) OF MolarEnthalpy
    X                           AS ARRAY(NSTAGE,NC) OF Fraction
    Y                           AS ARRAY(NSTAGE,NC) OF Fraction
    K                           AS ARRAY(NSTAGE,NC) OF Value
```

```
  DISTILLOUT              AS MolarFlowRate
  QR                      AS EnergyFlowRate
  QC                      AS EnergyFlowRate
  REFLUXRATIO             AS Value
  P                       AS ARRAY(NSTAGE) OF Pressure
  DPSTAT                  AS ARRAY(NSTAGE) OF Pressure
  DPDRY                   AS ARRAY(NSTAGE) OF Pressure
  DPTRAY                  AS ARRAY(NSTAGE) OF Pressure
  T                       AS ARRAY(NSTAGE) OF Temperature
  FEED                    AS ARRAY(NSTAGE) OF MolarFlowRate
  BOTTOMS                 AS MolarFlowRate
  XFEED                   AS ARRAY(NC) OF Fraction
  FEEDENTHALPY            AS MolarEnthalpy
  VOLUMEHOLDUP            AS ARRAY(NSTAGE) OF VOLUME
  HEAD                    AS ARRAY(NSTAGE) OF Length
  LIQHEIGHT               AS ARRAY(NSTAGE) OF Length
  LIQDENS                 AS ARRAY(NSTAGE) OF Molar_Concentration
  VAPDENS                 AS ARRAY(NSTAGE) OF Molar_Concentration
  LIQMDENS                AS ARRAY(NSTAGE) OF Molar_Concentration
  VAPMDENS                AS ARRAY(NSTAGE) OF Molar_Concentration
 # FeedTemp                AS Temperature
  MWV                     AS ARRAY(NSTAGE) OF Value
  MWL                     AS ARRAY(NSTAGE) OF Value
EQUATION

# Condenser Model

  # Material Balance
  V(2)=DISTILLOUT+L(1)  ;
  V(2)*REFLUXRATIO=L(1) ;
  X(1,)=Y(2,) ;

  # No holdup and no vapor
  V(1)=0 ;
  M(1)=0 ;

  # Energy Balance
  QC=V(2)*VAPORENTHALPY(2)-(L(1)+DISTILLOUT)*LIQUIDENTHALPY(1) ;

  # Phase Equilibrium
  Y(1,)=K(1,)*X(1,);
  SIGMA(Y(1,))=1 ;

  # Liquid and vapor densities
  LIQDENS(1)=SIGMA(X(1,)/MOLAR_VOLUME);
  VAPDENS(1)=(P(1)*100000/(RGAS*T(1)))/1000 ;
  LIQMDENS(1)=SIGMA(X(1,)*MW/MOLAR_VOLUME);
  VAPMDENS(1)=VAPDENS(1)*SIGMA(Y(1,)*MW) ;
  MWV(1)=SIGMA(Y(1,)*MW);
  MWL(1)=SIGMA(X(1,)*MW);

  # Relate liquid overflow rate to holdup
  VOLUMEHOLDUP(1)=M(1)/LIQDENS(1);
  LIQHEIGHT(1)=0;
```

```
  HEAD(1)=0 ;
  ((V(2)*MWV(2)/3600)/VAPMDENS(1)*ALINE)^2=2.0*DPTRAY(1)/VAPMDENS(1)/KLOSS;

  # No static or dry pressure loss
  DPSTAT(1)=0;
  DPDRY(1)=0;

# Tray Model
  FOR I:=2 TO NSTAGE-1 DO

    # Material Balance
    $M(I)=L(I-1)-L(I)+V(I+1)-V(I)+FEED(I) ;
    $M(I)*X(I,)+M(I)*$X(I,)=
       L(I-1)*X(I-1,)-L(I)*X(I,)+
       V(I+1)*Y(I+1,)-V(I)*Y(I,)+
       FEED(I)*XFEED ;

    # Energy Balance
    #$M(I)*LIQUIDENTHALPY(I)+M(I)*$LIQUIDENTHALPY(I)=
    0=
    L(I-1)*LIQUIDENTHALPY(I-1)-L(I)*LIQUIDENTHALPY(I)+
      V(I+1)*VAPORENTHALPY(I+1)-V(I)*VAPORENTHALPY(I)+
      FEED(I)*FEEDENTHALPY;

    # Phase Equilibrium
    Y(I,)=K(I,)*X(I,);

    # Normalize liquid mole fraction
    SIGMA(Y(I,))=1;

    # Liquid and vapor densities
    LIQDENS(I)=SIGMA(X(I,)/MOLAR_VOLUME);
    VAPDENS(I)=(P(I)*100000/(RGAS*T(I)))/1000 ;
    LIQMDENS(I)=SIGMA(X(I,)*MW/MOLAR_VOLUME);
    VAPMDENS(I)=VAPDENS(I)*SIGMA(Y(I,)*MW) ;
    MWV(I)=SIGMA(Y(I,)*MW);
    MWL(I)=SIGMA(X(I,)*MW);

    # Relate liquid overflow rate to holdup
    VOLUMEHOLDUP(I)=M(I)/LIQDENS(I);
    LIQHEIGHT(I)=VOLUMEHOLDUP(I)/STAGE_AREA;
    HEAD(I)=(LIQHEIGHT(I)-WEIR_HEIGHT)*1000 ;
    L(I)*MWL(I)/3600=LIQMDENS(I)*WEIR_LENGTH*(HEAD(I)/750)^1.5 ;

    # Relate vapor flow to plate pressure drop
    DPSTAT(I)=LIQHEIGHT(I)*9.81*LIQMDENS(I)/100000;
    DPDRY(I)=(1/100000)*(0.50031*VAPMDENS(I))*
      ((V(I+1)*MWV(I)/3600)/VAPMDENS(I)*AFREE*ORIFCON)^2  ;
    DPTRAY(I)=DPSTAT(I)+DPDRY(I)+(12.5*9.81)/100000 ;
    P(I)=P(I-1)+DPTRAY(I-1) ;
  END

  #Reboiler Model
    # Material Balance
```

```
      $M(NSTAGE)=L(NSTAGE-1)-L(NSTAGE)-V(NSTAGE) ;
      BOTTOMS=L(NSTAGE) ;
      $M(NSTAGE)*X(NSTAGE,)+M(NSTAGE)*$X(NSTAGE,)=
         L(NSTAGE-1)*X(NSTAGE-1,)-L(NSTAGE)*X(NSTAGE,)-
         V(NSTAGE)*Y(NSTAGE,) ;

      # Energy Balance
      #$M(NSTAGE)*LIQUIDENTHALPY(NSTAGE)+M(NSTAGE)*$LIQUIDENTHALPY(NSTAGE)=
      0=
        L(NSTAGE-1)*LIQUIDENTHALPY(NSTAGE-1)-L(NSTAGE)*LIQUIDENTHALPY(NSTAGE)
        -V(NSTAGE)*VAPORENTHALPY(NSTAGE)+QR;

      # Phase Equilibrium
      Y(NSTAGE,)=K(NSTAGE,)*X(NSTAGE,);
      SIGMA(Y(NSTAGE,))=SIGMA(X(NSTAGE,));

      # Liquid and vapor densities
      LIQDENS(NSTAGE)=SIGMA(X(NSTAGE,)/MOLAR_VOLUME);
      VAPDENS(NSTAGE)=(P(NSTAGE)*100000/(RGAS*T(NSTAGE)))/1000 ;
      LIQMDENS(NSTAGE)=SIGMA(X(NSTAGE,)*MW/MOLAR_VOLUME);
      VAPMDENS(NSTAGE)=VAPDENS(NSTAGE)*SIGMA(Y(NSTAGE,)*MW) ;
      MWV(NSTAGE)=SIGMA(Y(NSTAGE,)*MW);
      MWL(NSTAGE)=SIGMA(X(NSTAGE,)*MW);

      # Relate liquid overflow rate to holdup
      VOLUMEHOLDUP(NSTAGE)=M(NSTAGE)/LIQDENS(NSTAGE);
      LIQHEIGHT(NSTAGE)=0;
      HEAD(NSTAGE)=0 ;

      # Relate vapor flow to plate pressure drop
      DPSTAT(NSTAGE)=0;
      DPDRY(NSTAGE)=0;
      DPTRAY(NSTAGE)=0 ;
      P(NSTAGE)=P(NSTAGE-1)+DPTRAY(NSTAGE-1) ;

    #Total Column holdup
    TotalHoldup=SIGMA(M);

END # Column

MODEL Flowsheet

  PARAMETER
    A, B, C, D                  AS INTEGER # identifiers for the components
    NR                          AS INTEGER # number of reactions
    NC                          AS INTEGER # number of components
    MOLAR_VOLUME                AS ARRAY(NC) of REAL
    MW                          AS ARRAY(NC) of REAL
    PRE_EXP_FACTOR              AS ARRAY(NR) of REAL
    RGAS                        AS REAL
    ACTIVATION_ENERGY           AS ARRAY(NR) of REAL
    STOICH_COEFF                AS ARRAY(NC,NR) of INTEGER
    #Enthalpy_Reaction           AS ARRAY(NR) OF REAL
    Heat_Formation           AS ARRAY(NC) OF REAL
```

```
  Heat_Transfer_Coeff          AS REAL
  Jacket_Area                  AS REAL
  CP_water                     AS REAL
  Twater_in                    AS REAL

  TREF                         AS REAL
  ENTHA                        AS ARRAY(NC) OF REAL
  ENTHB                        AS ARRAY(NC) OF REAL
  ENTHC                        AS ARRAY(NC) OF REAL
  ENTHD                        AS ARRAY(NC) OF REAL
  HVAP                         AS ARRAY(NC) OF REAL

  NSTAGE                       AS INTEGER
  FEEDSTAGE                    AS INTEGER
  STAGE_AREA                   AS REAL
  WEIR_LENGTH                  AS REAL
  WEIR_HEIGHT                  AS REAL

  VPA                          AS ARRAY(NC) OF REAL
  VPB                          AS ARRAY(NC) OF REAL
  AFREE                        AS REAL
  ORIFCON                      AS REAL
  ALINE                         AS REAL
  KLOSS                         AS REAL

SET
  NR                        := 2 ;
  NC                        := 4 ;

  A                         := 1;
  B                         := 2;
  C                         := 3;
  D                         := 4;

  MOLAR_VOLUME(A)           := 1.0/11.0 ;
  MOLAR_VOLUME(B)           := 1.0/16.0 ;
  MOLAR_VOLUME(C)           := 1.0/10.4 ;
  MOLAR_VOLUME(D)           := 1.0/10.0 ;

  MW(A):= 76 ;
  MW(B):= 52 ;
  MW(C):= 170 ;
  MW(D):= 264 ;

  PRE_EXP_FACTOR(1)         := 100.0 ;
  PRE_EXP_FACTOR(2)         := 120.0 ;

  ACTIVATION_ENERGY(1)      := 17000 ;
  ACTIVATION_ENERGY(2)      := 20000 ;

  RGAS                      := 8.314 ;
```

```
                    # R1  R2
 STOICH_COEFF(A,)  := [-1,  0] ;
 STOICH_COEFF(B,)  := [-1, -1] ;
 STOICH_COEFF(C,)  := [ 1, -1] ;
 STOICH_COEFF(D,)  := [ 0,  1] ;

# Enthalpy_Reaction(1) := -60000 ;
# Enthalpy_Reaction(2) := -50000 ;

 Heat_Formation(A) := -30000;
 Heat_Formation(B) := -50000;
 Heat_Formation(C) := -20000;
 Heat_Formation(D) := -20000;

 Heat_Transfer_Coeff := 3000 ;
 Jacket_Area         := 30      ;
 CP_water            := 4200    ;
 Twater_in           := 298     ;

 TREF                := 298   ;
 ENTHA(A)            := 172.3 ;
 ENTHA(B)            := 200.0 ;
 ENTHA(C)            := 160.0 ;
 ENTHA(D)            := 155.0 ;

 ENTHB(1:NC)         := 0.0 ;
 ENTHC(1:NC)         := 0.0 ;
 ENTHD(1:NC)         := 0.0 ;

 HVAP(A)             := 31000 ;
 HVAP(B)             := 26000 ;
 HVAP(C)             := 28000 ;
 HVAP(D)             := 34000 ;

 NSTAGE              := 10 ;
 FEEDSTAGE           := 5 ;
 STAGE_AREA          := 1.05;
 AFREE               := 0.1*STAGE_AREA ;
 ALINE               := .03;
 KLOSS               := 1 ;
 ORIFCON             :=0.6 ;
 WEIR_LENGTH         :=1 ;
 WEIR_HEIGHT         :=0.25;

 VPA(A)              := 4142.75;
 VPA(B)              := 3474.56;
 VPA(C)              := 3500;
 VPA(D)              := 4543.71;

 VPB(A)              := 11.7158;
 VPB(B)              :=  9.9404;
 VPB(C)              :=  8.9;
 VPB(D)              := 11.2599;
```

```
END #Flowsheet

MODEL ReactorFlowsheet INHERITS Flowsheet
  UNIT
    Reactor AS RD1_Reactions
    Jacket  AS Reactor_Jacket
    LiqEnthalpy AS LiquidEnthalpy
    ReactorFeedEnthalpy AS LiquidEnthalpy

  EQUATION
    Reactor.Temp IS Jacket.Temp_Reactor ;
    Reactor.Temp IS LiqEnthalpy.Temp ;
    Reactor.Specific_Enthalpy IS LiqEnthalpy.Specific_Enthalpy_Liquid ;
    Reactor.QJacket IS Jacket.QJacket ;
    Reactor.FeedTemp IS ReactorFeedEnthalpy.Temp ;
    Reactor.FeedEnthalpy IS ReactorFeedEnthalpy.Specific_Enthalpy_Liquid ;
    Reactor.$TotalMols=0 ;
    Reactor.Feed_A=Reactor.Feed_B;
END # ReactorFlowsheet

MODEL ColumnFlowsheet INHERITS Flowsheet
  UNIT
    Column AS Column
    VLE    AS ColumnVLE
    ColumnFeedEnthalpy AS LiquidEnthalpy

  EQUATION
    Column.X IS VLE.X ;
    Column.Y IS VLE.Y ;
    Column.K IS VLE.K ;
    Column.T IS VLE.Temp ;
    Column.P IS VLE.P ;
    Column.VAPORENTHALPY IS VLE.VAPORENTHALPY ;
    Column.LIQUIDENTHALPY IS VLE.LIQUIDENTHALPY ;
    #Column.$TotalHoldup=0 ;
    #Column.TotalHoldup=43.911382-17.325 ;
    Column.BOTTOMS=0.1*(Column.M(Column.NSTAGE)-22) ;
    #Column.BOTTOMS=0.1*Column.M(Column.NSTAGE) ;
    #Column.FeedTemp IS ColumnFeedEnthalpy.Temp;
    Column.FeedEnthalpy=
      SIGMA(Column.XFEED*ColumnFeedEnthalpy.Specific_Enthalpy_Liquid)*
      SIGMA(Column.FEED);
    Column.$TotalHoldup=0 ;
END # ColumnFlowsheet

MODEL BothFlowsheet INHERITS Flowsheet

  UNIT
    Column AS Column
    VLE    AS ColumnVLE
    Reactor AS RD1_Reactions
    Jacket  AS Reactor_Jacket
    LiqEnthalpy AS LiquidEnthalpy
```

```
  MakeupEnthalpy AS LiquidEnthalpy

VARIABLE
   BFraction              AS  Fraction
   Makeup             AS  ARRAY(NC) OF MolarFlowRate

EQUATION
  WITHIN Column DO
    X IS VLE.X ;
    Y IS VLE.Y ;
    K IS VLE.K ;
    T IS VLE.Temp ;
    P IS VLE.P ;
    VAPORENTHALPY IS VLE.VAPORENTHALPY ;
    LIQUIDENTHALPY IS VLE.LIQUIDENTHALPY ;
    BOTTOMS=0.1*Column.M(Column.NSTAGE);
    $TotalHoldup=0 ;
  END

  WITHIN Reactor DO
    Temp = Jacket.Temp_Reactor+0 ;
    Temp= LiqEnthalpy.Temp+0 ;
    Specific_Enthalpy IS LiqEnthalpy.Specific_Enthalpy_Liquid ;
    QJacket IS Jacket.QJacket ;
    #FeedTemp IS ReactorFeedEnthalpy.Temp ;
    #FeedEnthalpy IS ReactorFeedEnthalpy.Specific_Enthalpy_Liquid ;
    FeedTemp IS MakeupEnthalpy.Temp;
    FeedEnthalpy =
      (Column.DISTILLOUT*Column.X(1,)*VLE.SPECIFIC_ENTHALPY_LIQUID(1,)+
      Makeup*MakeupEnthalpy.Specific_Enthalpy_Liquid);
    $TotalMols=0 ;
    Feed_B=BFraction*Feed_A;
    # Need to change 0.01 in the figure below to 1
    Feed_A=Column.DISTILLOUT*Column.X(1,A)+Makeup(A);
    Feed_B=Column.DISTILLOUT*Column.X(1,B)+Makeup(B);
    Feed_C=Column.DISTILLOUT*Column.X(1,C);
    Feed_D=Column.DISTILLOUT*Column.X(1,D);
    Makeup(C)=0 ;
    Makeup(D)=0 ;
  END

  WITHIN Column DO
   # FeedTemp IS Reactor.Temp;
    FeedEnthalpy=
      SIGMA(XFEED*Reactor.Specific_Enthalpy)*SIGMA(FEED);
    XFEED IS Reactor.X ;
    FEED(1)=0 ;
    FOR I:=2 TO FEEDSTAGE-1 DO
      FEED(I)=0  ;
    END
    FEED(FEEDSTAGE)= Reactor.FlowOut;
    FOR I:=FEEDSTAGE+1 TO NSTAGE-1 DO
      FEED(I)=0 ;
    END
```

```
      FEED(NSTAGE)=0 ;
    END

END # BothFlowsheet

SIMULATION RunReactor

  UNIT
    System As ReactorFlowSheet

  INPUT
    WITHIN System.Reactor DO
      FlowOut := 10.0 ;
      FeedTemp := 300 ;
      Feed_C := 0.0 ;
      Feed_D := 0.0 ;
    END
    WITHIN System.Jacket DO
      Flow_Water := 1 ;
    END
  INITIAL
     WITHIN System.Reactor DO
       No_Mols(A) = 22.27 ;
       No_Mols(B) = 36.81 ;
       No_Mols(C) = 0.0 ;
       No_Mols(D) = 0.0 ;
       Temp=300 ;
     END

   SCHEDULE
     CONTINUE FOR 0

END

SIMULATION RunColumn

  UNIT
    System As ColumnFlowSheet

  INPUT
    WITHIN System.Column DO
      P(1)          :=1.01325 ;
      REFLUXRATIO   :=0.5 ;
      FEED(1):=0 ;
      FOR I:=2 TO FEEDSTAGE-1 DO
        FEED(I):=0  ;
      END
      FEED(FEEDSTAGE):=(10/6+8*10/6) ;
      FOR I:=FEEDSTAGE+1 TO NSTAGE-1 DO
        FEED(I):=0 ;
      END
      FEED(NSTAGE):=0 ;
      XFEED(1) := 1 ;
      XFEED(2) := 0 ;
```

```
      XFEED(3) := 0 ;
      XFEED(4) := 0 ;
     # FEEDTEMP := 300 ;
      #TotalHoldup:=3.6586382E+01+10 ;
    END

PRESET
##############################
# Values of All Active Variables #
##############################

SYSTEM.COLUMNFEEDENTHALPY.TEMP := 3.0000000E+02 ;
SYSTEM.COLUMNFEEDENTHALPY.SPECIFIC_ENTHALPY_VAPOR(1) := 3.4460000E+02 ;
SYSTEM.COLUMNFEEDENTHALPY.SPECIFIC_ENTHALPY_VAPOR(2) := 4.0000000E+02 ;
SYSTEM.COLUMNFEEDENTHALPY.SPECIFIC_ENTHALPY_VAPOR(3) := 3.2000000E+02 ;
SYSTEM.COLUMNFEEDENTHALPY.SPECIFIC_ENTHALPY_VAPOR(4) := 3.1000000E+02 ;
SYSTEM.COLUMNFEEDENTHALPY.SPECIFIC_ENTHALPY_LIQUID(1) := -3.0655400E+04 ;
SYSTEM.COLUMNFEEDENTHALPY.SPECIFIC_ENTHALPY_LIQUID(2) := -2.5600000E+04 ;
SYSTEM.COLUMNFEEDENTHALPY.SPECIFIC_ENTHALPY_LIQUID(3) := -2.7680000E+04 ;
SYSTEM.COLUMNFEEDENTHALPY.SPECIFIC_ENTHALPY_LIQUID(4) := -3.3690000E+04 ;
SYSTEM.COLUMN.MWV(1) := 7.6000000E+01 ;
SYSTEM.COLUMN.MWV(2) := 7.6000000E+01 ;
SYSTEM.COLUMN.MWV(3) := 7.6000000E+01 ;
SYSTEM.COLUMN.MWV(4) := 7.6000000E+01 ;
SYSTEM.COLUMN.MWV(5) := 7.6000000E+01 ;
SYSTEM.COLUMN.MWV(6) := 7.6000000E+01 ;
SYSTEM.COLUMN.MWV(7) := 7.6000000E+01 ;
SYSTEM.COLUMN.MWV(8) := 7.6000000E+01 ;
SYSTEM.COLUMN.MWV(9) := 7.6000000E+01 ;
SYSTEM.COLUMN.MWV(10) := 7.6000000E+01 ;
SYSTEM.COLUMN.BOTTOMS := 2.2005890E+00 ;
SYSTEM.COLUMN.VAPORENTHALPY(1) := 9.6490429E+03 ;
SYSTEM.COLUMN.VAPORENTHALPY(2) := 9.6493013E+03 ;
SYSTEM.COLUMN.VAPORENTHALPY(3) := 9.7613871E+03 ;
SYSTEM.COLUMN.VAPORENTHALPY(4) := 9.8713496E+03 ;
SYSTEM.COLUMN.VAPORENTHALPY(5) := 9.9792593E+03 ;
SYSTEM.COLUMN.VAPORENTHALPY(6) := 1.0094036E+04 ;
SYSTEM.COLUMN.VAPORENTHALPY(7) := 1.0206730E+04 ;
SYSTEM.COLUMN.VAPORENTHALPY(8) := 1.0317422E+04 ;
SYSTEM.COLUMN.VAPORENTHALPY(9) := 1.0426188E+04 ;
SYSTEM.COLUMN.VAPORENTHALPY(10) := 1.0533100E+04 ;
SYSTEM.COLUMN.LIQMDENS(1) := 8.3600000E+02 ;
SYSTEM.COLUMN.LIQMDENS(2) := 8.3600000E+02 ;
SYSTEM.COLUMN.LIQMDENS(3) := 8.3600000E+02 ;
SYSTEM.COLUMN.LIQMDENS(4) := 8.3600000E+02 ;
SYSTEM.COLUMN.LIQMDENS(5) := 8.3600000E+02 ;
SYSTEM.COLUMN.LIQMDENS(6) := 8.3600000E+02 ;
SYSTEM.COLUMN.LIQMDENS(7) := 8.3600000E+02 ;
SYSTEM.COLUMN.LIQMDENS(8) := 8.3600000E+02 ;
SYSTEM.COLUMN.LIQMDENS(9) := 8.3600000E+02 ;
SYSTEM.COLUMN.LIQMDENS(10) := 8.3600000E+02 ;
SYSTEM.COLUMN.VAPMDENS(1) := 2.6164666E+00 ;
SYSTEM.COLUMN.VAPMDENS(2) := 2.6165852E+00 ;
SYSTEM.COLUMN.VAPMDENS(3) := 2.6684553E+00 ;
```

```
SYSTEM.COLUMN.VAPMDENS(4) := 2.7201430E+00 ;
SYSTEM.COLUMN.VAPMDENS(5) := 2.7716449E+00 ;
SYSTEM.COLUMN.VAPMDENS(6) := 2.8272809E+00 ;
SYSTEM.COLUMN.VAPMDENS(7) := 2.8827759E+00 ;
SYSTEM.COLUMN.VAPMDENS(8) := 2.9381325E+00 ;
SYSTEM.COLUMN.VAPMDENS(9) := 2.9933533E+00 ;
SYSTEM.COLUMN.VAPMDENS(10) := 3.0484406E+00 ;
SYSTEM.COLUMN.QR := 6.4072745E+06 ;
SYSTEM.COLUMN.DPSTAT(1) := 0.0000000E+00 ;
SYSTEM.COLUMN.DPSTAT(2) := 2.0759991E-02 ;
SYSTEM.COLUMN.DPSTAT(3) := 2.0726226E-02 ;
SYSTEM.COLUMN.DPSTAT(4) := 2.0689946E-02 ;
SYSTEM.COLUMN.DPSTAT(5) := 2.2496062E-02 ;
SYSTEM.COLUMN.DPSTAT(6) := 2.2483939E-02 ;
SYSTEM.COLUMN.DPSTAT(7) := 2.2471846E-02 ;
SYSTEM.COLUMN.DPSTAT(8) := 2.2459782E-02 ;
SYSTEM.COLUMN.DPSTAT(9) := 2.2447747E-02 ;
SYSTEM.COLUMN.DPSTAT(10) := 0.0000000E+00 ;
SYSTEM.COLUMN.DPTRAY(1) := 5.0229571E-05 ;
SYSTEM.COLUMN.DPTRAY(2) := 2.1986243E-02 ;
SYSTEM.COLUMN.DPTRAY(3) := 2.1952478E-02 ;
SYSTEM.COLUMN.DPTRAY(4) := 2.1916198E-02 ;
SYSTEM.COLUMN.DPTRAY(5) := 2.3722479E-02 ;
SYSTEM.COLUMN.DPTRAY(6) := 2.3710350E-02 ;
SYSTEM.COLUMN.DPTRAY(7) := 2.3698251E-02 ;
SYSTEM.COLUMN.DPTRAY(8) := 2.3686181E-02 ;
SYSTEM.COLUMN.DPTRAY(9) := 2.3674140E-02 ;
SYSTEM.COLUMN.DPTRAY(10) := 0.0000000E+00 ;
SYSTEM.COLUMN.LIQUIDENTHALPY(1) := -2.1350957E+04 ;
SYSTEM.COLUMN.LIQUIDENTHALPY(2) := -2.1350699E+04 ;
SYSTEM.COLUMN.LIQUIDENTHALPY(3) := -2.1238613E+04 ;
SYSTEM.COLUMN.LIQUIDENTHALPY(4) := -2.1128650E+04 ;
SYSTEM.COLUMN.LIQUIDENTHALPY(5) := -2.1020741E+04 ;
SYSTEM.COLUMN.LIQUIDENTHALPY(6) := -2.0905964E+04 ;
SYSTEM.COLUMN.LIQUIDENTHALPY(7) := -2.0793270E+04 ;
SYSTEM.COLUMN.LIQUIDENTHALPY(8) := -2.0682578E+04 ;
SYSTEM.COLUMN.LIQUIDENTHALPY(9) := -2.0573812E+04 ;
SYSTEM.COLUMN.LIQUIDENTHALPY(10) := -2.0466900E+04 ;
SYSTEM.COLUMN.LIQHEIGHT(1) := 0.0000000E+00 ;
SYSTEM.COLUMN.LIQHEIGHT(2) := 2.5313481E-01 ;
SYSTEM.COLUMN.LIQHEIGHT(3) := 2.5272311E-01 ;
SYSTEM.COLUMN.LIQHEIGHT(4) := 2.5228073E-01 ;
SYSTEM.COLUMN.LIQHEIGHT(5) := 2.7430342E-01 ;
SYSTEM.COLUMN.LIQHEIGHT(6) := 2.7415560E-01 ;
SYSTEM.COLUMN.LIQHEIGHT(7) := 2.7400814E-01 ;
SYSTEM.COLUMN.LIQHEIGHT(8) := 2.7386104E-01 ;
SYSTEM.COLUMN.LIQHEIGHT(9) := 2.7371429E-01 ;
SYSTEM.COLUMN.LIQHEIGHT(10) := 0.0000000E+00 ;
SYSTEM.COLUMN.HEAD(1) := 0.0000000E+00 ;
SYSTEM.COLUMN.HEAD(2) := 3.1348103E+00 ;
SYSTEM.COLUMN.HEAD(3) := 2.7231056E+00 ;
SYSTEM.COLUMN.HEAD(4) := 2.2807310E+00 ;
SYSTEM.COLUMN.HEAD(5) := 2.4303419E+01 ;
SYSTEM.COLUMN.HEAD(6) := 2.4155598E+01 ;
```

