CONDITIONAL MODELING

Joseph James Zaher

November 1995

A Dissertation Submitted to the
Graduate School in Partial Fulfillment
of the Requirements for the Degree

DOCTOR OF PHILOSOPHY

in

CHEMICAL ENGINEERING

Carnegie Mellon University
Pittsburgh, Pennsylvania

Abstract

Conditional models exist where the defining equations differ depending on where the model solution lies. In this work, an equation-based approach is studied for formulating, structurally analyzing, and solving this class of problems.

Formulation is achieved using a disjunctive statement where at least one set of equations, among any number of alternatives, must be satisfied at the solution. Each set is associated with certain ranges in values of the model parameters and consists of the same number of equations. Provision for models involving mixed differential and algebraic equations is made.

A structural analysis is devised to ensure a well-posed formulation. A consistent subset of the variable parameters of the model are sought to be considered independent such that the solution for the remaining dependent variables may be computed no matter which set of equations are ultimately used. Furthermore, a partitioning pattern is devised in order to decompose the problem also without any knowledge of the correct set of equations.

An optimization algorithm is first presented for equality constrained nonlinear programs. It is based on applying a quasi-newton sequential quadratic programming algorithm toward the minimization of the augmented Lagrangian. For conditional models, the search space is broken up into search regions. The feasible region of each search region is given by different sets of equations. The solution of conditional models is carried out whereby the choosing of which equations to solve and their solving are done simultaneously.

Acknowledgements

This dissertation is the product of determination and dedication, both of which are not inherent in human nature. They must be upheld with much hard work, and in many instances, require the assistance of others with whom we may sometimes be so lucky to come in contact.

I would like to first thank my advisor, Professor Arthur Westerberg, for recognizing my ideas as an area of worthy research and giving me the confidence to pursue them. My thesis committee members, consisting of Professors Ignacio Grossmann, Dennis Prieve, and James Garrett, have been instrumental in molding this thesis into a both contributive and readable product. Research was made fun for me by all my colleagues at school, including Peter Piela, George Serghiou, and Manuel Dufresne. I am especially grateful for having worked with the creative mind of Dr. Piela. I will not soon forget being a part of his ASCEND team as a user and later joining with the developers Ben Allan and Kirk Abbott. Much of my inspiration comes from the diligence of my dear friend, James Donley, who shall always be my best man. You are the brother I never had.

My wife and friend of many years, Elizabeth, has been that important someone who believes in me and, in turn, is someone in whom I believe as well. We feel that by nurturing a person with love and understanding, that person can do great things and this thesis, I hope, is simply one of them. Most of all, I am proud to be the first person in the history of my family to obtain a doctoral degree and hope that there will be more to follow. My immediate family has survived much loss through the years and their strength has helped me greatly. I wish to dedicate this contribution to the loving free-spirited memory of my sister Dianne.

Contents

List of Figures

Chapter 1

Introduction

1.1 Functionality

The modeling of chemical engineering processes will in general involve a set of parameters both variable and constant and a set of equations to provide a mapping from the constants to the variables. In cases where there are more variables than equations, an objective function may be used to determine optimal values for the excess variables at the solution.

The functionality of a model is the dependence of the values of the variables at the solution on the values of the constants. It is largely influenced by the value types of the parameters. All parameters of a model may be either real or integer. The functionality of a model is discontinuous and nondifferentiable with respect to the integer constants. More interestingly, when integer variables exist, the functionality becomes discontinuous and nondifferentiable with respect to the real constants as well. Furthermore, even though all of the variables may be real, the functionality may be continuous yet nondifferentiable with respect to the real constants if the equations are conditional. In conditional modeling, the solution lies in one of a multitude of different states, each classified by certain ranges in the values of the parameters and characterized by a unique set of equations. It is transitions of the solution among these states brought on by a continuous change in the values of the constants of the model which introduces nondifferentiable functionality. An example arising in thermodynamics is the phenomenon of phase transition where, for different values in temperature of a closed system at fixed pressure and composition, one or more phases may exist.

1.2 Applications

Conditional models arise in chemical engineering design mostly due to the phenomena of phase and fluid flow transition. There are also cases where conditional cost functions are used in optimizing designs. As will be demonstrated, conditional models may consist of both algebraic and differential equations.

1.2.1 Distillation

Distillation is a well understood unit operation whose parameters are of both the real (*e.g.*, the boilup and reflux rates, reboiler and condenser duties, tray compositions) and integer type (*e.g.*, number of trays, the tray on which feed is introduced). When the number of trays is a variable parameter, a mixed integer nonlinear formulation must be used to ensure integer solution values. In observing the functionality of distillation models having a known number of trays and a fixed feed tray location, it is evident that an increase in purity demand of one of the products leads to an increase in required reflux. This in turn translates to a higher operating cost. In Figure 1.1, this behavior is observed for different feed tray locations. What can be inferred is that for a given range in purity demands, there exists an optimal feed tray location which will yield the lowest required reflux. The functionality of distillation models ought to include, therefore, transitions among feed tray locations brought on by changes in purity demand.

One of the assumptions in solving a distillation model has been that the phases

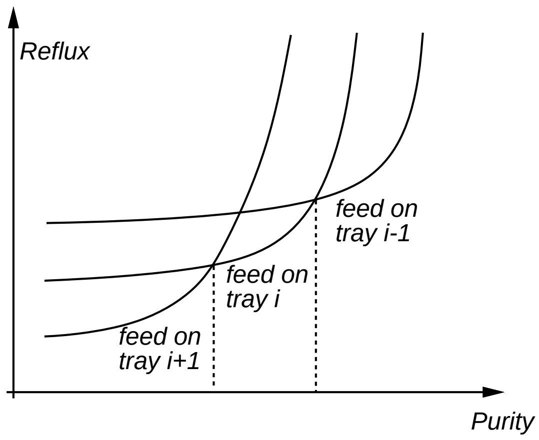

Figure 1.1: Distillation column operating curves

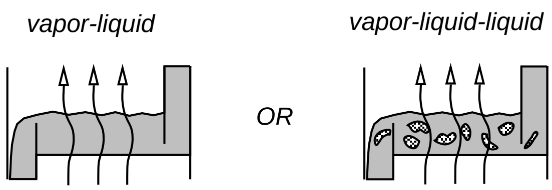

Figure 1.2: Phase transition on a distillation tray

existing on all trays be the same, namely one vapor phase and one liquid phase. It is not uncommon, however, for distillation towers to contain two liquid phases on one or more trays [9] [48]. The components may form two liquid phases under certain conditions which may be met on some of the trays while the immiscibility may disappear on other trays (Figure 1.2). Each tray should be modeled conditionally with unknown phases at the solution. An example of a conditional flash model will be given later.

1.2.2 Integration

Commonly in chemical engineering, the equations of a model may be both differential and algebraic. When differential equations appear in a conditional model, care must be taken during integration. It is possible that multiple transitions may occur within the domain of integration by the state variables. Transitions within the solution profile

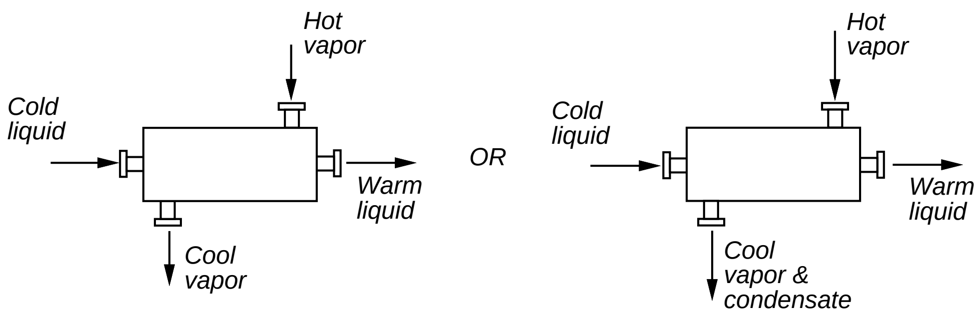

Figure 1.3: Phase transition in a heat exchanger

are classified to be reversible or irreversible [3]. Reversible transitions are further classified to be symmetric or asymmetric. An example of an irreversible transition is the dynamic modeling of the pressurization of a vessel fitted with a bursting disc. If the fitting is initially intact and the integrated pressure profile increases fast enough from some low initial pressure, there will exist a point in the time domain at which a set value in the vessel pressure is reached causing the fitting to burst. At all future times in the profile, the fitting will resume the burst state regardless of the vessel pressure. If the fitting were a safety relief valve, reversibility would exist for, as the pressure decreases below a reseat pressure, the valve is capable of resuming the closed position. However, because the reseat pressure is often at some value lower than the set value required to open the valve, this is termed an asymmetric reversible transition.

Phase transition is reversible and symmetric because it occurs in a well mixed system at the same temperature in either direction. This is of interest in modeling a counter current heat exchanger like the one in Figure 1.3. A cold liquid is heated using a hot vapor which may or may not condense within the shell of the heat exchanger. The problem here is to determine not just if phase transition will occur but at what position inside the heat exchanger it does occur in order to integrate the differential heat transfer equation accurately. This example is investigated further later on.

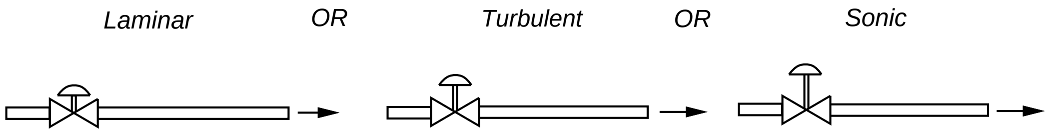

Figure 1.4: Fluid flow transition

1.2.3 Piping

Fluid flow transition in constant diameter pipes is another common instance where conditional equations would be necessary (Figure 1.4). At very low flow rates, a laminar type of flow is observed. As the flow rate is increased, a transition may be observed as the flow type becomes turbulent. Even at very high flow rates, another transition may be observed for compressible gases where the flow may become choked due to the sonic barrier. This example is also discussed further later on.

1.3 Outline

What follows in this report are the results of this research. In all facets of modeling including formulation, structural analysis, and solution, discussion will be on the comparison between continuous and differentiable models (nonlinear equality constrained programming) and conditional models. There exists much more similarity to exploit between these classes of problems than when comparing conditional models to mixed integer nonlinear programs predominantly because of the lack of continuous functionality in problems involving integer variables.

In chapter 2, formulation is discussed. A set of notation is introduced first for a continuous and differentiable model having only equality constraints and then for a conditional model with a means of unambiguously specifying alternative sets of equations. Associated with each set of equations must be a unique set of conditions. Other alternative formulations which introduce nondifferentiable functionality are discussed, namely those where inequality constraints are posed and those where integer variables are incorporated. It is mentioned that although all inequality constrained nonlinear programs can be posed as a conditional model, not all conditional models can be formulated using inequality constraints. Furthermore, although all conditional models can be written as a mixed integer nonlinear program, not all models involving integer variables can be written or solved using conditional modeling techniques. This is to identify conditional modeling as only a subclass of discrete optimization. Three examples are constructed for the benefit of consistent illustration throughout the remainder of the report, one in the area of phase equilibria, one involving heat exchange, and one involving adiabatic compressible flow. The heat exchange example is particularly interesting because it demonstrates how a mixed differential and algebraic conditional model is discretized using a method developed in this work, ensuring that transitions of the solution among states does not introduce error in the integration.

In chapter 3, structural analysis is discussed. All formulations must be analyzed structurally before solving. Criteria and algorithms for recognizing a well-posed problem are available for continuous and differentiable models and they are reviewed in this chapter. A notion of structural consistency exists for the set of equality constraints in a continuous and differentiable nonlinear program and is therefore applicable as well to the individual sets of equations found in a conditional model. The analysis becomes more complicated when there are excess variables to be optimized along with variables computed from the equations. The partitioning of the variables into the two groups is not unique and for conditional models, it must be done carefully. An extended notion of consistency for conditional models is defined in this work which essentially deals with the finding of a consistent partitioning of the variables which agree structurally with any of the alternative sets of equations. Another interesting discussion offered in this chapter is on the partitioning of the equations to decompose large conditional models even in an optimization mode. Before now, equation partitioning was only discussed in nonlinear equation solving where an equal number of variables and equations leaves no degrees of freedom. All of the algorithms are applied to the examples introduced in the first chapter and the results are displayed graphically as will be explained later.

Chapter 4 deals directly with the development of a solution algorithm for conditional models. First, conventional nonlinear programming techniques are reviewed in detail to uncover areas of improvement. A solution algorithm for continuous and differentiable models is presented because of the anticipation that, most of the time, the solution path for a conditional model will visit a series of points where the functionality is differentiable. It is only nondifferentiable points, or points of transition, which must be treated differently and are using a gradient analysis. Boundary crossing is the algorithm developed in this work and presented in this chapter which allows solution paths for conditional models to be directed across state transitions. The algorithm is also shown to terminate at solutions which happen to lie at points of transition. The specifics of the algorithm are demonstrated when applied to the three examples which have been up to that point only structurally analyzed.

Chapter 5 gives a thorough summary of the contributions made in this research. It is felt that prior to this work, problems belonging to the subclass of conditional modeling had not been adequately identified. As will be seen in this report, a framework has been established to efficiently characterize conditional modeling in the three main areas of formulation, structural analysis, and solution, all of which are equally capable of attracting continued research in the future.

Chapter 2

Formulation

2.1 Introduction

In this chapter, a foundation of notation is established. Generic representations of the parameters of a model as well as the objective function and equations are offered for reference throughout the remainder of this report.

2.2 Continuous and differentiable models

Conventional continuous and differentiable models consist of a set of continuous parameters of which n are variable and the rest specified as constants. The variables $\mathbf{x}$ appear in a set of m equality constraints $\mathbf{g}(\mathbf{x}) = \mathbf{0}$ with the constant parameters already dissolved. At most, these equations can produce solution values for m variables. In cases where n exceeds m, an objective function $f(\mathbf{x})$ is needed in order for solution values for the remaining variables to be optimal.

$$
\begin{aligned}
\min \quad & f(\mathbf{x}) \\
s.t. \quad & \mathbf{g}(\mathbf{x}) = \mathbf{0} \\
& \mathbf{x} \in R^n \\
& f : R^n \rightarrow R^1 \\
& \mathbf{g} : R^n \rightarrow R^m
\end{aligned}
\tag{2.1}
$$

The search space consists of all n-dimensional real vectors. The feasible region is a subset of the search space, consisting of all points satisfying the equations. The constraints are referred to as being consistent if and only if the feasible region is nonempty. A solution will then be a member of the set of feasible points.

An example is taken from a widely used collection of test problems [23]. Problem number 114 is used. As originally formulated, there are 8 inequality constraints and only 3 equality constraints along with lower and upper bounds on all of the variables. The reported solution, however, indicates only 4 of the inequality constraints and 2 of the variable upper bounds are active at the solution. These can therefore be included as equations.

$$
\begin{aligned}
\min \quad & 5.04x_1 + 0.035x_2 + 10.0x_3 + 3.36x_5 - 0.063x_4x_7 \\
s.t. \quad & 3x_7 = 0.99x_{10} + 133.0 \\
& x_9 + 0.1998x_{10} = 32.238 \\
& 1.12x_1 + 0.13167x_1x_8 = 0.00667x_1x_8^2 + 0.99x_4 \\
& 57.425 + 1.098x_8 + 0.325x_6 = 0.038x_8^2 + 0.99x_7 \\
& 1.22x_4 = x_1 + x_5 \\
& 98000x_3 = x_6(x_4x_9 + 1000x_3) \\
& x_2 + x_5 = x_1x_8 \\
& x_5 = 2000 \\
& x_7 = 95
\end{aligned}
\tag{2.2}
$$

As a result, with 10 variables and 9 equations, there is only 1 degree of freedom.

2.3 Conditional models

Conditional models, on the other hand, contain a means to formulate alternative sets of m equations involving the common set of parameters. The equations within each set must be locally defined and confined to some region of values of the model parameters [58]. Such a region is referred to as a search region and is a subset of the search space. A set of l logical conditions of the form $\mathbf{b}(\mathbf{x}) \geq \mathbf{0}$ is used to associate the conditional equations to search regions. Each condition, which may or may not be satisfied, disects the search space into two neighboring search regions, each characterized by either positive or negative values of its boundary expression $b_i(\mathbf{x})$, where i is an element of the set of indices $\{1 \ldots l\}$. Each search region is identified by a unique combination of boolean values for all of the conditions. This can accommodate up to 2^l different search regions. A compact way of differentiating a search region from other search regions is by using a set s where $s \subseteq \{1 \ldots l\}$. The set s can be used to contain the indices of all of the conditions which must be met (implying that all of the remaining conditions must not be met) in order for the solution to reside in a specific search region. To generate all of the possible combinations of boolean values for the conditions, the power set $\mathcal{P}(\{1 \ldots l\})$ is used which contains all of the possible subsets s, including the empty set $\emptyset$ and the set $\{1 \ldots l\}$ itself. For most conditional models, though, it is physically not meaningful for the solution to lie in some of the search regions. These search regions, therefore, can be removed from the search space and no conditional equations need be associated with them. Let $\mathcal{R}$, where $\mathcal{R} \subseteq \mathcal{P}(\{1 \ldots l\})$, be the set of all subsets s with which m conditional equations are to be associated in the form $\mathbf{r}_s(\mathbf{x}) = \mathbf{0}$. The statement that the solution reside in only one of the search regions is formulated disjunctively.

$$
\begin{aligned}
\min \quad & f(\mathbf{x}) \\
s.t. \quad & \bigvee_{s \in \mathcal{R}} \left\{ \begin{array}{l} b_i(\mathbf{x}) \geq 0, \ \forall_{i \in s} \\ b_i(\mathbf{x}) < 0, \ \forall_{i \in \{1 \ldots l\} - s} \\ \mathbf{r}_s(\mathbf{x}) = \mathbf{0} \end{array} \right\} \\
& \mathbf{x} \in R^n \\
& f : R^n \rightarrow R^1 \\
& \mathbf{b} : R^n \rightarrow R^l \\
& \mathbf{r}_s : R^n \rightarrow R^m, \ \forall_{s \in \mathcal{R}}
\end{aligned}
\tag{2.3}
$$

The disjunctive constraints are consistent if there exists at least one search region whose equations yield a nonempty feasible region. Furthermore, continuity is preserved if the feasible regions of all consistent search regions are aligned at search region boundaries. It should be pointed out that, although similarities exist between formulation (2.3) and that of disjunctive programming, there are important differences as well [1] [45]. The boundary expressions which appear above are not necessarily posed as disjunctive inequality constraints but rather are used to associate parameter values with a correct set of equations. This is efficiently done by defining a common boundary expression which changes sign from one search region to another. In addition, continuity in the feasible region is not required in disjunctive programming and is assumed in (2.3). Because of continuity, what has been looked at in this work is the ability to preserve the constraints as equations instead of replacing them with inequality constraints along with nontrivially augmenting the objective function which would be required if the model is to adhere to the theory which exists for disjunctive programs.

When the number of logical conditions, l, is zero, only one search region is created using $s = \emptyset$ and formulation (2.3) reduces to that of (2.1) since no boundary expressions exist. Otherwise, it is true that often a large number of equations are found to be common to some if not all of the search regions. Those common to all search regions are said to be globally defined because they must hold for all values of the parameters just as the equality constraints of (2.1) do. It is therefore convenient to include unconditional equations outside of the disjunction.

Formulation (2.3) enables the modeling of nonsmooth functions such as the common absolute value function. In this case, the domain of the argument of the absolute value function is divided into positive and negative values. For positive values, the value of the function is simply the value of the argument whereas for negative values, the value of the function is the negative of the value of the argument. Using the variables x_1 and x_2 and some objective function $f(x_1, x_2)$, a simple conditional model where x_2 is constrained to be equal to the absolute value of x_1 can be constructed with two search regions, each having a feasible region. In this small example, it is easy to confirm continuity by noticing that the point $(0, 0)$ lies in both feasible regions.

$$\begin{aligned} \min \quad & f(x_1, x_2) \\ s.t. \quad & \left\{ \begin{array}{c} x_1 \geq 0 \\ x_2 = x_1 \end{array} \right\} \bigvee \left\{ \begin{array}{c} x_1 < 0 \\ x_2 = -x_1 \end{array} \right\} \end{aligned} \tag{2.4}$$

2.3.1 Inequality constraints

Another common way of introducing nondifferentiable functionality to a model is by incorporating inequality constraints. With l inequality constraints added to (2.1), the more general model formulation is created.

$$\begin{aligned} \min \quad & f(\mathbf{x}) \\ s.t. \quad & \mathbf{g}(\mathbf{x}) = \mathbf{0} \\ & \mathbf{h}(\mathbf{x}) \geq \mathbf{0} \\ & \mathbf{x} \in R^n \\ & f : R^n \rightarrow R^1 \\ & \mathbf{g} : R^n \rightarrow R^m \\ & \mathbf{h} : R^n \rightarrow R^l \end{aligned} \tag{2.5}$$

This is first recognized as a special case of formulation (2.3) where the number of search regions (*i.e.*, the cardinality of $\mathcal{R}$) is one, namely the one denoted by $s = \{1 \ldots l\}$ and the notation $\mathbf{r_s}(\mathbf{x})$ and $\mathbf{b}(\mathbf{x})$ are more generically replaced by $\mathbf{g}(\mathbf{x})$ and $\mathbf{h}(\mathbf{x})$ respectively. Thus the inequality constraints can be thought of as logical conditions for which only one combination of boolean values is allowed at the solution. For that reason, the solution may frequently be found at a point on the search region boundary where at least one of the conditions has a boundary expression equal to zero. Those inequalities for which this is true are referred to as being active at the solution. The set of active constraints could change if the model were to be solved again using different values for the constants, hence the nondifferentiable functionality. With inequality constrained nonlinear programs, techniques are available which can determine if the solution resides away from the boundary of the search region or not while solving. The active-set strategy is based on the idea that, if the correct active set were known for the solution, then the solution could be computed from reformulating the inequality constraints which are known to be active at the solution as equations and discarding those which are expected to be inactive [17].

When there are more than one search region, as is normally the case with conditional models, the solution is permitted to reside on either side of some boundaries. In this case, the logical conditions can no longer be used as inequality constraints. Instead, the conditional equations involved in the disjunction need to be replaced by a strategic combination of inequality constraints [8]. This can only be done in the special case where the alternative sets of m equations are able to be broken up into m independent decisions of single equation alternatives. Furthermore, the single equation alternatives must be of the same form with a common variable isolated on the left hand side. As a result, only disjunctions which effectively swap right hand side expressions are handled. This has direct application to modeling with constraints involving the nonsmooth maximum and minimum operators which can each be constructed through the superimposition of multiple inequalities. For each operator, at least one of the inequality constraints must be active at the solution as required by the disjunction that one of the arguments be chosen. This can be enforced by minimizing an objective consisting of a sum of weighted terms involving the left hand side variables. If an objective function already exists in the conditional model, then, as illustrated next, augmentation with the weighted terms is necessary. Choosing sufficiently high values for the weights is difficult.

To re-formulate (2.4) as an inequality constrained nonlinear program, another parameter ω must be introduced which is treated like a constant. It will be used in augmenting the objective function. To generate the correct inequality constraints, an equivalence is made between stating that x_2 is equal to the absolute value of x_1 and that x_2 is equal to the maximum of x_1 and $-x_1$. For values of ω beyond some threshold value, x_2 will be forced equal to either x_1 or $-x_1$.

$$
\begin{aligned}
\min \quad & f(x_1, x_2) + \omega x_2 \\
s.t. \quad & x_2 \geq x_1 \\
& x_2 \geq -x_1
\end{aligned}
\tag{2.6}
$$

Although not all conditional models can be expressed this way, it will be shown that all nonlinear programs like (2.5) which involve inequality constraints can be reformulated as a conditional model. The main assumption in doing so is that the solution to (2.5) can be alternatively found by solving the equations corresponding to the Kuhn-Tucker necessary conditions [4]. This assumption has convexity property implications on nonlinear programs and will be discussed later. It suffices to say that some nonlinear programs can be adequately transformed to feasibility problems where the equations which must be satisfied at an optimum point are solved. The derivation is shown using the notation of (2.5) and by introducing a vector of multipliers, $\boldsymbol{\lambda}$, for the equality constraints and a vector of multipliers, $\boldsymbol{\mu}$, for the inequality constraints. A necessary condition for optimality is

$$
\nabla_{\mathbf{x}} f(\mathbf{x}) + \nabla_{\mathbf{x}} \mathbf{g}(\mathbf{x}) \boldsymbol{\lambda} + \nabla_{\mathbf{x}} \mathbf{h}(\mathbf{x}) \boldsymbol{\mu} = \mathbf{0} \tag{2.7}
$$

The additional stipulations on $\boldsymbol{\mu}$ are that they must be less than or equal to zero and complementary to the inequality constraint expressions $\mathbf{h}(\mathbf{x})$ which must be greater than or equal to zero. Complementarity means that for any index i in the set of inequality constraints $\{1 \ldots l\}$, the product $\mu_i h_i(\mathbf{x})$ must be zero.

When the objective function $f(\mathbf{x})$ is quadratic and the constraint expressions $\mathbf{g}(\mathbf{x})$ and $\mathbf{h}(\mathbf{x})$ are linear in the variables $\mathbf{x}$, the problem can be reduced to the familiar linear complementarity form [12] [29]. By taking advantage of the linearity, the complementarity reduces to determining for each of a set of pairs of variables, which, if not both, is to be set to zero at the solution. Here, though, the task is to determine for each index $i \in \{1 \ldots l\}$ whether μ_i is zero and $h_i(\mathbf{x})$ is positive or μ_i is negative and $h_i(\mathbf{x})$ is zero. This is recognized as a disjunctive statement. What is needed is a common boundary expression to act as a definitive indicator which will associate the solution with the correct choice. Mathematically, the choices of the disjunction can be identified by the sign of the expression $\mu_i + h_i(\mathbf{x})$. At the solution, it must be positive when μ_i is set to zero and negative otherwise.

$$\left\{ \begin{array}{l} \mu_i + h_i(\mathbf{x}) \geq 0 \\ \mu_i = 0 \end{array} \right\} \bigvee \left\{ \begin{array}{l} \mu_i + h_i(\mathbf{x}) < 0 \\ h_i(\mathbf{x}) = 0 \end{array} \right\} \tag{2.8}$$

As a result, the optimization problem (2.5) is transformed to the conditional model consisting of the equality constraints of (2.5) and (2.7) along with a disjunctive statement which accounts for all combinations of choices of the disjunctions (2.8).

2.3.2 Binary variables

The determination of the state at which the solution resides can also be considered a discrete decision. To imbed the decision making process into the formulation, binary variables are needed, one for each condition. A solution value of 0 for a binary variable implies that the corresponding condition is not met and a solution value of 1 for the binary variable implies that the condition is met.

Taking the example (2.4) introduced above, a binary variable y is included along with a necessary constant M whose value must be chosen large enough so as to not affect the solution. Then the conditions and the conditional equations can all be written as inequality constraints involving y and M.

$$
\begin{aligned}
\min \quad & f(x_1, x_2) \\
s.t. \quad & x_1 \geq -M(1-y) \\
& x_1 \leq My \\
& x_2 - x_1 \geq -M(1-y) \\
& x_2 - x_1 \leq M(1-y) \\
& x_2 + x_1 \geq -My \\
& x_2 + x_1 \leq My
\end{aligned}
\tag{2.9}
$$

Techniques have been developed to solve mixed integer nonlinear programs rigorously using a two-phase strategy [19]. The sequence of search regions to visit in searching for the correct state is given by successive solutions of a mixed integer linear subproblem solved by branch and bound methods while each nonlinear subproblem attempts to find a solution confined to each visited search region using conventional nonlinear programming techniques. More recently, disjunctive programming techniques have been successfully developed to enhance convergence and avoid the need to introduce M [5] [46]. The difficulty with formulation (2.9) is that during a branch and bound search, combinations of values for the binary variables are not necessarily associated with the correct set of equations. Disjunctive programming helps to tighten the formulation by using boolean variables which are treated special at solution time. By exploiting continuity, the association of parameter values to equations is enforced more easily in this work through the use of a set of boundary expressions.

Termination of solving the mixed integer nonlinear program involves finding the search region and the point in that search region where the objective function is minimized. By not exploiting continuity, the solution algorithm will typically jump from one search region to another which is remote from it. For highly nonlinear conditional models, jumps to remote search regions may result in the poor initialization of some of the nonlinear subproblems. Furthermore, the nonlinear subproblems insist on converging the equations of what might be an incorrect search region. This method employs a highly mathematical approach which remains as a rigorous general purpose algorithm designed for problems where discrete decision making cannot be avoided.

2.4 Examples

To better demonstrate how disjunctive formulations are constructed, three examples are used which will be referenced throughout the remainder of this report. The first two are based on phase transition, the first consisting of purely algebraic equations while the second illustrates the effect of differential equations. The last example is based on fluid flow transition from turbulent or plug flow to choked or sonic flow.

2.4.1 Phase equilibria

In the application of distillation, Figure 1.2 was included to introduce the phenomenon of phase transition which can drastically change the separation. In order to model equilibrium on each tray of a distillation column correctly, an ability to determine the phases present as well as the distribution of the components among the existing phases needs to be first developed. This is then incorporated in a conditional model for each tray of the distillation column where phase transition is anticipated.

The isothermal flash problem has attracted much research [20] [32] [33] [34] [38]. It is recognized in classical thermodynamics that a closed system, if given sufficient time to equilibrate, will reside at a state at which the total Gibbs free energy is minimized. For a system at a fixed temperature, T, pressure, P, and composition, $\mathbf{z}$, over the set of components, C, this may result in the formation of multiple phases, F, with each phase having a nonnegative molar amount, $\phi^j, j \in F$, relative to the total number of moles in the system, and composition $\mathbf{y}^j, j \in F$. Phase j does not exist if its corresponding mole fraction ϕ^j is zero. Regardless of the phase separation, a material balance for the closed system requires that the sum of the number of moles of a component over all of the existing phases must be the same as the total number of moles of the component in the system. For any component i in some phase j, the partial molar Gibbs free energy, $\overline{G}_i^j(T,P,\mathbf{y}^j)$, will in general be a function of the temperature, pressure, and the mole fractions of all components in phase j. The solution of the following dimensionless nonlinear optimization program will therefore dictate the phase distribution to be expected for a specified temperature and pressure.

$$\begin{aligned}
\min \quad & \frac{1}{RT}\sum_{j\in F}\sum_{i\in C}\phi^j y_i^j \overline{G}_i^j(T,P,\mathbf{y}^j) \\
s.t. \quad & \sum_{j\in F}\phi^j y_i^j = z_i,\ \forall_{i\in C} \\
& \sum_{i\in C} y_i^j = 1,\ \forall_{j\in F} \\
& \phi^j \geq 0,\ \forall_{j\in F}
\end{aligned} \tag{2.10}$$

Problem (2.10) is recognized as being in the form of (2.5). In an attempt to reduce (2.10) to a feasibility problem, the necessary conditions for optimality to supplement the constraints in (2.10) must be derived by introducing constraint multipliers. The variables α_i are introduced for the material balance of each component i. The variables β^j are introduced for the normalization of component mole fractions in each phase j. For the inequality constraints, the variables η^j are introduced to enforce nonnegativity on the fractions of each phase j. With the addition of these variables, the procedure explained in transforming problem (2.5) is applied to supplement the equations of problem (2.10) with the following set of constraints.

$$\left\{\begin{array}{l} \eta^j + \phi^j \geq 0 \\ \eta^j = 0 \end{array}\right\} \bigvee \left\{\begin{array}{l} \eta^j + \phi^j < 0 \\ \phi^j = 0 \end{array}\right\},\ \forall_{j\in F} \tag{2.11}$$

$$\phi^j\left(\frac{\overline{G}_i^j(T,P,\mathbf{y}^j)}{RT} + \frac{1}{RT}\sum_{k\in C} y_k^j \frac{\partial \overline{G}_k^j(T,P,\mathbf{y}^j)}{\partial y_i^j} + \alpha_i\right) + \beta^j = 0,\ \forall_{i\in C},\ \forall_{j\in F} \tag{2.12}$$

$$\sum_{i\in C} y_i^j\left(\frac{\overline{G}_i^j(T,P,\mathbf{y}^j)}{RT}+\alpha_i\right)+\eta^j=0,\ \forall_{j\in F} \tag{2.13}$$

The summation term in (2.12) evaluates to 1 when the component mole fractions of each phase are treated as independent variables. Unlike in the derivation of the Gibbs-Duhem equation, the dependency among the component mole fractions of each phase that they must be normalized has been instead incorporated as a constraint on the feasible region. To further simplify, the complementarity condition of (2.11) must be analyzed. It is interesting to find the result that for each phase j, either η^j is zero, implying that the phase exists, or ϕ^j is zero, implying that the phase vanishes. Either way, their sum makes an adequate transition indicator.

For all existing phases j, the variable η^j will be zero, ϕ^j will be more than zero and equation (2.12) can be rearranged.

$$\frac{\overline{G}_i^j(T,P,\mathbf{y}^j)}{RT}+\alpha_i=-\frac{\phi^j+\beta^j}{\phi^j},\ \forall_{i\in C} \tag{2.14}$$

Relation (2.14) dictates that the expression on the left hand side is the same for all components i in phase j. In equation (2.13), this allows the expression to be factored out of the summation and, since the mole fractions of each phase are constrained to add to one, it is concluded that

$$\frac{\overline{G}_i^j(T,P,\mathbf{y}^j)}{RT}+\alpha_i=-\eta^j,\ \forall_{i\in C} \tag{2.15}$$

which is currently zero. Equation (2.15) amounts to expressing equilibrium among all of the existing phases since for each component i, the partial molar Gibbs energy will be equated to the same value, $-\alpha_i RT$, among all of the existing phases. It is further concluded from (2.14) that β^j will be $-\phi^j$ for all existing phases j.

For all absent phases j, there is a different reasoning. This time, the variable ϕ^j will be zero and η^j will be less than zero. It is readily seen from equation (2.12) that β^j will be zero, suggesting that the expression $\phi^j+\beta^j$, which is evidently zero under all conditions, can be eliminated. However, with most of the information multiplied by zero, there is no further conclusion to be drawn as to a unique solution. One solution is given by (2.15) which is now nonzero and also satisfies (2.13). As a result, a means now exists to compute the fictitious compositions of the nonexistent phases. This is crucial in providing continuity in the solution across boundaries. Problem (2.10) is now posed as a feasibility problem similar to how phase equilibrium calculations are done conventionally. The difference here is that, through the additional variables η^j, the phases present at equilibrium need not be known *a priori*.

$$\begin{aligned}
&\left\{ \begin{array}{l} \eta^j + \phi^j \geq 0 \\ \eta^j = 0 \end{array} \right\} \bigvee \left\{ \begin{array}{l} \eta^j + \phi^j < 0 \\ \phi^j = 0 \end{array} \right\}, \ \forall_{j \in F} \\
&\frac{\overline{G}_i^j(T, P, \mathbf{y}^j)}{RT} + \alpha_i + \eta^j = 0, \ \forall_{i \in C}, \ \forall_{j \in F} \\
&\sum_{j \in F} \phi^j y_i^j = z_i, \ \forall_{i \in C} \\
&\sum_{i \in C} y_i^j = 1, \ \forall_{j \in F}
\end{aligned} \tag{2.16}$$

In order to eliminate the introduced variables α_i and η^j, further analysis is needed. It is known that solution values of η^j for all nonexistent phases will be strictly negative. That being the case, by inspection of the equilibrium constraints in problem (2.16), if η^j were to be increased to zero as it is for all existing phases, what must result in order to keep the equations satisfied is that all $\overline{G}_i^j(T, P, \mathbf{y}^j)$ values of the nonexisting phases will be reduced. In order to accomplish this, the component mole fractions will be reduced because the partial molar Gibbs energy, expressed as

$$\overline{G}_i^j(T, P, \mathbf{y}^j) = G_i^j(T, P) + RT \ln(y_i^j) + \overline{G}_i^{Ej}(T, P, \mathbf{y}^j) \tag{2.17}$$

where $G_i^j(T, P)$ is the Gibbs energy of pure component i in phase j and $\overline{G}_i^{Ej}(T, P, \mathbf{y}^j)$ is the partial molar excess Gibbs energy, is an increasing function of composition. Therefore, instead of being normalized, the component mole fractions will be forced to satisfy the condition

$$\sum_{i \in C} y_i^j < 1 \tag{2.18}$$

This is precisely the result of Michelsen's phase stability analysis [32] [34]. In the process, the variable η^j is no longer needed since it can be set to zero at the solution under all conditions. However, a new boundary expression will need to replace $\eta^j + \phi^j$. To mimic the property that η^j used to be less than zero for all nonexisting phases, the expression $\sum_{i \in C} y_i^j - 1$ is used as its replacement. Finally, in expressing equilibrium now among all phases, existent or not, it is typical to choose some reference phase $r \in F$. This enables the elimination of the unwanted computation of α_i by subtracting away some of the equations. Problem (2.16) is presented again in compact form.

$$\begin{array}{l}
\left\{ \begin{array}{l} \sum_{i \in C} y_i^j + \phi^j \geq 1 \\ \sum_{i \in C} y_i^j = 1 \end{array} \right\} \bigvee \left\{ \begin{array}{l} \sum_{i \in C} y_i^j + \phi^j < 1 \\ \phi^j = 0 \end{array} \right\}, \ \forall_{j \in F} \\
\overline{G}_i^j(T, P, \mathbf{y}^j) = \overline{G}_i^r(T, P, \mathbf{y}^r), \ \forall_{i \in C}, \ \forall_{j \in F - \{r\}} \\
\sum_{j \in F} \phi^j y_i^j = z_i, \ \forall_{i \in C}
\end{array} \tag{2.19}$$

The equilibrium constraints of (2.19) can be written in alternative forms such as through the familiar use of fugacity coefficients which are further expressed as products of contributing factors such as activity coefficients. Although this is very common in procedural modeling, it has been the experience of this work that, in an equation based modeling environment, expressions involving terms consisting of the product of many variables is not desireable because of the severe nonlinearities that are introduced. Excess and residual properties have the advantage of being added to ideal properties which may preserve the degree of nonlinearity.

Formulation (2.19) is applied to a ternary system involving benzene, ethanol, and water ($C = \{B, E, W\}$). At most, three phases are expected to exist simultaneously,

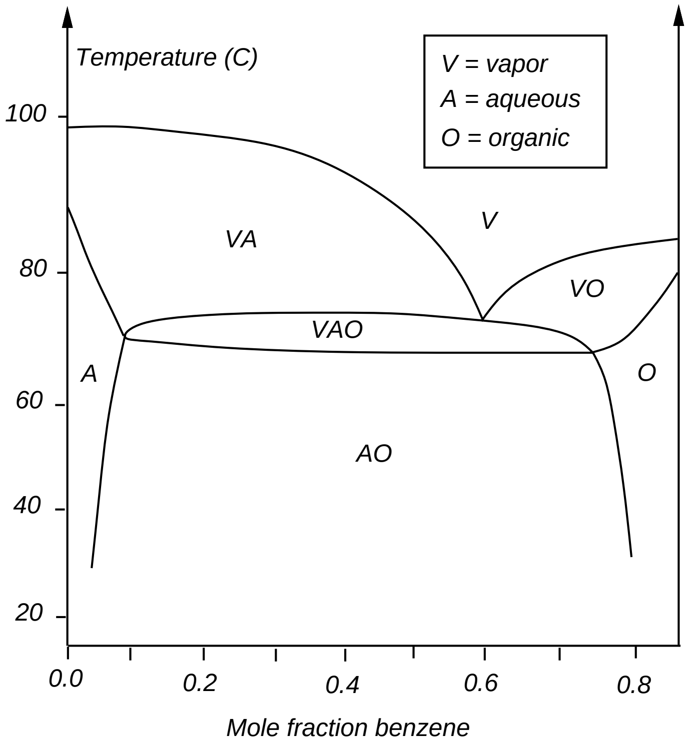

Figure 2.1: Phase diagram for a benzene-ethanol-water mixture

an aqueous liquid phase, an organic liquid phase, and a vapor phase ($F = \{A, O, V\}$). The problem will be to solve for the complete thermodynamic state of the system at a fixed temperature T and pressure P, determining in the process which of the three phases are present. With three potential phases, there are three logical conditions which give rise to eight possible states. However, one state in which all phases are absent is physically unrealizeable and the search region corresponding to that combination can be removed from the problem. Thus there are only seven different possibilities in the phase distribution. Pham and Doherty report qualitatively the data shown in Figure 2.1 which was reproduced with a fixed ethanol concentration of fifteen mole percent and at a pressure of one atmosphere [38]. The seven search regions are identified as areas on the phase diagram.