```
SYSTEM.COLUMN.HEAD(7) := 2.4008142E+01 ;
SYSTEM.COLUMN.HEAD(8) := 2.3861041E+01 ;
SYSTEM.COLUMN.HEAD(9) := 2.3714286E+01 ;
SYSTEM.COLUMN.HEAD(10) := 0.0000000E+00 ;
SYSTEM.COLUMN.K(1,1) := 1.0000000E+00 ;
SYSTEM.COLUMN.K(1,2) := 1.1186637E+00 ;
SYSTEM.COLUMN.K(1,3) := 3.6783189E-01 ;
SYSTEM.COLUMN.K(1,4) := 2.0422134E-01 ;
SYSTEM.COLUMN.K(2,1) := 1.0000000E+00 ;
SYSTEM.COLUMN.K(2,2) := 1.1186548E+00 ;
SYSTEM.COLUMN.K(2,3) := 3.6782906E-01 ;
SYSTEM.COLUMN.K(2,4) := 2.0422232E-01 ;
SYSTEM.COLUMN.K(3,1) := 1.0000000E+00 ;
SYSTEM.COLUMN.K(3,2) := 1.1147884E+00 ;
SYSTEM.COLUMN.K(3,3) := 3.6660608E-01 ;
SYSTEM.COLUMN.K(3,4) := 2.0464705E-01 ;
SYSTEM.COLUMN.K(4,1) := 1.0000000E+00 ;
SYSTEM.COLUMN.K(4,2) := 1.1110220E+00 ;
SYSTEM.COLUMN.K(4,3) := 3.6541455E-01 ;
SYSTEM.COLUMN.K(4,4) := 2.0506307E-01 ;
SYSTEM.COLUMN.K(5,1) := 1.0000000E+00 ;
SYSTEM.COLUMN.K(5,2) := 1.1073514E+00 ;
SYSTEM.COLUMN.K(5,3) := 3.6425316E-01 ;
SYSTEM.COLUMN.K(5,4) := 2.0547070E-01 ;
SYSTEM.COLUMN.K(6,1) := 1.0000000E+00 ;
SYSTEM.COLUMN.K(6,2) := 1.1034745E+00 ;
SYSTEM.COLUMN.K(6,3) := 3.6302637E-01 ;
SYSTEM.COLUMN.K(6,4) := 2.0590358E-01 ;
SYSTEM.COLUMN.K(7,1) := 1.0000000E+00 ;
SYSTEM.COLUMN.K(7,2) := 1.0996952E+00 ;
SYSTEM.COLUMN.K(7,3) := 3.6183030E-01 ;
SYSTEM.COLUMN.K(7,4) := 2.0632791E-01 ;
SYSTEM.COLUMN.K(8,1) := 1.0000000E+00 ;
SYSTEM.COLUMN.K(8,2) := 1.0960090E+00 ;
SYSTEM.COLUMN.K(8,3) := 3.6066354E-01 ;
SYSTEM.COLUMN.K(8,4) := 2.0674404E-01 ;
SYSTEM.COLUMN.K(9,1) := 1.0000000E+00 ;
SYSTEM.COLUMN.K(9,2) := 1.0924118E+00 ;
SYSTEM.COLUMN.K(9,3) := 3.5952480E-01 ;
SYSTEM.COLUMN.K(9,4) := 2.0715229E-01 ;
SYSTEM.COLUMN.K(10,1) := 1.0000000E+00 ;
SYSTEM.COLUMN.K(10,2) := 1.0888996E+00 ;
SYSTEM.COLUMN.K(10,3) := 3.5841286E-01 ;
SYSTEM.COLUMN.K(10,4) := 2.0755297E-01 ;
SYSTEM.COLUMN.L(1) := 1.2799411E+01 ;
SYSTEM.COLUMN.L(2) := 1.0700897E+01 ;
SYSTEM.COLUMN.L(3) := 8.6636325E+00 ;
SYSTEM.COLUMN.L(4) := 6.6407080E+00 ;
SYSTEM.COLUMN.L(5) := 2.3099594E+02 ;
SYSTEM.COLUMN.L(6) := 2.2889167E+02 ;
SYSTEM.COLUMN.L(7) := 2.2679899E+02 ;
SYSTEM.COLUMN.L(8) := 2.2471775E+02 ;
SYSTEM.COLUMN.L(9) := 2.2264778E+02 ;
SYSTEM.COLUMN.L(10) := 2.2005890E+00 ;
```

```
SYSTEM.COLUMN.$M(2) := 1.2242624E-18 ;
SYSTEM.COLUMN.$M(3) := -4.6540611E-19 ;
SYSTEM.COLUMN.$M(4) := -5.9937802E-19 ;
SYSTEM.COLUMN.$M(5) := 1.1802807E-18 ;
SYSTEM.COLUMN.$M(6) := 1.9926115E-19 ;
SYSTEM.COLUMN.$M(7) := -3.8043140E-19 ;
SYSTEM.COLUMN.$M(8) := -8.5739929E-19 ;
SYSTEM.COLUMN.$M(9) := -4.9483541E-19 ;
SYSTEM.COLUMN.M(1) := 0.0000000E+00 ;
SYSTEM.COLUMN.M(2) := 2.9237071E+00 ;
SYSTEM.COLUMN.M(3) := 2.9189519E+00 ;
SYSTEM.COLUMN.M(4) := 2.9138424E+00 ;
SYSTEM.COLUMN.M(5) := 3.1682045E+00 ;
SYSTEM.COLUMN.M(6) := 3.1664972E+00 ;
SYSTEM.COLUMN.M(7) := 3.1647940E+00 ;
SYSTEM.COLUMN.M(8) := 3.1630950E+00 ;
SYSTEM.COLUMN.M(9) := 3.1614000E+00 ;
SYSTEM.COLUMN.M(10) := 2.2005890E+01 ;
SYSTEM.COLUMN.LIQDENS(1) := 1.1000000E+01 ;
SYSTEM.COLUMN.LIQDENS(2) := 1.1000000E+01 ;
SYSTEM.COLUMN.LIQDENS(3) := 1.1000000E+01 ;
SYSTEM.COLUMN.LIQDENS(4) := 1.1000000E+01 ;
SYSTEM.COLUMN.LIQDENS(5) := 1.1000000E+01 ;
SYSTEM.COLUMN.LIQDENS(6) := 1.1000000E+01 ;
SYSTEM.COLUMN.LIQDENS(7) := 1.1000000E+01 ;
SYSTEM.COLUMN.LIQDENS(8) := 1.1000000E+01 ;
SYSTEM.COLUMN.LIQDENS(9) := 1.1000000E+01 ;
SYSTEM.COLUMN.LIQDENS(10) := 1.1000000E+01 ;
SYSTEM.COLUMN.VAPDENS(1) := 3.4427192E-02 ;
SYSTEM.COLUMN.VAPDENS(2) := 3.4428753E-02 ;
SYSTEM.COLUMN.VAPDENS(3) := 3.5111254E-02 ;
SYSTEM.COLUMN.VAPDENS(4) := 3.5791355E-02 ;
SYSTEM.COLUMN.VAPDENS(5) := 3.6469012E-02 ;
SYSTEM.COLUMN.VAPDENS(6) := 3.7201064E-02 ;
SYSTEM.COLUMN.VAPDENS(7) := 3.7931261E-02 ;
SYSTEM.COLUMN.VAPDENS(8) := 3.8659638E-02 ;
SYSTEM.COLUMN.VAPDENS(9) := 3.9386227E-02 ;
SYSTEM.COLUMN.VAPDENS(10) := 4.0111061E-02 ;
SYSTEM.COLUMN.P(2) := 1.0133002E+00 ;
SYSTEM.COLUMN.P(3) := 1.0352865E+00 ;
SYSTEM.COLUMN.P(4) := 1.0572390E+00 ;
SYSTEM.COLUMN.P(5) := 1.0791551E+00 ;
SYSTEM.COLUMN.P(6) := 1.1028776E+00 ;
SYSTEM.COLUMN.P(7) := 1.1265880E+00 ;
SYSTEM.COLUMN.P(8) := 1.1502862E+00 ;
SYSTEM.COLUMN.P(9) := 1.1739724E+00 ;
SYSTEM.COLUMN.P(10) := 1.1976465E+00 ;
SYSTEM.COLUMN.DISTILLOUT := 1.2799411E+01 ;
SYSTEM.COLUMN.VOLUMEHOLDUP(1) := 0.0000000E+00 ;
SYSTEM.COLUMN.VOLUMEHOLDUP(2) := 2.6579155E-01 ;
SYSTEM.COLUMN.VOLUMEHOLDUP(3) := 2.6535926E-01 ;
SYSTEM.COLUMN.VOLUMEHOLDUP(4) := 2.6489477E-01 ;
SYSTEM.COLUMN.VOLUMEHOLDUP(5) := 2.8801859E-01 ;
SYSTEM.COLUMN.VOLUMEHOLDUP(6) := 2.8786338E-01 ;
```

```
SYSTEM.COLUMN.VOLUMEHOLDUP(7) := 2.8770855E-01 ;
SYSTEM.COLUMN.VOLUMEHOLDUP(8) := 2.8755409E-01 ;
SYSTEM.COLUMN.VOLUMEHOLDUP(9) := 2.8740000E-01 ;
SYSTEM.COLUMN.VOLUMEHOLDUP(10) := 2.0005354E+00 ;
SYSTEM.COLUMN.T(1) := 3.5400141E+02 ;
SYSTEM.COLUMN.T(2) := 3.5400291E+02 ;
SYSTEM.COLUMN.T(3) := 3.5465344E+02 ;
SYSTEM.COLUMN.T(4) := 3.5529164E+02 ;
SYSTEM.COLUMN.T(5) := 3.5591793E+02 ;
SYSTEM.COLUMN.T(6) := 3.5658408E+02 ;
SYSTEM.COLUMN.T(7) := 3.5723813E+02 ;
SYSTEM.COLUMN.T(8) := 3.5788057E+02 ;
SYSTEM.COLUMN.T(9) := 3.5851183E+02 ;
SYSTEM.COLUMN.T(10) := 3.5913233E+02 ;
SYSTEM.COLUMN.V(1) := 0.0000000E+00 ;
SYSTEM.COLUMN.V(2) := 2.5598822E+01 ;
SYSTEM.COLUMN.V(3) := 2.3500308E+01 ;
SYSTEM.COLUMN.V(4) := 2.1463044E+01 ;
SYSTEM.COLUMN.V(5) := 1.9440119E+01 ;
SYSTEM.COLUMN.V(6) := 2.2879535E+02 ;
SYSTEM.COLUMN.V(7) := 2.2669108E+02 ;
SYSTEM.COLUMN.V(8) := 2.2459840E+02 ;
SYSTEM.COLUMN.V(9) := 2.2251716E+02 ;
SYSTEM.COLUMN.V(10) := 2.2044719E+02 ;
SYSTEM.COLUMN.DPDRY(1) := 0.0000000E+00 ;
SYSTEM.COLUMN.DPDRY(2) := 1.8679046E-09 ;
SYSTEM.COLUMN.DPDRY(3) := 1.5277952E-09 ;
SYSTEM.COLUMN.DPDRY(4) := 1.2295567E-09 ;
SYSTEM.COLUMN.DPDRY(5) := 1.6714726E-07 ;
SYSTEM.COLUMN.DPDRY(6) := 1.6085789E-07 ;
SYSTEM.COLUMN.DPDRY(7) := 1.5486202E-07 ;
SYSTEM.COLUMN.DPDRY(8) := 1.4914136E-07 ;
SYSTEM.COLUMN.DPDRY(9) := 1.4367911E-07 ;
SYSTEM.COLUMN.DPDRY(10) := 0.0000000E+00 ;
SYSTEM.COLUMN.X(1,1) := 1.0000000E+00 ;
SYSTEM.COLUMN.X(1,2) := -1.4211639E-30 ;
SYSTEM.COLUMN.X(1,3) := -7.7119210E-33 ;
SYSTEM.COLUMN.X(1,4) := -6.9770011E-33 ;
SYSTEM.COLUMN.X(2,1) := 1.0000000E+00 ;
SYSTEM.COLUMN.$X(2,1) := -1.1837221E-27 ;
SYSTEM.COLUMN.X(2,2) := -1.2704223E-30 ;
SYSTEM.COLUMN.$X(2,2) := 5.7989273E-33 ;
SYSTEM.COLUMN.X(2,3) := -2.0966046E-32 ;
SYSTEM.COLUMN.$X(2,3) := -2.6702840E-35 ;
SYSTEM.COLUMN.X(2,4) := -6.1284288E-32 ;
SYSTEM.COLUMN.$X(2,4) := -1.6449487E-34 ;
SYSTEM.COLUMN.X(3,1) := 1.0000000E+00 ;
SYSTEM.COLUMN.$X(3,1) := 1.2030481E-26 ;
SYSTEM.COLUMN.X(3,2) := -1.2080724E-30 ;
SYSTEM.COLUMN.$X(3,2) := 5.5717889E-33 ;
SYSTEM.COLUMN.X(3,3) := -3.8197066E-32 ;
SYSTEM.COLUMN.$X(3,3) := -1.0199658E-34 ;
SYSTEM.COLUMN.X(3,4) := -8.1682874E-32 ;
SYSTEM.COLUMN.$X(3,4) := -2.0340822E-34 ;
```

```
SYSTEM.COLUMN.X(4,1) := 1.0000000E+00 ;
SYSTEM.COLUMN.$X(4,1) := -1.3084953E-26 ;
SYSTEM.COLUMN.X(4,2) := -1.0440124E-30 ;
SYSTEM.COLUMN.$X(4,2) := 5.5665390E-33 ;
SYSTEM.COLUMN.X(4,3) := -6.9179846E-32 ;
SYSTEM.COLUMN.$X(4,3) := -1.7839640E-34 ;
SYSTEM.COLUMN.X(4,4) := -6.5639146E-32 ;
SYSTEM.COLUMN.$X(4,4) := -1.6350563E-34 ;
SYSTEM.COLUMN.X(5,1) := 1.0000000E+00 ;
SYSTEM.COLUMN.$X(5,1) := -1.9839325E-24 ;
SYSTEM.COLUMN.X(5,2) := -9.9188031E-31 ;
SYSTEM.COLUMN.$X(5,2) := 5.6461486E-33 ;
SYSTEM.COLUMN.X(5,3) := -9.4816353E-32 ;
SYSTEM.COLUMN.$X(5,3) := -2.4050174E-34 ;
SYSTEM.COLUMN.X(5,4) := -4.3865365E-33 ;
SYSTEM.COLUMN.$X(5,4) := -1.1121583E-35 ;
SYSTEM.COLUMN.X(6,1) := 1.0000000E+00 ;
SYSTEM.COLUMN.$X(6,1) := 7.4938318E-24 ;
SYSTEM.COLUMN.X(6,2) := -8.9965879E-31 ;
SYSTEM.COLUMN.$X(6,2) := 4.9062142E-33 ;
SYSTEM.COLUMN.X(6,3) := -2.6625428E-31 ;
SYSTEM.COLUMN.$X(6,3) := -6.7514852E-34 ;
SYSTEM.COLUMN.X(6,4) := -1.2628664E-32 ;
SYSTEM.COLUMN.$X(6,4) := -3.2429701E-35 ;
SYSTEM.COLUMN.X(7,1) := 1.0000000E+00 ;
SYSTEM.COLUMN.$X(7,1) := -2.4396441E-24 ;
SYSTEM.COLUMN.X(7,2) := -8.2074687E-31 ;
SYSTEM.COLUMN.$X(7,2) := 4.4714835E-33 ;
SYSTEM.COLUMN.X(7,3) := -1.0224736E-30 ;
SYSTEM.COLUMN.$X(7,3) := -1.8627965E-33 ;
SYSTEM.COLUMN.X(7,4) := -5.2858763E-32 ;
SYSTEM.COLUMN.$X(7,4) := -1.3644363E-34 ;
SYSTEM.COLUMN.X(8,1) := 1.0000000E+00 ;
SYSTEM.COLUMN.$X(8,1) := 1.9593269E-24 ;
SYSTEM.COLUMN.X(8,2) := -7.5079031E-31 ;
SYSTEM.COLUMN.$X(8,2) := 4.0864429E-33 ;
SYSTEM.COLUMN.X(8,3) := -2.3610164E-30 ;
SYSTEM.COLUMN.$X(8,3) := -4.3205815E-33 ;
SYSTEM.COLUMN.X(8,4) := -2.4917836E-31 ;
SYSTEM.COLUMN.$X(8,4) := -6.4403095E-34 ;
SYSTEM.COLUMN.X(9,1) := 1.0000000E+00 ;
SYSTEM.COLUMN.$X(9,1) := -4.8662107E-23 ;
SYSTEM.COLUMN.X(9,2) := -6.8857471E-31 ;
SYSTEM.COLUMN.$X(9,2) := 3.7443424E-33 ;
SYSTEM.COLUMN.X(9,3) := -6.1456693E-30 ;
SYSTEM.COLUMN.$X(9,3) := -1.1199828E-32 ;
SYSTEM.COLUMN.X(9,4) := -1.2057443E-30 ;
SYSTEM.COLUMN.$X(9,4) := -3.1172520E-33 ;
SYSTEM.COLUMN.X(10,1) := 1.0000000E+00 ;
SYSTEM.COLUMN.$X(10,1) := 6.2732799E-24 ;
SYSTEM.COLUMN.X(10,2) := -6.3306961E-31 ;
SYSTEM.COLUMN.$X(10,2) := 3.4394567E-33 ;
SYSTEM.COLUMN.X(10,3) := -1.6841651E-29 ;
SYSTEM.COLUMN.$X(10,3) := -3.0684627E-32 ;
```

```
SYSTEM.COLUMN.X(10,4) := -5.7768554E-30 ;
SYSTEM.COLUMN.$X(10,4) := -1.5124539E-32 ;
SYSTEM.COLUMN.Y(1,1) := 1.0000000E+00 ;
SYSTEM.COLUMN.Y(1,2) := -1.5898045E-30 ;
SYSTEM.COLUMN.Y(1,3) := -2.8366905E-33 ;
SYSTEM.COLUMN.Y(1,4) := -1.4248525E-33 ;
SYSTEM.COLUMN.Y(2,1) := 1.0000000E+00 ;
SYSTEM.COLUMN.Y(2,2) := -1.4211639E-30 ;
SYSTEM.COLUMN.Y(2,3) := -7.7119210E-33 ;
SYSTEM.COLUMN.Y(2,4) := -6.9770011E-33 ;
SYSTEM.COLUMN.Y(3,1) := 1.0000000E+00 ;
SYSTEM.COLUMN.Y(3,2) := -1.3467451E-30 ;
SYSTEM.COLUMN.Y(3,3) := -1.4003277E-32 ;
SYSTEM.COLUMN.Y(3,4) := -1.6716159E-32 ;
SYSTEM.COLUMN.Y(4,1) := 1.0000000E+00 ;
SYSTEM.COLUMN.Y(4,2) := -1.1599208E-30 ;
SYSTEM.COLUMN.Y(4,3) := -2.5279322E-32 ;
SYSTEM.COLUMN.Y(4,4) := -1.3460165E-32 ;
SYSTEM.COLUMN.Y(5,1) := 1.0000000E+00 ;
SYSTEM.COLUMN.Y(5,2) := -1.0983600E-30 ;
SYSTEM.COLUMN.Y(5,3) := -3.4537156E-32 ;
SYSTEM.COLUMN.Y(5,4) := -9.0130471E-34 ;
SYSTEM.COLUMN.Y(6,1) := 1.0000000E+00 ;
SYSTEM.COLUMN.Y(6,2) := -9.9275052E-31 ;
SYSTEM.COLUMN.Y(6,3) := -9.6657325E-32 ;
SYSTEM.COLUMN.Y(6,4) := -2.6002870E-33 ;
SYSTEM.COLUMN.Y(7,1) := 1.0000000E+00 ;
SYSTEM.COLUMN.Y(7,2) := -9.0257137E-31 ;
SYSTEM.COLUMN.Y(7,3) := -3.6996193E-31 ;
SYSTEM.COLUMN.Y(7,4) := -1.0906238E-32 ;
SYSTEM.COLUMN.Y(8,1) := 1.0000000E+00 ;
SYSTEM.COLUMN.Y(8,2) := -8.2287294E-31 ;
SYSTEM.COLUMN.Y(8,3) := -8.5153253E-31 ;
SYSTEM.COLUMN.Y(8,4) := -5.1516140E-32 ;
SYSTEM.COLUMN.Y(9,1) := 1.0000000E+00 ;
SYSTEM.COLUMN.Y(9,2) := -7.5220712E-31 ;
SYSTEM.COLUMN.Y(9,3) := -2.2095205E-30 ;
SYSTEM.COLUMN.Y(9,4) := -2.4977270E-31 ;
SYSTEM.COLUMN.Y(10,1) := 1.0000000E+00 ;
SYSTEM.COLUMN.Y(10,2) := -6.8934926E-31 ;
SYSTEM.COLUMN.Y(10,3) := -6.0362642E-30 ;
SYSTEM.COLUMN.Y(10,4) := -1.1990035E-30 ;
SYSTEM.COLUMN.FEEDENTHALPY := -4.5983100E+05 ;
SYSTEM.COLUMN.MWL(1) := 7.6000000E+01 ;
SYSTEM.COLUMN.MWL(2) := 7.6000000E+01 ;
SYSTEM.COLUMN.MWL(3) := 7.6000000E+01 ;
SYSTEM.COLUMN.MWL(4) := 7.6000000E+01 ;
SYSTEM.COLUMN.MWL(5) := 7.6000000E+01 ;
SYSTEM.COLUMN.MWL(6) := 7.6000000E+01 ;
SYSTEM.COLUMN.MWL(7) := 7.6000000E+01 ;
SYSTEM.COLUMN.MWL(8) := 7.6000000E+01 ;
SYSTEM.COLUMN.MWL(9) := 7.6000000E+01 ;
SYSTEM.COLUMN.MWL(10) := 7.6000000E+01 ;
SYSTEM.COLUMN.TOTALHOLDUP := 4.6586382E+01 ;
```

```
SYSTEM.COLUMN.$TOTALHOLDUP := 0.0000000E+00 ;
SYSTEM.COLUMN.QC := 7.9357010E+05 ;
SYSTEM.VLE.VAPORENTHALPY(1) := 9.6490429E+03 ;
SYSTEM.VLE.VAPORENTHALPY(2) := 9.6493013E+03 ;
SYSTEM.VLE.VAPORENTHALPY(3) := 9.7613871E+03 ;
SYSTEM.VLE.VAPORENTHALPY(4) := 9.8713496E+03 ;
SYSTEM.VLE.VAPORENTHALPY(5) := 9.9792593E+03 ;
SYSTEM.VLE.VAPORENTHALPY(6) := 1.0094036E+04 ;
SYSTEM.VLE.VAPORENTHALPY(7) := 1.0206730E+04 ;
SYSTEM.VLE.VAPORENTHALPY(8) := 1.0317422E+04 ;
SYSTEM.VLE.VAPORENTHALPY(9) := 1.0426188E+04 ;
SYSTEM.VLE.VAPORENTHALPY(10) := 1.0533100E+04 ;
SYSTEM.VLE.TEMP(1) := 3.5400141E+02 ;
SYSTEM.VLE.TEMP(2) := 3.5400291E+02 ;
SYSTEM.VLE.TEMP(3) := 3.5465344E+02 ;
SYSTEM.VLE.TEMP(4) := 3.5529164E+02 ;
SYSTEM.VLE.TEMP(5) := 3.5591793E+02 ;
SYSTEM.VLE.TEMP(6) := 3.5658408E+02 ;
SYSTEM.VLE.TEMP(7) := 3.5723813E+02 ;
SYSTEM.VLE.TEMP(8) := 3.5788057E+02 ;
SYSTEM.VLE.TEMP(9) := 3.5851183E+02 ;
SYSTEM.VLE.TEMP(10) := 3.5913233E+02 ;
SYSTEM.VLE.LIQUIDENTHALPY(1) := -2.1350957E+04 ;
SYSTEM.VLE.LIQUIDENTHALPY(2) := -2.1350699E+04 ;
SYSTEM.VLE.LIQUIDENTHALPY(3) := -2.1238613E+04 ;
SYSTEM.VLE.LIQUIDENTHALPY(4) := -2.1128650E+04 ;
SYSTEM.VLE.LIQUIDENTHALPY(5) := -2.1020741E+04 ;
SYSTEM.VLE.LIQUIDENTHALPY(6) := -2.0905964E+04 ;
SYSTEM.VLE.LIQUIDENTHALPY(7) := -2.0793270E+04 ;
SYSTEM.VLE.LIQUIDENTHALPY(8) := -2.0682578E+04 ;
SYSTEM.VLE.LIQUIDENTHALPY(9) := -2.0573812E+04 ;
SYSTEM.VLE.LIQUIDENTHALPY(10) := -2.0466900E+04 ;
SYSTEM.VLE.K(1,1) := 1.0000000E+00 ;
SYSTEM.VLE.K(1,2) := 1.1186637E+00 ;
SYSTEM.VLE.K(1,3) := 3.6783189E-01 ;
SYSTEM.VLE.K(1,4) := 2.0422134E-01 ;
SYSTEM.VLE.K(2,1) := 1.0000000E+00 ;
SYSTEM.VLE.K(2,2) := 1.1186548E+00 ;
SYSTEM.VLE.K(2,3) := 3.6782906E-01 ;
SYSTEM.VLE.K(2,4) := 2.0422232E-01 ;
SYSTEM.VLE.K(3,1) := 1.0000000E+00 ;
SYSTEM.VLE.K(3,2) := 1.1147884E+00 ;
SYSTEM.VLE.K(3,3) := 3.6660608E-01 ;
SYSTEM.VLE.K(3,4) := 2.0464705E-01 ;
SYSTEM.VLE.K(4,1) := 1.0000000E+00 ;
SYSTEM.VLE.K(4,2) := 1.1110220E+00 ;
SYSTEM.VLE.K(4,3) := 3.6541455E-01 ;
SYSTEM.VLE.K(4,4) := 2.0506307E-01 ;
SYSTEM.VLE.K(5,1) := 1.0000000E+00 ;
SYSTEM.VLE.K(5,2) := 1.1073514E+00 ;
SYSTEM.VLE.K(5,3) := 3.6425316E-01 ;
SYSTEM.VLE.K(5,4) := 2.0547070E-01 ;
SYSTEM.VLE.K(6,1) := 1.0000000E+00 ;
SYSTEM.VLE.K(6,2) := 1.1034745E+00 ;
```