The vapor phase is modeled as an ideal gas mixture. There is no partial molar excess contribution to the partial molar Gibbs energy of the components in the vapor phase and as a result, (2.17) becomes

$$\overline{G}_i^V(T,P,\mathbf{y}^V) = G_i^V(T,P) + RT\ln(y_i^V) \tag{2.20}$$

The liquid mixtures are modeled as regular solutions in order to estimate the partial molar excess quantities which, for this system of components, are expected to be largely different from zero.

$$\overline{G}_i^A(T,P,\mathbf{y}^A) = G_i^A(T,P) + RT\ln(y_i^A) + \overline{G}_i^{EA}(T,P,\mathbf{y}^A) \tag{2.21}$$

$$\overline{G}_i^O(T,P,\mathbf{y}^O) = G_i^O(T,P) + RT\ln(y_i^O) + \overline{G}_i^{EO}(T,P,\mathbf{y}^O) \tag{2.22}$$

The liquid pure components are modeled as incompressible liquids with constant molar densities ρ_i and saturation pressures $P_i^{sat}(T)$ which behave as a function of temperature as given by Reid *et. al.* [47]. The free energies are derived by condensation from the ideal gas state via isothermal expansion and compression.

$$G_i^A(T,P) = G_i^V(T,P) + RT\ln(\frac{P_i^{sat}(T)}{P}) + \frac{P - P_i^{sat}(T)}{\rho_i} \tag{2.23}$$

$$G_i^O(T,P) = G_i^V(T,P) + RT\ln(\frac{P_i^{sat}(T)}{P}) + \frac{P - P_i^{sat}(T)}{\rho_i} \tag{2.24}$$

The term involving ρ_i in equation (2.23) and (2.24) is better known as the Poynting contribution and is often left out because of its relatively small influence on the free energy calculation. By choosing phase V as the reference phase and substituting functionality for the liquid saturation pressures and partial molar excess energies, the equilibrium relations are simplified and, as a result, the problem is formulated below. In order to more clearly illustrate the formulation for this example, only 3 search regions are listed, namely those where exactly one phase is absent. The specifications of temperature and pressure can always be designed for this example to make sure the solution lies in one of these search regions. These search regions correspond to the designated areas VA, VO, and AO in Figure 2.1.

$$\left\{\begin{array}{l}\sum\limits_{i\in C} y_i^A + \phi^A \geq 1 \\ \sum\limits_{i\in C} y_i^O + \phi^O \geq 1 \\ \sum\limits_{i\in C} y_i^V + \phi^V < 1 \\ \sum\limits_{i\in C} y_i^A = 1 \\ \sum\limits_{i\in C} y_i^O = 1 \\ \phi^V = 0\end{array}\right\} \bigvee \left\{\begin{array}{l}\sum\limits_{i\in C} y_i^A + \phi^A \geq 1 \\ \sum\limits_{i\in C} y_i^O + \phi^O < 1 \\ \sum\limits_{i\in C} y_i^V + \phi^V \geq 1 \\ \sum\limits_{i\in C} y_i^A = 1 \\ \phi^O = 0 \\ \sum\limits_{i\in C} y_i^V = 1\end{array}\right\} \bigvee \left\{\begin{array}{l}\sum\limits_{i\in C} y_i^A + \phi^A < 1 \\ \sum\limits_{i\in C} y_i^O + \phi^O \geq 1 \\ \sum\limits_{i\in C} y_i^V + \phi^V \geq 1 \\ \phi^A = 0 \\ \sum\limits_{i\in C} y_i^O = 1 \\ \sum\limits_{i\in C} y_i^V = 1\end{array}\right\} \tag{2.25}$$

$$\ln\left(\frac{y_i^V P}{y_i^A P_i^c}\right) = \frac{1}{T}\sum_{j\in C}\left(A_{ij} - \frac{1}{2}\sum_{k\in C} A_{jk} y_k^A\right) y_j^A + \frac{T_i^c}{T}\sum_{k\in\{1\ldots4\}} B_{ik}\left(1 - \frac{T}{T_i^c}\right)^{c_k},\ \forall_{i\in C}$$

$$\ln\left(\frac{y_i^V P}{y_i^O P_i^c}\right) = \frac{1}{T}\sum_{j\in C}\left(A_{ij} - \frac{1}{2}\sum_{k\in C} A_{jk} y_k^O\right) y_j^O + \frac{T_i^c}{T}\sum_{k\in\{1\ldots4\}} B_{ik}\left(1 - \frac{T}{T_i^c}\right)^{c_k},\ \forall_{i\in C}$$

$$\phi^A y_i^A + \phi^O y_i^O + \phi^V y_i^V = z_i,\ \forall_{i\in C}$$

The following data are applicable for the benzene-ethanol-water system [38] [47].

$$\begin{aligned}
\mathbf{A} &= \begin{bmatrix} 0.0\{K\} & 576.3\{K\} & 1074.5\{K\} \\ 576.3\{K\} & 0.0\{K\} & 351.8\{K\} \\ 1074.5\{K\} & 351.8\{K\} & 0.0\{K\} \end{bmatrix} \\
\mathbf{B} &= \begin{bmatrix} -6.98273 & 1.33213 & -2.62863 & -3.33399 \\ -8.51838 & 0.34163 & -5.73683 & 8.32581 \\ -7.76451 & 1.45838 & -2.77580 & -1.23303 \end{bmatrix} \\
\mathbf{c} &= \begin{bmatrix} 1.0 & 1.5 & 3.0 & 6.0 \end{bmatrix}^T \\
\mathbf{T^c} &= \begin{bmatrix} 562.2\{K\} & 516.2\{K\} & 647.4\{K\} \end{bmatrix}^T \\
\mathbf{P^c} &= \begin{bmatrix} 48.3\{atm\} & 63.0\{atm\} & 217.6\{atm\} \end{bmatrix}^T
\end{aligned}$$

The additional constant parameters for problem (2.25) for which specifications are required are T, P, and $\mathbf{z}$. This leaves alternative sets of $(|C|+1)|F|$ equations in a consistent set of $(|C|+1)|F|$ variables. Among all the phases, it is not clear based on the specifications which phase will disappear. It should be noted that formulation

(2.10) is known to exhibit many local minima when two or more of the candidate phases are liquid. This is because the same functionality is used in estimating the partial molar excess free energy of all liquid phases [57]. In the vapor-liquid-liquid equilibrium problem described above, it is often difficult to avoid obtaining the trivial solution where, for two liquid phases present at equilibrium, the compositions are identical. Proper initialization can help to obtain the desired solution where both liquid phases may exist with different compositions.

2.4.2 Heat exchange

In Figure 1.3, it was mentioned that a difficulty exists in integrating conditional models having differential equations. The main difficulty is preserving rigor when the integrated variable is nonsmooth. To accomplish this, the domain of integration must be partitioned into finite elements in such a way that transition of the integrated variables between states does not occur internal to any element. This is because the numerical methods for integrating across an element assume smoothness by using gradient evaluations at only a finite number of points in the element. Therefore, should transition occur within the domain of integration, the elements will almost surely need to be adjusted in width to accommodate. For cases where the initial conditions of all of the integrated variables are given, tailored algorithms are offered which integrate cautiously, adjusting the advancement as transitions become detected [3]. This has direct applications to solving processes which involve discrete events taking place at certain points in time. What is more interesting is formulating and solving models where both initial and final conditions are specified for the integrating variables. In this case, no efficient tailored algorithm can be devised to determine if and where transition takes place.

A general method for integrating conditional models having differential equations is to discretize the differential equations first and treat the problem as a conditional model with only algebraic equations as originally posed in (2.3). To accomplish this, a relay method is introduced. The name emerges from the observation that the point in the domain of integration at the solution where transition occurs will get continuously passed along, as a baton in a relay race, from one element to another by successive contractions and expansions of the individual elements, similar to a compression wave moving along a spring, if driven by a continuous change in the values of the model constants. Switching stations at which the analogous baton transfer occurs must first be positioned. This can be best demonstrated using the heat exchanger example introduced in Figure 1.3.

At any given position within a heat exchanger, a differential amount of heat from a hot stream, flowing through the shell, is transfered to a cold stream, flowing countercurrently through the tubes, across a differential amount of area per unit time. When integrated over each finite element, the net amount of heat transfered will enforce a change in the enthalpies of the two flowing streams and therefore result in temperature changes as well. Integration across the elements is typically carried out using a quadrature method which applies a common propagation formula to predict changes in the heat transfer rate [11]. Propagation is carried out assuming a smooth profile for the heat transfer rate throughout the element which can be accurately described by a polynomial. In this example, the trapezoidal rule is used. Difficulty arises when the temperature change of one of the streams is enough to cause phase change as phase change creates a nonsmooth profile for the heat transfer rate. In that case, it is expected that the integration will be more accurate if the element were able to be broken up into two subelements at precisely the point where phase change first occurs.

In this example, three finite elements are chosen with one switching station but the formulation can be generalized for more. To outline the three elements, four positions referenced by the indices $\{0 \ldots 3\}$ are used. The domain of integration is transformed to the dimensionless variable η which varies from zero to one so that the differential area can be expressed as $Ad\eta$ where A is the total heat transfer area in the heat exchanger. To model countercurrent heat exchange and utilize all of the heat transfer area, the cold stream is introduced at position $\eta^0 = 0$ while the hot stream is introduced at position $\eta^3 = 1$. The placement of η^1 and η^2 will depend on phase change. The molar flows of the hot and cold streams are given by F_h and F_c respectively. There is negligible loss of pressure in both streams and the same pressure P, set to $1\{atm\}$, is used. The instantaneous heat transfer rate is Q. Due to possible shell-side condensation, a variable heat transfer coefficient is used to relate dQ/dA to the local temperature driving force. The presence of mist and subsequently condensation droplets on the shell side of the tube walls will have a sharply increasing effect on the heat transfer coefficient [10]. It is adequate to express the heat transfer coefficient as a linear function of ϕ, the fraction of the hot stream which is condensed. The composition of the condensation droplets, $\mathbf{x}$, and the composition of the vapor, $\mathbf{y}$, are related to each other by phase equilibria and to the overall stream composition, $\mathbf{z}$, via a material balance involving ϕ. The set of components C consists of butane, pentane, and hexane ($C = \{B, P, H\}$). The additional thermal properties for the hot stream include temperature T_h and molar enthalpy H_h. The cold stream is made up entirely of water and is expected not to undergo phase change throughout the length of the heat exchanger. The parameters needed at all positions for the cooling water are the temperature T_c and the molar enthalpy H_c. For reference, the molar enthalpy of both streams is set to zero at a temperature of $540\{R\}$.

The difficulty with this model is that, in addition to solving for the temperature profile, the dimensions of the finite elements are to be solved for as well simultaneously. This is done by solving for the placement of the interior evaluation positions η^1 and η^2 such that at least one of them locates the point of condensation should condensation occur internally while the other resides at the switching station, positioned at the midpoint of the domain. It is assumed that the hot stream entering the shell side is superheated. Therefore, there are three alternatives of interest. Either condensation does not occur, condensation appears somewhere between the outlet of the shell side and the switching station, or it first appears somewhere between the switching station and the inlet of the shell side. With continuous heat transfer occurring down the length of the heat exchanger, the alternative that condensation appear in the inlet side of the switching station and disappear in the outlet side of the switching station is not physically meaningful. The temperature profiles are assumed to be monotone. By applying the conditions developed for phase equilibria at the outlet of the shell side and at the switching station, the presence of condensate in each of the three elements can be deduced.

The first alternative is that not enough heat is drawn from the hot stream to cause condensation. In this case, the hot stream will be in the vapor state throughout the heat exchanger and the leftmost element (bordered by η_0 and η_1) will collapse by setting η_1 to η_0 since the other two elements, separated by position $\eta^2 = 0.5$, are sufficient to integrate accurately. This is shown in Figure 2.2a. If more heat were transfered to cause condensation, the point at which it occurs may be located to the left of the switching station. In this case, while keeping η_2 fixed at the switching station, η_1 is sent to go fetch the point of condensation. The result is that condensate will exist in the hot stream within the leftmost element only. This is shown in Figure 2.2b. If still more heat is transfered, the condensation point may be isolated to the right of the switching station in which case η_1 and η_2 switch roles. Here, pure vapor will exist in the hot stream within the rightmost element only. This is shown in Figure 2.2c. For this example, $500000\{BTU/hour\}$ are to be removed from a $600\{R\}$ paraffin stream. There exists a refrigeration unit whose operating cost is assumed to increase quadratically with increasing cooling duty. An optimal design for a water cooler is sought to be used upstream of the existing refrigeration unit so that its annualized installed cost along with its operating cost still results in a beneficial reduction in the operating cost of the existing unit [14].

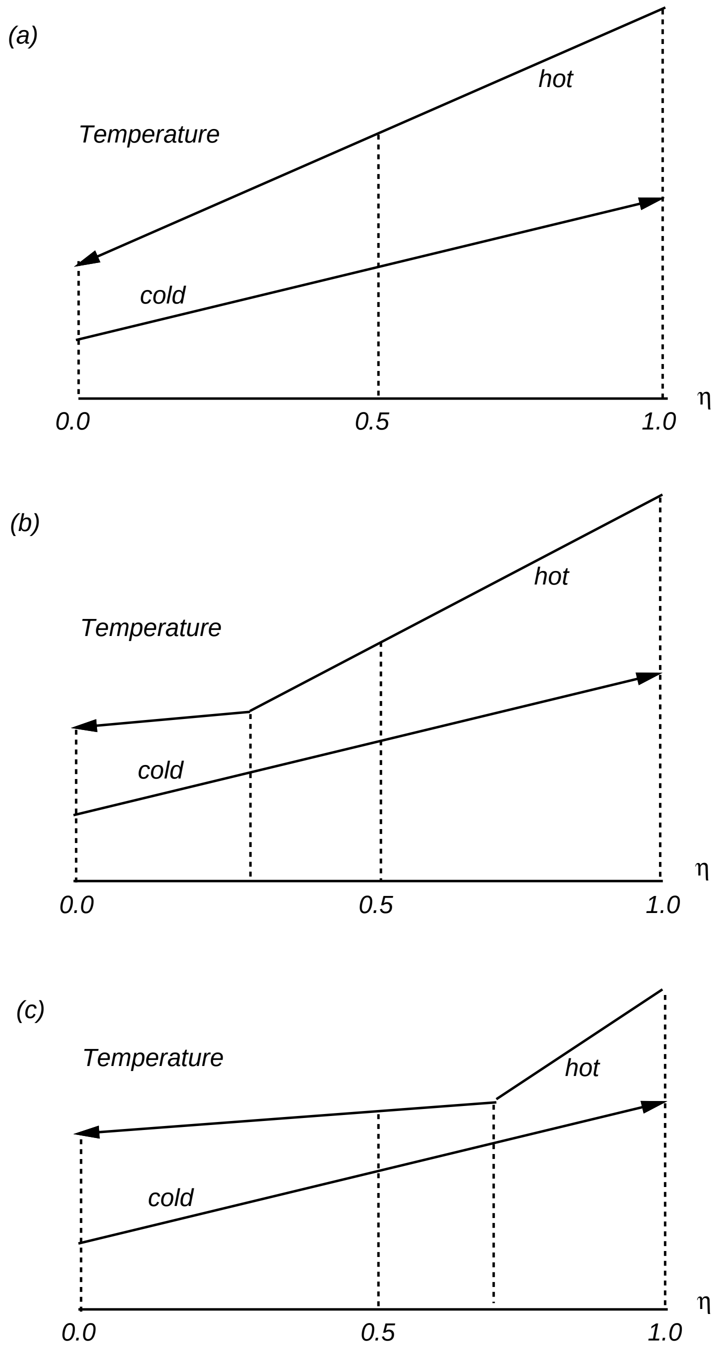

Figure 2.2: Alternative heat exchanger temperature profiles

$$\min \quad 0.238\left(\frac{A}{1.0\{ft^2\}}\right)^{0.65} + \frac{0.0013F_c}{1.0\{\frac{lbmole}{hour}\}} + 17.0\left(1 - \frac{F_c(H_c^3 - H_c^0)}{500000\{\frac{BTU}{hour}\}}\right)^2 \tag{2.26}$$

$$s.t. \quad \left\{\begin{array}{l} \sum_{i\in C} x_i^0 + \phi^0 < 1 \\ \sum_{i\in C} x_i^2 + \phi^1 < 1 \\ \eta^1 = \eta^0 \\ \phi^0 = 0 \\ \eta^2 = 0.5 \\ \phi^1 = 0 \end{array}\right\} \bigvee \left\{\begin{array}{l} \sum_{i\in C} x_i^0 + \phi^0 \geq 1 \\ \sum_{i\in C} x_i^2 + \phi^1 < 1 \\ \sum_{i\in C} x_i^0 = 1 \\ \sum_{i\in C} x_i^1 = 1 \\ \eta^2 = 0.5 \\ \phi^1 = 0 \end{array}\right\} \bigvee \left\{\begin{array}{l} \sum_{i\in C} x_i^0 + \phi^0 \geq 1 \\ \sum_{i\in C} x_i^2 + \phi^1 \geq 1 \\ \sum_{i\in C} x_i^0 = 1 \\ \sum_{i\in C} x_i^1 = 1 \\ \eta^1 = 0.5 \\ \sum_{i\in C} x_i^2 = 1 \end{array}\right\}$$

$$\phi^2 = 0$$

$$\phi^3 = 0$$

$$\phi^j x_i^j + (1-\phi^j) y_i^j = z_i, \ \forall_{i\in C}, \ \forall_{j\in\{0\ldots3\}}$$

$$\ln\left(\frac{y_i^j P}{x_i^j P_i^c}\right) = \frac{T_i^c}{T_h^j} \sum_{k\in\{1\ldots4\}} B_{ik}\left(1 - \frac{T_h^j}{T_i^c}\right)^{c_k}, \ \forall_{i\in C}, \ \forall_{j\in\{0\ldots3\}}$$

$$\frac{dQ^j}{d\eta} = 20\left\{\frac{BTU}{hr\,ft^2\,R}\right\}(1+10\phi^j)A(T_h^j - T_c^j), \ \forall_{j\in\{0\ldots3\}}$$

$$F_h(H_h^{j+1} - H_h^j) = F_c(H_c^{j+1} - H_c^j), \ \forall_{j\in\{0\ldots2\}}$$

$$F_c(H_c^{j+1} - H_c^j) = \frac{1}{2}\left(\frac{dQ^j}{d\eta} + \frac{dQ^{j+1}}{d\eta}\right)(\eta^{j+1} - \eta^j), \ \forall_{j\in\{0\ldots2\}}$$

$$H_c^j = 9720\left\{\frac{BTU}{lbmole}\right\}\left(\frac{T_c^j}{540\{R\}} - 1\right), \ \forall_{j\in\{0\ldots3\}}$$

$$H_h^j = \sum_{i\in C}\left[z_i \sum_{k\in\{1\ldots4\}} D_{ik}\left(\left(\frac{T_h^j}{540\{R\}}\right)^k - 1\right) - \phi^j x_i^j H_i^{vap}\right], \ \forall_{j\in\{0\ldots3\}}$$

The following data is applicable for the paraffin mixture [47].

$$\mathbf{B} = \begin{bmatrix} -6.88709 & 1.15157 & -1.99873 & -3.13003 \\ -7.28936 & 1.53679 & -3.08367 & -1.02456 \\ -7.46765 & 1.44211 & -3.28222 & -2.50941 \end{bmatrix}$$

$$\mathbf{D} = \begin{bmatrix} 1224\left\{\frac{BTU}{lbmole}\right\} & 6410\left\{\frac{BTU}{lbmole}\right\} & -429\left\{\frac{BTU}{lbmole}\right\} & -2\left\{\frac{BTU}{lbmole}\right\} \\ -468\left\{\frac{BTU}{lbmole}\right\} & 9428\left\{\frac{BTU}{lbmole}\right\} & -998\left\{\frac{BTU}{lbmole}\right\} & 46\left\{\frac{BTU}{lbmole}\right\} \\ -569\left\{\frac{BTU}{lbmole}\right\} & 11260\left\{\frac{BTU}{lbmole}\right\} & -1207\left\{\frac{BTU}{lbmole}\right\} & 57\left\{\frac{BTU}{lbmole}\right\} \end{bmatrix}$$

$$\mathbf{c} = \begin{bmatrix} 1.0 & 1.5 & 3.0 & 6.0 \end{bmatrix}^T$$

$$\mathbf{T^c} = \begin{bmatrix} 765.4\{R\} & 845.3\{R\} & 913.3\{R\} \end{bmatrix}^T$$

$$\mathbf{P^c} = \begin{bmatrix} 37.5\{atm\} & 33.3\{atm\} & 29.3\{atm\} \end{bmatrix}^T$$

$$\mathbf{H^{vap}} = \begin{bmatrix} 9634\left\{\frac{BTU}{lbmole}\right\} & 11088\left\{\frac{BTU}{lbmole}\right\} & 12413\left\{\frac{BTU}{lbmole}\right\} \end{bmatrix}^T$$

To summarize, formulation (2.26) consists of 3 alternative sets of 48 equations of which at least 44 equations are common among all of the sets. With specification of $\mathbf{z}$, T_h^3, T_c^0, F_h, η^0, η^3, and P, there are 50 variables which leaves two degrees of freedom. A common choice for the decision variables with which to optimize are A and F_c.

2.4.3 Adiabatic compressible flow

As pointed out in Figure 1.4, fluid flow transition occurs in some conditional models. One such model is constructed here where the laminar state is avoided. It describes the flow of a compressible gas in an adiabatic frictional circular pipe of constant diameter. The parameters of the model are the initial and final Mach numbers M_i and M_f, the initial and final temperatures T_i and T_f, the initial and final pressures P_i and P_f, the discharge pressure P_d, the pipe diameter and length, D and L, and the molar flow rate F. Other quantities include the molecular weight mw, the heat capacity ratio γ, the universal gas constant R, and fanning friction factor f. The gas is to be delivered from a well at a pressure of P_i and a temperature of T_i through a

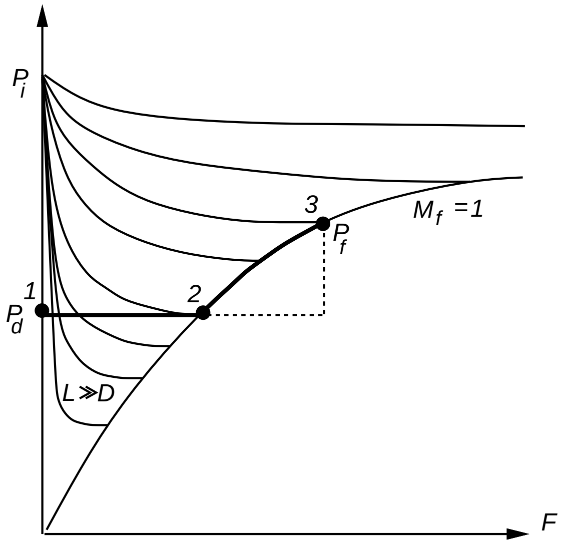

Figure 2.3: Adiabatic compressible flow in a constant diameter pipe

pipe yet to be installed to a reservoir at a pressure of P_d.

Nonsmooth functionality occurs due to possible transition to sonic flow at the outlet when investigating designs of varying diameters. This is shown more clearly in Figure 2.3. Starting at point 1 and gradually increasing the diameter size, the flow will be observed to increase while the pressure, P_f, just inside the end of the pipe remains equal to the discharge pressure, P_d. Point 2 will eventually be reached where the velocity of the fluid exiting the pipe is equal to that of sound in the fluid at the exiting conditions. Any attempt to further increase the diameter will follow along the path toward point 3 where P_f will be forced to be higher than P_d, yielding shock waves which are produced by the sudden isothermal and irreversible drop in pressure. In formulating this model, the condition $P_d - P_f \geq M_f - 1$ can indicate if the flow is sonic or not based on the above information. Below critical flow, no shock waves are produced ($P_d = P_f$), the outlet Mach number M_f will be less than one, and the condition will be met. During sonic flow, however, $P_d < P_f$, the outlet Mach number M_f will be equal to one, and the condition will be violated. The equations for an ideal gas are derived to govern the behavior described above [31]. The problem is to design the pipe to maximize the profit obtained by subtracting the annualized installed cost of the pipe from the value of the gas produced. The same solution can be found by minimizing the negative of the profit.

$$\begin{aligned}
\min \quad & 0.425\left(\frac{D}{1.0\{cm\}}\right)^{2.5}\left(\frac{L}{1.0\{m\}}\right) - \frac{mw\,F}{1.0\{\frac{g}{s}\}} \\
s.t. \quad & \left\{\begin{array}{l} P_d - P_f \geq M_f - 1 \\ P_d = P_f \end{array}\right\} \bigvee \left\{\begin{array}{l} P_d - P_f < M_f - 1 \\ M_f = 1 \end{array}\right\} \\
& \frac{FRT_i}{P_i} = \frac{\pi D^2 M_i}{4}\sqrt{\frac{\gamma R T_i}{mw}} \\
& \frac{FRT_f}{P_f} = \frac{\pi D^2 M_f}{4}\sqrt{\frac{\gamma R T_f}{mw}} \\
& \frac{T_f}{T_i} = \frac{1 + \frac{\gamma-1}{2}M_i^2}{1 + \frac{\gamma-1}{2}M_f^2} \\
& \frac{1}{M_i^2} - \frac{1}{M_f^2} - \frac{4\gamma f L}{D} = \frac{\gamma+1}{2}\ln\left(\frac{M_f^2(1+\frac{\gamma-1}{2}M_i^2)}{M_i^2(1+\frac{\gamma-1}{2}M_f^2)}\right)
\end{aligned} \tag{2.27}$$

Problem (2.27) consists of 2 sets of 5 equations of which 4 equations are common. Using methane gas, specifications include $mw = 16\{g/gmole\}$ and $\gamma = 1.292$. For the pipe, a length of $L = 100\{m\}$ and $f = 0.01$ are chosen. Finally, the inlet conditions $T_i = 300\{K\}$ and $P_i = 10\{atm\}$ as well as the discharge pressure $P_d = 5\{atm\}$ are known. This leaves 6 variables, one of which (D, for example) can be selected as a decision variable for the optimization.

Chapter 3

Structural Analysis

3.1 Introduction

Prior to solving, a model formulation is analyzed to determine if it is well-posed. Techniques are available which may detect if any structural inconsistencies exist among the equations which in turn indicates a likely empty feasible region. These techniques are based on the idea that models are sparse. In other words, all equations are likely to involve only a few of the variables. The subset of variables appearing in an equation are called incident variables. Although each variable may appear in more than one equation, most equations are expected to have different incident sets. Structural analysis of the equations is the process of exploiting model sparsity.

3.2 Continuous and differentiable models

An incidence matrix offers an effective medium with which the sparsity of (2.1) is exploited [60]. The elements of the vector, or enumerated set, of variables, $\mathbf{x}$, each have a specific index from the set $\{1 \ldots n\}$ and the elements of the vector of equation residuals, $\mathbf{g}(\mathbf{x})$, each have a specific index from the set $\{1 \ldots m\}$. A matrix $\mathbf{I}$ is constructed with n columns and m rows so that initially each column can refer to the variable of the same index and each row can refer to the equation residual of the same index. A sparse incidence pattern is then created by deleting all elements I_{ij} for which $\partial g_i(\mathbf{x})/\partial x_j = 0,\ \forall_{\mathbf{x}\in R^n}$ and inserting all others. This condition that the partial derivative vanishes everywhere is simply a mathematical statement that variable x_j does not contribute to the evaluation of residual $g_i(\mathbf{x})$. An algorithm to generate the incidence pattern can be more clearly stated in a stepwise procedure.

Algorithm 1 *Constructing an incidence matrix*

1 **for each** $i \in \{1 \ldots m\}$
2   **for each** $j \in \{1 \ldots n\}$
3   **if** $(\partial g_i(\mathbf{x})/\partial x_j = 0,\ \forall_{\mathbf{x}\in R^n})$
4   **delete** I_{ij}
5   **else**
6   **insert** I_{ij}

The nested loops of lines 1 and 2 visit all positions of the matrix elements so that the condition of line 3 can determine whether or not an element should be placed. To illustrate, an incidence matrix is constructed for problem (2.2). The formulation as written produces the incidence pattern given in Figure 3.1. The equations are indexed in the order in which they are listed in the formulation and the indices appear to the left of their corresponding rows. The variable names appear above their respective columns. As will be shown next, by rearranging the variables among the columns in a systematic manner, tests for conditions of a well-posed model are conducted.

3.2.1 Structural consistency

One of the requirements for a well-posed model is that it not be over-specified. This requires $m \leq n$. However, this is only a necessary condition and not a sufficient one. Even in situations where the number of variables exceeds the number of equations, there may exist a subset of equations which exceed in number the combined variables on which they are incident. All those equations which if removed from the subset

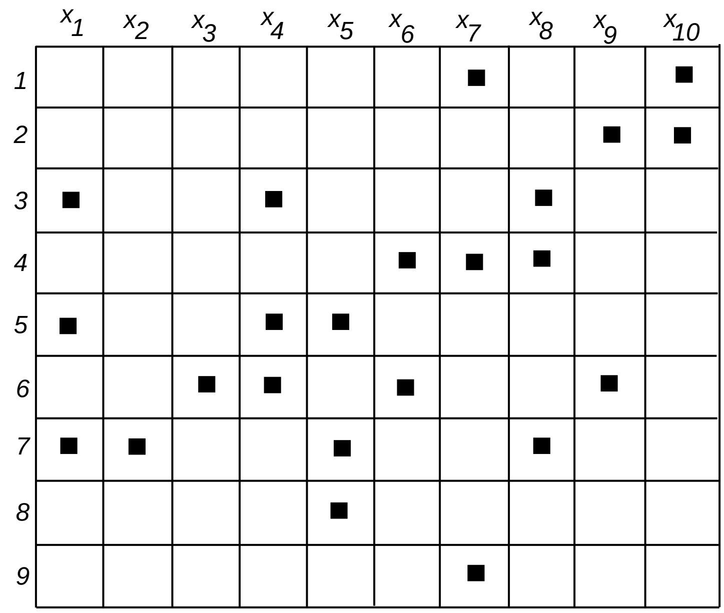

Figure 3.1: Incidence matrix

would restore the proper specification are referred to as being structurally singular or inconsistent. This suggests that with the equations not removed, no matter what values are used for the variables, the Jacobian matrix, $\nabla_{\mathbf{x}}\mathbf{g}(\mathbf{x})^T$, will have numerically singular rows. Numerical singularity may be encountered even with well-posed systems so it should be noted that structural consistency is only a necessary condition for numerical consistency, not a sufficient one.

Structural singularities are detected by performing an output assignment. This is the process of assigning each equation to one of its incident variables. The structural rank, which will be computed using the quantity $rank$, is the largest number of equations which can be assigned such that no two equations are assigned to the same variable. If $rank = m$, the equations are structurally nonsingular. An algorithm for conducting an output assignment, and, at the same time, testing for structural singularity, employs a depth-first search and is described using the incidence matrix [15] [62]. Initially, $rank$ is set to m. Starting with the first row, all of the variables are initially unassigned so any one of the variables incident on the equation of the first row can be assigned to it. This is done by moving the chosen variable over to the first column so that this first assignment starts the forming of a diagonal. As the algorithm proceeds to the next row, the number of unassigned variables is decremented and the diagonal is extended. Eventually, a row may be encountered where all of the incident variables of the associated equation have been already assigned to some of the equations of the previous rows. In this case, it must be investigated to see if any of these previously assigned equations have incident variables which are currently available for assignment. If not, the investigation must be continued recursively before concluding that an assignment simply cannot be made. In this case, the equation of the current row is structurally singular and is moved to the last row before decrementing *rank* so that it will not be visited again.

This algorithm is stated more clearly by first declaring a recursive call-by-reference integer function *assign* whose task is to assign an equation to one of its incident variables. An implementation of such a function is presented by Karl Westerberg [62]. The function takes, as arguments, the row i where the equation to assign currently resides, a set *range* of columns in which to search for a variable available for assignment, and a set V of rows which have been previously visited. It is assumed that all of the equations situated in rows $\{1 \ldots (i-1)\}$ have already been assigned. The function will return the column where the variable available for assignment was ultimately found. The incidence matrix $\mathbf{I}$ is assumed to have been prepared as prescribed above. It is also convenient to assume that the matrix is accessible from within the function along with its dimensions m and n.

Algorithm 2 *Function to assign an equation to a variable*

integer function $assign(i, range, V)$
    for each $j \in range$
        if (I_{ij} **exists**)
            exchange variables of columns i **and** j
            return j
    set $V = V \vee \{i\}$
    for each $j \in \{1 \ldots n\} - range$

```
8        if ((I_ij exists) AND (j ∉ V))
9           set k = assign(j, range, V)
10          if k ≠ 0
11             exchange variables of columns i and k
12             return k
13    return 0
```

Lines 2 through 5 check for the possibility that an incident variable, which is not yet assigned to another equation, exists for the equation in question. In this case, the column where it was found is returned to the caller of the function to notify that a successful assignment has been made. If line 6 is reached, the current row must be added to the set of visited rows V so that it doesn't get visited again. Then lines 8 through 12 search among the previously assigned rows for one whose equation could be reassigned to any of the variables located in *range*. If line 13 is reached, the search has been exhausted and a 0 is returned to inform the caller that the equation in question is structurally singular. It is important to set V to the empty set prior to calling the function with a new equation. To output assign as many equations as possible, the following loop is executed, as proposed by Karl Westerberg[62].

Algorithm 3 *Achieving an output assignment*

```
set i = 1
set rank = m
while (i ≤ rank)
     set V = ∅
     set range = {i ... n}
     if (assign(i, range, V) ≠ 0)
        set i = i + 1
     else
        exchange equations of rows i and rank
        set rank = rank − 1
```

If, at the end of the loop, $rank < m$, then the equations residing in rows $\{(rank + 1) \ldots m\}$ are causing structural singularities in the model. Furthermore, as an added feature, at the time an equation is concluded to be singular, the set V will contain

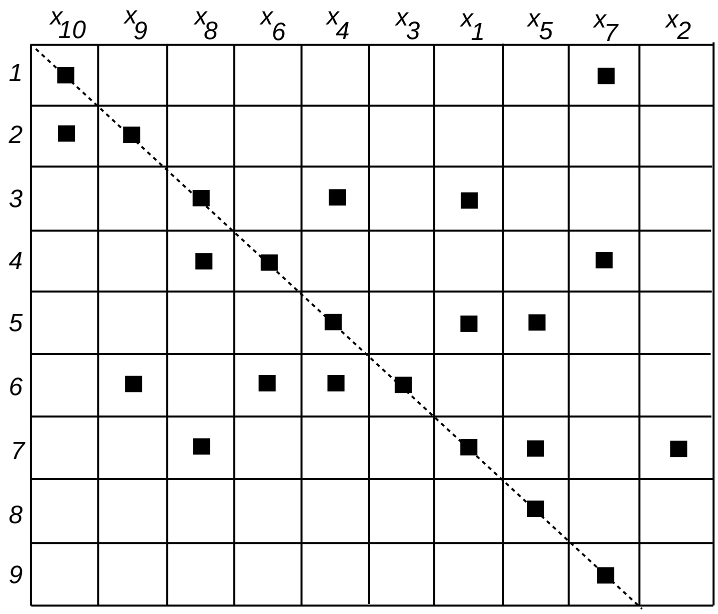

Figure 3.2: Output assignment

all the rows whose equations contribute to the structural dependency, as it is also sometimes called. Essentially, this means that the singularity will be resolved if either the singular equation or in fact any one of the equations residing at the rows of V were removed from the formulation. It is good practice to present all of them to the user for confirmation of the formulation error. It is assumed hereafter, then, that formulations have full structural rank since a means of correcting rank deficiency now exists.

The algorithm described above is applied to the incidence matrix of Figure 3.1 and the columns end up being rearranged to the order shown in Figure 3.2. The diagonal formed gives a plausible assignment, and it is concluded that the model is structurally consistent.

3.2.2 Partitioning the variables

When a structurally consistent output assignment is achieved, the variables are effectively partitioned into 2 groups. Those variables which are assigned make up m

dependent or state variables $\mathbf{y}$ while those left unassigned (in the case $m < n$) make up independent or decision variables $\mathbf{z}$. The independent variables are equal in number to the degrees of freedom. Incidentally, when inequality constraints are imposed, the degrees of freedom in the solution are not known since it cannot be guaranteed which constraints will be active at the solution. Nevertheless, the active set strategy can adjust the number of degrees of freedom as solution progress is made.

The partitioning of the variables is not unique. Furthermore, among the dependent variables of a given partition, the assignment of variables to equations is not unique. This confirms that the Jacobian can be inverted using different pivot sequences, the preferred choice being the sequence which achieves a desired balance between preserving sparsity and avoiding ill-conditioning in the inverse. This is a linear analysis issue and will be addressed later. As far as partitioning the variables, the largest number of partitions is obtained when all variables are incident upon all equations. In this case, there are $C_m^n = n!/(m!(n-m)!)$ combinations of n variables taken m at a time and for each combination, there are $P_m^m = m!$ permutations for assigning the equations among the dependent variables. Fortunately, though, model sparsity reduces the number of possible combinations and permutations of the variables and, as a result, some combinations of variables are restricted from being independent.

With multiple choices for the decision variables available, it is expected that some choices may serve to enhance convergence when solving better than others. Some arguments for this include sparsity and linearity. The first issue suggests that it is preferable to select decision variables in such a way that the assigned region of the matrix is as sparse as possible so that working with the Jacobian while solving will involve less work. However, from a convergence point of view, it might also be preferable to select decision variables in such a way that the equations are as linear as possible in the assigned variables so that it will be easier to maintain feasibility during solution. Without a truly definitive criterion to select the best choice, the

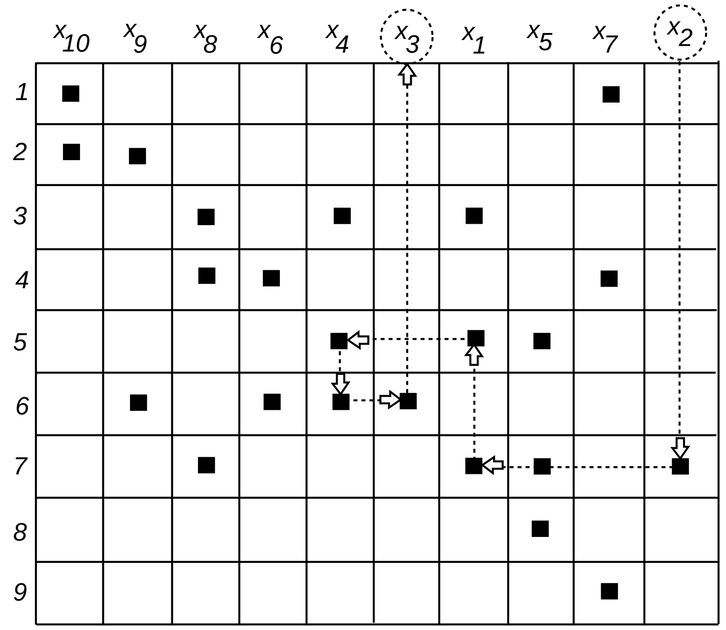

Figure 3.3: Steward path

ability to at least generate all choices and present them to the user is important. All selections of decision variables can be generated from an initial partition through the use of eligibility analysis. The initial partition is obtained from finding an output assignment. To generate another partition, some of the variables **y** are determined if they are eligible to be made independent in exchange for one of the variables **z** without altering the structural rank of the equations. This is accomplished by following all Steward paths, as they are called, stemming from the current decision variables and marking all state variables encountered along the paths as being eligible [60].

In the output assignment of Figure 3.2, x_2 is found to be an eligible independent variable placed in the end column outside of the assigned region. Variable x_2 is incident only in the equation in row 7. To find other possible choices for the independent variable, variable x_2 will need to be assigned to this equation. If row 7 is to be assigned to the last column, then the variable of column 7 (x_1) will need to be unassigned. Therefore, x_1 is an eligible variable. It is noticed that x_1 also appears in the equation of row 5. It is therefore further possible to re-assign equation residual

$g_5(\mathbf{x})$ to x_1 and unassign x_4, thereby making x_4 an eligible candidate as well. Variable x_4 is also shared by $g_6(\mathbf{x})$ which is currently assigned to x_3. Since x_3 does not appear in any other equation, the path terminates and other branches which were bypassed along the way are traversed to see if other variable destinations exist. The path just described is shown in Figure 3.3 and is one of many Steward paths which can be made. All of the Steward paths are essentially branches of a directed graph originating from the currently unassigned variables and terminating at some of the assigned variables. An algorithm for traversing all possible Steward paths and, in the process, gathering all eligible variables encountered along the way can best be written as a recursive function. Similar to how the function *assign* described above has the capability of recursively searching all existing paths to find a variable available for assignment, the function needed here must recursively exhaust all existing paths in order to gather the variables eligible for unassignment. A function *eligible* is created in this work which takes, as arguments, a column j, where a variable which is currently known to be eligible resides, and a set E, which is used to store the indices of eligible variables as they are found. A more lengthy nonrecursive version of the algorithm is presented by Piela [39].

Algorithm 4 *Function to collect all eligible dependent variables from one known independent variable*

1 **function** *eligible*(j, E)
2   **set** $E = E \vee \{$**index of variable in column** $j\}$
3   **for each** $i \in \{1 \ldots m\}$
4   **if** $((I_{ij}$ **exists**$)$ **AND** $($**index of variable in column** $i \notin E))$
5   *eligible*(i, E)

With a partition to start with, the set E is initialized to the empty set. The indices of the variables which are initially unassigned can immediately be added to E. But a more thorough approach would be to call the function *eligible* on the columns of each of these unassigned variables. Taking one variable at a time, in line

2, the index of the variable is sure to be a member of E. Then line 3 starts a loop over all equations in which the variable also appears because the function can then be called with the variables to which these equations are assigned, assuming that their indices have not already been added to E. As a result, by calling the function in a loop over all columns outside the assigned region, not only will the indices of independent variables be added to E but so will the index of any dependent variable which can be reached from them.

Algorithm 5 *Generating all eligible variables*

```
set E = ∅
for each j ∈ {(m + 1) ... n}
    eligible(j, E)
```

For the example (2.2), function *eligible* need only be called once with x_2 as the arbitrary independent variable. Upon returning, the complete set of eligible variables for this problem corresponding to the indices of E is $\{x_1, x_2, x_3, x_4, x_6, x_8\}$. It is interesting to point out that no matter which of these variables was used as the independent variable for the initial output assignment, the function *eligible* will generate the same set E.

3.2.3 Partitioning the equations

Because of sparsity, it will often be found among the equations, at least in instances where $m = n$, that a subset of some small number of equations can be solved for an equally small number of variables. With the solution values of these variables achieved, another subset may also be found in the remaining equations. This process is continued until no more subsets exist and results in a decomposition of the original problem into partitions which can be individually solved in some precedence order [60]. This is again efficiently accomplished through manipulation of the incidence matrix. This time, the equations will be rearranged among the rows as well in order to transform the matrix into lower block triangular form. When in this form, the region of the matrix above and to the right of the blocks, which are positioned along the diagonal, is empty. As a result, the equations and variables within each block make up each partition.