```
SYSTEM.VLE.K(6,3) := 3.6302637E-01 ;
SYSTEM.VLE.K(6,4) := 2.0590358E-01 ;
SYSTEM.VLE.K(7,1) := 1.0000000E+00 ;
SYSTEM.VLE.K(7,2) := 1.0996952E+00 ;
SYSTEM.VLE.K(7,3) := 3.6183030E-01 ;
SYSTEM.VLE.K(7,4) := 2.0632791E-01 ;
SYSTEM.VLE.K(8,1) := 1.0000000E+00 ;
SYSTEM.VLE.K(8,2) := 1.0960090E+00 ;
SYSTEM.VLE.K(8,3) := 3.6066354E-01 ;
SYSTEM.VLE.K(8,4) := 2.0674404E-01 ;
SYSTEM.VLE.K(9,1) := 1.0000000E+00 ;
SYSTEM.VLE.K(9,2) := 1.0924118E+00 ;
SYSTEM.VLE.K(9,3) := 3.5952480E-01 ;
SYSTEM.VLE.K(9,4) := 2.0715229E-01 ;
SYSTEM.VLE.K(10,1) := 1.0000000E+00 ;
SYSTEM.VLE.K(10,2) := 1.0888996E+00 ;
SYSTEM.VLE.K(10,3) := 3.5841286E-01 ;
SYSTEM.VLE.K(10,4) := 2.0755297E-01 ;
SYSTEM.VLE.VAPORPRESSURE(1,1) := 1.0132500E+00 ;
SYSTEM.VLE.VAPORPRESSURE(1,2) := 1.1334860E+00 ;
SYSTEM.VLE.VAPORPRESSURE(1,3) := 3.7270566E-01 ;
SYSTEM.VLE.VAPORPRESSURE(1,4) := 2.0692728E-01 ;
SYSTEM.VLE.VAPORPRESSURE(2,1) := 1.0133002E+00 ;
SYSTEM.VLE.VAPORPRESSURE(2,2) := 1.1335331E+00 ;
SYSTEM.VLE.VAPORPRESSURE(2,3) := 3.7272127E-01 ;
SYSTEM.VLE.VAPORPRESSURE(2,4) := 2.0693853E-01 ;
SYSTEM.VLE.VAPORPRESSURE(3,1) := 1.0352865E+00 ;
SYSTEM.VLE.VAPORPRESSURE(3,2) := 1.1541254E+00 ;
SYSTEM.VLE.VAPORPRESSURE(3,3) := 3.7954232E-01 ;
SYSTEM.VLE.VAPORPRESSURE(3,4) := 2.1186832E-01 ;
SYSTEM.VLE.VAPORPRESSURE(4,1) := 1.0572390E+00 ;
SYSTEM.VLE.VAPORPRESSURE(4,2) := 1.1746158E+00 ;
SYSTEM.VLE.VAPORPRESSURE(4,3) := 3.8633050E-01 ;
SYSTEM.VLE.VAPORPRESSURE(4,4) := 2.1680067E-01 ;
SYSTEM.VLE.VAPORPRESSURE(5,1) := 1.0791551E+00 ;
SYSTEM.VLE.VAPORPRESSURE(5,2) := 1.1950039E+00 ;
SYSTEM.VLE.VAPORPRESSURE(5,3) := 3.9308568E-01 ;
SYSTEM.VLE.VAPORPRESSURE(5,4) := 2.2173476E-01 ;
SYSTEM.VLE.VAPORPRESSURE(6,1) := 1.1028776E+00 ;
SYSTEM.VLE.VAPORPRESSURE(6,2) := 1.2169973E+00 ;
SYSTEM.VLE.VAPORPRESSURE(6,3) := 4.0037366E-01 ;
SYSTEM.VLE.VAPORPRESSURE(6,4) := 2.2708645E-01 ;
SYSTEM.VLE.VAPORPRESSURE(7,1) := 1.1265880E+00 ;
SYSTEM.VLE.VAPORPRESSURE(7,2) := 1.2389034E+00 ;
SYSTEM.VLE.VAPORPRESSURE(7,3) := 4.0763366E-01 ;
SYSTEM.VLE.VAPORPRESSURE(7,4) := 2.3244654E-01 ;
SYSTEM.VLE.VAPORPRESSURE(8,1) := 1.1502862E+00 ;
SYSTEM.VLE.VAPORPRESSURE(8,2) := 1.2607240E+00 ;
SYSTEM.VLE.VAPORPRESSURE(8,3) := 4.1486630E-01 ;
SYSTEM.VLE.VAPORPRESSURE(8,4) := 2.3781482E-01 ;
SYSTEM.VLE.VAPORPRESSURE(9,1) := 1.1739724E+00 ;
SYSTEM.VLE.VAPORPRESSURE(9,2) := 1.2824613E+00 ;
SYSTEM.VLE.VAPORPRESSURE(9,3) := 4.2207220E-01 ;
SYSTEM.VLE.VAPORPRESSURE(9,4) := 2.4319108E-01 ;
```

```
SYSTEM.VLE.VAPORPRESSURE(10,1) := 1.1976465E+00 ;
SYSTEM.VLE.VAPORPRESSURE(10,2) := 1.3041169E+00 ;
SYSTEM.VLE.VAPORPRESSURE(10,3) := 4.2925192E-01 ;
SYSTEM.VLE.VAPORPRESSURE(10,4) := 2.4857510E-01 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_VAPOR(1,1) := 9.6490429E+03 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_VAPOR(1,2) := 1.1200282E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_VAPOR(1,3) := 8.9602256E+03 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_VAPOR(1,4) := 8.6802185E+03 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_VAPOR(2,1) := 9.6493013E+03 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_VAPOR(2,2) := 1.1200582E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_VAPOR(2,3) := 8.9604655E+03 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_VAPOR(2,4) := 8.6804509E+03 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_VAPOR(3,1) := 9.7613871E+03 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_VAPOR(3,2) := 1.1330687E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_VAPOR(3,3) := 9.0645499E+03 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_VAPOR(3,4) := 8.7812827E+03 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_VAPOR(4,1) := 9.8713496E+03 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_VAPOR(4,2) := 1.1458328E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_VAPOR(4,3) := 9.1666624E+03 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_VAPOR(4,4) := 8.8802042E+03 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_VAPOR(5,1) := 9.9792593E+03 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_VAPOR(5,2) := 1.1583586E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_VAPOR(5,3) := 9.2668688E+03 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_VAPOR(5,4) := 8.9772791E+03 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_VAPOR(6,1) := 1.0094036E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_VAPOR(6,2) := 1.1716815E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_VAPOR(6,3) := 9.3734520E+03 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_VAPOR(6,4) := 9.0805316E+03 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_VAPOR(7,1) := 1.0206730E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_VAPOR(7,2) := 1.1847626E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_VAPOR(7,3) := 9.4781009E+03 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_VAPOR(7,4) := 9.1819102E+03 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_VAPOR(8,1) := 1.0317422E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_VAPOR(8,2) := 1.1976113E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_VAPOR(8,3) := 9.5808907E+03 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_VAPOR(8,4) := 9.2814879E+03 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_VAPOR(9,1) := 1.0426188E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_VAPOR(9,2) := 1.2102365E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_VAPOR(9,3) := 9.6818924E+03 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_VAPOR(9,4) := 9.3793332E+03 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_VAPOR(10,1) := 1.0533100E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_VAPOR(10,2) := 1.2226466E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_VAPOR(10,3) := 9.7811726E+03 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_VAPOR(10,4) := 9.4755110E+03 ;
SYSTEM.VLE.P(1) := 1.0132500E+00 ;
SYSTEM.VLE.P(2) := 1.0133002E+00 ;
SYSTEM.VLE.P(3) := 1.0352865E+00 ;
SYSTEM.VLE.P(4) := 1.0572390E+00 ;
SYSTEM.VLE.P(5) := 1.0791551E+00 ;
SYSTEM.VLE.P(6) := 1.1028776E+00 ;
SYSTEM.VLE.P(7) := 1.1265880E+00 ;
SYSTEM.VLE.P(8) := 1.1502862E+00 ;
SYSTEM.VLE.P(9) := 1.1739724E+00 ;
SYSTEM.VLE.P(10) := 1.1976465E+00 ;
```

```
SYSTEM.VLE.X(1,1) := 1.0000000E+00 ;
SYSTEM.VLE.X(1,2) := -1.4211639E-30 ;
SYSTEM.VLE.X(1,3) := -7.7119210E-33 ;
SYSTEM.VLE.X(1,4) := -6.9770011E-33 ;
SYSTEM.VLE.X(2,1) := 1.0000000E+00 ;
SYSTEM.VLE.X(2,2) := -1.2704223E-30 ;
SYSTEM.VLE.X(2,3) := -2.0966046E-32 ;
SYSTEM.VLE.X(2,4) := -6.1284288E-32 ;
SYSTEM.VLE.X(3,1) := 1.0000000E+00 ;
SYSTEM.VLE.X(3,2) := -1.2080724E-30 ;
SYSTEM.VLE.X(3,3) := -3.8197066E-32 ;
SYSTEM.VLE.X(3,4) := -8.1682874E-32 ;
SYSTEM.VLE.X(4,1) := 1.0000000E+00 ;
SYSTEM.VLE.X(4,2) := -1.0440124E-30 ;
SYSTEM.VLE.X(4,3) := -6.9179846E-32 ;
SYSTEM.VLE.X(4,4) := -6.5639146E-32 ;
SYSTEM.VLE.X(5,1) := 1.0000000E+00 ;
SYSTEM.VLE.X(5,2) := -9.9188031E-31 ;
SYSTEM.VLE.X(5,3) := -9.4816353E-32 ;
SYSTEM.VLE.X(5,4) := -4.3865365E-33 ;
SYSTEM.VLE.X(6,1) := 1.0000000E+00 ;
SYSTEM.VLE.X(6,2) := -8.9965879E-31 ;
SYSTEM.VLE.X(6,3) := -2.6625428E-31 ;
SYSTEM.VLE.X(6,4) := -1.2628664E-32 ;
SYSTEM.VLE.X(7,1) := 1.0000000E+00 ;
SYSTEM.VLE.X(7,2) := -8.2074687E-31 ;
SYSTEM.VLE.X(7,3) := -1.0224736E-30 ;
SYSTEM.VLE.X(7,4) := -5.2858763E-32 ;
SYSTEM.VLE.X(8,1) := 1.0000000E+00 ;
SYSTEM.VLE.X(8,2) := -7.5079031E-31 ;
SYSTEM.VLE.X(8,3) := -2.3610164E-30 ;
SYSTEM.VLE.X(8,4) := -2.4917836E-31 ;
SYSTEM.VLE.X(9,1) := 1.0000000E+00 ;
SYSTEM.VLE.X(9,2) := -6.8857471E-31 ;
SYSTEM.VLE.X(9,3) := -6.1456693E-30 ;
SYSTEM.VLE.X(9,4) := -1.2057443E-30 ;
SYSTEM.VLE.X(10,1) := 1.0000000E+00 ;
SYSTEM.VLE.X(10,2) := -6.3306961E-31 ;
SYSTEM.VLE.X(10,3) := -1.6841651E-29 ;
SYSTEM.VLE.X(10,4) := -5.7768554E-30 ;
SYSTEM.VLE.Y(1,1) := 1.0000000E+00 ;
SYSTEM.VLE.Y(1,2) := -1.5898045E-30 ;
SYSTEM.VLE.Y(1,3) := -2.8366905E-33 ;
SYSTEM.VLE.Y(1,4) := -1.4248525E-33 ;
SYSTEM.VLE.Y(2,1) := 1.0000000E+00 ;
SYSTEM.VLE.Y(2,2) := -1.4211639E-30 ;
SYSTEM.VLE.Y(2,3) := -7.7119210E-33 ;
SYSTEM.VLE.Y(2,4) := -6.9770011E-33 ;
SYSTEM.VLE.Y(3,1) := 1.0000000E+00 ;
SYSTEM.VLE.Y(3,2) := -1.3467451E-30 ;
SYSTEM.VLE.Y(3,3) := -1.4003277E-32 ;
SYSTEM.VLE.Y(3,4) := -1.6716159E-32 ;
SYSTEM.VLE.Y(4,1) := 1.0000000E+00 ;
SYSTEM.VLE.Y(4,2) := -1.1599208E-30 ;
```

```
SYSTEM.VLE.Y(4,3) := -2.5279322E-32 ;
SYSTEM.VLE.Y(4,4) := -1.3460165E-32 ;
SYSTEM.VLE.Y(5,1) := 1.0000000E+00 ;
SYSTEM.VLE.Y(5,2) := -1.0983600E-30 ;
SYSTEM.VLE.Y(5,3) := -3.4537156E-32 ;
SYSTEM.VLE.Y(5,4) := -9.0130471E-34 ;
SYSTEM.VLE.Y(6,1) := 1.0000000E+00 ;
SYSTEM.VLE.Y(6,2) := -9.9275052E-31 ;
SYSTEM.VLE.Y(6,3) := -9.6657325E-32 ;
SYSTEM.VLE.Y(6,4) := -2.6002870E-33 ;
SYSTEM.VLE.Y(7,1) := 1.0000000E+00 ;
SYSTEM.VLE.Y(7,2) := -9.0257137E-31 ;
SYSTEM.VLE.Y(7,3) := -3.6996193E-31 ;
SYSTEM.VLE.Y(7,4) := -1.0906238E-32 ;
SYSTEM.VLE.Y(8,1) := 1.0000000E+00 ;
SYSTEM.VLE.Y(8,2) := -8.2287294E-31 ;
SYSTEM.VLE.Y(8,3) := -8.5153253E-31 ;
SYSTEM.VLE.Y(8,4) := -5.1516140E-32 ;
SYSTEM.VLE.Y(9,1) := 1.0000000E+00 ;
SYSTEM.VLE.Y(9,2) := -7.5220712E-31 ;
SYSTEM.VLE.Y(9,3) := -2.2095205E-30 ;
SYSTEM.VLE.Y(9,4) := -2.4977270E-31 ;
SYSTEM.VLE.Y(10,1) := 1.0000000E+00 ;
SYSTEM.VLE.Y(10,2) := -6.8934926E-31 ;
SYSTEM.VLE.Y(10,3) := -6.0362642E-30 ;
SYSTEM.VLE.Y(10,4) := -1.1990035E-30 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_LIQUID(1,1) := -2.1350957E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_LIQUID(1,2) := -1.4799718E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_LIQUID(1,3) := -1.9039774E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_LIQUID(1,4) := -2.5319781E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_LIQUID(2,1) := -2.1350699E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_LIQUID(2,2) := -1.4799418E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_LIQUID(2,3) := -1.9039535E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_LIQUID(2,4) := -2.5319549E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_LIQUID(3,1) := -2.1238613E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_LIQUID(3,2) := -1.4669313E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_LIQUID(3,3) := -1.8935450E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_LIQUID(3,4) := -2.5218717E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_LIQUID(4,1) := -2.1128650E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_LIQUID(4,2) := -1.4541672E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_LIQUID(4,3) := -1.8833338E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_LIQUID(4,4) := -2.5119796E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_LIQUID(5,1) := -2.1020741E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_LIQUID(5,2) := -1.4416414E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_LIQUID(5,3) := -1.8733131E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_LIQUID(5,4) := -2.5022721E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_LIQUID(6,1) := -2.0905964E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_LIQUID(6,2) := -1.4283185E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_LIQUID(6,3) := -1.8626548E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_LIQUID(6,4) := -2.4919468E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_LIQUID(7,1) := -2.0793270E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_LIQUID(7,2) := -1.4152374E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_LIQUID(7,3) := -1.8521899E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_LIQUID(7,4) := -2.4818090E+04 ;
```

```
SYSTEM.VLE.SPECIFIC_ENTHALPY_LIQUID(8,1) := -2.0682578E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_LIQUID(8,2) := -1.4023887E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_LIQUID(8,3) := -1.8419109E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_LIQUID(8,4) := -2.4718512E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_LIQUID(9,1) := -2.0573812E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_LIQUID(9,2) := -1.3897635E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_LIQUID(9,3) := -1.8318108E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_LIQUID(9,4) := -2.4620667E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_LIQUID(10,1) := -2.0466900E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_LIQUID(10,2) := -1.3773534E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_LIQUID(10,3) := -1.8218827E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_LIQUID(10,4) := -2.4524489E+04 ;


  INITIAL
    WITHIN System.Column DO
      FOR I:=2 TO NSTAGE DO
        X(I,D)= 0.0 ;
        X(I,B)= 0.0 ;
        X(I,C)= 0.0 ;
      END
      TotalHoldup=4.6586382E+01 ;
      M(2)=2.9237071E+00  ;
      M(3)=2.9189519E+00  ;
      M(4)=2.9138424E+00  ;
      M(5)=3.1682045E+00  ;
      M(6)=3.1664972E+00  ;
      M(7)=3.1647940E+00  ;
      M(8)=3.1630950E+00  ;
      M(9)=3.1614000E+00  ;
      FOR I:=2 TO NSTAGE DO
        SIGMA(X(I,))=1 ;
      END

    END

   SCHEDULE
     CONTINUE FOR 1000

END

SIMULATION RunBoth

  UNIT
    System As BothFlowSheet

  INPUT
    WITHIN System.Reactor DO
      FlowOut := 15 ;
      FeedTemp := 300 ;
    END
    WITHIN System.Jacket DO
      Flow_Water := 3 ;
    END
```

```
    WITHIN System.Column DO
      P(1)          :=1.01325 ;
      REFLUXRATIO   :=0.5 ;
    END
    WITHIN System DO
      BFraction:=0.15 ;
    END

PRESET
##############################
# Values of All Active Variables #
##############################

SYSTEM.MAKEUP(1) := 7.6341614E-02 ;
SYSTEM.MAKEUP(2) := 7.5000000E+00 ;
SYSTEM.MAKEUP(3) := 0.0000000E+00 ;
SYSTEM.MAKEUP(4) := 0.0000000E+00 ;
SYSTEM.COLUMN.MWV(1) := 7.6000000E+01 ;
SYSTEM.COLUMN.MWV(2) := 7.6000000E+01 ;
SYSTEM.COLUMN.MWV(3) := 7.6000000E+01 ;
SYSTEM.COLUMN.MWV(4) := 7.6000000E+01 ;
SYSTEM.COLUMN.MWV(5) := 7.6000000E+01 ;
SYSTEM.COLUMN.MWV(6) := 7.6000000E+01 ;
SYSTEM.COLUMN.MWV(7) := 7.6000000E+01 ;
SYSTEM.COLUMN.MWV(8) := 7.6000000E+01 ;
SYSTEM.COLUMN.MWV(9) := 7.6000000E+01 ;
SYSTEM.COLUMN.MWV(10) := 7.6000000E+01 ;
SYSTEM.COLUMN.BOTTOMS := 2.2005890E+00 ;
SYSTEM.COLUMN.VAPORENTHALPY(1) := 9.6490429E+03 ;
SYSTEM.COLUMN.VAPORENTHALPY(2) := 9.6493013E+03 ;
SYSTEM.COLUMN.VAPORENTHALPY(3) := 9.7613871E+03 ;
SYSTEM.COLUMN.VAPORENTHALPY(4) := 9.8713496E+03 ;
SYSTEM.COLUMN.VAPORENTHALPY(5) := 9.9792593E+03 ;
SYSTEM.COLUMN.VAPORENTHALPY(6) := 1.0094036E+04 ;
SYSTEM.COLUMN.VAPORENTHALPY(7) := 1.0206730E+04 ;
SYSTEM.COLUMN.VAPORENTHALPY(8) := 1.0317422E+04 ;
SYSTEM.COLUMN.VAPORENTHALPY(9) := 1.0426188E+04 ;
SYSTEM.COLUMN.VAPORENTHALPY(10) := 1.0533100E+04 ;
SYSTEM.COLUMN.LIQMDENS(1) := 8.3600000E+02 ;
SYSTEM.COLUMN.LIQMDENS(2) := 8.3600000E+02 ;
SYSTEM.COLUMN.LIQMDENS(3) := 8.3600000E+02 ;
SYSTEM.COLUMN.LIQMDENS(4) := 8.3600000E+02 ;
SYSTEM.COLUMN.LIQMDENS(5) := 8.3600000E+02 ;
SYSTEM.COLUMN.LIQMDENS(6) := 8.3600000E+02 ;
SYSTEM.COLUMN.LIQMDENS(7) := 8.3600000E+02 ;
SYSTEM.COLUMN.LIQMDENS(8) := 8.3600000E+02 ;
SYSTEM.COLUMN.LIQMDENS(9) := 8.3600000E+02 ;
SYSTEM.COLUMN.LIQMDENS(10) := 8.3600000E+02 ;
SYSTEM.COLUMN.VAPMDENS(1) := 2.6164666E+00 ;
SYSTEM.COLUMN.VAPMDENS(2) := 2.6165852E+00 ;
SYSTEM.COLUMN.VAPMDENS(3) := 2.6684553E+00 ;
SYSTEM.COLUMN.VAPMDENS(4) := 2.7201430E+00 ;
SYSTEM.COLUMN.VAPMDENS(5) := 2.7716449E+00 ;
SYSTEM.COLUMN.VAPMDENS(6) := 2.8272809E+00 ;
```

```
SYSTEM.COLUMN.VAPMDENS(7) := 2.8827759E+00 ;
SYSTEM.COLUMN.VAPMDENS(8) := 2.9381325E+00 ;
SYSTEM.COLUMN.VAPMDENS(9) := 2.9933533E+00 ;
SYSTEM.COLUMN.VAPMDENS(10) := 3.0484406E+00 ;
SYSTEM.COLUMN.QR := 6.4072745E+06 ;
SYSTEM.COLUMN.DPSTAT(1) := 0.0000000E+00 ;
SYSTEM.COLUMN.DPSTAT(2) := 2.0759991E-02 ;
SYSTEM.COLUMN.DPSTAT(3) := 2.0726226E-02 ;
SYSTEM.COLUMN.DPSTAT(4) := 2.0689946E-02 ;
SYSTEM.COLUMN.DPSTAT(5) := 2.2496062E-02 ;
SYSTEM.COLUMN.DPSTAT(6) := 2.2483940E-02 ;
SYSTEM.COLUMN.DPSTAT(7) := 2.2471846E-02 ;
SYSTEM.COLUMN.DPSTAT(8) := 2.2459782E-02 ;
SYSTEM.COLUMN.DPSTAT(9) := 2.2447747E-02 ;
SYSTEM.COLUMN.DPSTAT(10) := 0.0000000E+00 ;
SYSTEM.COLUMN.XFEED(1) := 1.0000000E+00 ;
SYSTEM.COLUMN.XFEED(2) := 0.0000000E+00 ;
SYSTEM.COLUMN.XFEED(3) := 0.0000000E+00 ;
SYSTEM.COLUMN.XFEED(4) := 0.0000000E+00 ;
SYSTEM.COLUMN.DPTRAY(1) := 5.0229571E-05 ;
SYSTEM.COLUMN.DPTRAY(2) := 2.1986243E-02 ;
SYSTEM.COLUMN.DPTRAY(3) := 2.1952478E-02 ;
SYSTEM.COLUMN.DPTRAY(4) := 2.1916197E-02 ;
SYSTEM.COLUMN.DPTRAY(5) := 2.3722479E-02 ;
SYSTEM.COLUMN.DPTRAY(6) := 2.3710350E-02 ;
SYSTEM.COLUMN.DPTRAY(7) := 2.3698251E-02 ;
SYSTEM.COLUMN.DPTRAY(8) := 2.3686181E-02 ;
SYSTEM.COLUMN.DPTRAY(9) := 2.3674140E-02 ;
SYSTEM.COLUMN.DPTRAY(10) := 0.0000000E+00 ;
SYSTEM.COLUMN.LIQUIDENTHALPY(1) := -2.1350957E+04 ;
SYSTEM.COLUMN.LIQUIDENTHALPY(2) := -2.1350699E+04 ;
SYSTEM.COLUMN.LIQUIDENTHALPY(3) := -2.1238613E+04 ;
SYSTEM.COLUMN.LIQUIDENTHALPY(4) := -2.1128650E+04 ;
SYSTEM.COLUMN.LIQUIDENTHALPY(5) := -2.1020741E+04 ;
SYSTEM.COLUMN.LIQUIDENTHALPY(6) := -2.0905964E+04 ;
SYSTEM.COLUMN.LIQUIDENTHALPY(7) := -2.0793270E+04 ;
SYSTEM.COLUMN.LIQUIDENTHALPY(8) := -2.0682578E+04 ;
SYSTEM.COLUMN.LIQUIDENTHALPY(9) := -2.0573812E+04 ;
SYSTEM.COLUMN.LIQUIDENTHALPY(10) := -2.0466900E+04 ;
SYSTEM.COLUMN.LIQHEIGHT(1) := 0.0000000E+00 ;
SYSTEM.COLUMN.LIQHEIGHT(2) := 2.5313481E-01 ;
SYSTEM.COLUMN.LIQHEIGHT(3) := 2.5272311E-01 ;
SYSTEM.COLUMN.LIQHEIGHT(4) := 2.5228073E-01 ;
SYSTEM.COLUMN.LIQHEIGHT(5) := 2.7430342E-01 ;
SYSTEM.COLUMN.LIQHEIGHT(6) := 2.7415560E-01 ;
SYSTEM.COLUMN.LIQHEIGHT(7) := 2.7400814E-01 ;
SYSTEM.COLUMN.LIQHEIGHT(8) := 2.7386104E-01 ;
SYSTEM.COLUMN.LIQHEIGHT(9) := 2.7371429E-01 ;
SYSTEM.COLUMN.LIQHEIGHT(10) := 0.0000000E+00 ;
SYSTEM.COLUMN.HEAD(1) := 0.0000000E+00 ;
SYSTEM.COLUMN.HEAD(2) := 3.1348139E+00 ;
SYSTEM.COLUMN.HEAD(3) := 2.7231082E+00 ;
SYSTEM.COLUMN.HEAD(4) := 2.2807273E+00 ;
SYSTEM.COLUMN.HEAD(5) := 2.4303420E+01 ;
```

```
SYSTEM.COLUMN.HEAD(6) := 2.4155602E+01 ;
SYSTEM.COLUMN.HEAD(7) := 2.4008139E+01 ;
SYSTEM.COLUMN.HEAD(8) := 2.3861039E+01 ;
SYSTEM.COLUMN.HEAD(9) := 2.3714286E+01 ;
SYSTEM.COLUMN.HEAD(10) := 0.0000000E+00 ;
SYSTEM.COLUMN.K(1,1) := 1.0000000E+00 ;
SYSTEM.COLUMN.K(1,2) := 1.1186637E+00 ;
SYSTEM.COLUMN.K(1,3) := 3.6783189E-01 ;
SYSTEM.COLUMN.K(1,4) := 2.0422134E-01 ;
SYSTEM.COLUMN.K(2,1) := 1.0000000E+00 ;
SYSTEM.COLUMN.K(2,2) := 1.1186548E+00 ;
SYSTEM.COLUMN.K(2,3) := 3.6782906E-01 ;
SYSTEM.COLUMN.K(2,4) := 2.0422232E-01 ;
SYSTEM.COLUMN.K(3,1) := 1.0000000E+00 ;
SYSTEM.COLUMN.K(3,2) := 1.1147884E+00 ;
SYSTEM.COLUMN.K(3,3) := 3.6660608E-01 ;
SYSTEM.COLUMN.K(3,4) := 2.0464705E-01 ;
SYSTEM.COLUMN.K(4,1) := 1.0000000E+00 ;
SYSTEM.COLUMN.K(4,2) := 1.1110220E+00 ;
SYSTEM.COLUMN.K(4,3) := 3.6541455E-01 ;
SYSTEM.COLUMN.K(4,4) := 2.0506307E-01 ;
SYSTEM.COLUMN.K(5,1) := 1.0000000E+00 ;
SYSTEM.COLUMN.K(5,2) := 1.1073514E+00 ;
SYSTEM.COLUMN.K(5,3) := 3.6425316E-01 ;
SYSTEM.COLUMN.K(5,4) := 2.0547070E-01 ;
SYSTEM.COLUMN.K(6,1) := 1.0000000E+00 ;
SYSTEM.COLUMN.K(6,2) := 1.1034745E+00 ;
SYSTEM.COLUMN.K(6,3) := 3.6302637E-01 ;
SYSTEM.COLUMN.K(6,4) := 2.0590358E-01 ;
SYSTEM.COLUMN.K(7,1) := 1.0000000E+00 ;
SYSTEM.COLUMN.K(7,2) := 1.0996952E+00 ;
SYSTEM.COLUMN.K(7,3) := 3.6183030E-01 ;
SYSTEM.COLUMN.K(7,4) := 2.0632791E-01 ;
SYSTEM.COLUMN.K(8,1) := 1.0000000E+00 ;
SYSTEM.COLUMN.K(8,2) := 1.0960090E+00 ;
SYSTEM.COLUMN.K(8,3) := 3.6066354E-01 ;
SYSTEM.COLUMN.K(8,4) := 2.0674404E-01 ;
SYSTEM.COLUMN.K(9,1) := 1.0000000E+00 ;
SYSTEM.COLUMN.K(9,2) := 1.0924118E+00 ;
SYSTEM.COLUMN.K(9,3) := 3.5952480E-01 ;
SYSTEM.COLUMN.K(9,4) := 2.0715229E-01 ;
SYSTEM.COLUMN.K(10,1) := 1.0000000E+00 ;
SYSTEM.COLUMN.K(10,2) := 1.0888996E+00 ;
SYSTEM.COLUMN.K(10,3) := 3.5841286E-01 ;
SYSTEM.COLUMN.K(10,4) := 2.0755297E-01 ;
SYSTEM.COLUMN.L(1) := 1.2799411E+01 ;
SYSTEM.COLUMN.L(2) := 1.0700915E+01 ;
SYSTEM.COLUMN.L(3) := 8.6636449E+00 ;
SYSTEM.COLUMN.L(4) := 6.6406918E+00 ;
SYSTEM.COLUMN.L(5) := 2.3099596E+02 ;
SYSTEM.COLUMN.L(6) := 2.2889172E+02 ;
SYSTEM.COLUMN.L(7) := 2.2679894E+02 ;
SYSTEM.COLUMN.L(8) := 2.2471772E+02 ;
SYSTEM.COLUMN.L(9) := 2.2264778E+02 ;
```

```
SYSTEM.COLUMN.L(10) := 2.2005890E+00 ;
SYSTEM.COLUMN.FEED(1) := 0.0000000E+00 ;
SYSTEM.COLUMN.FEED(2) := 0.0000000E+00 ;
SYSTEM.COLUMN.FEED(3) := 0.0000000E+00 ;
SYSTEM.COLUMN.FEED(4) := 0.0000000E+00 ;
SYSTEM.COLUMN.FEED(5) := 1.5000000E+01 ;
SYSTEM.COLUMN.FEED(6) := 0.0000000E+00 ;
SYSTEM.COLUMN.FEED(7) := 0.0000000E+00 ;
SYSTEM.COLUMN.FEED(8) := 0.0000000E+00 ;
SYSTEM.COLUMN.FEED(9) := 0.0000000E+00 ;
SYSTEM.COLUMN.FEED(10) := 0.0000000E+00 ;
SYSTEM.COLUMN.M(1) := 0.0000000E+00 ;
SYSTEM.COLUMN.$M(1) := 0.0000000E+00 ;
SYSTEM.COLUMN.M(2) := 2.9237071E+00 ;
SYSTEM.COLUMN.$M(2) := -5.8431630E-05 ;
SYSTEM.COLUMN.M(3) := 2.9189519E+00 ;
SYSTEM.COLUMN.$M(3) := 1.9291184E-05 ;
SYSTEM.COLUMN.M(4) := 2.9138424E+00 ;
SYSTEM.COLUMN.$M(4) := 8.9519949E-05 ;
SYSTEM.COLUMN.M(5) := 3.1682045E+00 ;
SYSTEM.COLUMN.$M(5) := -1.1165674E-04 ;
SYSTEM.COLUMN.M(6) := 3.1664972E+00 ;
SYSTEM.COLUMN.$M(6) := -1.0041827E-04 ;
SYSTEM.COLUMN.M(7) := 3.1647940E+00 ;
SYSTEM.COLUMN.$M(7) := 3.0944158E-04 ;
SYSTEM.COLUMN.M(8) := 3.1630950E+00 ;
SYSTEM.COLUMN.$M(8) := -5.6979389E-05 ;
SYSTEM.COLUMN.M(9) := 3.1614000E+00 ;
SYSTEM.COLUMN.$M(9) := -7.1432749E-05 ;
SYSTEM.COLUMN.M(10) := 2.2005890E+01 ;
SYSTEM.COLUMN.$M(10) := -1.9333939E-05 ;
SYSTEM.COLUMN.LIQDENS(1) := 1.1000000E+01 ;
SYSTEM.COLUMN.LIQDENS(2) := 1.1000000E+01 ;
SYSTEM.COLUMN.LIQDENS(3) := 1.1000000E+01 ;
SYSTEM.COLUMN.LIQDENS(4) := 1.1000000E+01 ;
SYSTEM.COLUMN.LIQDENS(5) := 1.1000000E+01 ;
SYSTEM.COLUMN.LIQDENS(6) := 1.1000000E+01 ;
SYSTEM.COLUMN.LIQDENS(7) := 1.1000000E+01 ;
SYSTEM.COLUMN.LIQDENS(8) := 1.1000000E+01 ;
SYSTEM.COLUMN.LIQDENS(9) := 1.1000000E+01 ;
SYSTEM.COLUMN.LIQDENS(10) := 1.1000000E+01 ;
SYSTEM.COLUMN.VAPDENS(1) := 3.4427192E-02 ;
SYSTEM.COLUMN.VAPDENS(2) := 3.4428753E-02 ;
SYSTEM.COLUMN.VAPDENS(3) := 3.5111254E-02 ;
SYSTEM.COLUMN.VAPDENS(4) := 3.5791355E-02 ;
SYSTEM.COLUMN.VAPDENS(5) := 3.6469012E-02 ;
SYSTEM.COLUMN.VAPDENS(6) := 3.7201064E-02 ;
SYSTEM.COLUMN.VAPDENS(7) := 3.7931261E-02 ;
SYSTEM.COLUMN.VAPDENS(8) := 3.8659638E-02 ;
SYSTEM.COLUMN.VAPDENS(9) := 3.9386227E-02 ;
SYSTEM.COLUMN.VAPDENS(10) := 4.0111061E-02 ;
SYSTEM.COLUMN.P(2) := 1.0133002E+00 ;
SYSTEM.COLUMN.P(3) := 1.0352865E+00 ;
SYSTEM.COLUMN.P(4) := 1.0572390E+00 ;
```

```
SYSTEM.COLUMN.P(5) := 1.0791551E+00 ;
SYSTEM.COLUMN.P(6) := 1.1028776E+00 ;
SYSTEM.COLUMN.P(7) := 1.1265880E+00 ;
SYSTEM.COLUMN.P(8) := 1.1502862E+00 ;
SYSTEM.COLUMN.P(9) := 1.1739724E+00 ;
SYSTEM.COLUMN.P(10) := 1.1976465E+00 ;
SYSTEM.COLUMN.DISTILLOUT := 1.2799411E+01 ;
SYSTEM.COLUMN.VOLUMEHOLDUP(1) := 0.0000000E+00 ;
SYSTEM.COLUMN.VOLUMEHOLDUP(2) := 2.6579155E-01 ;
SYSTEM.COLUMN.VOLUMEHOLDUP(3) := 2.6535926E-01 ;
SYSTEM.COLUMN.VOLUMEHOLDUP(4) := 2.6489476E-01 ;
SYSTEM.COLUMN.VOLUMEHOLDUP(5) := 2.8801859E-01 ;
SYSTEM.COLUMN.VOLUMEHOLDUP(6) := 2.8786338E-01 ;
SYSTEM.COLUMN.VOLUMEHOLDUP(7) := 2.8770855E-01 ;
SYSTEM.COLUMN.VOLUMEHOLDUP(8) := 2.8755409E-01 ;
SYSTEM.COLUMN.VOLUMEHOLDUP(9) := 2.8740000E-01 ;
SYSTEM.COLUMN.VOLUMEHOLDUP(10) := 2.0005354E+00 ;
SYSTEM.COLUMN.T(1) := 3.5400141E+02 ;
SYSTEM.COLUMN.T(2) := 3.5400291E+02 ;
SYSTEM.COLUMN.T(3) := 3.5465344E+02 ;
SYSTEM.COLUMN.T(4) := 3.5529164E+02 ;
SYSTEM.COLUMN.T(5) := 3.5591793E+02 ;
SYSTEM.COLUMN.T(6) := 3.5658408E+02 ;
SYSTEM.COLUMN.T(7) := 3.5723813E+02 ;
SYSTEM.COLUMN.T(8) := 3.5788057E+02 ;
SYSTEM.COLUMN.T(9) := 3.5851183E+02 ;
SYSTEM.COLUMN.T(10) := 3.5913233E+02 ;
SYSTEM.COLUMN.V(1) := 0.0000000E+00 ;
SYSTEM.COLUMN.V(2) := 2.5598822E+01 ;
SYSTEM.COLUMN.V(3) := 2.3500268E+01 ;
SYSTEM.COLUMN.V(4) := 2.1463017E+01 ;
SYSTEM.COLUMN.V(5) := 1.9440153E+01 ;
SYSTEM.COLUMN.V(6) := 2.2879531E+02 ;
SYSTEM.COLUMN.V(7) := 2.2669097E+02 ;
SYSTEM.COLUMN.V(8) := 2.2459850E+02 ;
SYSTEM.COLUMN.V(9) := 2.2251722E+02 ;
SYSTEM.COLUMN.V(10) := 2.2044721E+02 ;
#SYSTEM.COLUMN.FEEDTEMP := 3.0000000E+02 ;
SYSTEM.COLUMN.DPDRY(1) := 0.0000000E+00 ;
SYSTEM.COLUMN.DPDRY(2) := 1.8678982E-09 ;
SYSTEM.COLUMN.DPDRY(3) := 1.5277914E-09 ;
SYSTEM.COLUMN.DPDRY(4) := 1.2295610E-09 ;
SYSTEM.COLUMN.DPDRY(5) := 1.6714719E-07 ;
SYSTEM.COLUMN.DPDRY(6) := 1.6085773E-07 ;
SYSTEM.COLUMN.DPDRY(7) := 1.5486215E-07 ;
SYSTEM.COLUMN.DPDRY(8) := 1.4914144E-07 ;
SYSTEM.COLUMN.DPDRY(9) := 1.4367913E-07 ;
SYSTEM.COLUMN.DPDRY(10) := 0.0000000E+00 ;
SYSTEM.COLUMN.X(1,1) := 1.0000000E+00 ;
SYSTEM.COLUMN.X(1,2) := 0.0000000E+00 ;
SYSTEM.COLUMN.X(1,3) := 0.0000000E+00 ;
SYSTEM.COLUMN.X(1,4) := 0.0000000E+00 ;
SYSTEM.COLUMN.X(2,1) := 1.0000000E+00 ;
SYSTEM.COLUMN.$X(2,1) := 1.0816707E-21 ;
```

```
SYSTEM.COLUMN.X(2,2) := 0.0000000E+00 ;
SYSTEM.COLUMN.$X(2,2) := 0.0000000E+00 ;
SYSTEM.COLUMN.X(2,3) := 0.0000000E+00 ;
SYSTEM.COLUMN.$X(2,3) := 0.0000000E+00 ;
SYSTEM.COLUMN.X(2,4) := 0.0000000E+00 ;
SYSTEM.COLUMN.$X(2,4) := 0.0000000E+00 ;
SYSTEM.COLUMN.X(3,1) := 1.0000000E+00 ;
SYSTEM.COLUMN.$X(3,1) := -1.5264713E-22 ;
SYSTEM.COLUMN.X(3,2) := 0.0000000E+00 ;
SYSTEM.COLUMN.$X(3,2) := 2.4869401E-23 ;
SYSTEM.COLUMN.X(3,3) := 0.0000000E+00 ;
SYSTEM.COLUMN.$X(3,3) := 0.0000000E+00 ;
SYSTEM.COLUMN.X(3,4) := 0.0000000E+00 ;
SYSTEM.COLUMN.$X(3,4) := 0.0000000E+00 ;
SYSTEM.COLUMN.X(4,1) := 1.0000000E+00 ;
SYSTEM.COLUMN.$X(4,1) := 2.1141297E-21 ;
SYSTEM.COLUMN.X(4,2) := 0.0000000E+00 ;
SYSTEM.COLUMN.$X(4,2) := -2.1385140E-23 ;
SYSTEM.COLUMN.X(4,3) := 0.0000000E+00 ;
SYSTEM.COLUMN.$X(4,3) := 0.0000000E+00 ;
SYSTEM.COLUMN.X(4,4) := 0.0000000E+00 ;
SYSTEM.COLUMN.$X(4,4) := 0.0000000E+00 ;
SYSTEM.COLUMN.X(5,1) := 1.0000000E+00 ;
SYSTEM.COLUMN.$X(5,1) := -1.2429451E-21 ;
SYSTEM.COLUMN.X(5,2) := 0.0000000E+00 ;
SYSTEM.COLUMN.$X(5,2) := -3.2446314E-24 ;
SYSTEM.COLUMN.X(5,3) := 0.0000000E+00 ;
SYSTEM.COLUMN.$X(5,3) := 0.0000000E+00 ;
SYSTEM.COLUMN.X(5,4) := 0.0000000E+00 ;
SYSTEM.COLUMN.$X(5,4) := 0.0000000E+00 ;
SYSTEM.COLUMN.X(6,1) := 1.0000000E+00 ;
SYSTEM.COLUMN.$X(6,1) := 8.0887492E-22 ;
SYSTEM.COLUMN.X(6,2) := 0.0000000E+00 ;
SYSTEM.COLUMN.$X(6,2) := 0.0000000E+00 ;
SYSTEM.COLUMN.X(6,3) := 0.0000000E+00 ;
SYSTEM.COLUMN.$X(6,3) := 0.0000000E+00 ;
SYSTEM.COLUMN.X(6,4) := 0.0000000E+00 ;
SYSTEM.COLUMN.$X(6,4) := 0.0000000E+00 ;
SYSTEM.COLUMN.X(7,1) := 1.0000000E+00 ;
SYSTEM.COLUMN.$X(7,1) := -6.1828866E-22 ;
SYSTEM.COLUMN.X(7,2) := 0.0000000E+00 ;
SYSTEM.COLUMN.$X(7,2) := 0.0000000E+00 ;
SYSTEM.COLUMN.X(7,3) := 0.0000000E+00 ;
SYSTEM.COLUMN.$X(7,3) := 4.1248869E-24 ;
SYSTEM.COLUMN.X(7,4) := 0.0000000E+00 ;
SYSTEM.COLUMN.$X(7,4) := 0.0000000E+00 ;
SYSTEM.COLUMN.X(8,1) := 1.0000000E+00 ;
SYSTEM.COLUMN.$X(8,1) := 1.2278487E-21 ;
SYSTEM.COLUMN.X(8,2) := 0.0000000E+00 ;
SYSTEM.COLUMN.$X(8,2) := 0.0000000E+00 ;
SYSTEM.COLUMN.X(8,3) := 0.0000000E+00 ;
SYSTEM.COLUMN.$X(8,3) := -4.1271025E-24 ;
SYSTEM.COLUMN.X(8,4) := 0.0000000E+00 ;
SYSTEM.COLUMN.$X(8,4) := 0.0000000E+00 ;
```

```
SYSTEM.COLUMN.X(9,1) := 1.0000000E+00 ;
SYSTEM.COLUMN.$X(9,1) := 1.0411440E-21 ;
SYSTEM.COLUMN.X(9,2) := 0.0000000E+00 ;
SYSTEM.COLUMN.$X(9,2) := 0.0000000E+00 ;
SYSTEM.COLUMN.X(9,3) := 0.0000000E+00 ;
SYSTEM.COLUMN.$X(9,3) := 0.0000000E+00 ;
SYSTEM.COLUMN.X(9,4) := 0.0000000E+00 ;
SYSTEM.COLUMN.$X(9,4) := 0.0000000E+00 ;
SYSTEM.COLUMN.X(10,1) := 1.0000000E+00 ;
SYSTEM.COLUMN.$X(10,1) := -8.9506882E-23 ;
SYSTEM.COLUMN.X(10,2) := 0.0000000E+00 ;
SYSTEM.COLUMN.$X(10,2) := 0.0000000E+00 ;
SYSTEM.COLUMN.X(10,3) := 0.0000000E+00 ;
SYSTEM.COLUMN.$X(10,3) := 0.0000000E+00 ;
SYSTEM.COLUMN.X(10,4) := 0.0000000E+00 ;
SYSTEM.COLUMN.$X(10,4) := 0.0000000E+00 ;
SYSTEM.COLUMN.Y(1,1) := 1.0000000E+00 ;
SYSTEM.COLUMN.Y(1,2) := 0.0000000E+00 ;
SYSTEM.COLUMN.Y(1,3) := 0.0000000E+00 ;
SYSTEM.COLUMN.Y(1,4) := 0.0000000E+00 ;
SYSTEM.COLUMN.Y(2,1) := 1.0000000E+00 ;
SYSTEM.COLUMN.Y(2,2) := 0.0000000E+00 ;
SYSTEM.COLUMN.Y(2,3) := 0.0000000E+00 ;
SYSTEM.COLUMN.Y(2,4) := 0.0000000E+00 ;
SYSTEM.COLUMN.Y(3,1) := 1.0000000E+00 ;
SYSTEM.COLUMN.Y(3,2) := 0.0000000E+00 ;
SYSTEM.COLUMN.Y(3,3) := 0.0000000E+00 ;
SYSTEM.COLUMN.Y(3,4) := 0.0000000E+00 ;
SYSTEM.COLUMN.Y(4,1) := 1.0000000E+00 ;
SYSTEM.COLUMN.Y(4,2) := 3.3822172E-24 ;
SYSTEM.COLUMN.Y(4,3) := 0.0000000E+00 ;
SYSTEM.COLUMN.Y(4,4) := 0.0000000E+00 ;
SYSTEM.COLUMN.Y(5,1) := 1.0000000E+00 ;
SYSTEM.COLUMN.Y(5,2) := 5.2878473E-25 ;
SYSTEM.COLUMN.Y(5,3) := 0.0000000E+00 ;
SYSTEM.COLUMN.Y(5,4) := 0.0000000E+00 ;
SYSTEM.COLUMN.Y(6,1) := 1.0000000E+00 ;
SYSTEM.COLUMN.Y(6,2) := 0.0000000E+00 ;
SYSTEM.COLUMN.Y(6,3) := 0.0000000E+00 ;
SYSTEM.COLUMN.Y(6,4) := 0.0000000E+00 ;
SYSTEM.COLUMN.Y(7,1) := 1.0000000E+00 ;
SYSTEM.COLUMN.Y(7,2) := 0.0000000E+00 ;
SYSTEM.COLUMN.Y(7,3) := 0.0000000E+00 ;
SYSTEM.COLUMN.Y(7,4) := 0.0000000E+00 ;
SYSTEM.COLUMN.Y(8,1) := 1.0000000E+00 ;
SYSTEM.COLUMN.Y(8,2) := 0.0000000E+00 ;
SYSTEM.COLUMN.Y(8,3) := 5.8123352E-26 ;
SYSTEM.COLUMN.Y(8,4) := 0.0000000E+00 ;
SYSTEM.COLUMN.Y(9,1) := 1.0000000E+00 ;
SYSTEM.COLUMN.Y(9,2) := 0.0000000E+00 ;
SYSTEM.COLUMN.Y(9,3) := 0.0000000E+00 ;
SYSTEM.COLUMN.Y(9,4) := 0.0000000E+00 ;
SYSTEM.COLUMN.Y(10,1) := 1.0000000E+00 ;
SYSTEM.COLUMN.Y(10,2) := 0.0000000E+00 ;
```

```
SYSTEM.COLUMN.Y(10,3) := 0.0000000E+00 ;
SYSTEM.COLUMN.Y(10,4) := 0.0000000E+00 ;
SYSTEM.COLUMN.FEEDENTHALPY := -4.5983100E+05 ;
SYSTEM.COLUMN.MWL(1) := 7.6000000E+01 ;
SYSTEM.COLUMN.MWL(2) := 7.6000000E+01 ;
SYSTEM.COLUMN.MWL(3) := 7.6000000E+01 ;
SYSTEM.COLUMN.MWL(4) := 7.6000000E+01 ;
SYSTEM.COLUMN.MWL(5) := 7.6000000E+01 ;
SYSTEM.COLUMN.MWL(6) := 7.6000000E+01 ;
SYSTEM.COLUMN.MWL(7) := 7.6000000E+01 ;
SYSTEM.COLUMN.MWL(8) := 7.6000000E+01 ;
SYSTEM.COLUMN.MWL(9) := 7.6000000E+01 ;
SYSTEM.COLUMN.MWL(10) := 7.6000000E+01 ;
SYSTEM.COLUMN.TOTALHOLDUP := 4.6586382E+01 ;
SYSTEM.COLUMN.$TOTALHOLDUP := 0.0000000E+00 ;
SYSTEM.COLUMN.QC := 7.9357010E+05 ;
SYSTEM.MAKEUPENTHALPY.TEMP := 3.0000000E+02 ;
SYSTEM.MAKEUPENTHALPY.SPECIFIC_ENTHALPY_VAPOR(1) := 3.4460000E+02 ;
SYSTEM.MAKEUPENTHALPY.SPECIFIC_ENTHALPY_VAPOR(2) := 4.0000000E+02 ;
SYSTEM.MAKEUPENTHALPY.SPECIFIC_ENTHALPY_VAPOR(3) := 3.2000000E+02 ;
SYSTEM.MAKEUPENTHALPY.SPECIFIC_ENTHALPY_VAPOR(4) := 3.1000000E+02 ;
SYSTEM.MAKEUPENTHALPY.SPECIFIC_ENTHALPY_LIQUID(1) := -3.0655400E+04 ;
SYSTEM.MAKEUPENTHALPY.SPECIFIC_ENTHALPY_LIQUID(2) := -2.5600000E+04 ;
SYSTEM.MAKEUPENTHALPY.SPECIFIC_ENTHALPY_LIQUID(3) := -2.7680000E+04 ;
SYSTEM.MAKEUPENTHALPY.SPECIFIC_ENTHALPY_LIQUID(4) := -3.3690000E+04 ;
SYSTEM.REACTOR.TOTALMOLS := 5.0000000E+01 ;
SYSTEM.REACTOR.$TOTALMOLS := 0.0000000E+00 ;
SYSTEM.REACTOR.TEMP := 3.0000000E+02 ;
SYSTEM.REACTOR.ENTHALPY := -1.5327700E+06 ;
SYSTEM.REACTOR.$ENTHALPY := -4.3344954E+06 ;
SYSTEM.REACTOR.TOTALFEED := 1.5000000E+01 ;
SYSTEM.REACTOR.QJACKET := 4.8152866E+04 ;
SYSTEM.REACTOR.NO_MOLS(1) := 5.0000000E+01 ;
SYSTEM.REACTOR.$NO_MOLS(1) := -7.5000000E+00 ;
SYSTEM.REACTOR.NO_MOLS(2) := 0.0000000E+00 ;
SYSTEM.REACTOR.$NO_MOLS(2) := 7.5000000E+00 ;
SYSTEM.REACTOR.NO_MOLS(3) := 0.0000000E+00 ;
SYSTEM.REACTOR.$NO_MOLS(3) := 0.0000000E+00 ;
SYSTEM.REACTOR.NO_MOLS(4) := 0.0000000E+00 ;
SYSTEM.REACTOR.$NO_MOLS(4) := 0.0000000E+00 ;
SYSTEM.REACTOR.VOLUME := 4.5454545E+00 ;
SYSTEM.REACTOR.CONCENTRATION(1) := 1.1000000E+01 ;
SYSTEM.REACTOR.CONCENTRATION(2) := 0.0000000E+00 ;
SYSTEM.REACTOR.CONCENTRATION(3) := 0.0000000E+00 ;
SYSTEM.REACTOR.CONCENTRATION(4) := 0.0000000E+00 ;
SYSTEM.REACTOR.FEED_A := 7.5000000E+00 ;
SYSTEM.REACTOR.FEED_B := 7.5000000E+00 ;
SYSTEM.REACTOR.FEED_C := 0.0000000E+00 ;
SYSTEM.REACTOR.X(1) := 1.0000000E+00 ;
SYSTEM.REACTOR.X(2) := 0.0000000E+00 ;
SYSTEM.REACTOR.X(3) := 0.0000000E+00 ;
SYSTEM.REACTOR.X(4) := 0.0000000E+00 ;
SYSTEM.REACTOR.FEED_D := 0.0000000E+00 ;
SYSTEM.REACTOR.FEEDENTHALPY(1) := -5.0553546E+05 ;
```

```
SYSTEM.REACTOR.FEEDENTHALPY(2) := -1.9395435E+05 ;
SYSTEM.REACTOR.FEEDENTHALPY(3) := -2.0971314E+05 ;
SYSTEM.REACTOR.FEEDENTHALPY(4) := -2.5524695E+05 ;
SYSTEM.REACTOR.SPECIFIC_ENTHALPY(1) := -3.0655400E+04 ;
SYSTEM.REACTOR.SPECIFIC_ENTHALPY(2) := -2.5600000E+04 ;
SYSTEM.REACTOR.SPECIFIC_ENTHALPY(3) := -2.7680000E+04 ;
SYSTEM.REACTOR.SPECIFIC_ENTHALPY(4) := -3.3690000E+04 ;
SYSTEM.REACTOR.REACTIONRATE(1) := 0.0000000E+00 ;
SYSTEM.REACTOR.REACTIONRATE(2) := 0.0000000E+00 ;
SYSTEM.VLE.VAPORENTHALPY(1) := 9.6490429E+03 ;
SYSTEM.VLE.VAPORENTHALPY(2) := 9.6493013E+03 ;
SYSTEM.VLE.VAPORENTHALPY(3) := 9.7613871E+03 ;
SYSTEM.VLE.VAPORENTHALPY(4) := 9.8713496E+03 ;
SYSTEM.VLE.VAPORENTHALPY(5) := 9.9792593E+03 ;
SYSTEM.VLE.VAPORENTHALPY(6) := 1.0094036E+04 ;
SYSTEM.VLE.VAPORENTHALPY(7) := 1.0206730E+04 ;
SYSTEM.VLE.VAPORENTHALPY(8) := 1.0317422E+04 ;
SYSTEM.VLE.VAPORENTHALPY(9) := 1.0426188E+04 ;
SYSTEM.VLE.VAPORENTHALPY(10) := 1.0533100E+04 ;
SYSTEM.VLE.TEMP(1) := 3.5400141E+02 ;
SYSTEM.VLE.TEMP(2) := 3.5400291E+02 ;
SYSTEM.VLE.TEMP(3) := 3.5465344E+02 ;
SYSTEM.VLE.TEMP(4) := 3.5529164E+02 ;
SYSTEM.VLE.TEMP(5) := 3.5591793E+02 ;
SYSTEM.VLE.TEMP(6) := 3.5658408E+02 ;
SYSTEM.VLE.TEMP(7) := 3.5723813E+02 ;
SYSTEM.VLE.TEMP(8) := 3.5788057E+02 ;
SYSTEM.VLE.TEMP(9) := 3.5851183E+02 ;
SYSTEM.VLE.TEMP(10) := 3.5913233E+02 ;
SYSTEM.VLE.LIQUIDENTHALPY(1) := -2.1350957E+04 ;
SYSTEM.VLE.LIQUIDENTHALPY(2) := -2.1350699E+04 ;
SYSTEM.VLE.LIQUIDENTHALPY(3) := -2.1238613E+04 ;
SYSTEM.VLE.LIQUIDENTHALPY(4) := -2.1128650E+04 ;
SYSTEM.VLE.LIQUIDENTHALPY(5) := -2.1020741E+04 ;
SYSTEM.VLE.LIQUIDENTHALPY(6) := -2.0905964E+04 ;
SYSTEM.VLE.LIQUIDENTHALPY(7) := -2.0793270E+04 ;
SYSTEM.VLE.LIQUIDENTHALPY(8) := -2.0682578E+04 ;
SYSTEM.VLE.LIQUIDENTHALPY(9) := -2.0573812E+04 ;
SYSTEM.VLE.LIQUIDENTHALPY(10) := -2.0466900E+04 ;
SYSTEM.VLE.K(1,1) := 1.0000000E+00 ;
SYSTEM.VLE.K(1,2) := 1.1186637E+00 ;
SYSTEM.VLE.K(1,3) := 3.6783189E-01 ;
SYSTEM.VLE.K(1,4) := 2.0422134E-01 ;
SYSTEM.VLE.K(2,1) := 1.0000000E+00 ;
SYSTEM.VLE.K(2,2) := 1.1186548E+00 ;
SYSTEM.VLE.K(2,3) := 3.6782906E-01 ;
SYSTEM.VLE.K(2,4) := 2.0422232E-01 ;
SYSTEM.VLE.K(3,1) := 1.0000000E+00 ;
SYSTEM.VLE.K(3,2) := 1.1147884E+00 ;
SYSTEM.VLE.K(3,3) := 3.6660608E-01 ;
SYSTEM.VLE.K(3,4) := 2.0464705E-01 ;
SYSTEM.VLE.K(4,1) := 1.0000000E+00 ;
SYSTEM.VLE.K(4,2) := 1.1110220E+00 ;
SYSTEM.VLE.K(4,3) := 3.6541455E-01 ;
```