As described above, partitioning has been for the most part applied to systems of equations with no degrees of freedom or at least with no objective function. Algorithms used for block triangularizing the incidence matrix, therefore, are based on square systems [55] [60] [62]. Without an objective function, if arbitrary values for the decision variables $\mathbf{z}$, if any, were specified, the equations along with the variables $\mathbf{y}$ make up a structurally nonsingular system. The system of size m is then typically decoupled if sparsity allows. If optimization is desired, however, the decoupling must be lessened in order to account for the fact that solution values for the decision variables are to be computed from the optimal evaluation of an objective function. One way to look at this problem is to imagine the existence of $n - m$ additional equations which, if added to the existing ones, would produce a structurally nonsingular system of size n. Then, if solved, this enlarged system should yield the correct solution values for all of the variables and therefore be subject to the same decoupling described above if sparsity allows. The structure of these imaginary equations is investigated.

An equation which is assigned to one of its incident variables will often have other incident variables as well. One way to interpret the significance of this is that the assigned variable can be computed by the equation only if values for the other incident variables are provided. Often, though, values for the other incident variables are not available and, instead, they rely on the information of other equations. Coupling is achieved when at least one of the other incident variables relies on information from another equation which ultimately requires values for the assigned variable.

The currently independent variables are unassigned. It can be the duty of the imaginary equations to compute them. This indicates that these equations must at least involve the independent variables so that they can be assigned to each other. By the argument above, these equations will also require information, at least enough information with which to evaluate the objective function, so that optimal values for the independent variables can be computed. This further indicates that the variables incident on the objective function must make up the other incident variables in the imaginary equations.

A procedure is now devised to prepare all model structures for decoupling. First, the previously output assigned incidence matrix is supplemented with $n - m$ rows. In these rows, elements are placed in the unassigned columns. If this is done, the output assignment, as given by the current diagonal of the matrix, will extend down to the last row and column. In addition, elements are placed in all other columns of the rows corresponding to variables which are incident on the objective function $f(\mathbf{x})$. To the resulting $n \times n$ sparse output assigned incidence matrix, a conventional algorithm to search for possible decoupling is applied.

By having the added rows all share the same incidence pattern, they will be coupled in the same partition. Furthermore, this partition will end up being the only one with degrees of freedom so the optimization problem will sometimes be significantly reduced while many smaller partitions are created for which the task is to simply search for feasibility only. The degree of coupling will strongly depend on the choice of decision variables. This is because no assumption is made as to the form of the objective function. If the objective is separable in the sense that it consists of the sum of two or more terms with different incidence, for example, then it may be possible to break up the calculation of the decision variables across multiple partitions, each of which having the duty of optimizing part of the objective. This is not considered here and, instead, the burden is increased on the user to select the variables wisely during the eligibility analysis. It is envisioned that, by having as many as possible of the variables incident on the objective function also serve as the decision variables, decoupling will be maximized.

An efficient algorithm for partitioning square systems of equations is now needed. It must involve searching for couplings which may exist among the equations. Analysis of an equation involves a depth first search requiring first the analysis of all other equations which are assigned to compute values for the incident variables of the equation. A recursive function *analyze* is created to do that. What is presented here is a modification of Karl Westerberg's original algorithm which was tailored for use with a specific sparse matrix representation [62]. It takes as arguments a row i, a range of unvisited rows bounded inclusively by the indices *low* and *high*, and a row vector **link** which identifies coupling among the equations. During the course of analysis, the rows are broken up into three groups. Those in the range $\{1 \ldots (low-1)\}$ house the equations which have already been placed into partitioned blocks so far and the same range of columns contain the variables ordered so that this area of the matrix will be in lower block triangular form. The rows in $\{(high+1) \ldots n\}$ contain equations which are currently on a stack for later revisiting because the completion of their analysis is pending the completion of the analysis of the equation in row i. For those rows corresponding to the stack, entries in the vector **link** are updated to keep track of the coupling among the equations on the stack. Essentially, an entry $link_i$ for some row i will indicate the deepest row on the stack below row i whose equation to which the equation of row i is coupled. This implies also that the equations of all rows in between i and $link_i$ are also coupled.

Algorithm 6 *Function to analyze an equation for coupling*

function $analyze(i, low, high, \mathbf{link})$
    exchange equations of rows i **and** $high$
    exchange variables of columns i **and** $high$
    set $i = high$

```
5     set high = high − 1
6     set link_i = i
7     set j = n
8     while (j ≥ low)
9         if (I_ij exists)
10            if (j > high)
11                for each k ∈ {i ... (j − 1)}
12                    if link_j > link_k
13                        set link_k = link_j
14                set j = high
15            else
16                analyze(j, low, high, link)
17        else
18            set j = j − 1
19    if (link_i = i)
20        while (high < i)
21            set high = high + 1
22            exchange equations of rows low and high
23            exchange variables of columns low and high
24            set low = low + 1
25        return
26    return
```

Lines 2 through 5 push the equation found at row i onto the stack. The index i is then reset to the equation's new row. The equation is initially assumed to be coupled with itself so $link_i$ is set to i. If, at the end of the analysis, the value is observed to be different, then the equation is concluded to be coupled with another one which is currently on the stack. Lines 7 through 18 describe a reverse loop over all incident variables for the equation which have yet to be placed in a partition. The equations to which these variables are assigned are classified according to their location. For all of the equations currently on the stack (those residing in rows $\{(high+1)\ldots n\}$), the vector **link** is adjusted to keep track of the coupling and the loop is continued outside the stack. On the other hand, for all equations which reside in rows $\{low \ldots high\}$, a necessary recursive analysis is performed. If, after the analysis of all relevant incident variables for an equation is complete, the condition of line 19 still indicates that the equation links back to itself, then a block is formed including all equations found in rows $\{(high+1)\ldots i\}$ and all variables to which they are output assigned. The block is removed from the stack and deposited immediately after the most recent addition to an accumulated list of blocks along the diagonal. Initially, *low* is set to 1 and *high* is set to n. All equations can then initially be found in the set of unvisited rows $\{low \ldots high\}$. The algorithm terminates when there are no more unvisited rows, as prescribed by Karl Westerberg [62].

Algorithm 7 *Partitioning the equations*

set $low = 1$
set $high = n$
while $(low \leq high)$
    set $i = low$
    $analyze(i, low, high, \textbf{link})$

For problem (2.2), only 1 row is needed to square the incidence matrix. The variable partitioning achieved in Figure 3.2 is used. Row 10 is given an incidence in column 10, thereby output assigning the last row to variable x_2. In addition, elements are created in the columns corresponding to variables x_1, x_3, x_4, x_5, and x_7, all of which appear in the objective function. The extended problem appears in Figure 3.4a. The index 10^* is used to indicate the added equation structure. The partitioning algorithm is then applied and the lower block triangular result is shown in Figure 3.4b. There are 5 partitions which result. The first 4 blocks are singletons because they each involve solving 1 equation for 1 unknown. This leaves the last block requiring optimization of 6 variables subject to 5 constraints.

3.3 Conditional models

Sparsity in conditional models can be exploited with some restrictions. Among the alternative search regions, the sparsity is expected to change. This implies that some combinations of variables which can successfully make up the independent variables

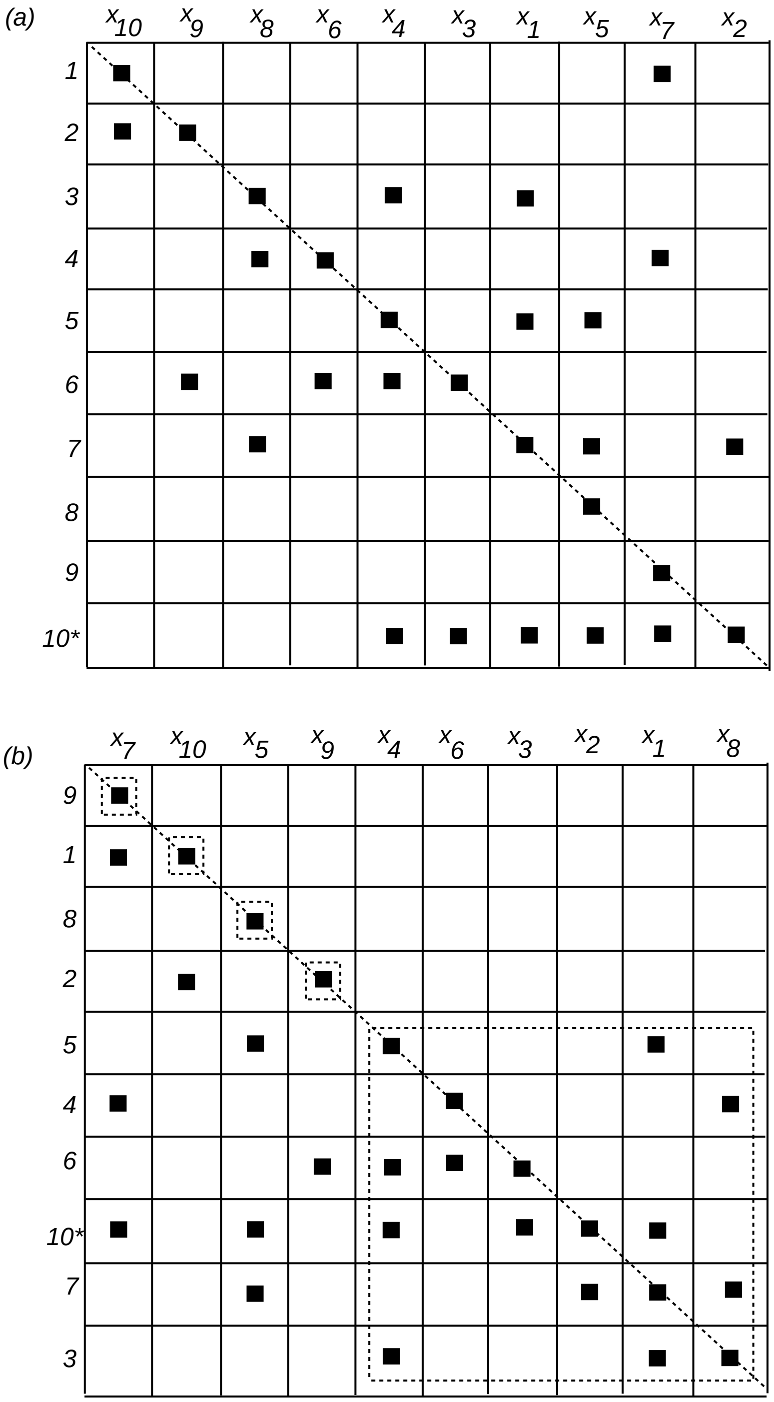

Figure 3.4: Lower block triangular form

for one search region may not be suitable for other search regions. In addition, isolation of a subset of variables from the other variables which may be successfully achieved through block partitioning of the equations of one search region may not be possible using the equations of another.

3.3.1 Search region consistency

A necessary condition for structural consistency with conditional models is that each of the alternative sets of equations are structurally consistent as prescribed above. However, this still does not guarantee that a consistent partitioning of the variables will exist such that output assignment of all of the equations in each search region can be completed using only the dependent variables. Search region consistency is assessed by confirming the existence of at least one such partitioning of the variables. To find a set of decision variables which are consistent with all search regions, each of the alternative sets of equations are first arbitrarily output assigned using all variables. For each output assignment, there is a partitioning of the variables which leads to generating a set of variables which are eligible to be independent, provided there are degrees of freedom. It may be that the alternative sets of eligible variables among the search regions will be different although some variables will be common to all sets. Since the correct set of equations needed to find the solution is not known, only those eligible variables which are common are safely regarded as being eligible to form a consistent set of choices for a decision variable. By selecting one, the number of decision variables left to find is reduced by one and the process is repeated using the remaining variables. Search region consistency is achieved only if a sequence is found which allows an eligible variable to be selected for each degree of freedom.

A function *consistent* is presented in this work to perform the task of finding one set of consistent decision variables. As described below, it takes, as a single argument, a column p. The equations are all assumed to have been tested for structural consistency so what will be done many times in this function is assigning equations to variables without checking whether or not there was success in doing so. The index p is expected to take on values upon input in the range $\{m \dots n\}$. If $p = m$, then no checking need be done as far as finding a consistent set of independent variables because there are no degrees of freedom. If however, $p > m$, lines 3 through 5 are responsible for initially adding all variable indices to the set of eligibles E. Then, each search region is visited, the incidence pattern is altered, new output assignments are made, and new sets of eligible variables e are computed. The set E is adjusted during each search region visit so that it will end up as the intersection of all sets e. Finally, in line 20, a loop over all eligible variables is commenced in search for one that can be removed from the problem while preserving consistency. To test each choice, the variable is moved to column p, p is then decremented, and the function is called recursively. If the loop terminates before finding a working set of independent variables, then search region inconsistency is concluded.

Algorithm 8 *Function to find a consistent set of independent variables*

1 **boolean function** $consistent(p)$
2   **if** $(p = m)$ **return** TRUE
3   **set** $E = \emptyset$
4   **for each** $j \in \{1 \dots p\}$
5   **set** $E = E \vee \{$**index of variable in column** $j\}$
6   **for each** $s \in \mathcal{R}$
7   **for each** $i \in \{1 \dots m\}$
8   **for each** $j \in \{1 \dots p\}$
9   **if** $(\partial r_{si}(\mathbf{x})/\partial x_j = 0,\ \forall_{\mathbf{x} \in R^n})$
10   **delete** I_{ij}
11   **else**
12   **insert** I_{ij}
13   **set** $V = \emptyset$
14   **set** $range = \{i \dots p\}$
15             $assign(i, range, V)$
16   **set** $e = \emptyset$
17   **for each** $j \in \{(m+1) \dots p\}$
18             $eligible(j, e)$
19   **set** $E = E \wedge e$

```
    for each j ∈ E
        exchange x_j with variable in column p
        set p = p − 1
        if (consistent(p))
            return TRUE
        set p = p + 1
    return FALSE
```

3.3.2 Search region partitioning

If all of the equations were dependent on all of the logical conditions, partitioning would not be possible. This is because without knowing the correct set of equations which describe the solution, decoupling cannot be performed. However, many of the equations will be unconditional or at least dependent on only some of the conditions. This is the reason why many of the same equations appear in more than one search region. Rather than attempt to incorrectly decouple the wrong set of equations, a representative incidence matrix is needed.

Consider defining l binary functions $\mathbf{y}(\mathbf{x})$ such that $y_i(\mathbf{x}) = 1$ if $b_i(\mathbf{x}) \geq 0$ and $y_i(\mathbf{x}) = 0$ otherwise. Then the feasible region of the conditional model can be written using all of the alternative sets of equations.

$$\sum_{s \in \mathcal{R}} \mathbf{r}_s(\mathbf{x}) \prod_{i \in s} y_i(\mathbf{x}) \prod_{i \in \{1 \ldots l\} - s} (1 - y_i(\mathbf{x})) = \mathbf{0} \tag{3.1}$$

This suggests that the representative incidence of the m alternative equations should at least involve the union over all incidence patterns of the equations in each search region. In order to output values for its assigned variable, each conditional equation requires other equations to compute values for its other incident variables as well as for the variables incident on the boundary expressions of the conditions on which it is dependent. This further suggests that additional incidences be placed accordingly to ensure that no variable included in the boundary expression of a condition be partitioned after a block containing an equation which depends on the condition.

Algorithm 9 *Creating a representative incidence pattern*

1 **for each** $i \in \{1 \dots m\}$
2   **for each** $j \in \{1 \dots n\}$
3   **delete** I_{ij}
4   **for each** $s \in \mathcal{R}$
5   **for each** $j \in \{1 \dots n\}$
6   **if NOT** $(\partial r_{si}(\mathbf{x})/\partial x_j = 0,\ \forall_{\mathbf{x} \in R^n})$
7   **insert** I_{ij}
8   **for each** $k \in \{1 \dots l\}$
9   **if equation** $r_{si}(\mathbf{x}) = 0$ **depends on condition** k
10   **for each** $j \in \{1 \dots n\}$
11   **if NOT** $(\partial b_k(\mathbf{x})/\partial x_j = 0,\ \forall_{\mathbf{x} \in R^n})$
12   **insert** I_{ij}
13   **set** $V = \emptyset$
14   **set** $range = \{i \dots m\}$
15     $assign(i, range, V)$
16 **for each** $i \in \{(m+1) \dots n\}$
17   **for each** $j \in \{1 \dots m\}$
18   **if** $(\partial f(\mathbf{x})/\partial x_j = 0,\ \forall_{\mathbf{x} \in R^n})$
19   **delete** I_{ij}
20   **else**
21   **insert** I_{ij}
22   **for each** $j \in \{(m+1) \dots n\}$
23   **insert** I_{ij}

A loop over all rows is commenced in line 1. Immediately, the row is cleared of any incidence. It will be accumulating incidence from various equations as different search regions are visited. It may also be filled with incidences of some boundary expressions. When filling incidences in the row is completed, it is then output assigned by executing the statements in lines 13 through 15. It is again assumed that output assigning will proceed without difficulty because tests were previously done to ensure it. It is also worth pointing out that more incidences exist in this representative pattern than does exist in any one search region pattern. In fact, the output assignments of all search regions have been merged so legal output assignment of the representative pattern is guaranteed. It is also necessary because of the assumptions of the decoupling algorithm in treating variables as inputs and outputs for the equations. Finally, the representative pattern achieved can be augmented with $n - m$ rows having incidence of $\mathbf{z}$ and incidence of the objective function as done earlier so that the decoupling algorithm will look at conditional optimization problems as well. This is done by executing the loop of line 16. To the resulting incidence pattern, the algorithm presented above which attempts to decouple the equations is applied.

3.4 Examples

Structural analysis is applied to each of the examples introduced earlier. Since the phase equilibria example has no degrees of freedom, search region consistency is easily verified by confirming structural consistency among each alternative set of equations. The heat exchange problem has two degrees of freedom. A working pair is needed for the decision variables. The adiabatic compressible flow example has one degree of freedom. Therefore, what will be demonstrated is how the intersection of the sets of eligible variables computed from the equations describing the two flow conditions will yield proper choices for the decision variable in optimizing the conditional model.

3.4.1 Phase equilibria

No matter which phase is absent, the equation sets are all structurally nonsingular. Figure 3.5 indicates an output assignment for the variables if the vapor phase is absent at the solution. Figures 3.6 and 3.7 give slightly different orderings for the variables when the organic and aqueous phase vanish respectively. With no degrees of freedom, it is easily concluded that the model is search region consistent. Phase equilibria models are often highly coupled. What is interesting here is how decoupling can exist in each individual search region (consider the singleton $\Phi^V = 0$ of the first search region) but when the partitioning patterns are merged, the decoupling goes

	y^A_B	y^O_B	Φ^V	y^A_E	y^V_E	y^V_W	y^V_B	y^O_E	y^O_W	Φ^A	Φ^O	y^A_W
1	■			■								■
2		■						■	■			
3			■									
4	■			■			■					■
5	■			■	■							■
6	■			■		■						■
7		■					■	■	■			
8		■			■			■	■			
9		■				■		■	■			
10	■	■	■				■			■	■	
11			■	■	■			■		■	■	
12			■			■			■	■	■	■

Figure 3.5: Output assignment with vapor phase absent

away as indicated in Figure 3.8.