```
SYSTEM.VLE.K(4,4) := 2.0506307E-01 ;
SYSTEM.VLE.K(5,1) := 1.0000000E+00 ;
SYSTEM.VLE.K(5,2) := 1.1073514E+00 ;
SYSTEM.VLE.K(5,3) := 3.6425316E-01 ;
SYSTEM.VLE.K(5,4) := 2.0547070E-01 ;
SYSTEM.VLE.K(6,1) := 1.0000000E+00 ;
SYSTEM.VLE.K(6,2) := 1.1034745E+00 ;
SYSTEM.VLE.K(6,3) := 3.6302637E-01 ;
SYSTEM.VLE.K(6,4) := 2.0590358E-01 ;
SYSTEM.VLE.K(7,1) := 1.0000000E+00 ;
SYSTEM.VLE.K(7,2) := 1.0996952E+00 ;
SYSTEM.VLE.K(7,3) := 3.6183030E-01 ;
SYSTEM.VLE.K(7,4) := 2.0632791E-01 ;
SYSTEM.VLE.K(8,1) := 1.0000000E+00 ;
SYSTEM.VLE.K(8,2) := 1.0960090E+00 ;
SYSTEM.VLE.K(8,3) := 3.6066354E-01 ;
SYSTEM.VLE.K(8,4) := 2.0674404E-01 ;
SYSTEM.VLE.K(9,1) := 1.0000000E+00 ;
SYSTEM.VLE.K(9,2) := 1.0924118E+00 ;
SYSTEM.VLE.K(9,3) := 3.5952480E-01 ;
SYSTEM.VLE.K(9,4) := 2.0715229E-01 ;
SYSTEM.VLE.K(10,1) := 1.0000000E+00 ;
SYSTEM.VLE.K(10,2) := 1.0888996E+00 ;
SYSTEM.VLE.K(10,3) := 3.5841286E-01 ;
SYSTEM.VLE.K(10,4) := 2.0755297E-01 ;
SYSTEM.VLE.VAPORPRESSURE(1,1) := 1.0132500E+00 ;
SYSTEM.VLE.VAPORPRESSURE(1,2) := 1.1334860E+00 ;
SYSTEM.VLE.VAPORPRESSURE(1,3) := 3.7270566E-01 ;
SYSTEM.VLE.VAPORPRESSURE(1,4) := 2.0692728E-01 ;
SYSTEM.VLE.VAPORPRESSURE(2,1) := 1.0133002E+00 ;
SYSTEM.VLE.VAPORPRESSURE(2,2) := 1.1335331E+00 ;
SYSTEM.VLE.VAPORPRESSURE(2,3) := 3.7272127E-01 ;
SYSTEM.VLE.VAPORPRESSURE(2,4) := 2.0693853E-01 ;
SYSTEM.VLE.VAPORPRESSURE(3,1) := 1.0352865E+00 ;
SYSTEM.VLE.VAPORPRESSURE(3,2) := 1.1541254E+00 ;
SYSTEM.VLE.VAPORPRESSURE(3,3) := 3.7954232E-01 ;
SYSTEM.VLE.VAPORPRESSURE(3,4) := 2.1186832E-01 ;
SYSTEM.VLE.VAPORPRESSURE(4,1) := 1.0572390E+00 ;
SYSTEM.VLE.VAPORPRESSURE(4,2) := 1.1746158E+00 ;
SYSTEM.VLE.VAPORPRESSURE(4,3) := 3.8633050E-01 ;
SYSTEM.VLE.VAPORPRESSURE(4,4) := 2.1680067E-01 ;
SYSTEM.VLE.VAPORPRESSURE(5,1) := 1.0791551E+00 ;
SYSTEM.VLE.VAPORPRESSURE(5,2) := 1.1950039E+00 ;
SYSTEM.VLE.VAPORPRESSURE(5,3) := 3.9308568E-01 ;
SYSTEM.VLE.VAPORPRESSURE(5,4) := 2.2173476E-01 ;
SYSTEM.VLE.VAPORPRESSURE(6,1) := 1.1028776E+00 ;
SYSTEM.VLE.VAPORPRESSURE(6,2) := 1.2169973E+00 ;
SYSTEM.VLE.VAPORPRESSURE(6,3) := 4.0037366E-01 ;
SYSTEM.VLE.VAPORPRESSURE(6,4) := 2.2708645E-01 ;
SYSTEM.VLE.VAPORPRESSURE(7,1) := 1.1265880E+00 ;
SYSTEM.VLE.VAPORPRESSURE(7,2) := 1.2389034E+00 ;
SYSTEM.VLE.VAPORPRESSURE(7,3) := 4.0763366E-01 ;
SYSTEM.VLE.VAPORPRESSURE(7,4) := 2.3244654E-01 ;
SYSTEM.VLE.VAPORPRESSURE(8,1) := 1.1502862E+00 ;
```

```
SYSTEM.VLE.VAPORPRESSURE(8,2) := 1.2607240E+00 ;
SYSTEM.VLE.VAPORPRESSURE(8,3) := 4.1486630E-01 ;
SYSTEM.VLE.VAPORPRESSURE(8,4) := 2.3781482E-01 ;
SYSTEM.VLE.VAPORPRESSURE(9,1) := 1.1739724E+00 ;
SYSTEM.VLE.VAPORPRESSURE(9,2) := 1.2824613E+00 ;
SYSTEM.VLE.VAPORPRESSURE(9,3) := 4.2207220E-01 ;
SYSTEM.VLE.VAPORPRESSURE(9,4) := 2.4319108E-01 ;
SYSTEM.VLE.VAPORPRESSURE(10,1) := 1.1976465E+00 ;
SYSTEM.VLE.VAPORPRESSURE(10,2) := 1.3041169E+00 ;
SYSTEM.VLE.VAPORPRESSURE(10,3) := 4.2925192E-01 ;
SYSTEM.VLE.VAPORPRESSURE(10,4) := 2.4857510E-01 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_VAPOR(1,1) := 9.6490429E+03 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_VAPOR(1,2) := 1.1200282E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_VAPOR(1,3) := 8.9602256E+03 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_VAPOR(1,4) := 8.6802185E+03 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_VAPOR(2,1) := 9.6493013E+03 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_VAPOR(2,2) := 1.1200582E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_VAPOR(2,3) := 8.9604655E+03 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_VAPOR(2,4) := 8.6804509E+03 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_VAPOR(3,1) := 9.7613871E+03 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_VAPOR(3,2) := 1.1330687E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_VAPOR(3,3) := 9.0645499E+03 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_VAPOR(3,4) := 8.7812827E+03 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_VAPOR(4,1) := 9.8713496E+03 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_VAPOR(4,2) := 1.1458328E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_VAPOR(4,3) := 9.1666624E+03 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_VAPOR(4,4) := 8.8802042E+03 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_VAPOR(5,1) := 9.9792593E+03 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_VAPOR(5,2) := 1.1583586E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_VAPOR(5,3) := 9.2668688E+03 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_VAPOR(5,4) := 8.9772791E+03 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_VAPOR(6,1) := 1.0094036E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_VAPOR(6,2) := 1.1716815E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_VAPOR(6,3) := 9.3734520E+03 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_VAPOR(6,4) := 9.0805316E+03 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_VAPOR(7,1) := 1.0206730E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_VAPOR(7,2) := 1.1847626E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_VAPOR(7,3) := 9.4781009E+03 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_VAPOR(7,4) := 9.1819102E+03 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_VAPOR(8,1) := 1.0317422E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_VAPOR(8,2) := 1.1976113E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_VAPOR(8,3) := 9.5808907E+03 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_VAPOR(8,4) := 9.2814879E+03 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_VAPOR(9,1) := 1.0426188E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_VAPOR(9,2) := 1.2102365E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_VAPOR(9,3) := 9.6818924E+03 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_VAPOR(9,4) := 9.3793332E+03 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_VAPOR(10,1) := 1.0533100E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_VAPOR(10,2) := 1.2226466E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_VAPOR(10,3) := 9.7811726E+03 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_VAPOR(10,4) := 9.4755110E+03 ;
SYSTEM.VLE.P(1) := 1.0132500E+00 ;
SYSTEM.VLE.P(2) := 1.0133002E+00 ;
SYSTEM.VLE.P(3) := 1.0352865E+00 ;
```

```
SYSTEM.VLE.P(4) := 1.0572390E+00 ;
SYSTEM.VLE.P(5) := 1.0791551E+00 ;
SYSTEM.VLE.P(6) := 1.1028776E+00 ;
SYSTEM.VLE.P(7) := 1.1265880E+00 ;
SYSTEM.VLE.P(8) := 1.1502862E+00 ;
SYSTEM.VLE.P(9) := 1.1739724E+00 ;
SYSTEM.VLE.P(10) := 1.1976465E+00 ;
SYSTEM.VLE.X(1,1) := 1.0000000E+00 ;
SYSTEM.VLE.X(1,2) := 0.0000000E+00 ;
SYSTEM.VLE.X(1,3) := 0.0000000E+00 ;
SYSTEM.VLE.X(1,4) := 0.0000000E+00 ;
SYSTEM.VLE.X(2,1) := 1.0000000E+00 ;
SYSTEM.VLE.X(2,2) := 0.0000000E+00 ;
SYSTEM.VLE.X(2,3) := 0.0000000E+00 ;
SYSTEM.VLE.X(2,4) := 0.0000000E+00 ;
SYSTEM.VLE.X(3,1) := 1.0000000E+00 ;
SYSTEM.VLE.X(3,2) := 0.0000000E+00 ;
SYSTEM.VLE.X(3,3) := 0.0000000E+00 ;
SYSTEM.VLE.X(3,4) := 0.0000000E+00 ;
SYSTEM.VLE.X(4,1) := 1.0000000E+00 ;
SYSTEM.VLE.X(4,2) := 0.0000000E+00 ;
SYSTEM.VLE.X(4,3) := 0.0000000E+00 ;
SYSTEM.VLE.X(4,4) := 0.0000000E+00 ;
SYSTEM.VLE.X(5,1) := 1.0000000E+00 ;
SYSTEM.VLE.X(5,2) := 0.0000000E+00 ;
SYSTEM.VLE.X(5,3) := 0.0000000E+00 ;
SYSTEM.VLE.X(5,4) := 0.0000000E+00 ;
SYSTEM.VLE.X(6,1) := 1.0000000E+00 ;
SYSTEM.VLE.X(6,2) := 0.0000000E+00 ;
SYSTEM.VLE.X(6,3) := 0.0000000E+00 ;
SYSTEM.VLE.X(6,4) := 0.0000000E+00 ;
SYSTEM.VLE.X(7,1) := 1.0000000E+00 ;
SYSTEM.VLE.X(7,2) := 0.0000000E+00 ;
SYSTEM.VLE.X(7,3) := 0.0000000E+00 ;
SYSTEM.VLE.X(7,4) := 0.0000000E+00 ;
SYSTEM.VLE.X(8,1) := 1.0000000E+00 ;
SYSTEM.VLE.X(8,2) := 0.0000000E+00 ;
SYSTEM.VLE.X(8,3) := 0.0000000E+00 ;
SYSTEM.VLE.X(8,4) := 0.0000000E+00 ;
SYSTEM.VLE.X(9,1) := 1.0000000E+00 ;
SYSTEM.VLE.X(9,2) := 0.0000000E+00 ;
SYSTEM.VLE.X(9,3) := 0.0000000E+00 ;
SYSTEM.VLE.X(9,4) := 0.0000000E+00 ;
SYSTEM.VLE.X(10,1) := 1.0000000E+00 ;
SYSTEM.VLE.X(10,2) := 0.0000000E+00 ;
SYSTEM.VLE.X(10,3) := 0.0000000E+00 ;
SYSTEM.VLE.X(10,4) := 0.0000000E+00 ;
SYSTEM.VLE.Y(1,1) := 1.0000000E+00 ;
SYSTEM.VLE.Y(1,2) := 0.0000000E+00 ;
SYSTEM.VLE.Y(1,3) := 0.0000000E+00 ;
SYSTEM.VLE.Y(1,4) := 0.0000000E+00 ;
SYSTEM.VLE.Y(2,1) := 1.0000000E+00 ;
SYSTEM.VLE.Y(2,2) := 0.0000000E+00 ;
SYSTEM.VLE.Y(2,3) := 0.0000000E+00 ;
```

```
SYSTEM.VLE.Y(2,4) := 0.0000000E+00 ;
SYSTEM.VLE.Y(3,1) := 1.0000000E+00 ;
SYSTEM.VLE.Y(3,2) := 0.0000000E+00 ;
SYSTEM.VLE.Y(3,3) := 0.0000000E+00 ;
SYSTEM.VLE.Y(3,4) := 0.0000000E+00 ;
SYSTEM.VLE.Y(4,1) := 1.0000000E+00 ;
SYSTEM.VLE.Y(4,2) := 3.3822172E-24 ;
SYSTEM.VLE.Y(4,3) := 0.0000000E+00 ;
SYSTEM.VLE.Y(4,4) := 0.0000000E+00 ;
SYSTEM.VLE.Y(5,1) := 1.0000000E+00 ;
SYSTEM.VLE.Y(5,2) := 5.2878473E-25 ;
SYSTEM.VLE.Y(5,3) := 0.0000000E+00 ;
SYSTEM.VLE.Y(5,4) := 0.0000000E+00 ;
SYSTEM.VLE.Y(6,1) := 1.0000000E+00 ;
SYSTEM.VLE.Y(6,2) := 0.0000000E+00 ;
SYSTEM.VLE.Y(6,3) := 0.0000000E+00 ;
SYSTEM.VLE.Y(6,4) := 0.0000000E+00 ;
SYSTEM.VLE.Y(7,1) := 1.0000000E+00 ;
SYSTEM.VLE.Y(7,2) := 0.0000000E+00 ;
SYSTEM.VLE.Y(7,3) := 0.0000000E+00 ;
SYSTEM.VLE.Y(7,4) := 0.0000000E+00 ;
SYSTEM.VLE.Y(8,1) := 1.0000000E+00 ;
SYSTEM.VLE.Y(8,2) := 0.0000000E+00 ;
SYSTEM.VLE.Y(8,3) := 5.8123352E-26 ;
SYSTEM.VLE.Y(8,4) := 0.0000000E+00 ;
SYSTEM.VLE.Y(9,1) := 1.0000000E+00 ;
SYSTEM.VLE.Y(9,2) := 0.0000000E+00 ;
SYSTEM.VLE.Y(9,3) := 0.0000000E+00 ;
SYSTEM.VLE.Y(9,4) := 0.0000000E+00 ;
SYSTEM.VLE.Y(10,1) := 1.0000000E+00 ;
SYSTEM.VLE.Y(10,2) := 0.0000000E+00 ;
SYSTEM.VLE.Y(10,3) := 0.0000000E+00 ;
SYSTEM.VLE.Y(10,4) := 0.0000000E+00 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_LIQUID(1,1) := -2.1350957E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_LIQUID(1,2) := -1.4799718E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_LIQUID(1,3) := -1.9039774E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_LIQUID(1,4) := -2.5319781E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_LIQUID(2,1) := -2.1350699E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_LIQUID(2,2) := -1.4799418E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_LIQUID(2,3) := -1.9039535E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_LIQUID(2,4) := -2.5319549E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_LIQUID(3,1) := -2.1238613E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_LIQUID(3,2) := -1.4669313E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_LIQUID(3,3) := -1.8935450E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_LIQUID(3,4) := -2.5218717E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_LIQUID(4,1) := -2.1128650E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_LIQUID(4,2) := -1.4541672E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_LIQUID(4,3) := -1.8833338E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_LIQUID(4,4) := -2.5119796E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_LIQUID(5,1) := -2.1020741E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_LIQUID(5,2) := -1.4416414E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_LIQUID(5,3) := -1.8733131E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_LIQUID(5,4) := -2.5022721E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_LIQUID(6,1) := -2.0905964E+04 ;
```

```
SYSTEM.VLE.SPECIFIC_ENTHALPY_LIQUID(6,2) := -1.4283185E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_LIQUID(6,3) := -1.8626548E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_LIQUID(6,4) := -2.4919468E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_LIQUID(7,1) := -2.0793270E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_LIQUID(7,2) := -1.4152374E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_LIQUID(7,3) := -1.8521899E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_LIQUID(7,4) := -2.4818090E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_LIQUID(8,1) := -2.0682578E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_LIQUID(8,2) := -1.4023887E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_LIQUID(8,3) := -1.8419109E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_LIQUID(8,4) := -2.4718512E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_LIQUID(9,1) := -2.0573812E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_LIQUID(9,2) := -1.3897635E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_LIQUID(9,3) := -1.8318108E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_LIQUID(9,4) := -2.4620667E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_LIQUID(10,1) := -2.0466900E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_LIQUID(10,2) := -1.3773534E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_LIQUID(10,3) := -1.8218827E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_LIQUID(10,4) := -2.4524489E+04 ;
SYSTEM.JACKET.TWATER_OUT := 2.9946497E+02 ;
SYSTEM.JACKET.QJACKET := 4.8152866E+04 ;
SYSTEM.JACKET.TEMP_REACTOR := 3.0000000E+02 ;
SYSTEM.LIQENTHALPY.TEMP := 3.0000000E+02 ;
SYSTEM.LIQENTHALPY.SPECIFIC_ENTHALPY_VAPOR(1) := 3.4460000E+02 ;
SYSTEM.LIQENTHALPY.SPECIFIC_ENTHALPY_VAPOR(2) := 4.0000000E+02 ;
SYSTEM.LIQENTHALPY.SPECIFIC_ENTHALPY_VAPOR(3) := 3.2000000E+02 ;
SYSTEM.LIQENTHALPY.SPECIFIC_ENTHALPY_VAPOR(4) := 3.1000000E+02 ;
SYSTEM.LIQENTHALPY.SPECIFIC_ENTHALPY_LIQUID(1) := -3.0655400E+04 ;
SYSTEM.LIQENTHALPY.SPECIFIC_ENTHALPY_LIQUID(2) := -2.5600000E+04 ;
SYSTEM.LIQENTHALPY.SPECIFIC_ENTHALPY_LIQUID(3) := -2.7680000E+04 ;
SYSTEM.LIQENTHALPY.SPECIFIC_ENTHALPY_LIQUID(4) := -3.3690000E+04 ;

  INITIAL
     WITHIN System.Reactor DO
      # No_Mols(A) = 22.27 ;
      # No_Mols(B) = 36.81 ;
       No_Mols(A) = 50 ;
       No_Mols(B) = 0 ;
       No_Mols(C) = 0.0 ;
       No_Mols(D) = 0.0 ;
       Temp=300 ;
     END
    WITHIN System.Column DO
      FOR I:=2 TO NSTAGE DO
        X(I,D)= 0.0 ;
        X(I,B)= 0.0 ;
        X(I,C)= 0.0 ;
      END
      TotalHoldup=4.6586382E+01 ;
      M(2)=2.9237071E+00  ;
      M(3)=2.9189519E+00  ,
      M(4)=2.9138424E+00  ;
      M(5)=3.1682045E+00  ;
      M(6)=3.1664972E+00  ;
```

```
      M(7)=3.1647940E+00  ;
      M(8)=3.1630950E+00  ;
      M(9)=3.1614000E+00  ;
      FOR I:=2 TO NSTAGE DO
        SIGMA(X(I,))=1 ;
        #SIGMA(K(I,)*X(I,))=1 ;
      END
    END

   SCHEDULE
     CONTINUE FOR 100

END


OPTIMIZATION OptBoth

  UNIT
    System As BothFlowSheet

  VARIABLE
    Mols_A_OUT  AS MolarFlowRate
    Mols_B_OUT  AS MolarFlowRate
    Mols_C_OUT  AS MolarFlowRate
    Mols_D_OUT  AS MolarFlowRate
    ChangeMols_A_Out AS Value
    ChangeMols_B_Out AS Value
    ChangeMols_C_Out AS Value
    ChangeMols_D_Out AS Value
    Total_C_OUT AS MolarHoldup
    Total_D_OUT AS MolarHoldup
    #Final_Time AS Value


  OBJECTIVE
    MAXIMIZE Total_C_OUT-0.1*Total_D_OUT;
    #MAXIMIZE Total_C_OUT;
    #MINIMIZE Final_Time ;
    #MINIMIZE 100000*(ChangeMols_A_Out^2 +ChangeMols_B_Out^2+
    #         ChangeMols_C_Out^2 +ChangeMols_D_Out^2);
    #MINIMIZE Final_Time+  100000*(ChangeMols_A_Out^2 +ChangeMols_B_Out^2+
    #         ChangeMols_C_Out^2 +ChangeMols_D_Out^2+
    #         System.Reactor.$No_Mols(1)^2+System.Reactor.$No_Mols(2)^2+
    #         System.Reactor.$No_Mols(3)^2+System.Reactor.$No_Mols(4)^2);

  EQUATION

    ChangeMols_A_Out= $Mols_A_OUT ;
    ChangeMols_B_Out= $Mols_B_OUT ;
    ChangeMols_C_Out= $Mols_C_OUT ;
    ChangeMols_D_Out= $Mols_D_OUT ;

    Mols_A_OUT=System.Column.BOTTOMS*System.Column.X(System.Column.NSTAGE,1) ;
```

```
    Mols_B_OUT=System.Column.BOTTOMS*System.Column.X(System.Column.NSTAGE,2) ;

    Mols_C_OUT=System.Column.BOTTOMS*System.Column.X(System.Column.NSTAGE,3) ;

    Mols_D_OUT=System.Column.BOTTOMS*System.Column.X(System.Column.NSTAGE,4) ;

    $Total_C_OUT=Mols_C_OUT;

    $Total_D_OUT=Mols_D_OUT;


  INEQUALITY
     System.Jacket.Temp_Reactor<=400 ;

  INPUT
    WITHIN System.Reactor DO
      FlowOut := 15 ;
      FeedTemp := 300 ;
    END
    WITHIN System.Column DO
      P(1)          :=1.01325 ;
     # REFLUXRATIO   :=0.5 ;
    END
    WITHIN System DO
      BFraction := .15 ;
    END

  CONTROL
    WITHIN System.Jacket DO
      Flow_Water := 3 :0 :5 ;
    END

  TIME_INVARIANT
    WITHIN System.Column DO
      REFLUXRATIO:=0.5 : 0.4 : 0.83;
    END
   #Final_Time:=90:71:500 ;

PRESET
################################
# Values of All Active Variables #
################################

SYSTEM.MAKEUP(1) := 7.6341614E-02 ;
SYSTEM.MAKEUP(2) := 7.5000000E+00 ;
SYSTEM.MAKEUP(3) := 0.0000000E+00 ;
SYSTEM.MAKEUP(4) := 0.0000000E+00 ;
SYSTEM.COLUMN.MWV(1) := 7.6000000E+01 ;
SYSTEM.COLUMN.MWV(2) := 7.6000000E+01 ;
SYSTEM.COLUMN.MWV(3) := 7.6000000E+01 ;
SYSTEM.COLUMN.MWV(4) := 7.6000000E+01 ;
SYSTEM.COLUMN.MWV(5) := 7.6000000E+01 ;
SYSTEM.COLUMN.MWV(6) := 7.6000000E+01 ;
SYSTEM.COLUMN.MWV(7) := 7.6000000E+01 ;
```

```
SYSTEM.COLUMN.MWV(8) := 7.6000000E+01 ;
SYSTEM.COLUMN.MWV(9) := 7.6000000E+01 ;
SYSTEM.COLUMN.MWV(10) := 7.6000000E+01 ;
SYSTEM.COLUMN.BOTTOMS := 2.2005890E+00 ;
SYSTEM.COLUMN.VAPORENTHALPY(1) := 9.6490429E+03 ;
SYSTEM.COLUMN.VAPORENTHALPY(2) := 9.6493013E+03 ;
SYSTEM.COLUMN.VAPORENTHALPY(3) := 9.7613871E+03 ;
SYSTEM.COLUMN.VAPORENTHALPY(4) := 9.8713496E+03 ;
SYSTEM.COLUMN.VAPORENTHALPY(5) := 9.9792593E+03 ;
SYSTEM.COLUMN.VAPORENTHALPY(6) := 1.0094036E+04 ;
SYSTEM.COLUMN.VAPORENTHALPY(7) := 1.0206730E+04 ;
SYSTEM.COLUMN.VAPORENTHALPY(8) := 1.0317422E+04 ;
SYSTEM.COLUMN.VAPORENTHALPY(9) := 1.0426188E+04 ;
SYSTEM.COLUMN.VAPORENTHALPY(10) := 1.0533100E+04 ;
SYSTEM.COLUMN.LIQMDENS(1) := 8.3600000E+02 ;
SYSTEM.COLUMN.LIQMDENS(2) := 8.3600000E+02 ;
SYSTEM.COLUMN.LIQMDENS(3) := 8.3600000E+02 ;
SYSTEM.COLUMN.LIQMDENS(4) := 8.3600000E+02 ;
SYSTEM.COLUMN.LIQMDENS(5) := 8.3600000E+02 ;
SYSTEM.COLUMN.LIQMDENS(6) := 8.3600000E+02 ;
SYSTEM.COLUMN.LIQMDENS(7) := 8.3600000E+02 ;
SYSTEM.COLUMN.LIQMDENS(8) := 8.3600000E+02 ;
SYSTEM.COLUMN.LIQMDENS(9) := 8.3600000E+02 ;
SYSTEM.COLUMN.LIQMDENS(10) := 8.3600000E+02 ;
SYSTEM.COLUMN.VAPMDENS(1) := 2.6164666E+00 ;
SYSTEM.COLUMN.VAPMDENS(2) := 2.6165852E+00 ;
SYSTEM.COLUMN.VAPMDENS(3) := 2.6684553E+00 ;
SYSTEM.COLUMN.VAPMDENS(4) := 2.7201430E+00 ;
SYSTEM.COLUMN.VAPMDENS(5) := 2.7716449E+00 ;
SYSTEM.COLUMN.VAPMDENS(6) := 2.8272809E+00 ;
SYSTEM.COLUMN.VAPMDENS(7) := 2.8827759E+00 ;
SYSTEM.COLUMN.VAPMDENS(8) := 2.9381325E+00 ;
SYSTEM.COLUMN.VAPMDENS(9) := 2.9933533E+00 ;
SYSTEM.COLUMN.VAPMDENS(10) := 3.0484406E+00 ;
SYSTEM.COLUMN.QR := 6.4072745E+06 ;
SYSTEM.COLUMN.DPSTAT(1) := 0.0000000E+00 ;
SYSTEM.COLUMN.DPSTAT(2) := 2.0759991E-02 ;
SYSTEM.COLUMN.DPSTAT(3) := 2.0726226E-02 ;
SYSTEM.COLUMN.DPSTAT(4) := 2.0689946E-02 ;
SYSTEM.COLUMN.DPSTAT(5) := 2.2496062E-02 ;
SYSTEM.COLUMN.DPSTAT(6) := 2.2483940E-02 ;
SYSTEM.COLUMN.DPSTAT(7) := 2.2471846E-02 ;
SYSTEM.COLUMN.DPSTAT(8) := 2.2459782E-02 ;
SYSTEM.COLUMN.DPSTAT(9) := 2.2447747E-02 ;
SYSTEM.COLUMN.DPSTAT(10) := 0.0000000E+00 ;
SYSTEM.COLUMN.XFEED(1) := 1.0000000E+00 ;
SYSTEM.COLUMN.XFEED(2) := 0.0000000E+00 ;
SYSTEM.COLUMN.XFEED(3) := 0.0000000E+00 ;
SYSTEM.COLUMN.XFEED(4) := 0.0000000E+00 ;
SYSTEM.COLUMN.DPTRAY(1) := 5.0229571E-05 ;
SYSTEM.COLUMN.DPTRAY(2) := 2.1986243E-02 ;
SYSTEM.COLUMN.DPTRAY(3) := 2.1952478E-02 ;
SYSTEM.COLUMN.DPTRAY(4) := 2.1916197E-02 ;
SYSTEM.COLUMN.DPTRAY(5) := 2.3722479E-02 ;
```