3.4.2 Heat exchange

This example becomes more interesting than the first because it introduces two degrees of freedom. Assuming the equation sets are all structurally consistent, search region consistency must be investigated. Output assignments are successfully conducted, confirming structural consistency, for each search region as shown in Figures 3.9, 3.10, and 3.11 where the variable names and equation numbers have been ommitted for clarity. The sets of indices of eligible variables computed from each output assignment are intersected to form the set of allowable decision variables

$$\left\{ \begin{array}{l} x^0_B, x^0_P, x^0_H, y^0_B, y^0_P, y^0_H, \\ T^1_c, T^2_c, T^3_c, T^0_h, H^1_c, H^2_c, H^3_c, H^0_h, \\ \frac{dQ}{d\eta}^0, \frac{dQ}{d\eta}^1, \frac{dQ}{d\eta}^2, \frac{dQ}{d\eta}^3, F_c, A \end{array} \right\} \tag{3.2}$$

Since it is not empty, search region inconsistency cannot be concluded. The model appears to be consistent. Intuitive choices for the decision variables are the total heat

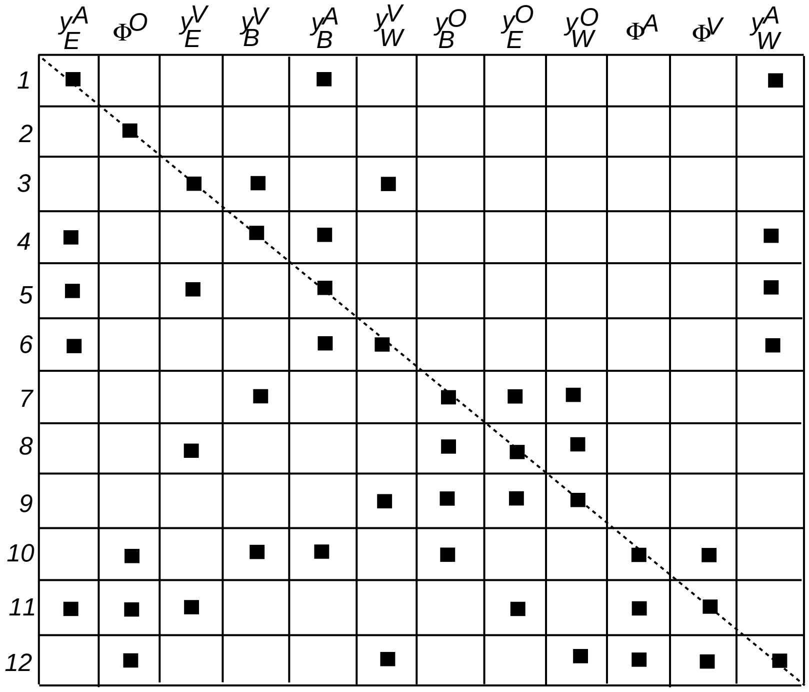

Figure 3.6: Output assignment with organic phase absent

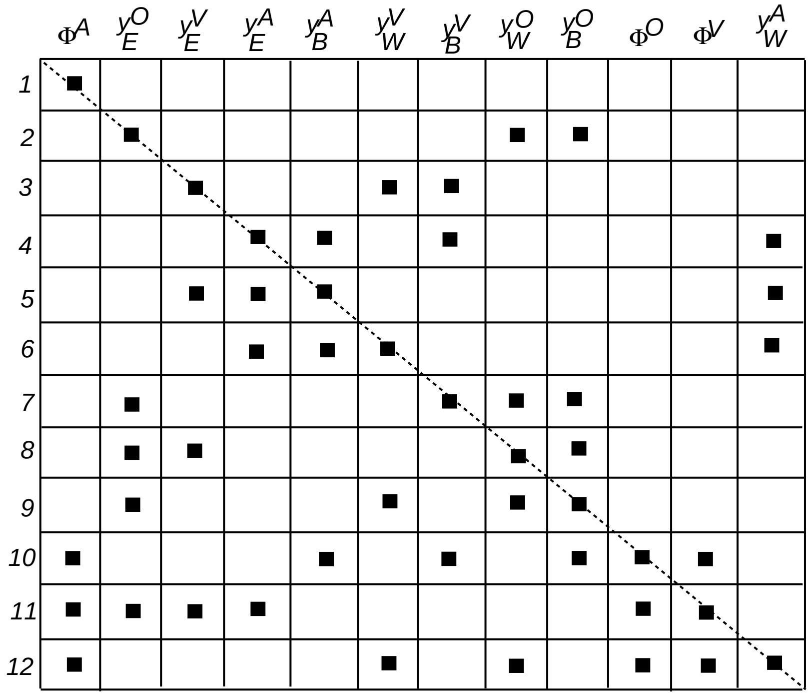

Figure 3.7: Output assignment with aqueous phase absent

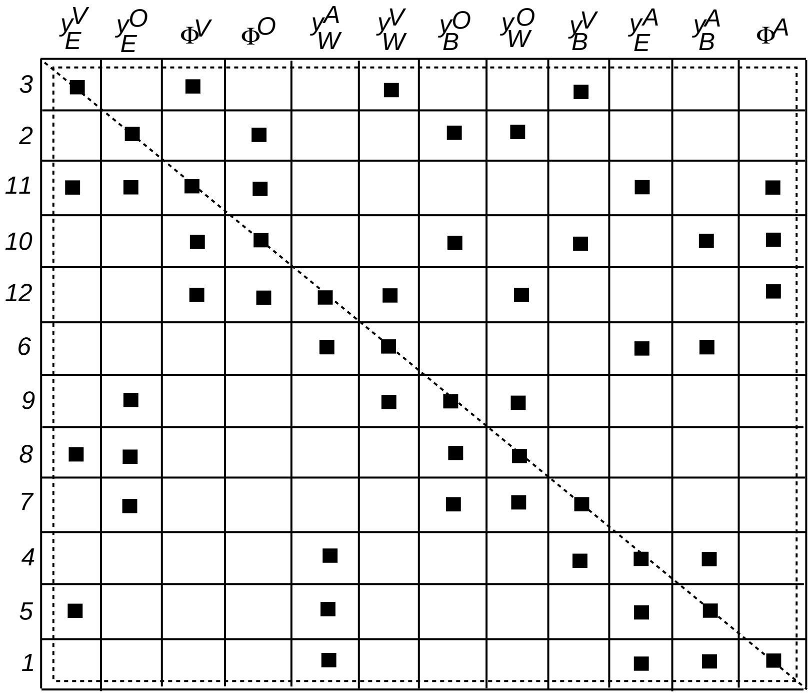

Figure 3.8: Representative incidence and coupling

transfer area A and the cooling water flow rate F_c. They are noticed to be members of the set above. However, this does not suggest that they as a pair can be used as decision variables. What is dictated by an eligibility analysis, when conducted as many times as there are degrees of freedom, is that choosing one of the set of eligible variables as a decision variable will not create structural singularities. Choosing more than one at a time is a risk. In this example, however, it turns out that if A were selected, the set of eligible variables for the remaining degree of freedom includes F_c.

Using these intuitive choices for the decision variables, a representative incidence is created and reordered in search of any possible partitioning among the equations. As a result, Figure 3.12 shows there being 8 blocks. The first 7 are considered trivial because they involve at most only 2 equations in 2 unknowns. In fact, the solution of each singleton block involves no iteration. The first block, for example, involves computing H_c^0 from the specified parameter T_c^0 using the given linear enthalpy expression for the cooling water. The majority of the model, however, remains coupled

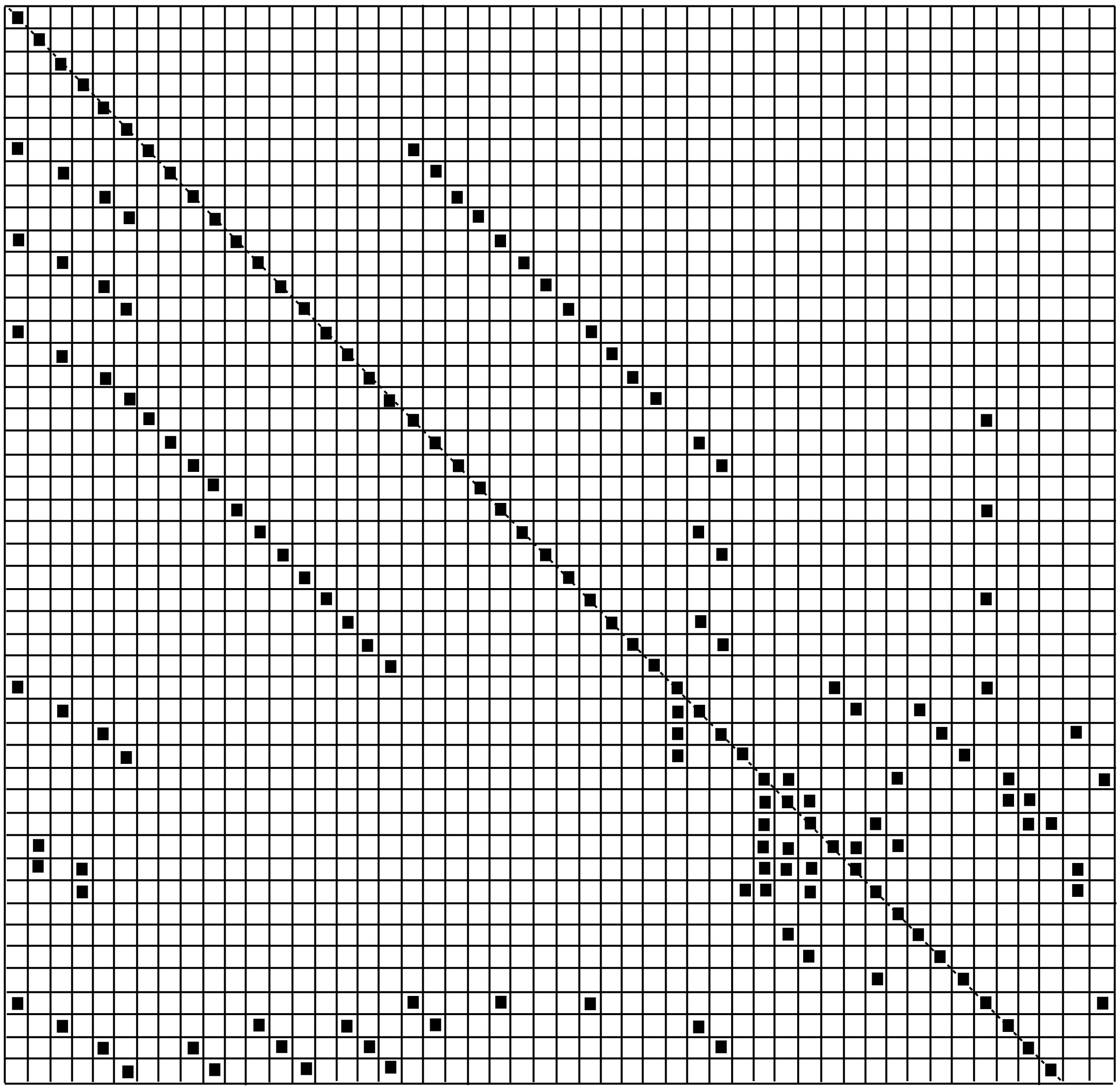

Figure 3.9: Output assignment with no condensation

and must be solved iteratively. The rows marked 49* and 50* are filled with the required incidence for binding.

3.4.3 Adiabatic compressible flow

With one degree of freedom, this model can be concluded as to its search region consistency simply by finding a variable which is common to the set of eligibles for each search region. For the case of plug flow, the output assignment is achieved and

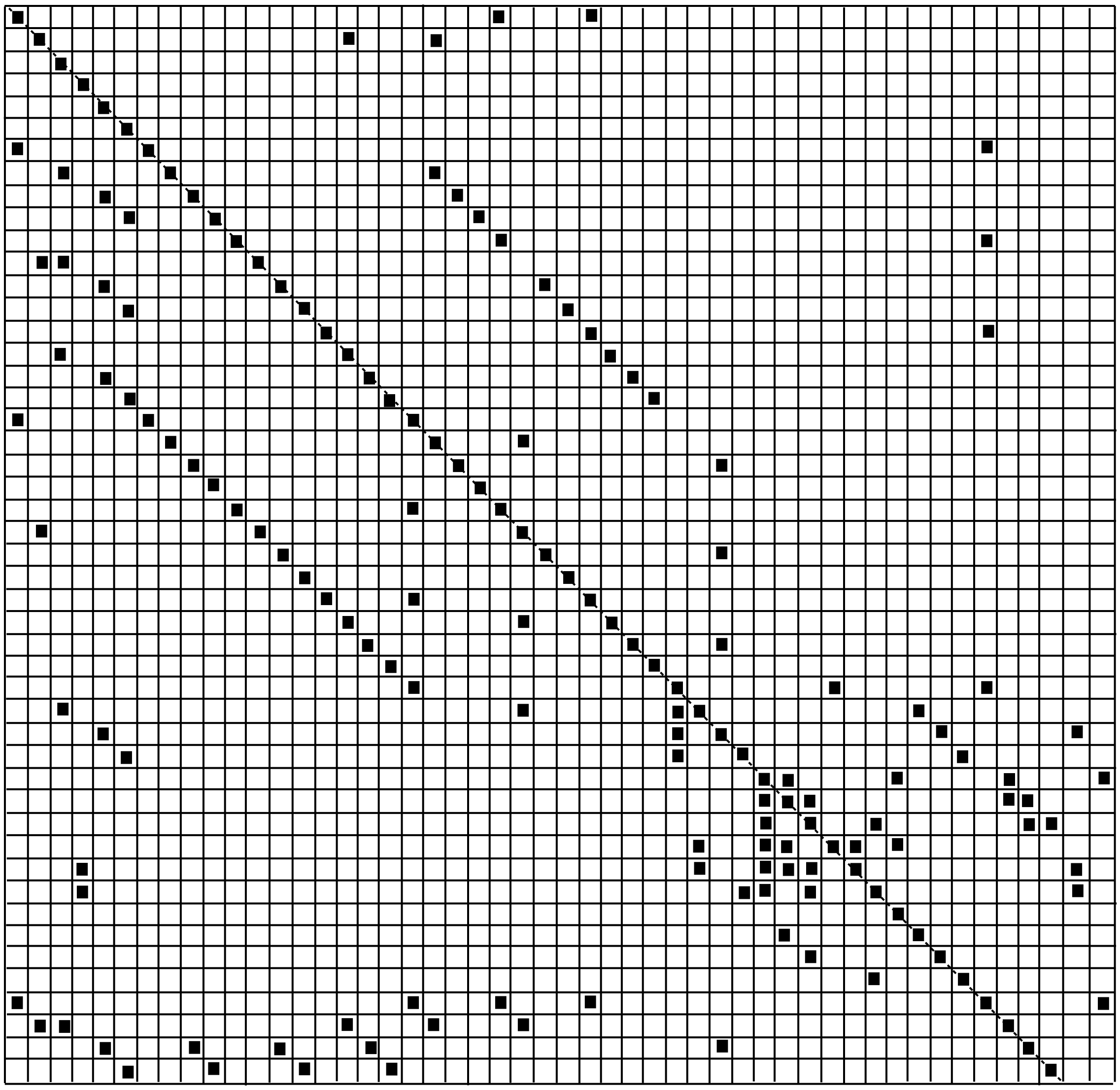

Figure 3.10: Output assignment with left side condensation

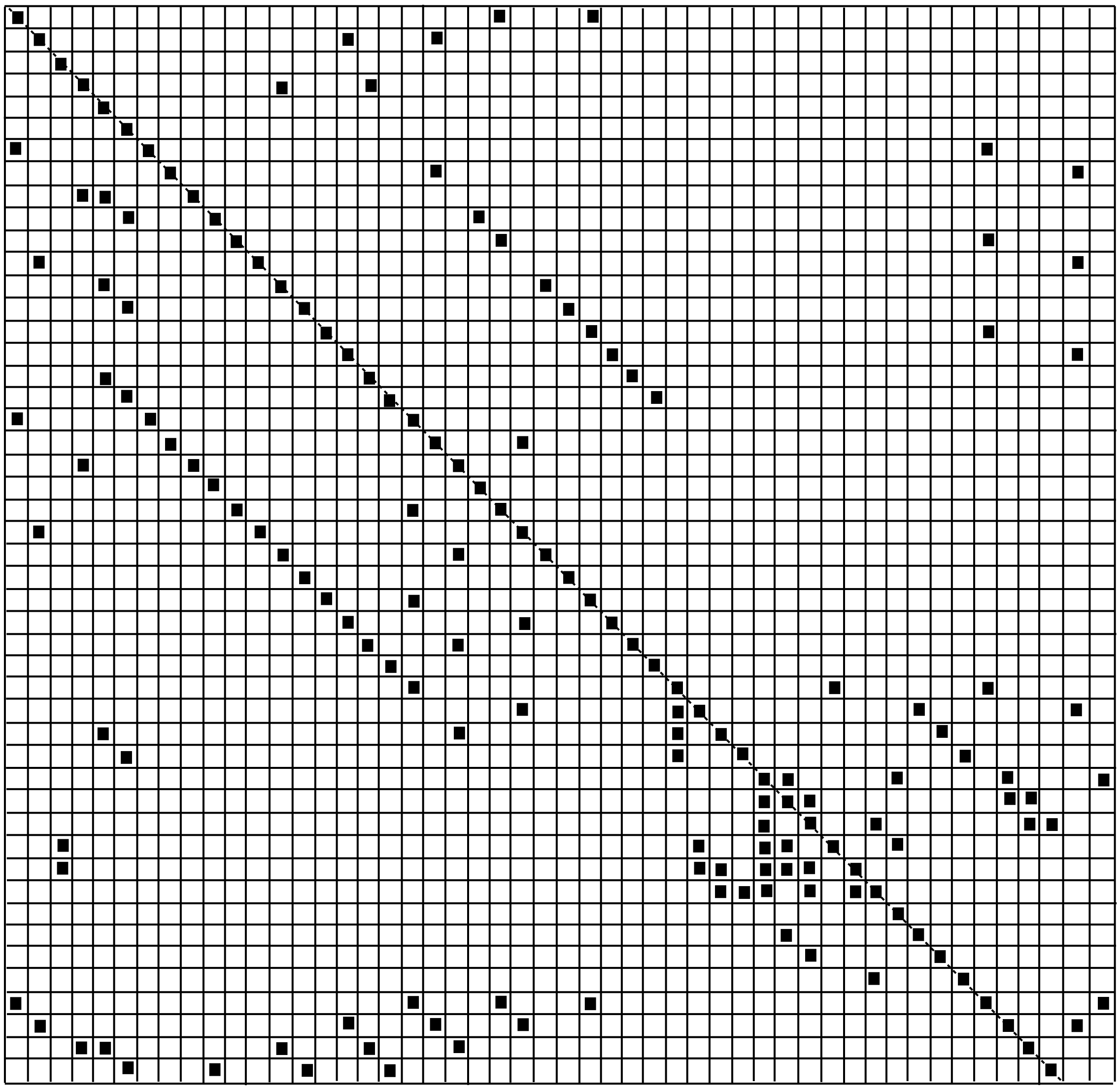

Figure 3.11: Output assignment with left and right side condensation

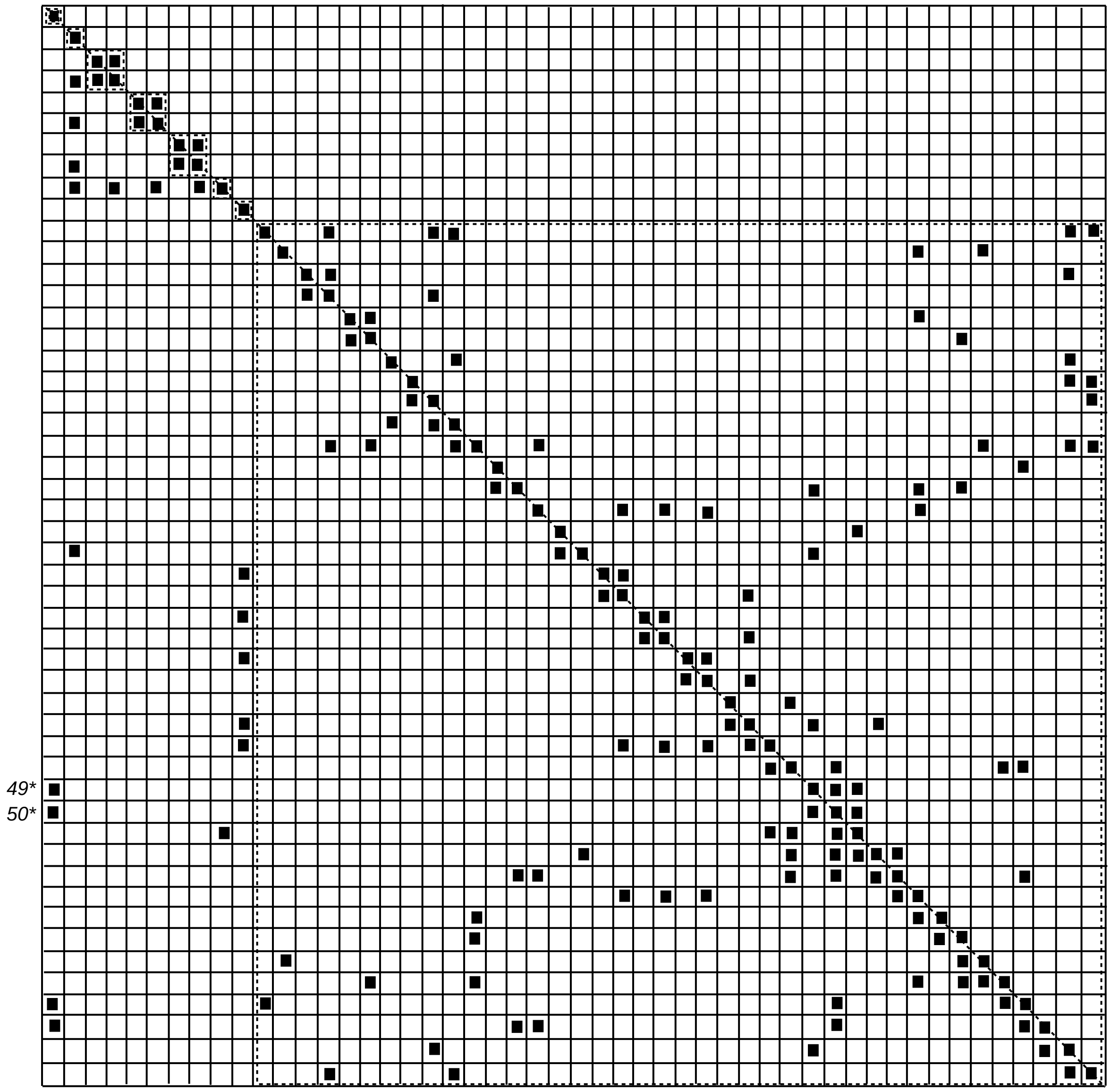

Figure 3.12: Representative incidence and coupling

is given in Figure 3.13a. The eligible variables in this search region were found to consist of F, M_i, M_f, T_f, and D. On the other hand, a different output assignment is obtained for sonic flow conditions as is shown in Figure 3.13b. Here, the eligible variables are found to be P_f, F, M_i, D, and T_f. Taking the intersection of these two sets,

$$\{F, M_i, M_f, T_f, D\} \wedge \{P_f, F, M_i, D, T_f\} \tag{3.3}$$

it is concluded that the model is search region consistent and that without knowing in which search region the solution will lie, it is safe to limit the choice of decision variables among F, M_i, T_f, and D. Choosing D as the decision variable, the representative incidence shown in Figure 3.13c is created and reordered as per the equation partitioning algorithm. The analysis dictates that the model cannot be decoupled.