```
SYSTEM.COLUMN.DPTRAY(6) := 2.3710350E-02 ;
SYSTEM.COLUMN.DPTRAY(7) := 2.3698251E-02 ;
SYSTEM.COLUMN.DPTRAY(8) := 2.3686181E-02 ;
SYSTEM.COLUMN.DPTRAY(9) := 2.3674140E-02 ;
SYSTEM.COLUMN.DPTRAY(10) := 0.0000000E+00 ;
SYSTEM.COLUMN.LIQUIDENTHALPY(1) := -2.1350957E+04 ;
SYSTEM.COLUMN.LIQUIDENTHALPY(2) := -2.1350699E+04 ;
SYSTEM.COLUMN.LIQUIDENTHALPY(3) := -2.1238613E+04 ;
SYSTEM.COLUMN.LIQUIDENTHALPY(4) := -2.1128650E+04 ;
SYSTEM.COLUMN.LIQUIDENTHALPY(5) := -2.1020741E+04 ;
SYSTEM.COLUMN.LIQUIDENTHALPY(6) := -2.0905964E+04 ;
SYSTEM.COLUMN.LIQUIDENTHALPY(7) := -2.0793270E+04 ;
SYSTEM.COLUMN.LIQUIDENTHALPY(8) := -2.0682578E+04 ;
SYSTEM.COLUMN.LIQUIDENTHALPY(9) := -2.0573812E+04 ;
SYSTEM.COLUMN.LIQUIDENTHALPY(10) := -2.0466900E+04 ;
SYSTEM.COLUMN.LIQHEIGHT(1) := 0.0000000E+00 ;
SYSTEM.COLUMN.LIQHEIGHT(2) := 2.5313481E-01 ;
SYSTEM.COLUMN.LIQHEIGHT(3) := 2.5272311E-01 ;
SYSTEM.COLUMN.LIQHEIGHT(4) := 2.5228073E-01 ;
SYSTEM.COLUMN.LIQHEIGHT(5) := 2.7430342E-01 ;
SYSTEM.COLUMN.LIQHEIGHT(6) := 2.7415560E-01 ;
SYSTEM.COLUMN.LIQHEIGHT(7) := 2.7400814E-01 ;
SYSTEM.COLUMN.LIQHEIGHT(8) := 2.7386104E-01 ;
SYSTEM.COLUMN.LIQHEIGHT(9) := 2.7371429E-01 ;
SYSTEM.COLUMN.LIQHEIGHT(10) := 0.0000000E+00 ;
SYSTEM.COLUMN.HEAD(1) := 0.0000000E+00 ;
SYSTEM.COLUMN.HEAD(2) := 3.1348139E+00 ;
SYSTEM.COLUMN.HEAD(3) := 2.7231082E+00 ;
SYSTEM.COLUMN.HEAD(4) := 2.2807273E+00 ;
SYSTEM.COLUMN.HEAD(5) := 2.4303420E+01 ;
SYSTEM.COLUMN.HEAD(6) := 2.4155602E+01 ;
SYSTEM.COLUMN.HEAD(7) := 2.4008139E+01 ;
SYSTEM.COLUMN.HEAD(8) := 2.3861039E+01 ;
SYSTEM.COLUMN.HEAD(9) := 2.3714286E+01 ;
SYSTEM.COLUMN.HEAD(10) := 0.0000000E+00 ;
SYSTEM.COLUMN.K(1,1) := 1.0000000E+00 ;
SYSTEM.COLUMN.K(1,2) := 1.1186637E+00 ;
SYSTEM.COLUMN.K(1,3) := 3.6783189E-01 ;
SYSTEM.COLUMN.K(1,4) := 2.0422134E-01 ;
SYSTEM.COLUMN.K(2,1) := 1.0000000E+00 ;
SYSTEM.COLUMN.K(2,2) := 1.1186548E+00 ;
SYSTEM.COLUMN.K(2,3) := 3.6782906E-01 ;
SYSTEM.COLUMN.K(2,4) := 2.0422232E-01 ;
SYSTEM.COLUMN.K(3,1) := 1.0000000E+00 ;
SYSTEM.COLUMN.K(3,2) := 1.1147884E+00 ;
SYSTEM.COLUMN.K(3,3) := 3.6660608E-01 ;
SYSTEM.COLUMN.K(3,4) := 2.0464705E-01 ;
SYSTEM.COLUMN.K(4,1) := 1.0000000E+00 ;
SYSTEM.COLUMN.K(4,2) := 1.1110220E+00 ;
SYSTEM.COLUMN.K(4,3) := 3.6541455E-01 ;
SYSTEM.COLUMN.K(4,4) := 2.0506307E-01 ;
SYSTEM.COLUMN.K(5,1) := 1.0000000E+00 ;
SYSTEM.COLUMN.K(5,2) := 1.1073514E+00 ;
SYSTEM.COLUMN.K(5,3) := 3.6425316E-01 ;
```

```
SYSTEM.COLUMN.K(5,4) := 2.0547070E-01 ;
SYSTEM.COLUMN.K(6,1) := 1.0000000E+00 ;
SYSTEM.COLUMN.K(6,2) := 1.1034745E+00 ;
SYSTEM.COLUMN.K(6,3) := 3.6302637E-01 ;
SYSTEM.COLUMN.K(6,4) := 2.0590358E-01 ;
SYSTEM.COLUMN.K(7,1) := 1.0000000E+00 ;
SYSTEM.COLUMN.K(7,2) := 1.0996952E+00 ;
SYSTEM.COLUMN.K(7,3) := 3.6183030E-01 ;
SYSTEM.COLUMN.K(7,4) := 2.0632791E-01 ;
SYSTEM.COLUMN.K(8,1) := 1.0000000E+00 ;
SYSTEM.COLUMN.K(8,2) := 1.0960090E+00 ;
SYSTEM.COLUMN.K(8,3) := 3.6066354E-01 ;
SYSTEM.COLUMN.K(8,4) := 2.0674404E-01 ;
SYSTEM.COLUMN.K(9,1) := 1.0000000E+00 ;
SYSTEM.COLUMN.K(9,2) := 1.0924118E+00 ;
SYSTEM.COLUMN.K(9,3) := 3.5952480E-01 ;
SYSTEM.COLUMN.K(9,4) := 2.0715229E-01 ;
SYSTEM.COLUMN.K(10,1) := 1.0000000E+00 ;
SYSTEM.COLUMN.K(10,2) := 1.0888996E+00 ;
SYSTEM.COLUMN.K(10,3) := 3.5841286E-01 ;
SYSTEM.COLUMN.K(10,4) := 2.0755297E-01 ;
SYSTEM.COLUMN.L(1) := 1.2799411E+01 ;
SYSTEM.COLUMN.L(2) := 1.0700915E+01 ;
SYSTEM.COLUMN.L(3) := 8.6636449E+00 ;
SYSTEM.COLUMN.L(4) := 6.6406918E+00 ;
SYSTEM.COLUMN.L(5) := 2.3099596E+02 ;
SYSTEM.COLUMN.L(6) := 2.2889172E+02 ;
SYSTEM.COLUMN.L(7) := 2.2679894E+02 ;
SYSTEM.COLUMN.L(8) := 2.2471772E+02 ;
SYSTEM.COLUMN.L(9) := 2.2264778E+02 ;
SYSTEM.COLUMN.L(10) := 2.2005890E+00 ;
SYSTEM.COLUMN.FEED(1) := 0.0000000E+00 ;
SYSTEM.COLUMN.FEED(2) := 0.0000000E+00 ;
SYSTEM.COLUMN.FEED(3) := 0.0000000E+00 ;
SYSTEM.COLUMN.FEED(4) := 0.0000000E+00 ;
SYSTEM.COLUMN.FEED(5) := 1.5000000E+01 ;
SYSTEM.COLUMN.FEED(6) := 0.0000000E+00 ;
SYSTEM.COLUMN.FEED(7) := 0.0000000E+00 ;
SYSTEM.COLUMN.FEED(8) := 0.0000000E+00 ;
SYSTEM.COLUMN.FEED(9) := 0.0000000E+00 ;
SYSTEM.COLUMN.FEED(10) := 0.0000000E+00 ;
SYSTEM.COLUMN.M(1) := 0.0000000E+00 ;
SYSTEM.COLUMN.$M(1) := 0.0000000E+00 ;
SYSTEM.COLUMN.M(2) := 2.9237071E+00 ;
SYSTEM.COLUMN.$M(2) := -5.8431630E-05 ;
SYSTEM.COLUMN.M(3) := 2.9189519E+00 ;
SYSTEM.COLUMN.$M(3) := 1.9291184E-05 ;
SYSTEM.COLUMN.M(4) := 2.9138424E+00 ;
SYSTEM.COLUMN.$M(4) := 8.9519949E-05 ;
SYSTEM.COLUMN.M(5) := 3.1682045E+00 ;
SYSTEM.COLUMN.$M(5) := -1.1165674E-04 ;
SYSTEM.COLUMN.M(6) := 3.1664972E+00 ;
SYSTEM.COLUMN.$M(6) := -1.0041827E-04 ;
SYSTEM.COLUMN.M(7) := 3.1647940E+00 ;
```

```
SYSTEM.COLUMN.$M(7) := 3.0944158E-04 ;
SYSTEM.COLUMN.M(8) := 3.1630950E+00 ;
SYSTEM.COLUMN.$M(8) := -5.6979389E-05 ;
SYSTEM.COLUMN.M(9) := 3.1614000E+00 ;
SYSTEM.COLUMN.$M(9) := -7.1432749E-05 ;
SYSTEM.COLUMN.M(10) := 2.2005890E+01 ;
SYSTEM.COLUMN.$M(10) := -1.9333939E-05 ;
SYSTEM.COLUMN.LIQDENS(1) := 1.1000000E+01 ;
SYSTEM.COLUMN.LIQDENS(2) := 1.1000000E+01 ;
SYSTEM.COLUMN.LIQDENS(3) := 1.1000000E+01 ;
SYSTEM.COLUMN.LIQDENS(4) := 1.1000000E+01 ;
SYSTEM.COLUMN.LIQDENS(5) := 1.1000000E+01 ;
SYSTEM.COLUMN.LIQDENS(6) := 1.1000000E+01 ;
SYSTEM.COLUMN.LIQDENS(7) := 1.1000000E+01 ;
SYSTEM.COLUMN.LIQDENS(8) := 1.1000000E+01 ;
SYSTEM.COLUMN.LIQDENS(9) := 1.1000000E+01 ;
SYSTEM.COLUMN.LIQDENS(10) := 1.1000000E+01 ;
SYSTEM.COLUMN.VAPDENS(1) := 3.4427192E-02 ;
SYSTEM.COLUMN.VAPDENS(2) := 3.4428753E-02 ;
SYSTEM.COLUMN.VAPDENS(3) := 3.5111254E-02 ;
SYSTEM.COLUMN.VAPDENS(4) := 3.5791355E-02 ;
SYSTEM.COLUMN.VAPDENS(5) := 3.6469012E-02 ;
SYSTEM.COLUMN.VAPDENS(6) := 3.7201064E-02 ;
SYSTEM.COLUMN.VAPDENS(7) := 3.7931261E-02 ;
SYSTEM.COLUMN.VAPDENS(8) := 3.8659638E-02 ;
SYSTEM.COLUMN.VAPDENS(9) := 3.9386227E-02 ;
SYSTEM.COLUMN.VAPDENS(10) := 4.0111061E-02 ;
SYSTEM.COLUMN.P(2) := 1.0133002E+00 ;
SYSTEM.COLUMN.P(3) := 1.0352865E+00 ;
SYSTEM.COLUMN.P(4) := 1.0572390E+00 ;
SYSTEM.COLUMN.P(5) := 1.0791551E+00 ;
SYSTEM.COLUMN.P(6) := 1.1028776E+00 ;
SYSTEM.COLUMN.P(7) := 1.1265880E+00 ;
SYSTEM.COLUMN.P(8) := 1.1502862E+00 ;
SYSTEM.COLUMN.P(9) := 1.1739724E+00 ;
SYSTEM.COLUMN.P(10) := 1.1976465E+00 ;
SYSTEM.COLUMN.DISTILLOUT := 1.2799411E+01 ;
SYSTEM.COLUMN.VOLUMEHOLDUP(1) := 0.0000000E+00 ;
SYSTEM.COLUMN.VOLUMEHOLDUP(2) := 2.6579155E-01 ;
SYSTEM.COLUMN.VOLUMEHOLDUP(3) := 2.6535926E-01 ;
SYSTEM.COLUMN.VOLUMEHOLDUP(4) := 2.6489476E-01 ;
SYSTEM.COLUMN.VOLUMEHOLDUP(5) := 2.8801859E-01 ;
SYSTEM.COLUMN.VOLUMEHOLDUP(6) := 2.8786338E-01 ;
SYSTEM.COLUMN.VOLUMEHOLDUP(7) := 2.8770855E-01 ;
SYSTEM.COLUMN.VOLUMEHOLDUP(8) := 2.8755409E-01 ;
SYSTEM.COLUMN.VOLUMEHOLDUP(9) := 2.8740000E-01 ;
SYSTEM.COLUMN.VOLUMEHOLDUP(10) := 2.0005354E+00 ;
SYSTEM.COLUMN.T(1) := 3.5400141E+02 ;
SYSTEM.COLUMN.T(2) := 3.5400291E+02 ;
SYSTEM.COLUMN.T(3) := 3.5465344E+02 ;
SYSTEM.COLUMN.T(4) := 3.5529164E+02 ;
SYSTEM.COLUMN.T(5) := 3.5591793E+02 ;
SYSTEM.COLUMN.T(6) := 3.5658408E+02 ;
SYSTEM.COLUMN.T(7) := 3.5723813E+02 ;
```

```
SYSTEM.COLUMN.T(8) := 3.5788057E+02 ;
SYSTEM.COLUMN.T(9) := 3.5851183E+02 ;
SYSTEM.COLUMN.T(10) := 3.5913233E+02 ;
SYSTEM.COLUMN.V(1) := 0.0000000E+00 ;
SYSTEM.COLUMN.V(2) := 2.5598822E+01 ;
SYSTEM.COLUMN.V(3) := 2.3500268E+01 ;
SYSTEM.COLUMN.V(4) := 2.1463017E+01 ;
SYSTEM.COLUMN.V(5) := 1.9440153E+01 ;
SYSTEM.COLUMN.V(6) := 2.2879531E+02 ;
SYSTEM.COLUMN.V(7) := 2.2669097E+02 ;
SYSTEM.COLUMN.V(8) := 2.2459850E+02 ;
SYSTEM.COLUMN.V(9) := 2.2251722E+02 ;
SYSTEM.COLUMN.V(10) := 2.2044721E+02 ;
#SYSTEM.COLUMN.FEEDTEMP := 3.0000000E+02 ;
SYSTEM.COLUMN.DPDRY(1) := 0.0000000E+00 ;
SYSTEM.COLUMN.DPDRY(2) := 1.8678982E-09 ;
SYSTEM.COLUMN.DPDRY(3) := 1.5277914E-09 ;
SYSTEM.COLUMN.DPDRY(4) := 1.2295610E-09 ;
SYSTEM.COLUMN.DPDRY(5) := 1.6714719E-07 ;
SYSTEM.COLUMN.DPDRY(6) := 1.6085773E-07 ;
SYSTEM.COLUMN.DPDRY(7) := 1.5486215E-07 ;
SYSTEM.COLUMN.DPDRY(8) := 1.4914144E-07 ;
SYSTEM.COLUMN.DPDRY(9) := 1.4367913E-07 ;
SYSTEM.COLUMN.DPDRY(10) := 0.0000000E+00 ;
SYSTEM.COLUMN.X(1,1) := 1.0000000E+00 ;
SYSTEM.COLUMN.X(1,2) := 0.0000000E+00 ;
SYSTEM.COLUMN.X(1,3) := 0.0000000E+00 ;
SYSTEM.COLUMN.X(1,4) := 0.0000000E+00 ;
SYSTEM.COLUMN.X(2,1) := 1.0000000E+00 ;
SYSTEM.COLUMN.$X(2,1) := 1.0816707E-21 ;
SYSTEM.COLUMN.X(2,2) := 0.0000000E+00 ;
SYSTEM.COLUMN.$X(2,2) := 0.0000000E+00 ;
SYSTEM.COLUMN.X(2,3) := 0.0000000E+00 ;
SYSTEM.COLUMN.$X(2,3) := 0.0000000E+00 ;
SYSTEM.COLUMN.X(2,4) := 0.0000000E+00 ;
SYSTEM.COLUMN.$X(2,4) := 0.0000000E+00 ;
SYSTEM.COLUMN.X(3,1) := 1.0000000E+00 ;
SYSTEM.COLUMN.$X(3,1) := -1.5264713E-22 ;
SYSTEM.COLUMN.X(3,2) := 0.0000000E+00 ;
SYSTEM.COLUMN.$X(3,2) := 2.4869401E-23 ;
SYSTEM.COLUMN.X(3,3) := 0.0000000E+00 ;
SYSTEM.COLUMN.$X(3,3) := 0.0000000E+00 ;
SYSTEM.COLUMN.X(3,4) := 0.0000000E+00 ;
SYSTEM.COLUMN.$X(3,4) := 0.0000000E+00 ;
SYSTEM.COLUMN.X(4,1) := 1.0000000E+00 ;
SYSTEM.COLUMN.$X(4,1) := 2.1141297E-21 ;
SYSTEM.COLUMN.X(4,2) := 0.0000000E+00 ;
SYSTEM.COLUMN.$X(4,2) := -2.1385140E-23 ;
SYSTEM.COLUMN.X(4,3) := 0.0000000E+00 ;
SYSTEM.COLUMN.$X(4,3) := 0.0000000E+00 ;
SYSTEM.COLUMN.X(4,4) := 0.0000000E+00 ;
SYSTEM.COLUMN.$X(4,4) := 0.0000000E+00 ;
SYSTEM.COLUMN.X(5,1) := 1.0000000E+00 ;
SYSTEM.COLUMN.$X(5,1) := -1.2429451E-21 ;
```

```
SYSTEM.COLUMN.X(5,2) := 0.0000000E+00 ;
SYSTEM.COLUMN.$X(5,2) := -3.2446314E-24 ;
SYSTEM.COLUMN.X(5,3) := 0.0000000E+00 ;
SYSTEM.COLUMN.$X(5,3) := 0.0000000E+00 ;
SYSTEM.COLUMN.X(5,4) := 0.0000000E+00 ;
SYSTEM.COLUMN.$X(5,4) := 0.0000000E+00 ;
SYSTEM.COLUMN.X(6,1) := 1.0000000E+00 ;
SYSTEM.COLUMN.$X(6,1) := 8.0887492E-22 ;
SYSTEM.COLUMN.X(6,2) := 0.0000000E+00 ;
SYSTEM.COLUMN.$X(6,2) := 0.0000000E+00 ;
SYSTEM.COLUMN.X(6,3) := 0.0000000E+00 ;
SYSTEM.COLUMN.$X(6,3) := 0.0000000E+00 ;
SYSTEM.COLUMN.X(6,4) := 0.0000000E+00 ;
SYSTEM.COLUMN.$X(6,4) := 0.0000000E+00 ;
SYSTEM.COLUMN.X(7,1) := 1.0000000E+00 ;
SYSTEM.COLUMN.$X(7,1) := -6.1828866E-22 ;
SYSTEM.COLUMN.X(7,2) := 0.0000000E+00 ;
SYSTEM.COLUMN.$X(7,2) := 0.0000000E+00 ;
SYSTEM.COLUMN.X(7,3) := 0.0000000E+00 ;
SYSTEM.COLUMN.$X(7,3) := 4.1248869E-24 ;
SYSTEM.COLUMN.X(7,4) := 0.0000000E+00 ;
SYSTEM.COLUMN.$X(7,4) := 0.0000000E+00 ;
SYSTEM.COLUMN.X(8,1) := 1.0000000E+00 ;
SYSTEM.COLUMN.$X(8,1) := 1.2278487E-21 ;
SYSTEM.COLUMN.X(8,2) := 0.0000000E+00 ;
SYSTEM.COLUMN.$X(8,2) := 0.0000000E+00 ;
SYSTEM.COLUMN.X(8,3) := 0.0000000E+00 ;
SYSTEM.COLUMN.$X(8,3) := -4.1271025E-24 ;
SYSTEM.COLUMN.X(8,4) := 0.0000000E+00 ;
SYSTEM.COLUMN.$X(8,4) := 0.0000000E+00 ;
SYSTEM.COLUMN.X(9,1) := 1.0000000E+00 ;
SYSTEM.COLUMN.$X(9,1) := 1.0411440E-21 ;
SYSTEM.COLUMN.X(9,2) := 0.0000000E+00 ;
SYSTEM.COLUMN.$X(9,2) := 0.0000000E+00 ;
SYSTEM.COLUMN.X(9,3) := 0.0000000E+00 ;
SYSTEM.COLUMN.$X(9,3) := 0.0000000E+00 ;
SYSTEM.COLUMN.X(9,4) := 0.0000000E+00 ;
SYSTEM.COLUMN.$X(9,4) := 0.0000000E+00 ;
SYSTEM.COLUMN.X(10,1) := 1.0000000E+00 ;
SYSTEM.COLUMN.$X(10,1) := -8.9506882E-23 ;
SYSTEM.COLUMN.X(10,2) := 0.0000000E+00 ;
SYSTEM.COLUMN.$X(10,2) := 0.0000000E+00 ;
SYSTEM.COLUMN.X(10,3) := 0.0000000E+00 ;
SYSTEM.COLUMN.$X(10,3) := 0.0000000E+00 ;
SYSTEM.COLUMN.X(10,4) := 0.0000000E+00 ;
SYSTEM.COLUMN.$X(10,4) := 0.0000000E+00 ;
SYSTEM.COLUMN.Y(1,1) := 1.0000000E+00 ;
SYSTEM.COLUMN.Y(1,2) := 0.0000000E+00 ;
SYSTEM.COLUMN.Y(1,3) := 0.0000000E+00 ;
SYSTEM.COLUMN.Y(1,4) := 0.0000000E+00 ;
SYSTEM.COLUMN.Y(2,1) := 1.0000000E+00 ;
SYSTEM.COLUMN.Y(2,2) := 0.0000000E+00 ;
SYSTEM.COLUMN.Y(2,3) := 0.0000000E+00 ;
SYSTEM.COLUMN.Y(2,4) := 0.0000000E+00 ;
```

```
SYSTEM.COLUMN.Y(3,1) := 1.0000000E+00 ;
SYSTEM.COLUMN.Y(3,2) := 0.0000000E+00 ;
SYSTEM.COLUMN.Y(3,3) := 0.0000000E+00 ;
SYSTEM.COLUMN.Y(3,4) := 0.0000000E+00 ;
SYSTEM.COLUMN.Y(4,1) := 1.0000000E+00 ;
SYSTEM.COLUMN.Y(4,2) := 3.3822172E-24 ;
SYSTEM.COLUMN.Y(4,3) := 0.0000000E+00 ;
SYSTEM.COLUMN.Y(4,4) := 0.0000000E+00 ;
SYSTEM.COLUMN.Y(5,1) := 1.0000000E+00 ;
SYSTEM.COLUMN.Y(5,2) := 5.2878473E-25 ;
SYSTEM.COLUMN.Y(5,3) := 0.0000000E+00 ;
SYSTEM.COLUMN.Y(5,4) := 0.0000000E+00 ;
SYSTEM.COLUMN.Y(6,1) := 1.0000000E+00 ;
SYSTEM.COLUMN.Y(6,2) := 0.0000000E+00 ;
SYSTEM.COLUMN.Y(6,3) := 0.0000000E+00 ;
SYSTEM.COLUMN.Y(6,4) := 0.0000000E+00 ;
SYSTEM.COLUMN.Y(7,1) := 1.0000000E+00 ;
SYSTEM.COLUMN.Y(7,2) := 0.0000000E+00 ;
SYSTEM.COLUMN.Y(7,3) := 0.0000000E+00 ;
SYSTEM.COLUMN.Y(7,4) := 0.0000000E+00 ;
SYSTEM.COLUMN.Y(8,1) := 1.0000000E+00 ;
SYSTEM.COLUMN.Y(8,2) := 0.0000000E+00 ;
SYSTEM.COLUMN.Y(8,3) := 5.8123352E-26 ;
SYSTEM.COLUMN.Y(8,4) := 0.0000000E+00 ;
SYSTEM.COLUMN.Y(9,1) := 1.0000000E+00 ;
SYSTEM.COLUMN.Y(9,2) := 0.0000000E+00 ;
SYSTEM.COLUMN.Y(9,3) := 0.0000000E+00 ;
SYSTEM.COLUMN.Y(9,4) := 0.0000000E+00 ;
SYSTEM.COLUMN.Y(10,1) := 1.0000000E+00 ;
SYSTEM.COLUMN.Y(10,2) := 0.0000000E+00 ;
SYSTEM.COLUMN.Y(10,3) := 0.0000000E+00 ;
SYSTEM.COLUMN.Y(10,4) := 0.0000000E+00 ;
SYSTEM.COLUMN.FEEDENTHALPY := -4.5983100E+05 ;
SYSTEM.COLUMN.MWL(1) := 7.6000000E+01 ;
SYSTEM.COLUMN.MWL(2) := 7.6000000E+01 ;
SYSTEM.COLUMN.MWL(3) := 7.6000000E+01 ;
SYSTEM.COLUMN.MWL(4) := 7.6000000E+01 ;
SYSTEM.COLUMN.MWL(5) := 7.6000000E+01 ;
SYSTEM.COLUMN.MWL(6) := 7.6000000E+01 ;
SYSTEM.COLUMN.MWL(7) := 7.6000000E+01 ;
SYSTEM.COLUMN.MWL(8) := 7.6000000E+01 ;
SYSTEM.COLUMN.MWL(9) := 7.6000000E+01 ;
SYSTEM.COLUMN.MWL(10) := 7.6000000E+01 ;
SYSTEM.COLUMN.TOTALHOLDUP := 4.6586382E+01 ;
SYSTEM.COLUMN.$TOTALHOLDUP := 0.0000000E+00 ;
SYSTEM.COLUMN.QC := 7.9357010E+05 ;
SYSTEM.MAKEUPENTHALPY.TEMP := 3.0000000E+02 ;
SYSTEM.MAKEUPENTHALPY.SPECIFIC_ENTHALPY_VAPOR(1) := 3.4460000E+02 ;
SYSTEM.MAKEUPENTHALPY.SPECIFIC_ENTHALPY_VAPOR(2) := 4.0000000E+02 ;
SYSTEM.MAKEUPENTHALPY.SPECIFIC_ENTHALPY_VAPOR(3) := 3.2000000E+02 ;
SYSTEM.MAKEUPENTHALPY.SPECIFIC_ENTHALPY_VAPOR(4) := 3.1000000E+02 ;
SYSTEM.MAKEUPENTHALPY.SPECIFIC_ENTHALPY_LIQUID(1) := -3.0655400E+04 ;
SYSTEM.MAKEUPENTHALPY.SPECIFIC_ENTHALPY_LIQUID(2) := -2.5600000E+04 ;
SYSTEM.MAKEUPENTHALPY.SPECIFIC_ENTHALPY_LIQUID(3) := -2.7680000E+04 ;
```