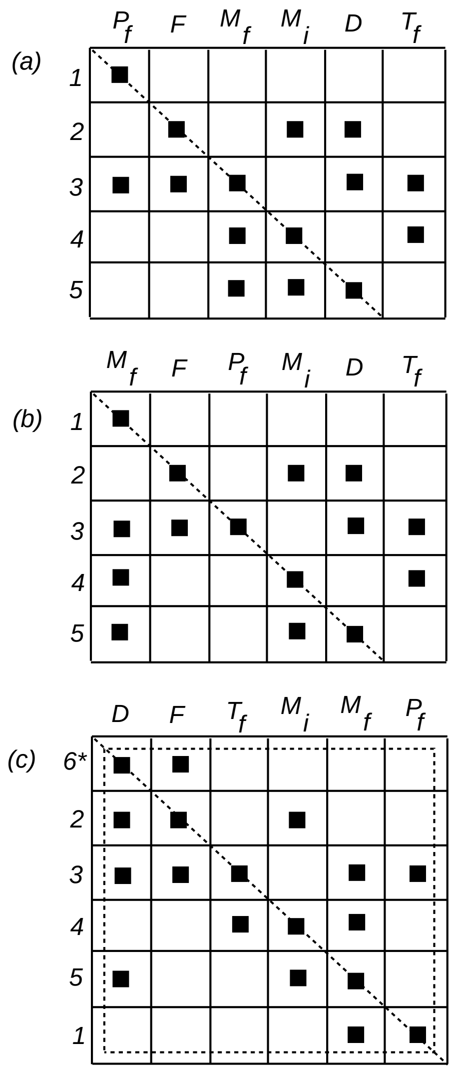

Figure 3.13: Analysis of the adiabatic compressible flow model

Chapter 4

Solution

4.1 Introduction

In this chapter, algorithms are developed to solve problems of the formulation types presented earlier after they have been structurally analyzed. Optimization theory has been very well defined for continuous and differentiable models as well as those involving inequality constraints [4] [16] [17] [37] [44] [49]. The theory will be briefly reviewed in order to provide an understanding of the material as the applicability is extended to conditional models.

4.2 Continuous and differentiable models

For unconstrained nonlinear programs, the sequential quadratic programming algorithm involves selecting initial values for the variables and iteratively minimizing a quadratic approximation to the objective function until a point which sufficiently minimizes the unapproximated objective is achieved. The difficulty with solving problems of the form (2.1) is ensuring that the equations are satisfied at the solution as well. Another common technique, known as the generalized reduced gradient method, enforces feasibility at the beginning of every iteration [53]. This is thought to involve more computation in arriving at a solution than should be necessary which is one of the reasons why the infeasible path sequential quadratic method was preferably used in this work.

4.2.1 Local Minima

To any optimization problem, there may be multiple local minima within the feasible region. In such cases, finding the global minimum is a very difficult problem and is not in the scope of this work. Much theory exists, however, in classifying local minima and can readily be used in developing a solution algorithm. When a local minimum is found, there should be termination of the algorithm. This implies that the obtained solution is the only or at least the global minimum. At the local minimum, there exists multipliers, $\boldsymbol{\lambda}$, for the equations such that

$$\nabla_{\mathbf{x}} f(\mathbf{x}) + \nabla_{\mathbf{x}} \mathbf{g}(\mathbf{x}) \boldsymbol{\lambda} = \mathbf{0} \tag{4.1}$$

To confirm that the solution is indeed a minimum and not a maximum or saddle point, the curvature of the objective function must be positive definite along all directions tangent to the feasible region at the solution point. The multipliers can be divided into positive and negative groups. The Kuhn-Tucker theorem states that if the objective function is pseudoconvex and furthermore if the residual functions are quasiconvex for all equations whose multiplier is positive and quasiconcave for all equations whose multiplier is negative, then the local minimum is indeed global [4]. The reverse need not be true, however, which implies that the Kuhn-Tucker theorem for globality is not a necessary condition [37]. This is best demonstrated with the following simple problems. In Figure 4.1a, the contours along with the feasible region are shown for the problem

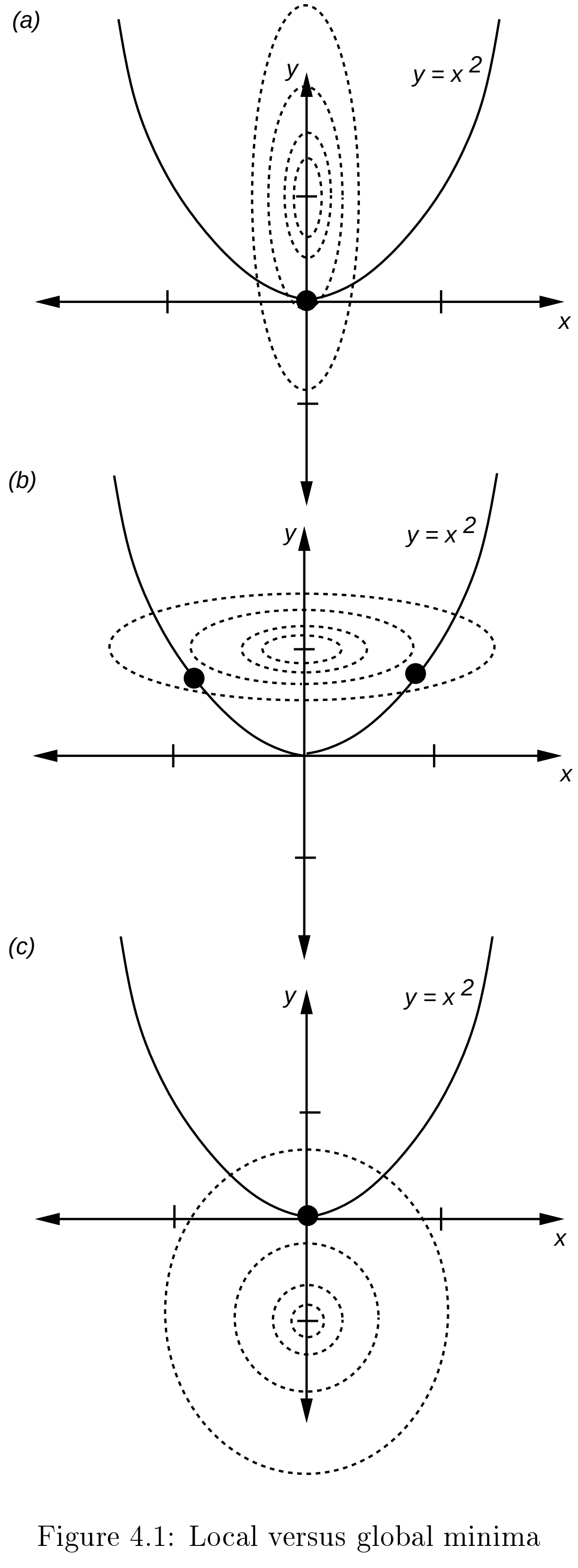

Figure 4.1: Local versus global minima

$$\begin{aligned} \min \quad & 2x^2 + (y-1)^2 \\ s.t. \quad & y = x^2 \end{aligned} \tag{4.2}$$

The solution is at $x = 0$ and $y = 0$. The multiplier for the single constraint is $\lambda = 2$. The objective is convex while the residual $y - x^2$ is quasiconcave. Although the multiplier is positive, it is still observed that the solution is in fact a global minimum. In Figure 4.1b, the objective is modified.

$$\begin{aligned} \min \quad & x^2 + 2(y-1)^2 \\ s.t. \quad & y = x^2 \end{aligned} \tag{4.3}$$

A solution exists near $x = 0.866$ and $y = 0.750$. At this point, the multiplier is $\lambda = 1$ which is still positive and therefore no conclusion can be drawn. In fact, the plot suggests there is another local minimum at $x = -0.866$ and $y = 0.750$. Finally, Figure 4.1c is constructed.

$$\begin{aligned} \min \quad & x^2 + (y+1)^2 \\ s.t. \quad & y = x^2 \end{aligned} \tag{4.4}$$

The solution here is at $x = 0$ and $y = 0$ with a multiplier of $\lambda = -2$. Based on the theorem and as suggested by the plot, this is in fact a global minimum. In conclusion, more often than not, there does exist a single minimum point which is sought.

4.2.2 Augmented Lagrangian

As was mentioned above, a difficulty with the sequential quadratic programming algorithm for solving equality constrained nonlinear programs is ensuring both feasibility and optimality in the solution. One method which has received much attention involves constructing an exact penalty function whose unconstrained minimum will satisfy the feasibility and optimality conditions of the original problem. The Lagrangian function defined by

$$L(\mathbf{x}, \boldsymbol{\lambda}) = f(\mathbf{x}) + \mathbf{g}(\mathbf{x})^T \boldsymbol{\lambda} \tag{4.5}$$

has the property that its stationary condition satisfies (4.1) and that its Hessian matrix, represented by $\nabla^2_{\mathbf{xx}} L(\mathbf{x}, \boldsymbol{\lambda})$ (which in other texts, may also be used to signify the Laplacian), is positive definite at least only along the subspace tangent to the feasible region at a local minimum point. This is the sufficient curvature requirement for optimality. But since successive quadratic programming employs an infeasible path searching algorithm, positive definiteness is required in all directions so that the algorithm can arrive at an unconstrained minimum from anywhere, not just from paths confined to the feasible region. A function that could augment the Lagrangian should have zero curvature along directions in the tangent subspace and positive curvature elsewhere. A function commonly used for augmentation is $\mathbf{g}(\mathbf{x})^T \mathbf{g}(\mathbf{x})/2$. Its Hessian along the feasible region, $\nabla_{\mathbf{x}} \mathbf{g}(\mathbf{x}) \nabla_{\mathbf{x}} \mathbf{g}(\mathbf{x})^T$, has strictly positive curvature in all nonzero directions $\mathbf{d}$ not satisfying $\nabla_{\mathbf{x}} \mathbf{g}(\mathbf{x})^T \mathbf{d} = \mathbf{0}$ and zero curvature otherwise.

$$\mathbf{d}^T \nabla_{\mathbf{x}} \mathbf{g}(\mathbf{x}) \nabla_{\mathbf{x}} \mathbf{g}(\mathbf{x})^T \mathbf{d} \geq \mathbf{0} \tag{4.6}$$

Such a function, multiplied by an appropriately large parameter ρ, can be added to the Lagrangian and the augmented Lagrangian

$$\Phi(\mathbf{x}, \boldsymbol{\lambda}) = f(\mathbf{x}) + \mathbf{g}(\mathbf{x})^T \boldsymbol{\lambda} + \frac{\rho}{2} \mathbf{g}(\mathbf{x})^T \mathbf{g}(\mathbf{x}) \tag{4.7}$$

will have a minimum at the desired solution when the appropriate values for the multipliers are known and a sufficiently large penalty parameter ρ is used in order to make the Hessian, $\nabla^2_{\mathbf{xx}}\Phi(\mathbf{x}, \boldsymbol{\lambda})$, arbitrarily positive definite. This differentiable function has global descent properties when the local minimum of the original problem is unique and can be used to generate search directions at each iteration as well as enforce descent during line search.

Conventionally, the sequential quadratic programming algorithm involves the solution of a quadratic program at each iteration to generate a search direction. The quadratic objective is typically based on the Hessian of the Lagrangian [2] [7] [43] [42]. For most industry problems, too many calculations are required to compute the matrix exactly. For these cases, the family of quasi-Newton methods were developed which accumulate, at each iteration, a positive definite approximation of the Hessian of the Lagrangian at the solution [13]. This reduces the convergence rate which is normally quadratic to what is termed superlinear. Along the search direction obtained, line search is employed which is the process of adjusting the distance traveled in order to ensure adequate decrease in an exact penalty function like the one introduced by Powell [43]. Others have demonstrated that Powell's nondifferentiable exact penalty function can be replaced by the augmented Lagrangian function above [28] [50].

One of the discrepancies neglected is the fact that the Hessian of the Lagrangian is not expected to be necessarily positive definite at the solution, as explained above. In order to circumvent the problem of applying a quasi-Newton positive definite approximation algorithm to a possibly indefinite matrix, the Hessian is deliberately modified. Powell introduces a technique which modifies the curvature information only at points where the positive definiteness of the Hessian is jeopardized [43] [42]. This is widely used throughout the literature. Recently, there have been attempts at computing the Hessian matrix analytically [2] [7]. In this case, Gershgorin eigen value analysis is applied to modify the diagonal elements of the matrix until positive definiteness is achieved.

Another discrepancy which also still exists is the fact that the function which is ultimately approximated quadratically in order to generate a search direction and the function used during line search are not the same. In this work, it is felt that the augmented Lagrangian function would be more appropriate for approximating quadratically in a neighborhood around the solution and also serve as a merit function for line search. One author sharing the same feeling is Tapia [54]. The Lagrangian by itself is known to produce a saddle at the solution for most problems. If this is so, positive curvature can always be achieved by adjusting the penalty parameter ρ rather than by distorting the space via *ad hoc* Hessian alteration. One of the advantages of using the augmented Lagrangian during line search over the nondifferentiable exact penalty function is that the Maratos effect is avoided [17]. The Maratos effect is a phenomenon observed when line search causes travel distances to be exceedingly small due to emphasis of feasibility over optimality and as a result superlinear convergence is impeded.

The convergence rate can be impeded using the augmented Lagrangian function as well because solution values for the multipliers are not known. At each iteration, estimates are made for the multipliers in a way that they converge to the solution values along with the variables. The line search function during each iteration is therefore dependent on how accurate the current estimates of the multipliers are. The generalization of methods which update the multipliers in an alternating fashion with the variables are called multiplier methods [22] [37] [49]. The common multiplier method updates the multipliers using a linear update formula even though the variables are updated superlinearly. As a result, the superlinear convergence rate is reduced. It is envisioned, then, that by combining the augmented Lagrangian of the multiplier method with the superlinear convergence characteristics of the sequential quadratic programming algorithm, a more efficient algorithm can be defined. In summary, one of the contributions of this work is to develop a quasi-Newton sequential quadratic programming algorithm based on the minimization of the augmented Lagrangian.

4.2.3 Limited memory quasi-Newton methods

Quasi-Newton methods provide a means of estimating the Hessian of the augmented Lagrangian at the solution. The popular BFGS method for updating the estimate involves secant information formed from evaluations of the gradient, $\nabla_{\mathbf{x}}\Phi(\mathbf{x}, \boldsymbol{\lambda})$, at two points which differ in the values for the variables $\mathbf{x}$ but not for the multipliers $\boldsymbol{\lambda}$. The algorithm operates on a dense matrix of n^2 elements. For problems involving many variables, this becomes too memory intensive. Modified optimization algorithms have been developed which attempt to store only the curvature in the decision variables tangent to the feasible region which amounts to a reduced Hessian of order equal to the degrees of freedom [27]. This is only accurate with a feasible path algorithm. The trouble with applying this update algorithm to sequential quadratic programming is that much of the important curvature information needed for superlinear convergence is lost.

The BFGS method allows an estimate from one iteration to be updated to the next through the use of two vectors, $\mathbf{u}$ and $\mathbf{v}$, and two scalars, $\beta_{\mathbf{u}}$ and $\beta_{\mathbf{v}}$. Letting $\mathbf{H}$ denote the most recent estimate of the Hessian and the vector $\mathbf{d}$ denote the change in the variable values for the next iteration, the BFGS method involves computing $\mathbf{u} = \mathbf{Hd}$, $\mathbf{v} = \nabla_{\mathbf{x}}\Phi(\mathbf{x}+\mathbf{d}, \boldsymbol{\lambda}) - \nabla_{\mathbf{x}}\Phi(\mathbf{x}, \boldsymbol{\lambda})$, $\beta_{\mathbf{u}} = -\mathbf{u}^T\mathbf{d}$ and $\beta_{\mathbf{v}} = \mathbf{v}^T\mathbf{d}$ so that a better estimate of the Hessian is obtained by the formula

$$\mathbf{H} \Leftarrow \mathbf{H} + \frac{\mathbf{u}\mathbf{u}^T}{\beta_{\mathbf{u}}} + \frac{\mathbf{v}\mathbf{v}^T}{\beta_{\mathbf{v}}} \tag{4.8}$$

At the first iteration, the Hessian is typically initialized to the familiar positive definite identity matrix.

To avoid allocation of a full $n \times n$ matrix, the limited memory BFGS method is used in this work [6] [26]. It is based on the premise that, for large problems, the total number of iterations needed to converge will be far less than the number of variables being solved. In these instances, it is beneficial then to accumulate storage of the update vectors $\mathbf{u}$ and $\mathbf{v}$ and the scalars $\beta_{\mathbf{u}}$ and $\beta_{\mathbf{v}}$ at each iteration rather than the actual matrix. If needed, the Hessian can always be reproduced at some iteration by combining the history of update information computed during previous iterations. The amount of history is controlled using a set Q which accumulates all of the iteration indices whose recorded update information is to be used in generating a current estimate of the Hessian via the relationship

$$\mathbf{H} = \mathbf{I} + \sum_{i \in Q} \left[\frac{(\mathbf{u}^i)(\mathbf{u}^i)^T}{\beta_{\mathbf{u}}^i} + \frac{(\mathbf{v}^i)(\mathbf{v}^i)^T}{\beta_{\mathbf{v}}^i} \right] \tag{4.9}$$

But, in fact, the Hessian matrix by itself is never needed. Instead, as will be shown, intermediate vectors are computed during the solution algorithm as a product between the Hessian matrix and other vectors. Considering some vector $\mathbf{d}$, for example, the formula for computing the vector $\mathbf{Hd}$ is obtained by postmultiplying relation (4.9) by $\mathbf{d}$.

$$\mathbf{Hd} = \mathbf{d} + \sum_{i \in Q} \left[\frac{(\mathbf{u}^i)(\mathbf{u}^i)^T\mathbf{d}}{\beta_{\mathbf{u}}^i} + \frac{(\mathbf{v}^i)(\mathbf{v}^i)^T\mathbf{d}}{\beta_{\mathbf{v}}^i} \right] \tag{4.10}$$

An additional feature of the limited memory BFGS method is that a limit can be set on how much update information is to be stored in case the number of iterations becomes exceedingly large. In this case, an upper bound is placed on the cardinality of Q and when the bound is reached, new data replaces the oldest data stored.

4.2.4 Initialization

To start the solution algorithm, a vector of initial values $\mathbf{a}$ for the variables is supplied. So that the system can be appropriately scaled, a vector of nominal values $\mathbf{n}$ which are an order of magnitude estimation of the solution values for the variables is supplied as well. A tolerance ϵ must be provided which is used to detect convergence. The multipliers, $\boldsymbol{\lambda}$, are initialized to zero, the penalty parameter, ρ, is set to one, the iteration counter $k = 0$, and the set $Q = \emptyset$. The objective function and all equation residuals, $f(\mathbf{a})$ and $\mathbf{g}(\mathbf{a})$, along with their derivatives, $\nabla_{\mathbf{x}} f(\mathbf{a})$ and $\nabla_{\mathbf{x}}\mathbf{g}(\mathbf{a})$ are evaluated at the point corresponding to these variable values. In order to scale the system, two diagonal matrices, $\mathbf{D_x}$ and $\mathbf{D_g}$ for variable and equation scaling respectively, and a scalar D_f for the objective function are introduced. The variable scaling matrix, $\mathbf{D_x}$, is generated by simply having the nominal values, which were provided, make up the matrix diagonal.

$$\mathbf{D_x} = \sum_{j \in \{1...n\}} n_j \mathbf{e}_j \mathbf{e}_j^T \tag{4.11}$$

The vector $\mathbf{e}_j \in R^n$ has the property that all of its elements are zero except element j which is one. The scaling factors for the equations are next obtained in such a way that, when they are applied to the Jacobian, the elements will end up not being much less than one and numerical ill-conditioning may be avoided. This involves computing the norm of the gradients with respect to the scaled variables of each equation residual.

$$\mathbf{D_g} = \sum_{i \in \{1...m\}} \|\mathbf{D_x}^T \nabla_\mathbf{x} g_i(\mathbf{a})\| \mathbf{e}_i \mathbf{e}_i^T \tag{4.12}$$

In its usage here, the vector $\mathbf{e}_i \in R^m$ has the same properties as described above. Finally, the scaling factor for the objective function is also computed as the norm of its gradient with respect to the scaled variables.

$$D_f = \|\mathbf{D_x}^T \nabla_\mathbf{x} f(\mathbf{a})\| \tag{4.13}$$

These three scaling factors are applied throughout the algorithm to keep the computations well-conditioned. They are applied first to the information evaluated at the initial point.

$$f(\mathbf{a}) \Leftarrow D_f^{-1} f(\mathbf{a}) \tag{4.14}$$

$$\nabla_\mathbf{x} f(\mathbf{a}) \Leftarrow \mathbf{D_x}^T \nabla_\mathbf{x} f(\mathbf{a}) D_f^{-1} \tag{4.15}$$

$$\mathbf{g}(\mathbf{a}) \Leftarrow \mathbf{D_g}^{-1} \mathbf{g}(\mathbf{a}) \tag{4.16}$$

$$\nabla_\mathbf{x} \mathbf{g}(\mathbf{a}) \Leftarrow \mathbf{D_x}^T \nabla_\mathbf{x} \mathbf{g}(\mathbf{a}) \mathbf{D_g}^{-1} \tag{4.17}$$

With the above sufficient information, the algorithm is started.