```
SYSTEM.MAKEUPENTHALPY.SPECIFIC_ENTHALPY_LIQUID(4) := -3.3690000E+04 ;
SYSTEM.REACTOR.TOTALMOLS := 5.0000000E+01 ;
SYSTEM.REACTOR.$TOTALMOLS := 0.0000000E+00 ;
SYSTEM.REACTOR.TEMP := 3.0000000E+02 ;
SYSTEM.REACTOR.ENTHALPY := -1.5327700E+06 ;
SYSTEM.REACTOR.$ENTHALPY := -4.3344954E+06 ;
SYSTEM.REACTOR.TOTALFEED := 1.5000000E+01 ;
SYSTEM.REACTOR.QJACKET := 4.8152866E+04 ;
SYSTEM.REACTOR.NO_MOLS(1) := 5.0000000E+01 ;
SYSTEM.REACTOR.$NO_MOLS(1) := -7.5000000E+00 ;
SYSTEM.REACTOR.NO_MOLS(2) := 0.0000000E+00 ;
SYSTEM.REACTOR.$NO_MOLS(2) := 7.5000000E+00 ;
SYSTEM.REACTOR.NO_MOLS(3) := 0.0000000E+00 ;
SYSTEM.REACTOR.$NO_MOLS(3) := 0.0000000E+00 ;
SYSTEM.REACTOR.NO_MOLS(4) := 0.0000000E+00 ;
SYSTEM.REACTOR.$NO_MOLS(4) := 0.0000000E+00 ;
SYSTEM.REACTOR.VOLUME := 4.5454545E+00 ;
SYSTEM.REACTOR.CONCENTRATION(1) := 1.1000000E+01 ;
SYSTEM.REACTOR.CONCENTRATION(2) := 0.0000000E+00 ;
SYSTEM.REACTOR.CONCENTRATION(3) := 0.0000000E+00 ;
SYSTEM.REACTOR.CONCENTRATION(4) := 0.0000000E+00 ;
SYSTEM.REACTOR.FEED_A := 7.5000000E+00 ;
SYSTEM.REACTOR.FEED_B := 7.5000000E+00 ;
SYSTEM.REACTOR.FEED_C := 0.0000000E+00 ;
SYSTEM.REACTOR.X(1) := 1.0000000E+00 ;
SYSTEM.REACTOR.X(2) := 0.0000000E+00 ;
SYSTEM.REACTOR.X(3) := 0.0000000E+00 ;
SYSTEM.REACTOR.X(4) := 0.0000000E+00 ;
SYSTEM.REACTOR.FEED_D := 0.0000000E+00 ;
SYSTEM.REACTOR.FEEDENTHALPY(1) := -5.0553546E+05 ;
SYSTEM.REACTOR.FEEDENTHALPY(2) := -1.9395435E+05 ;
SYSTEM.REACTOR.FEEDENTHALPY(3) := -2.0971314E+05 ;
SYSTEM.REACTOR.FEEDENTHALPY(4) := -2.5524695E+05 ;
SYSTEM.REACTOR.SPECIFIC_ENTHALPY(1) := -3.0655400E+04 ;
SYSTEM.REACTOR.SPECIFIC_ENTHALPY(2) := -2.5600000E+04 ;
SYSTEM.REACTOR.SPECIFIC_ENTHALPY(3) := -2.7680000E+04 ;
SYSTEM.REACTOR.SPECIFIC_ENTHALPY(4) := -3.3690000E+04 ;
SYSTEM.REACTOR.REACTIONRATE(1) := 0.0000000E+00 ;
SYSTEM.REACTOR.REACTIONRATE(2) := 0.0000000E+00 ;
SYSTEM.VLE.VAPORENTHALPY(1) := 9.6490429E+03 ;
SYSTEM.VLE.VAPORENTHALPY(2) := 9.6493013E+03 ;
SYSTEM.VLE.VAPORENTHALPY(3) := 9.7613871E+03 ;
SYSTEM.VLE.VAPORENTHALPY(4) := 9.8713496E+03 ;
SYSTEM.VLE.VAPORENTHALPY(5) := 9.9792593E+03 ;
SYSTEM.VLE.VAPORENTHALPY(6) := 1.0094036E+04 ;
SYSTEM.VLE.VAPORENTHALPY(7) := 1.0206730E+04 ;
SYSTEM.VLE.VAPORENTHALPY(8) := 1.0317422E+04 ;
SYSTEM.VLE.VAPORENTHALPY(9) := 1.0426188E+04 ;
SYSTEM.VLE.VAPORENTHALPY(10) := 1.0533100E+04 ;
SYSTEM.VLE.TEMP(1) := 3.5400141E+02 ;
SYSTEM.VLE.TEMP(2) := 3.5400291E+02 ;
SYSTEM.VLE.TEMP(3) := 3.5465344E+02 ;
SYSTEM.VLE.TEMP(4) := 3.5529164E+02 ;
SYSTEM.VLE.TEMP(5) := 3.5591793E+02 ;
```

```
SYSTEM.VLE.TEMP(6) := 3.5658408E+02 ;
SYSTEM.VLE.TEMP(7) := 3.5723813E+02 ;
SYSTEM.VLE.TEMP(8) := 3.5788057E+02 ;
SYSTEM.VLE.TEMP(9) := 3.5851183E+02 ;
SYSTEM.VLE.TEMP(10) := 3.5913233E+02 ;
SYSTEM.VLE.LIQUIDENTHALPY(1) := -2.1350957E+04 ;
SYSTEM.VLE.LIQUIDENTHALPY(2) := -2.1350699E+04 ;
SYSTEM.VLE.LIQUIDENTHALPY(3) := -2.1238613E+04 ;
SYSTEM.VLE.LIQUIDENTHALPY(4) := -2.1128650E+04 ;
SYSTEM.VLE.LIQUIDENTHALPY(5) := -2.1020741E+04 ;
SYSTEM.VLE.LIQUIDENTHALPY(6) := -2.0905964E+04 ;
SYSTEM.VLE.LIQUIDENTHALPY(7) := -2.0793270E+04 ;
SYSTEM.VLE.LIQUIDENTHALPY(8) := -2.0682578E+04 ;
SYSTEM.VLE.LIQUIDENTHALPY(9) := -2.0573812E+04 ;
SYSTEM.VLE.LIQUIDENTHALPY(10) := -2.0466900E+04 ;
SYSTEM.VLE.K(1,1) := 1.0000000E+00 ;
SYSTEM.VLE.K(1,2) := 1.1186637E+00 ;
SYSTEM.VLE.K(1,3) := 3.6783189E-01 ;
SYSTEM VLE.K(1,4) := 2.0422134E-01 ;
SYSTEM.VLE.K(2,1) := 1.0000000E+00 ;
SYSTEM.VLE.K(2,2) := 1.1186548E+00 ;
SYSTEM.VLE.K(2,3) := 3.6782906E-01 ;
SYSTEM.VLE.K(2,4) := 2.0422232E-01 ;
SYSTEM.VLE.K(3,1) := 1.0000000E+00 ;
SYSTEM.VLE.K(3,2) := 1.1147884E+00 ;
SYSTEM.VLE.K(3,3) := 3.6660608E-01 ;
SYSTEM.VLE.K(3,4) := 2.0464705E-01 ;
SYSTEM.VLE.K(4,1) := 1.0000000E+00 ;
SYSTEM.VLE.K(4,2) := 1.1110220E+00 ;
SYSTEM.VLE.K(4,3) := 3.6541455E-01 ;
SYSTEM.VLE.K(4,4) := 2.0506307E-01 ;
SYSTEM.VLE.K(5,1) := 1.0000000E+00 ;
SYSTEM.VLE.K(5,2) := 1.1073514E+00 ;
SYSTEM.VLE.K(5,3) := 3.6425316E-01 ;
SYSTEM.VLE.K(5,4) := 2.0547070E-01 ;
SYSTEM.VLE.K(6,1) := 1.0000000E+00 ;
SYSTEM.VLE.K(6,2) := 1.1034745E+00 ;
SYSTEM.VLE.K(6,3) := 3.6302637E-01 ;
SYSTEM.VLE.K(6,4) := 2.0590358E-01 ;
SYSTEM.VLE.K(7,1) := 1.0000000E+00 ;
SYSTEM.VLE.K(7,2) := 1.0996952E+00 ;
SYSTEM.VLE.K(7,3) := 3.6183030E-01 ;
SYSTEM.VLE.K(7,4) := 2.0632791E-01 ;
SYSTEM.VLE.K(8,1) := 1.0000000E+00 ;
SYSTEM.VLE.K(8,2) := 1.0960090E+00 ;
SYSTEM.VLE.K(8,3) := 3.6066354E-01 ;
SYSTEM.VLE.K(8,4) := 2.0674404E-01 ;
SYSTEM.VLE.K(9,1) := 1.0000000E+00 ;
SYSTEM.VLE.K(9,2) := 1.0924118E+00 ;
SYSTEM.VLE.K(9,3) := 3.5952480E-01 ;
SYSTEM.VLE.K(9,4) := 2.0715229E-01 ;
SYSTEM.VLE.K(10,1) := 1.0000000E+00 ;
SYSTEM.VLE.K(10,2) := 1.0888996E+00 ;
SYSTEM.VLE.K(10,3) := 3.5841286E-01 ;
```

```
SYSTEM.VLE.K(10,4) := 2.0755297E-01 ;
SYSTEM.VLE.VAPORPRESSURE(1,1) := 1.0132500E+00 ;
SYSTEM.VLE.VAPORPRESSURE(1,2) := 1.1334860E+00 ;
SYSTEM.VLE.VAPORPRESSURE(1,3) := 3.7270566E-01 ;
SYSTEM.VLE.VAPORPRESSURE(1,4) := 2.0692728E-01 ;
SYSTEM.VLE.VAPORPRESSURE(2,1) := 1.0133002E+00 ;
SYSTEM.VLE.VAPORPRESSURE(2,2) := 1.1335331E+00 ;
SYSTEM.VLE.VAPORPRESSURE(2,3) := 3.7272127E-01 ;
SYSTEM.VLE.VAPORPRESSURE(2,4) := 2.0693853E-01 ;
SYSTEM.VLE.VAPORPRESSURE(3,1) := 1.0352865E+00 ;
SYSTEM.VLE.VAPORPRESSURE(3,2) := 1.1541254E+00 ;
SYSTEM.VLE.VAPORPRESSURE(3,3) := 3.7954232E-01 ;
SYSTEM.VLE.VAPORPRESSURE(3,4) := 2.1186832E-01 ;
SYSTEM.VLE.VAPORPRESSURE(4,1) := 1.0572390E+00 ;
SYSTEM.VLE.VAPORPRESSURE(4,2) := 1.1746158E+00 ;
SYSTEM.VLE.VAPORPRESSURE(4,3) := 3.8633050E-01 ;
SYSTEM.VLE.VAPORPRESSURE(4,4) := 2.1680067E-01 ;
SYSTEM.VLE.VAPORPRESSURE(5,1) := 1.0791551E+00 ;
SYSTEM.VLE.VAPORPRESSURE(5,2) := 1.1950039E+00 ;
SYSTEM.VLE.VAPORPRESSURE(5,3) := 3.9308568E-01 ;
SYSTEM.VLE.VAPORPRESSURE(5,4) := 2.2173476E-01 ;
SYSTEM.VLE.VAPORPRESSURE(6,1) := 1.1028776E+00 ;
SYSTEM.VLE.VAPORPRESSURE(6,2) := 1.2169973E+00 ;
SYSTEM.VLE.VAPORPRESSURE(6,3) := 4.0037366E-01 ;
SYSTEM.VLE.VAPORPRESSURE(6,4) := 2.2708645E-01 ;
SYSTEM.VLE.VAPORPRESSURE(7,1) := 1.1265880E+00 ;
SYSTEM.VLE.VAPORPRESSURE(7,2) := 1.2389034E+00 ;
SYSTEM.VLE.VAPORPRESSURE(7,3) := 4.0763366E-01 ;
SYSTEM.VLE.VAPORPRESSURE(7,4) := 2.3244654E-01 ;
SYSTEM.VLE.VAPORPRESSURE(8,1) := 1.1502862E+00 ;
SYSTEM.VLE.VAPORPRESSURE(8,2) := 1.2607240E+00 ;
SYSTEM.VLE.VAPORPRESSURE(8,3) := 4.1486630E-01 ;
SYSTEM.VLE.VAPORPRESSURE(8,4) := 2.3781482E-01 ;
SYSTEM.VLE.VAPORPRESSURE(9,1) := 1.1739724E+00 ;
SYSTEM.VLE.VAPORPRESSURE(9,2) := 1.2824613E+00 ;
SYSTEM.VLE.VAPORPRESSURE(9,3) := 4.2207220E-01 ;
SYSTEM.VLE.VAPORPRESSURE(9,4) := 2.4319108E-01 ;
SYSTEM.VLE.VAPORPRESSURE(10,1) := 1.1976465E+00 ;
SYSTEM.VLE.VAPORPRESSURE(10,2) := 1.3041169E+00 ;
SYSTEM.VLE.VAPORPRESSURE(10,3) := 4.2925192E-01 ;
SYSTEM.VLE.VAPORPRESSURE(10,4) := 2.4857510E-01 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_VAPOR(1,1) := 9.6490429E+03 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_VAPOR(1,2) := 1.1200282E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_VAPOR(1,3) := 8.9602256E+03 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_VAPOR(1,4) := 8.6802185E+03 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_VAPOR(2,1) := 9.6493013E+03 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_VAPOR(2,2) := 1.1200582E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_VAPOR(2,3) := 8.9604655E+03 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_VAPOR(2,4) := 8.6804509E+03 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_VAPOR(3,1) := 9.7613871E+03 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_VAPOR(3,2) := 1.1330687E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_VAPOR(3,3) := 9.0645499E+03 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_VAPOR(3,4) := 8.7812827E+03 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_VAPOR(4,1) := 9.8713496E+03 ;
```

```
SYSTEM.VLE.SPECIFIC_ENTHALPY_VAPOR(4,2) := 1.1458328E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_VAPOR(4,3) := 9.1666624E+03 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_VAPOR(4,4) := 8.8802042E+03 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_VAPOR(5,1) := 9.9792593E+03 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_VAPOR(5,2) := 1.1583586E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_VAPOR(5,3) := 9.2668688E+03 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_VAPOR(5,4) := 8.9772791E+03 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_VAPOR(6,1) := 1.0094036E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_VAPOR(6,2) := 1.1716815E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_VAPOR(6,3) := 9.3734520E+03 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_VAPOR(6,4) := 9.0805316E+03 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_VAPOR(7,1) := 1.0206730E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_VAPOR(7,2) := 1.1847626E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_VAPOR(7,3) := 9.4781009E+03 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_VAPOR(7,4) := 9.1819102E+03 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_VAPOR(8,1) := 1.0317422E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_VAPOR(8,2) := 1.1976113E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_VAPOR(8,3) := 9.5808907E+03 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_VAPOR(8,4) := 9.2814879E+03 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_VAPOR(9,1) := 1.0426188E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_VAPOR(9,2) := 1.2102365E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_VAPOR(9,3) := 9.6818924E+03 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_VAPOR(9,4) := 9.3793332E+03 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_VAPOR(10,1) := 1.0533100E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_VAPOR(10,2) := 1.2226466E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_VAPOR(10,3) := 9.7811726E+03 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_VAPOR(10,4) := 9.4755110E+03 ;
SYSTEM.VLE.P(1) := 1.0132500E+00 ;
SYSTEM.VLE.P(2) := 1.0133002E+00 ;
SYSTEM.VLE.P(3) := 1.0352865E+00 ;
SYSTEM.VLE.P(4) := 1.0572390E+00 ;
SYSTEM.VLE.P(5) := 1.0791551E+00 ;
SYSTEM.VLE.P(6) := 1.1028776E+00 ;
SYSTEM.VLE.P(7) := 1.1265880E+00 ;
SYSTEM.VLE.P(8) := 1.1502862E+00 ;
SYSTEM.VLE.P(9) := 1.1739724E+00 ;
SYSTEM.VLE.P(10) := 1.1976465E+00 ;
SYSTEM.VLE.X(1,1) := 1.0000000E+00 ;
SYSTEM.VLE.X(1,2) := 0.0000000E+00 ;
SYSTEM.VLE.X(1,3) := 0.0000000E+00 ;
SYSTEM.VLE.X(1,4) := 0.0000000E+00 ;
SYSTEM.VLE.X(2,1) := 1.0000000E+00 ;
SYSTEM.VLE.X(2,2) := 0.0000000E+00 ;
SYSTEM.VLE.X(2,3) := 0.0000000E+00 ;
SYSTEM.VLE.X(2,4) := 0.0000000E+00 ;
SYSTEM.VLE.X(3,1) := 1.0000000E+00 ;
SYSTEM.VLE.X(3,2) := 0.0000000E+00 ;
SYSTEM.VLE.X(3,3) := 0.0000000E+00 ;
SYSTEM.VLE.X(3,4) := 0.0000000E+00 ;
SYSTEM.VLE.X(4,1) := 1.0000000E+00 ;
SYSTEM.VLE.X(4,2) := 0.0000000E+00 ;
SYSTEM.VLE.X(4,3) := 0.0000000E+00 ;
SYSTEM.VLE.X(4,4) := 0.0000000E+00 ;
SYSTEM.VLE.X(5,1) := 1.0000000E+00 ;
```

```
SYSTEM.VLE.X(5,2) := 0.0000000E+00 ;
SYSTEM.VLE.X(5,3) := 0.0000000E+00 ;
SYSTEM.VLE.X(5,4) := 0.0000000E+00 ;
SYSTEM.VLE.X(6,1) := 1.0000000E+00 ;
SYSTEM.VLE.X(6,2) := 0.0000000E+00 ;
SYSTEM.VLE.X(6,3) := 0.0000000E+00 ;
SYSTEM.VLE.X(6,4) := 0.0000000E+00 ;
SYSTEM.VLE.X(7,1) := 1.0000000E+00 ;
SYSTEM.VLE.X(7,2) := 0.0000000E+00 ;
SYSTEM.VLE.X(7,3) := 0.0000000E+00 ;
SYSTEM.VLE.X(7,4) := 0.0000000E+00 ;
SYSTEM.VLE.X(8,1) := 1.0000000E+00 ;
SYSTEM.VLE.X(8,2) := 0.0000000E+00 ;
SYSTEM.VLE.X(8,3) := 0.0000000E+00 ;
SYSTEM.VLE.X(8,4) := 0.0000000E+00 ;
SYSTEM.VLE.X(9,1) := 1.0000000E+00 ;
SYSTEM.VLE.X(9,2) := 0.0000000E+00 ;
SYSTEM.VLE.X(9,3) := 0.0000000E+00 ;
SYSTEM.VLE.X(9,4) := 0.0000000E+00 ;
SYSTEM.VLE.X(10,1) := 1.0000000E+00 ;
SYSTEM.VLE.X(10,2) := 0.0000000E+00 ;
SYSTEM.VLE.X(10,3) := 0.0000000E+00 ;
SYSTEM.VLE.X(10,4) := 0.0000000E+00 ;
SYSTEM.VLE.Y(1,1) := 1.0000000E+00 ;
SYSTEM.VLE.Y(1,2) := 0.0000000E+00 ;
SYSTEM.VLE.Y(1,3) := 0.0000000E+00 ;
SYSTEM.VLE.Y(1,4) := 0.0000000E+00 ;
SYSTEM.VLE.Y(2,1) := 1.0000000E+00 ;
SYSTEM.VLE.Y(2,2) := 0.0000000E+00 ;
SYSTEM.VLE.Y(2,3) := 0.0000000E+00 ;
SYSTEM.VLE.Y(2,4) := 0.0000000E+00 ;
SYSTEM.VLE.Y(3,1) := 1.0000000E+00 ;
SYSTEM.VLE.Y(3,2) := 0.0000000E+00 ;
SYSTEM.VLE.Y(3,3) := 0.0000000E+00 ;
SYSTEM.VLE.Y(3,4) := 0.0000000E+00 ;
SYSTEM.VLE.Y(4,1) := 1.0000000E+00 ;
SYSTEM.VLE.Y(4,2) := 3.3822172E-24 ;
SYSTEM.VLE.Y(4,3) := 0.0000000E+00 ;
SYSTEM.VLE.Y(4,4) := 0.0000000E+00 ;
SYSTEM.VLE.Y(5,1) := 1.0000000E+00 ;
SYSTEM.VLE.Y(5,2) := 5.2878473E-25 ;
SYSTEM.VLE.Y(5,3) := 0.0000000E+00 ;
SYSTEM.VLE.Y(5,4) := 0.0000000E+00 ;
SYSTEM.VLE.Y(6,1) := 1.0000000E+00 ;
SYSTEM.VLE.Y(6,2) := 0.0000000E+00 ;
SYSTEM.VLE.Y(6,3) := 0.0000000E+00 ;
SYSTEM.VLE.Y(6,4) := 0.0000000E+00 ;
SYSTEM.VLE.Y(7,1) := 1.0000000E+00 ;
SYSTEM.VLE.Y(7,2) := 0.0000000E+00 ;
SYSTEM.VLE.Y(7,3) := 0.0000000E+00 ;
SYSTEM.VLE.Y(7,4) := 0.0000000E+00 ;
SYSTEM.VLE.Y(8,1) := 1.0000000E+00 ;
SYSTEM.VLE.Y(8,2) := 0.0000000E+00 ;
SYSTEM.VLE.Y(8,3) := 5.8123352E-26 ;
```

```
SYSTEM.VLE.Y(8,4) := 0.0000000E+00 ;
SYSTEM.VLE.Y(9,1) := 1.0000000E+00 ;
SYSTEM.VLE.Y(9,2) := 0.0000000E+00 ;
SYSTEM.VLE.Y(9,3) := 0.0000000E+00 ;
SYSTEM.VLE.Y(9,4) := 0.0000000E+00 ;
SYSTEM.VLE.Y(10,1) := 1.0000000E+00 ;
SYSTEM.VLE.Y(10,2) := 0.0000000E+00 ;
SYSTEM.VLE.Y(10,3) := 0.0000000E+00 ;
SYSTEM.VLE.Y(10,4) := 0.0000000E+00 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_LIQUID(1,1) := -2.1350957E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_LIQUID(1,2) := -1.4799718E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_LIQUID(1,3) := -1.9039774E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_LIQUID(1,4) := -2.5319781E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_LIQUID(2,1) := -2.1350699E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_LIQUID(2,2) := -1.4799418E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_LIQUID(2,3) := -1.9039535E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_LIQUID(2,4) := -2.5319549E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_LIQUID(3,1) := -2.1238613E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_LIQUID(3,2) := -1.4669313E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_LIQUID(3,3) := -1.8935450E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_LIQUID(3,4) := -2.5218717E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_LIQUID(4,1) := -2.1128650E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_LIQUID(4,2) := -1.4541672E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_LIQUID(4,3) := -1.8833338E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_LIQUID(4,4) := -2.5119796E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_LIQUID(5,1) := -2.1020741E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_LIQUID(5,2) := -1.4416414E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_LIQUID(5,3) := -1.8733131E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_LIQUID(5,4) := -2.5022721E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_LIQUID(6,1) := -2.0905964E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_LIQUID(6,2) := -1.4283185E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_LIQUID(6,3) := -1.8626548E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_LIQUID(6,4) := -2.4919468E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_LIQUID(7,1) := -2.0793270E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_LIQUID(7,2) := -1.4152374E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_LIQUID(7,3) := -1.8521899E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_LIQUID(7,4) := -2.4818090E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_LIQUID(8,1) := -2.0682578E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_LIQUID(8,2) := -1.4023887E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_LIQUID(8,3) := -1.8419109E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_LIQUID(8,4) := -2.4718512E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_LIQUID(9,1) := -2.0573812E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_LIQUID(9,2) := -1.3897635E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_LIQUID(9,3) := -1.8318108E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_LIQUID(9,4) := -2.4620667E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_LIQUID(10,1) := -2.0466900E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_LIQUID(10,2) := -1.3773534E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_LIQUID(10,3) := -1.8218827E+04 ;
SYSTEM.VLE.SPECIFIC_ENTHALPY_LIQUID(10,4) := -2.4524489E+04 ;
SYSTEM.JACKET.TWATER_OUT := 2.9946497E+02 ;
SYSTEM.JACKET.QJACKET := 4.8152866E+04 ;
SYSTEM.JACKET.TEMP_REACTOR := 3.0000000E+02 ;
SYSTEM.LIQENTHALPY.TEMP := 3.0000000E+02 ;
SYSTEM.LIQENTHALPY.SPECIFIC_ENTHALPY_VAPOR(1) := 3.4460000E+02 ;
```

```
SYSTEM.LIQENTHALPY.SPECIFIC_ENTHALPY_VAPOR(2) := 4.0000000E+02 ;
SYSTEM.LIQENTHALPY.SPECIFIC_ENTHALPY_VAPOR(3) := 3.2000000E+02 ;
SYSTEM.LIQENTHALPY.SPECIFIC_ENTHALPY_VAPOR(4) := 3.1000000E+02 ;
SYSTEM.LIQENTHALPY.SPECIFIC_ENTHALPY_LIQUID(1) := -3.0655400E+04 ;
SYSTEM.LIQENTHALPY.SPECIFIC_ENTHALPY_LIQUID(2) := -2.5600000E+04 ;
SYSTEM.LIQENTHALPY.SPECIFIC_ENTHALPY_LIQUID(3) := -2.7680000E+04 ;
SYSTEM.LIQENTHALPY.SPECIFIC_ENTHALPY_LIQUID(4) := -3.3690000E+04 ;

  INITIAL
    TOTAL_C_OUT = 0;
    TOTAL_D_OUT = 0;
     WITHIN System.Reactor DO
       No_Mols(A) = 50 ;
       No_Mols(B) = 0 ;
       No_Mols(C) = 0.0 ;
       No_Mols(D) = 0.0 ;
       Temp=300 ;
     END
    WITHIN System.Column DO
      #FOR I:=2 TO NSTAGE-1 DO
      #  $M(I)=0;
      #END
      FOR I:=2 TO NSTAGE DO
        X(I,D)= 0.0 ;
        X(I,B)= 0.0 ;
        X(I,C)= 0.0 ;
      END
      TotalHoldup=4.6586382E+01 ;
      M(2)=2.9237071E+00  ;
      M(3)=2.9189519E+00  ;
      M(4)=2.9138424E+00  ;
      M(5)=3.1682045E+00  ;
      M(6)=3.1664972E+00  ;
      M(7)=3.1647940E+00  ;
      M(8)=3.1630950E+00  ;
      M(9)=3.1614000E+00  ;
      FOR I:=2 TO NSTAGE DO
        SIGMA(X(I,))=1 ;
        #SIGMA(K(I,)*X(I,))=1 ;
      END
    END

  # FINAL
     #TOTAL_C_OUT>=130 ;
     #ChangeMols_A_Out^2 +ChangeMols_B_Out^2+
     #        ChangeMols_C_Out^2 +ChangeMols_D_Out^2<=0.00001;
     #SYSTEM.REACTOR.$ENTHALPY<=0.01 ;
     #SYSTEM.REACTOR.$NO_Mols(1)<=0.01 ;
     #SYSTEM.REACTOR.$NO_Mols(2)<=0.01 ;
     #SYSTEM.REACTOR.$NO_Mols(3)<=0.01 ;
     #SYSTEM.REACTOR.$NO_Mols(4)<=0.01 ;
     #MOLS_D_OUT<=0.065 ;

  SCHEDULE
```

```
     CONTINUE FOR 100
    #CONTINUE FOR FINAL_TIME

END
```

References

[1] R. ALLGOR, *Modeling and Computational Issues in the Development of Batch Processes*, PhD thesis, Department of Chemical Engineering, Massachusetts Institute of Technology, Cambridge, MA, 1997.

[2] R. ALLGOR AND P. BARTON, *Accurate solution of batch distillation models: Implications of ill-conditioned corrector iterations*, AIChE 1995 Annual Meeting, Miami, (1995).

[3] ——, *Mixed-integer dynamic optimization I: Problem formulation*, Computers chem. Engng., (submitted, 1997).

[4] U. ASCHER AND L. PETZOLD, *Collocation software for boundary-value differential-algebraic equations*, SIAM J. Sci. Comput., 15 (1994), pp. 938–952.

[5] R. BACHMAN, L. BRÜLL, T. MRZIGLOD, AND U. PALLASKE, *On methods for reducing the index of differential algebraic equations*, Computers chem. Engng., 14 (1990), pp. 1271–1273.

[6] A. BACK, J. GUCKENHEIMER, AND M. MYERS, *A dynamical simulation facility for hybrid systems*, Hybrid Systems, Lecture Notes in Computer Science, 736 (1993), pp. 255–267.

[7] J. BANGA AND J. CASARES, *Integrated controlled random search: Application to a wastewater treatment plant model*, Inst. Chem. Eng. Symp. Ser., 100 (1987), pp. 183–192.

[8] J. BANGA AND W. SEIDER, *Global optimization of chemical processes using stochastic algorithms*, in State of the Art in Global Optimization: Computational Methods and Applications, Princeton, NJ, April 1995, Princeton University.

[9] J. BARLOW AND U. VEMULAPATI, *Rank detection methods for sparse matrices*, SIAM J. Matrix Anal. Appl., 13 (1992), pp. 1279–1297.

[10] M. BARRERA AND L. EVANS, *Optimal design and operation of batch processes*, Chem. Eng. Comm., 82 (1989), pp. 45–66.

[11] A. BARRLUND AND B. KÄGSTRÖM, *Analytical and numerical solutions to higher index linear variable coefficient DAE systems*, J. Comp. Appl. Math, 31 (1990), pp. 305–330.

[12] M. BARTHOLOMEW-BIGGS, *A penalty method for point and path state constraints in trajectory optimization*, Opt. Cont. Appl. and Meth., 16 (1995), pp. 291–297.

[13] P. BARTON, *The Modelling and Simulation of Combined Discrete/Continuous Processes*, PhD thesis, University of London, London, U.K., 1992.

[14] P. BARTON, R. ALLGOR, W. FEEHERY, AND S. GALÁN, *Dynamic optimization in a discontinuous world*, Ind. Eng. Chem. Res., 37 (1998).

[15] P. BARTON AND C. PANTELIDES, *Modeling of combined discrete/continuous processes*, AIChE Journal, 40 (1994), pp. 966–979.

[16] M. BAZARAA, H. SHERALI, AND C. SHETTY, *Nonlinear Programming: Theory and Applications*, Wiley, New York, 1993.

[17] J. BEAUMGARTE, *Stabilization of constraints and numerical solutions to higher index linear variable coefficient DAE systems*, J. Comp. Appl. Math., 31 (1990), pp. 305–330.

[18] R. BELLMAN, *Dynamic Programming*, Princeton University Press, Princeton, New Jersey, 1957.

[19] T. BHATIA AND L. BIEGLER, *Dynamic optimization in the planning and scheduling of multi-product batch plants*, Ind. Eng. Chem. Res., 35 (1996), pp. 2234–2246.

[20] J. BOSTON, H. BRITT, S. JIRAPONGHAN, AND V. SHAH, *An advanced system for simulation of batch distillation columns*, in Proceedings of the Conference on Computer-Aided Process Design, R. Mah and W. Seader, eds., 1981, pp. 203–207.

[21] K. BRENAN, S. CAMPBELL, AND L. PETZOLD, *Numerical Solution of Initial Value Problems in Differential-Algebraic Equations*, SIAM, Philadelphia, PA, 1996.

[22] L. BRÜLL AND R. PALLASKE, *On consistent initialization of differential-algebraic equations with discontinuities*, in Proceedings of the Fourth European Conference on Mathematics in Industry, Teubner, Stuttgart, 1992, pp. 213–217.

[23] R. BRUSCH AND R. SCHAPELLE, *Solution of highly constrained optimal control problems using nonlinear programming*, AIAA J., 11 (1973), pp. 135–136.

[24] A. BRYSON AND Y. HO, *Applied Optimal Control*, Hemisphere, New York, 1975.

[25] P. BUJAKIEWICZ, *Maximum weighted matching for high-index differential algebraic equations*, PhD thesis, Technical University of Delft, The Netherlands, 1994.

[26] S. CAMPBELL, *Consistent initial conditions for singular nonlinear systems*, Circuits, Systems and Signal Processing, 2 (1983), pp. 45–55.

[27] ——, *Least squares completions for nonlinear differential algebraic equations*, Numer. Math., 65 (1993), pp. 77–94.

[28] ——, *Numerical methods for unstructured higher-index DAEs*, Annals of Numerical Mathematics, 1 (1994), pp. 265–278.

[29] S. CAMPBELL AND E. GRIEPENTROG, *Solvability of general differential-algebraic equations*, SIAM J. Sci. Comput., 16 (1995), pp. 257–270.

[30] M. CARACOTSIOS AND W. STEWART, *Sensitivity analysis of initial value problems with mixed ODEs and algebraic constraints*, Computers chem. Engng., 9 (1985), pp. 359–365.

[31] M. CARVER, *Efficient integration over discontinuities in ordinary differential equation simulations*, Math. Comput. Sim., XX (1978), pp. 190–196.

[32] J. CASARES AND J. RODRIGUEZ, *Analysis and evolution of a wastewater treatment plant by stochastic optimization*, Appl. Math. Model, 13 (1989), pp. 420–424.

[33] M. CHARALAMBIDES, *Optimal Design of Integrated Batch Processes*, PhD thesis, University of London, London, U.K., 1996.

[34] J. CUTHRELL AND L. BIEGLER, *On the optimization of differential-algebraic process systems*, AIChE J., 33 (1987), pp. 1257–1270.

[35] W. DE BACKER, *Jump conditions for sensitivity coefficients*, in Sensitivity methods in control theory, L. Radanović, ed., Dubrovnik, Aug. 31 - Sep. 5 1964, Pergamon Press, pp. 168–175.

[36] M. DE PINHO, R. SARGENT, AND R. VINTNER, *Optimal control of nonlinear DAE systems*, in Proceedings of 32nd IEEE Conference on Decision and Control, 1993.

[37] W. DENHAM AND A. BRYSON, *Optimal programming problems with inequality constraints. II: Solution by steepest ascent*, AIAA J., 2 (1964), pp. 25–34.

[38] J. DENNIS AND R. SCHNABEL, *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*, Prentice Hall, Englewood Cliffs, NJ, 1983.

[39] P. DEUFLHARD, *Recent advances in multiple shooting techniques*, in Computational Techniques for Ordinary Differential Equations, Academic Press, 1980, pp. 217–272.

[40] P. DEUFLHARD, E. HAIRER, AND J. ZUGCK, *One-step and extrapolation methods for differential-algebraic systems*, Numer. Math., 51 (1987), pp. 501–516.

[41] L. DIXON AND M. BARTHOLOMEW-BIGGS, *Adjoint-control transformations for solving practical optimal control problems*, Opt. Cont. Appl. and Meth., 2 (1981), pp. 365–381.

[42] S. DREYFUS, *Variational problems with state variable inequality constraints*, J. Math. Anal. Appl., 4 (1962), pp. 297–308.

[43] I. DUFF, *On algorithms for obtaining a maximum transversal*, ACM Trans. Math. Software, 7 (1981), pp. 315–330.

[44] I. DUFF, A. ERISMAN, AND J. REID, *Direct Methods for Sparse Matrices*, Oxford University Press, 1986.

[45] I. DUFF AND J. REID, *MA48, a FORTRAN code for direct solution of sparse unsymmetric linear systems of equations*, Tech. Rep. RAL-93-072, Rutherford Appleton Laboratory, Oxon, UK, 1993.

[46] W. FEEHERY, J. BANGA, AND P. BARTON, *A novel approach to dynamic optimization of ODE and DAE systems as high-index problems*, AIChE Annual Meeting, Miami, (1995).

[47] W. FEEHERY AND P. BARTON, *A differentiation-based approach to dynamic simulation and optimization with high-index differential-algebraic equations*, in Computational Differentiation: Techniques, Applications, and Tools, M. Berz, C. Bischof, G. Corliss, and A. Griewank, eds., SIAM, Philadelphia, Penn., 1996, pp. 239–252.

[48] ——, *Dynamic simulation and optimization with inequality path constraints*, Computers chem. Engng., 20 (1996), pp. S707–S712.

[49] W. FEEHERY, J. TOLSMA, AND P. BARTON, *Efficient sensitivity analysis of large-scale differential-algebraic equations*, Applied Numerical Mathematics, 25 (1997), pp. 41–54.

[50] R. FELDER AND R. ROUSSEAU, *Elementary Principles of Chemical Processes*, Wiley, New York, 1986.

[51] A. FENG, C. HOLLAND, AND S. GALLUN, *Development and comparison of a generalized implicit Runge-Kutta method with Gear's method for systems of coupled differential and algebraic equations*, Computers chem. Engng., 8 (1984), pp. 51–59.

[52] C. FLOUDAS, *Nonlinear and Mixed-Integer Optimization*, Oxford University Press, New York, 1995.

[53] R. FRANKS, *Modeling and Simulation in Chemical Engineering*, Wiley-Interscience, New York, 1972.

[54] C. FÜHRER AND B. LEIMKUHLER, *Numerical solution of differential-algebraic equations for constrained mechanical motion*, Numer. Math., 59 (1991), pp. 55–69.

[55] S. GALÁN, W. FEEHERY, AND P. BARTON, *Parametric sensitivity functions for hybrid discrete/continuous systems*, submitted to Applied Numerical Mathematics, (1997).

[56] R. GANI AND T. CAMERON, *Modelling for dynamic simulation of chemical processes- the index problem*, Chem. Engng. Sci., 47 (1992), pp. 1311–1315.

[57] C. GEAR, *Maintaining solution invariants in the numerical solution of ODEs*, SIAM J. Sci. Statist. Comput., 7 (1986), pp. 734–743.

[58] ——, *Differential-algebraic equation index transformation*, SIAM J. Sci. Statist. Comput., 9 (1988), pp. 39–47.

[59] C. GEAR, G. GUPTA, AND B. LEIMKUHLER, *Automatic integration of the Euler-Lagrange equations with constraints*, J. Comp. Appl. Math., 12 (1985), pp. 77–90.

[60] C. GEAR AND L. PETZOLD, *ODE methods for the solution of differential-algebraic equations*, SIAM J. Numer. Anal., 21 (1984), pp. 716–728.

[61] P. GILL, W. MURRAY, AND M. SAUNDERS, *Large-scale SQP methods and their application in trajectory optimization*, in Computational Optimal Control, R. Bulirsch and D. Kraft, eds., Birkhäuser Verlag, Basel, 1994, ch. 1, pp. 29–42.

[62] P. GILL, W. MURRAY, AND M. WRIGHT, *Practical Optimization*, Academic Press, New York, 1981.

[63] C. GOH AND L. TEO, *Control parametrization: A unified approach to optimal control problems with general constraints*, Automatica, 24 (1988), pp. 3–18.

[64] V. GOPAL AND L. BIEGLER, *A successive linear programming approach for initialization and reinitialization after discontinuities of differential algebraic equations*, SIAM Journal on Scientific Computing, (to appear).

[65] A. GRIEWANK, *On automatic differentiation*, in Mathematical Programming: Recent Developments and Applications, T. Coleman and Y. Li, eds., Kluwer Academic Publishers, 1989, pp. 83–108.

[66] D. GRITSIS, *The Dynamic Simulation and Optimal Control of Systems Described by Index Two Differential-Algebraic Equations*, PhD thesis, University of London, 1990

[67] D. GRITSIS, C. PANTELIDES, AND R. SARGENT, *Optimal control of systems described by index-2 differential-algebriac equations*, SIAM J. Sci. Comput., 16 (1995), pp. 1349–1366.

[68] T. GRONWALL, *Note on the derivatives with respect to a parameter of the solutions of a system of differential equations*, Annals of Mathematics, 20 (1919), pp. 292–296.

[69] J. GUCKENHEIMER AND S. JOHNSON, *Planar hybrid systems*, Hybrid Systems II, Lecture Notes in Computer Science, 999 (1995), pp. 202–225.

[70] E. HAIRER, C. LUBICH, AND M. ROCHE, *The numerical solution of differential-algebriac systems by Runge-Kutta methods*, in Lecture Notes in Mathematics, Springer-Verlag, 1989.

[71] E. HAIRER, S. NØRSETT, AND G. WANNER, *Solving Ordinary Differential Equations*, vol. I Nonstiff Problems, Springer-Verlag, Berlin, 1987. pp. 94-98.

[72] E. HAUG AND P. EHLE, *Second-order design sensitivity analysis of mechanical system dynamics*, Internat. J. Numer. Methods Engrg., 18 (1982), pp. 1699–1717.

[73] E. HAUG, R. WEHAGE, AND N. BARMAN, *Design sensitivity analysis of planar mechanism and machine dynamics*, Trans. ASME, 103 (1981).

[74] D. JACOBSON AND M. LELE, *A transformation technique for optimal control problems with a state variable inequality constraint*, IEEE Trans. Automatic Control, 5 (1969), pp. 457–464.

[75] D. JACOBSON, M. LELE, AND J. SPEYER, *New necessary conditions of optimality for control problems with state variable inequality constraints*, J. Math. Anal. Appl., 35 (1971), pp. 255–284.

[76] R. JARVIS AND C. PANTELIDES, *A differentiation-free algorithm for solving high-index DAE systems*, AIChE Annual Meeting, Miami, (1992).

[77] B. KEEPING AND C. PANTELIDES, *On the implementation of optimisation algorithms for large-scale transient systems on a MIMD computer*, AIChE Annual Meeting, Miami, (1995).

[78] J. KEIPER AND C. GEAR, *The analysis of generalized backward-difference formula methods applied to Hessenberg form differential-algebraic equations*, SIAM J. Numer. Anal., 28 (1991), pp. 833–858.

[79] H. KELLER, *Numerical Methods for Two-Point Boundary Value Problems*, Blaisdell, Waltham, MA, 1968.

[80] H. KELLEY, *A trajectory optimization technique based upon the theory of the second variation*, Progress in Astronautics and Aeronautics, 14 (1964), pp. 559–582.

[81] E. KIKKINIDES AND R. YANG, *Simultaneous SO_2/NO_2 removal and SO_2 recovery from flue gas by pressure swing adsorption*, Ind. Eng. Chem. Res., 30 (1991), pp. 1981–1989.

[82] D. KIRK, *Optimal Control Theory: An Introduction*, Prentice-Hall, Englewood Cliffs, New Jersey, 1970.

[83] P. KOKOTOVIĆ AND R. RUTMAN, *Sensitivity of automatic control systems (survey)*, Automation and Remote Control, 4 (1965), pp. 727–749.

[84] D. KRAFT, *On converting optimal control problems into nonlinear programming problems*, Comp. Math. Prog., 15 (1985), pp. 261–280.

[85] ——, *Algorithm 733: TOMP- Fortran modules for optimal control calculations*, ACM Trans. Math. Software, 20 (1994), pp. 263–281.

[86] M. KRAMER AND J. LEIS, *The simultaneous solution and sensitivity analysis of systems described by ordinary differential equations*, ACM Trans. Math Software, 14 (1988), pp. 45–60.

[87] A. LEFKOPOULOS AND M. STADTHERR, *Index analysis of unsteady-state chemical processes- I. an algorithm for problem formulation*, Computers chem. Engng., 17 (1993), pp. 399–413.

[88] J. LEIS AND M. KRAMER, *Sensitivity analysis of systems of differential and algebraic equations*, Computers chem. Engng., 9 (1985), pp. 93–96.

[89] M. LENTINI AND V. PEREYRA, *An adaptive finite-difference solver for nonlinear two-point boundary value problems with mild boundary layers*, SIAM J. Numer. Anal., 14 (1977), pp. 91–111.

[90] J. LOGSDON AND L. BIEGLER, *Accurate solution of differential-algebraic optimization problems*, Ind. Eng. Chem. Res., 28 (1989), pp. 1628–1639.

[91] R. LUUS, *Piecewise linear continuous optimal control by iterative dynamic programming*, Ind. Eng. Chem. Res., 32 (1993), pp. 859–865.

[92] L. LYNN, E. PARKIN, AND R. ZAHRADNIK, *Near-optimal control by trajectory approximation*, Ind. Eng. Chem. Fundam., 18 (1979).

[93] C. MAJER, W. MARQUARDT, AND E. GILLES, *Reinitialization of DAEs after discontinuities*, Computers chem. Engng., 19 (1995), pp. S507–S512.

[94] T. MALY AND L. PETZOLD, *Numerical methods and software for sensitivity analysis of differential-algebraic systems*, Applied Numerical Mathematics, 20 (1996), pp. 57–79.

[95] W. MARQUARDT, *Numerical methods for the simulation of differential-algebraic process models*, tech. rep., Rheinisch-Westfälische Technische Hochschule Aachen, 1994.

[96] R. MÄRZ, *Numerical methods for differential-algebraic equations*, Acta Numerica, (1991), pp. 141–198.

[97] S. MATTSSON AND G. SÖDERLIND, *Index reduction in differential-algebraic equations using dummy derivatives*, SIAM J. Sci. Stat. Comput., 14 (1993), pp. 677–92.

[98] R. MEHRA AND R. DAVIS, *A generalized gradient method for optimal control problems with inequality constraints and singular arcs*, IEEE Trans. Autom. Control, (1972).

[99] A. MENGEL, *Optimum trajectories*, in Proceedings of Project Cyclone Symposium 1 on REAC Techniques, New York, March 1951, Reeves Instrument Corp. and U.S. Navy Special Devices Center, pp. 7–13.

[100] A. MIELE, B. MOHANTY, P. VENKATARAMAN, AND Y. KUO, *Numerical solution of minimax problems of optimal control, part 2*, JOTA, 38 (1982), pp. 111–135.

[101] M. MOHIDEEN, J. PERKINS, AND E. PISTIKOPOULOS, *Optimal synthesis and design of dynamic systems under uncertainty*, AIChE Journal, 42 (1994), pp. 2251–2272.

[102] ——, *Optimal synthesis and design of dynamic systems under uncertainty*, Computers chem. Engng., 20 (1996), pp. S895–S900.

[103] ——, *Towards an efficient numerical procedure for optimal control*, Computers chem. Engng., 21 (1997), pp. S457–S462.

[104] K. MORISON AND R. SARGENT, *Optimization of multistage processes described by differential-algebraic equations*, Lect. Notes Math., 1230 (1986), pp. 86–102.

[105] K. R. MORISON, *Optimal Control of Processes Described by Systems of Differential-Algebraic Equations*, PhD thesis, University of London, 1984.

[106] C. NEUMAN AND A. SEN, *A suboptimal control algorithm for constrained problems using cubic splines*, Automatica, 9 (1973), pp. 601–603.

[107] C. PANTELIDES, *The consistent initialization of differential-algebraic systems*, SIAM J. Sci. Stat. Comput., 9 (1988), pp. 213–231.

[108] C. PANTELIDES, D. GRITSIS, K. MORISON, AND R. SARGENT, *The mathematical modelling of transient systems using differential-algebraic equations*, Computers chem. Engng., 12 (1988), pp. 449–454.

[109] C. PANTELIDES, V. VASSILIADIS, AND R. SARGENT, *Optimal control of multistage systems described by high-index differential-algebraic equations*, in Computational Optimal Control, R. Bulirsch and D. Kraft, eds., Birkhaüser, Basel, Germany, 1994, pp. 177–191.

[110] T. PARK AND P. BARTON, *State event location in differential-algebraic models*, ACM Transactions on Modeling and Computer Simulation, 6 (1996), pp. 137–165.

[111] H. PESCH, *Real-time computation of feedback controls for constrained optimal control problems. Part 2: A correction method based on multiple shooting*, Opt. Cont. Appl. and Meth., 10 (1989), pp. 147–171.

[112] L. PETZOLD, *Differential/algebraic equations are not ODEs*, SIAM J. Sci. Stat. Comput., 3 (1982), pp. 367–384.

[113] ——, *A description of DASSL: A differential/algebraic equation solver*, in Scientific Computing, R. Stepelman, ed., North-Holland, 1983, pp. 65–68.

[114] L. PETZOLD, Y. REN, AND T. MALY, *Regularization of higher-index differential-algebraic equations with rank-deficient constraints*, SIAM J. Sci. Comput., 18 (1997), pp. 753–774.

[115] L. PETZOLD, J. ROSEN, P. GILL, L. JAY, AND K. PARK, *Numerical optimal control of parabolic PDEs using DASOPT*, Proc. IMA Workshop on Large-Scale Optimization, (1996).

[116] P. PIELA, *ASCEND: An Object-Oriented Computer Environment for Modeling and Analysis*, PhD thesis, Carnegie–Mellon University, Pittsburgh, 1989.

[117] J. PONTON AND P. GAWTHROP, *Systematic construction of dynamic models for phase equilibrium processes*, Computers chem. Engng., 15 (1991), pp. 803–808.

[118] L. PONTRYAGIN, V. BOLTYANSKII, R. GAMKRELIDZE, AND E. MISHENKO, *The Mathematical Theory of Optimal Processes*, Interscience Publishers Inc., New York, 1962.

[119] H. RABITZ, M. KRAMER, AND D. DACOL, *Sensitivity analysis in chemical kinetics*, Ann. Rev. Phys. Chem., 34 (1983), pp. 419–61.

[120] W. RAY, *Advanced Process Control*, McGraw-Hill, New York, 1981.

[121] K. REINSCHKE, *Multivariable control: A graph theoretic approach*, in Lecture Notes in Control and Information Sciences, M. Thoma and A. Wyner, eds., vol. 108, Springer-Verlag, 1988.

[122] S. ROBERTS AND J. SHIPMAN, *Two Point Boundary-Value Problems: Shooting Methods*, American Elsevier, New York, 1972.

[123] O. ROSEN AND R. LUUS, *Evaluation of gradients for piecewise optimal control*, Computers chem. Engng., 15 (1991), pp. 273–281.

[124] E. ROZENVASSER, *General sensitivity equations of discontinuous systems*, Automation and Remote Control, (1967), pp. 400–404.

[125] I. RUSS, *Sensitivity function generation from nonlinear systems with jump discontinuities*, Buletinul Institutului Politehnic "Gheorghe Gheorghiu-Dej" Bucuresti. Seria Electrotehnica, 39 (1977), pp. 91–100.

[126] C. SAGAN, *Cosmos*, Random House, New York, 1980.

[127] K. SAN AND G. STEPHANOPOULOS, *Optimization of fed-batch penicillin fermentation: A case of singular optimal control with state constraints*, Biotechnology and Bioengineering, 34 (1989), pp. 72–78.

[128] R. SARGENT AND G. SULLIVAN, *The development of an efficient optimal control package*, in Proc. 8th IFIP Conf. Optimization Tech. Pt. 2, 1977.

[129] R. SCHOPF AND P. DEUFLHARD, *OCCAL a mixed symbolic-numerical optimal control calculator*, in Computational Optimal Control, R. Bulirsch and D. Kraft, eds., Birkhaüser, Basel, Germany, 1994.

[130] R. SINCOVEC, A. ERISMAN, E. YIP, AND M. EPTON, *Analysis of descriptior systems using numerical algorithms*, IEEE Trans. Automatic Control, AC-26 (1981), pp. 139–147.

[131] G. STEWART, *Rank degeneracy*, SIAM J. Sci. Stat. Comput., 5 (1984), pp. 403–413.

[132] P. TANARTKIT AND L. BIEGLER, *Stable decomposition for dynamic optimization*, Ind. Eng. Chem. Res., 34 (1995), pp. 1253–1266.

[133] K. TEO, C. GOH, AND K. WONG, *A Unified Computational Approach to Optimal Control Problems*, Pitman Monographs and Surveys in Pure and Applied Mathematics, Wiley, New York, 1991.

[134] J. TOLSMA AND P. BARTON, *Efficient calculation of sparse Jacobians*, submitted to SIAM Journal on Scientific Computing, (1997).

[135] ——, *On computational derivatives*, Computers chem. Engng., (in press, 1997).

[136] R. TOMOVIĆ AND M. VUKOBRATOVIĆ, *General Sensitivity Theory*, American Elsevier, New York, 1972.

[137] T. TSANG, D. HIMMELBLAU, AND T. EDGAR, *Optimal control via collocation and non-linear programming*, Int. J. Control, 21 (1975), pp. 763–768.

[138] Y. Z. TSYPKIN AND R. S. RUTMAN, *Sensitivity equations for discontinuous systems*, in Sensitivity methods in control theory, L. Radanović, ed., Dubrovnik, Aug. 31 - Sep. 5 1964, Pergamon Press, pp. 195–196.

[139] J. UNGER, A. KRÖNER, AND W. MARQUARDT, *Structural analysis of differential-algebraic equation systems- theory and application*, Computers chem. Engng., 19 (1995), pp. 867–882.

[140] F. VALENTINE, *The problem of Lagrange with differential inequalities as added side conditions*, in Contributions to the Calculus of Variations, Chicago University Press, 1937, pp. 407–448.

[141] S. VASANTHARAJAN AND L. BIEGLER, *Simultaneous strategies for optimization of differential-algebraic systems with enforcement of error criteria*, Computers chem. Engng., 14 (1990), pp. 1083–1100.

[142] V. VASSILIADIS, *Computational Solution of Dynamic Optimization Problems with General Differential-Algebraic Constraints*, PhD thesis, University of London, London, U.K., 1993.

[143] V. VASSILIADIS, R. SARGENT, AND C. PANTELIDES, *Solution of a class of multistage dynamic optimization problems. 2. Problems with path constraints*, Ind. Eng. Chem. Res., 33 (1994), pp. 2123–2133.

[144] O. VON STRYK, *Numerical solution of optimal control problems by direct collocation*, in Optimal Control and Variational Calculus, R. Bulirsch, A. Miele, J. Stoer, and K. Well, eds., Birkhaüser, Basel, Germany, 1993.

[145] H. WU, *Generalized maximum principle for optimal control of generalized state-space systems*, Int. J. Control, 47 (1988), pp. 373–380.

[146] A. XING AND C. WANG, *Applications of the exterior penalty method in constrained optimal control problems*, Opt. Cont. Appl. and Meth., 10 (1989), pp. 333–345.

[147] V. YEN AND M. NAGURKA, *Optimal control of linearly constrained linear systems via state parameterization*, Opt. Cont. Appl. and Meth., 13 (1992), pp. 155–167.

THESIS PROCESSING SLIP

FIXED FIELD ill _____ name _____

index _____ biblio _____

► COPIES (Archives) Aero Dewey Eng Hum

Lindgren Music Rotch (Science)

TITLE VARIES ►☐ _____

_____

_____

NAME VARIES: ►☐ _____

_____

IMPRINT: (COPYRIGHT) _____

►COLLATION: 391 p

_____

►ADD. DEGREE: _____ ►DEPT.: _____

SUPERVISORS: _____

_____

_____ _____

_____ _____

_____ _____

_____ _____

_____ _____

_____ _____

_____ _____

NOTES:

cat'r: date:

page: 0171

►DEPT: Chem. Eng

►YEAR: 1998 ►DEGREE: Ph.D.

►NAME: FEEHERY, William Francis

_____