4.2.5 Termination

A local minimum can be detected by testing for stationarity in the augmented Lagrangian. The gradient of the augmented Lagrangian is computed by

$$\nabla_{\mathbf{x}}\Phi(\mathbf{a},\boldsymbol{\lambda}) = \nabla_{\mathbf{x}} f(\mathbf{a}) + \nabla_{\mathbf{x}}\mathbf{g}(\mathbf{a})(\boldsymbol{\lambda} + \rho\mathbf{g}(\mathbf{a})) \tag{4.18}$$

If $\|\nabla_{\mathbf{x}}\Phi(\mathbf{a},\boldsymbol{\lambda})\| \leq \epsilon$, then the algorithm is terminated. It is noticed that a measure of infeasibility is incorporated in the gradient so it is hoped that stationarity implies feasibility. It is always safe, though, to claim convergence only after checking for feasibility first when the algorithm terminates. There may exist instances where the augmented Lagrangian becomes stationary at an infeasible point.

4.2.6 Search direction

If the norm of the gradient of the augmented Lagrangian is not small enough for termination, the gradient is linearized (using the current approximation to the Hessian) in both the variables and the multipliers and set to zero in order to generate superlinear updates. In sequential quadratic programming, this is equivalent to solving the following quadratic subproblem for $\boldsymbol{\eta}$, the superlinear update for the variables.

$$\begin{aligned} \min \quad & [\nabla_{\mathbf{x}} f(\mathbf{a}) + \rho\nabla_{\mathbf{x}}\mathbf{g}(\mathbf{a})\mathbf{g}(\mathbf{a})]^T\boldsymbol{\eta} + \frac{1}{2}\boldsymbol{\eta}^T\mathbf{H}\boldsymbol{\eta} \\ s.t. \quad & \mathbf{g}(\mathbf{a}) + \nabla_{\mathbf{x}}\mathbf{g}(\mathbf{a})^T\boldsymbol{\eta} = \mathbf{0} \end{aligned} \tag{4.19}$$

The multipliers corresponding to the solution of (4.19) provide a superlinear update for the multipliers $\boldsymbol{\lambda}$ of the original problem. The variable partitioning of $\mathbf{x}$ into $\mathbf{y}$ and $\mathbf{z}$, which is accomplished during a structural analysis, is used here for solving the quadratic subproblem. The equivalent system of linear equations resulting from zeroing the linearization of the gradient are shown in Figure 4.2a. When solved, it produces values for the vector $\boldsymbol{\eta}$ and a new estimate for the multipliers $\boldsymbol{\lambda}$.

Solving the linear system of Figure 4.2a numerically is not practical because the full dense Hessian matrix is not available. Instead, the linear system is first partially pivoted symbolically in an effort to reduce the system down to order equal to the degrees of freedom. Figure 4.2b shows the effect of one pivot and the result after another pivot is shown in Figure 4.2c. What results in the middle of the matrix after elimination is what is called the reduced Hessian and is of order $(n-m) \times (n-m)$. The right hand side of this reduced system is also shown. Thus, without any loss of curvature information from the original linear system, this reduced system (enclosed in parenthesis in Figure 4.2c) is solved for the components of $\boldsymbol{\eta}$ corresponding to the variables $\mathbf{z}$. Back calculation for the remaining quantities is then done.

Another common practice of reducing the system via decomposition is to introduce two vectors $\mathbf{p_y}$ of length m and $\mathbf{p_z}$ of length $n-m$. A matrix $\mathbf{Y} \in R^{n \times m}$ and a matrix $\mathbf{Z} \in R^{n \times (n-m)}$ are also defined so that $\boldsymbol{\eta} = \mathbf{Y}\mathbf{p_y} + \mathbf{Z}\mathbf{p_z}$ [17] [51] [56]. The only requirements on the matrices for the decomposition to be useful are that $\nabla_\mathbf{x}\mathbf{g}(\mathbf{a})^T\mathbf{Y}$ is nonsingular and $\nabla_\mathbf{x}\mathbf{g}(\mathbf{a})^T\mathbf{Z} = \mathbf{0}$. In other words, the columns of $\mathbf{Z}$ span the null space of the Jacobian. An additional optional restriction on the matrices are that they be orthogonal to each other or that $\mathbf{Y}^T\mathbf{Z} = \mathbf{0}$ [17] [56]. Sparsity issues with large models have led to a decline in popularity with orthogonal decomposition. The coordinate based decomposition, as it has been called, is gaining attention recently due to the significant calculation savings that are obtained [6] [27] [51]. With coordinate based decomposition, the matrices are defined as follows [51].

$$\mathbf{Z} = \begin{bmatrix} -\nabla_\mathbf{z}\mathbf{g}(\mathbf{a})\nabla_\mathbf{y}\mathbf{g}(\mathbf{a})^{-1} & \mathbf{I} \end{bmatrix}^T \tag{4.20}$$

$$\mathbf{Y} = \begin{bmatrix} \mathbf{I} & \mathbf{0} \end{bmatrix}^T \tag{4.21}$$

Because $\mathbf{Z}$ spans the null space or tangent space of the Jacobian, the reduced Hes-

(a)

$$\begin{bmatrix} \boldsymbol{H}_{yy} & \boldsymbol{H}_{yz} & \nabla_y \boldsymbol{g}(\boldsymbol{a}) \\ \boldsymbol{H}_{zy} & \boldsymbol{H}_{zz} & \nabla_z \boldsymbol{g}(\boldsymbol{a}) \\ \nabla_y \boldsymbol{g}(\boldsymbol{a})^T & \nabla_z \boldsymbol{g}(\boldsymbol{a})^T & \boldsymbol{0} \end{bmatrix} \begin{bmatrix} \eta_y \\ \eta_z \\ \lambda \end{bmatrix} = \begin{bmatrix} -\nabla_y f(\boldsymbol{a}) - \rho \nabla_y \boldsymbol{g}(\boldsymbol{a}) \boldsymbol{g}(\boldsymbol{a}) \\ -\nabla_z f(\boldsymbol{a}) - \rho \nabla_z \boldsymbol{g}(\boldsymbol{a}) \boldsymbol{g}(\boldsymbol{a}) \\ -\boldsymbol{g}(\boldsymbol{a}) \end{bmatrix}$$

(b)

$$\begin{bmatrix} \boldsymbol{0} & \boldsymbol{H}_{yz} - \boldsymbol{H}_{yy} \nabla_y \boldsymbol{g}(\boldsymbol{a})^{-T} \nabla_z \boldsymbol{g}(\boldsymbol{a})^T & \nabla_y \boldsymbol{g}(\boldsymbol{a}) \\ \boldsymbol{0} & \boldsymbol{H}_{zz} - \boldsymbol{H}_{zy} \nabla_y \boldsymbol{g}(\boldsymbol{a})^{-T} \nabla_z \boldsymbol{g}(\boldsymbol{a})^T & \nabla_z \boldsymbol{g}(\boldsymbol{a}) \\ \nabla_y \boldsymbol{g}(\boldsymbol{a})^T & \nabla_z \boldsymbol{g}(\boldsymbol{a})^T & \boldsymbol{0} \end{bmatrix} \begin{bmatrix} \eta_y \\ \eta_z \\ \lambda \end{bmatrix} = \begin{bmatrix} -\nabla_y f(\boldsymbol{a}) - \rho \nabla_y \boldsymbol{g}(\boldsymbol{a}) \boldsymbol{g}(\boldsymbol{a}) + \boldsymbol{H}_{yy} \nabla_y \boldsymbol{g}(\boldsymbol{a})^{-T} \boldsymbol{g}(\boldsymbol{a}) \\ -\nabla_z f(\boldsymbol{a}) - \rho \nabla_z \boldsymbol{g}(\boldsymbol{a}) \boldsymbol{g}(\boldsymbol{a}) + \boldsymbol{H}_{zy} \nabla_y \boldsymbol{g}(\boldsymbol{a})^{-T} \boldsymbol{g}(\boldsymbol{a}) \\ -\boldsymbol{g}(\boldsymbol{a}) \end{bmatrix}$$

(c)

$$\begin{bmatrix} \boldsymbol{0} & \boldsymbol{H}_{yz} - \boldsymbol{H}_{yy} \nabla_y \boldsymbol{g}(\boldsymbol{a})^{-T} \nabla_z \boldsymbol{g}(\boldsymbol{a})^T & \nabla_y \boldsymbol{g}(\boldsymbol{a}) \\ \boldsymbol{0} & \begin{pmatrix} \boldsymbol{H}_{zz} - \boldsymbol{H}_{zy} \nabla_y \boldsymbol{g}(\boldsymbol{a})^{-T} \nabla_z \boldsymbol{g}(\boldsymbol{a})^T \\ - \nabla_z \boldsymbol{g}(\boldsymbol{a}) \nabla_y \boldsymbol{g}(\boldsymbol{a})^{-1} \boldsymbol{H}_{yz} + \\ \nabla_z \boldsymbol{g}(\boldsymbol{a}) \nabla_y \boldsymbol{g}(\boldsymbol{a})^{-1} \boldsymbol{H}_{yy} \nabla_y \boldsymbol{g}(\boldsymbol{a})^{-T} \nabla_z \boldsymbol{g}(\boldsymbol{a})^T \end{pmatrix} & \boldsymbol{0} \\ \nabla_y \boldsymbol{g}(\boldsymbol{a})^T & \nabla_z \boldsymbol{g}(\boldsymbol{a})^T & \boldsymbol{0} \end{bmatrix} \begin{bmatrix} \eta_y \\ \eta_z \\ \lambda \end{bmatrix} = \begin{bmatrix} -\nabla_y f(\boldsymbol{a}) - \rho \nabla_y \boldsymbol{g}(\boldsymbol{a}) \boldsymbol{g}(\boldsymbol{a}) + \boldsymbol{H}_{yy} \nabla_y \boldsymbol{g}(\boldsymbol{a})^{-T} \boldsymbol{g}(\boldsymbol{a}) \\ \begin{pmatrix} -\nabla_z f(\boldsymbol{a}) - \nabla_z \boldsymbol{g}(\boldsymbol{a}) \nabla_y \boldsymbol{g}(\boldsymbol{a})^{-1} \boldsymbol{H}_{yy} \nabla_y \boldsymbol{g}(\boldsymbol{a})^{-T} \boldsymbol{g}(\boldsymbol{a}) \\ + \boldsymbol{H}_{zy} \nabla_y \boldsymbol{g}(\boldsymbol{a})^{-T} \boldsymbol{g}(\boldsymbol{a}) + \nabla_z \boldsymbol{g}(\boldsymbol{a}) \nabla_y \boldsymbol{g}(\boldsymbol{a})^{-1} \nabla_y f(\boldsymbol{a}) \end{pmatrix} \\ -\boldsymbol{g}(\boldsymbol{a}) \end{bmatrix}$$

Figure 4.2: Derivation of the reduced Hessian

sian is given by the expression $\mathbf{Z}^T\mathbf{HZ}$. When the definition for $\mathbf{Z}$ given above is substituted, it is significant to find that the reduced Hessian derived in Figure 4.2c is obtained. The linear system is now more concisely written in compact form using these new vectors and matrices.

$$\nabla_{\mathbf{x}}\mathbf{g}(\mathbf{a})^T\mathbf{Y}\mathbf{p_y} = -\mathbf{g}(\mathbf{a}) \tag{4.22}$$

$$\mathbf{Z}^T\mathbf{HZ} = \mathbf{Z}^T\mathbf{Z} + \sum_{i\in Q}\left[\frac{(\mathbf{Z}^T\mathbf{u}^i)(\mathbf{Z}^T\mathbf{u}^i)^T}{\beta_{\mathbf{u}}^i} + \frac{(\mathbf{Z}^T\mathbf{v}^i)(\mathbf{Z}^T\mathbf{v}^i)^T}{\beta_{\mathbf{v}}^i}\right] \tag{4.23}$$

$$\mathbf{HY}\mathbf{p_y} = \mathbf{Y}\mathbf{p_y} + \sum_{i\in Q}\left[\frac{(\mathbf{u}^i)(\mathbf{u}^i)^T\mathbf{Y}\mathbf{p_y}}{\beta_{\mathbf{u}}^i} + \frac{(\mathbf{v}^i)(\mathbf{v}^i)^T\mathbf{Y}\mathbf{p_y}}{\beta_{\mathbf{v}}^i}\right] \tag{4.24}$$

$$\mathbf{Z}^T\mathbf{HZ}\mathbf{p_z} = -\mathbf{Z}^T(\nabla_{\mathbf{x}}f(\mathbf{a}) + \mathbf{HY}\mathbf{p_y}) \tag{4.25}$$

$$\boldsymbol{\eta} = \mathbf{Y}\mathbf{p_y} + \mathbf{Z}\mathbf{p_z} \tag{4.26}$$

$$\mathbf{H}\boldsymbol{\eta} = \boldsymbol{\eta} + \sum_{i\in Q}\left[\frac{(\mathbf{u}^i)(\mathbf{u}^i)^T\boldsymbol{\eta}}{\beta_{\mathbf{u}}^i} + \frac{(\mathbf{v}^i)(\mathbf{v}^i)^T\boldsymbol{\eta}}{\beta_{\mathbf{v}}^i}\right] \tag{4.27}$$

$$\mathbf{Y}^T\nabla_{\mathbf{x}}\mathbf{g}(\mathbf{a})\boldsymbol{\lambda} = -\mathbf{Y}^T(\nabla_{\mathbf{x}}f(\mathbf{a}) + \rho\nabla_{\mathbf{x}}\mathbf{g}(\mathbf{a})\mathbf{g}(\mathbf{a}) + \mathbf{H}\boldsymbol{\eta}) \tag{4.28}$$

Before solving equations (4.22)-(4.28), the matrices $\mathbf{Y}$ and $\mathbf{Z}$ will need to be computed. The matrix $\mathbf{Y}$ is known and so are the last $n-m$ rows of $\mathbf{Z}$. To get the first m rows of $\mathbf{Z}$ (or in other words, $\mathbf{Y}^T\mathbf{Z}$), the matrix $\nabla_{\mathbf{x}}\mathbf{g}(\mathbf{a})^T\mathbf{Y}$ is first factorized into lower and upper triangular matrices $\mathbf{L}$ and $\mathbf{U}$ using sparse matrix techniques [62]. With the factors obtained, the following linear system is solved for each column of $\mathbf{Y}^T\mathbf{Z}$.

$$\nabla_{\mathbf{x}}\mathbf{g}(\mathbf{a})^T\mathbf{YY}^T\mathbf{Z} = -\nabla_{\mathbf{z}}\mathbf{g}(\mathbf{a})^T \tag{4.29}$$

With the factorization completed and the matrices $\mathbf{Y}$ and $\mathbf{Z}$ known, equation (4.22) is solved using the $\mathbf{L}$ and $\mathbf{U}$ factors to obtain $\mathbf{p_y}$. Next the reduced Hessian is derived from the current estimate of the Hessian using formula (4.23). The computation of (4.24) must be carried out to compute $\mathbf{HY}\mathbf{p_y}$ using all of the saved update vectors. This allows solution of the reduced space equations (4.25) to get values for $\mathbf{p_z}$. But the reduced Hessian is symmetric so a special technique for solving these $n-m$ equations involves the Cholesky factorization of the reduced Hessian. Essentially, the Cholesky factorization differs from the factorization referred to above in that it seeks a single lower triangular matrix which when multiplied by its transpose, gives the reduced Hessian. Enough information is now known to generate the variable update vector $\boldsymbol{\eta}$ using (4.26). In order to solve for the multipliers, the vector $\mathbf{H}\boldsymbol{\eta}$ is needed and is computed using the update vectors as shown in (4.27). Finally, by using the $\mathbf{L}$ and $\mathbf{U}$ factors again, the multipliers are now updated by solving (4.28).

Using these new values for the multipliers, the augmented Lagrangian is approximated quadratically in the variables to generate a search direction $\mathbf{d}$.

$$\min \quad \Phi(\mathbf{a}, \boldsymbol{\lambda}) + \nabla_{\mathbf{x}}\Phi(\mathbf{a}, \boldsymbol{\lambda})^T\mathbf{d} + \frac{1}{2}\mathbf{d}^T\mathbf{H}\mathbf{d} \tag{4.30}$$

As would be expected, the solution to the unconstrained minimization is given by $\boldsymbol{\eta}$. However, the approximation may be poor so that the augmented Lagrangian computed from function evaluations of the objective and equation residuals may increase at the new point when $\boldsymbol{\eta}$ is applied. This is where line search is traditionally employed. To enforce descent in the augmented Lagrangian, the length of $\boldsymbol{\eta}$ is adjusted during line search by successively shortening it until it is found that updating the variables over a fraction of the original distance of $\boldsymbol{\eta}$ leads to adequate decrease in the evaluation of the augmented Lagrangian.

Although distances are adjusted, line search restricts the search direction from changing. Levenberg and Marquadt were among the earliest to show that, at least with least squares problems, step reductions should be considered as constraints to impose on (4.30), the minimization of the quadratic approximation, thereby generating a new direction in the process [18] [25] [30] [35]. In fact, in the limit that step reductions get exceedingly small, the direction tends to that of steepest descent at the current point. A vector $\boldsymbol{\gamma}$ is introduced to contain the steepest descent direction. It is simply the negative of the gradient which has already been computed by recognizing the equivalence $\boldsymbol{\gamma} = \mathbf{H}\boldsymbol{\eta}$. Limiting the search directions to lie in the subspace spanned by the two vectors, $\boldsymbol{\eta}$ and $\boldsymbol{\gamma}$, is a common technique for reducing the work involved in resolving (4.30) subject to a reduced step length constraint, as has been seen with systems of nonlinear equations [41] [59]. Powell's hybrid method was later extended to optimization [21]. In this work, the Westerberg and Director method was chosen to be made applicable to optimization. Two quantities α_η and α_γ act as coefficients of the two normalized directions which will give the direction of the step $\mathbf{d}$. To control step length, the quantity M is adjusted. M can also be considered as what is called a trust region radius. The quadratic program to be solved for the step vector $\mathbf{d}$ under varying specifications for M is

$$\begin{aligned} \min \quad & -\boldsymbol{\gamma}^T\mathbf{d} + \frac{1}{2}\mathbf{d}^T\mathbf{H}\mathbf{d} \\ s.t. \quad & \mathbf{d} = M(\alpha_\eta \frac{\boldsymbol{\eta}}{\|\boldsymbol{\eta}\|} + \alpha_\gamma \frac{\boldsymbol{\gamma}}{\|\boldsymbol{\gamma}\|}) \\ & \mathbf{d}^T\mathbf{d} = M^2 \end{aligned} \tag{4.31}$$

The optimality conditions for (4.31) are derived and simplified with the introduction of an additional parameter θ.

$$\begin{bmatrix} \frac{\boldsymbol{\eta}^T\boldsymbol{\gamma}+\theta\boldsymbol{\eta}^T\boldsymbol{\eta}}{\boldsymbol{\eta}^T\boldsymbol{\eta}} & \frac{\boldsymbol{\gamma}^T\boldsymbol{\gamma}+\theta\boldsymbol{\eta}^T\boldsymbol{\gamma}}{\|\boldsymbol{\eta}\|\|\boldsymbol{\gamma}\|} \\ \frac{\boldsymbol{\gamma}^T\boldsymbol{\gamma}+\theta\boldsymbol{\eta}^T\boldsymbol{\gamma}}{\|\boldsymbol{\eta}\|\|\boldsymbol{\gamma}\|} & \frac{\boldsymbol{\gamma}^T\mathbf{H}\boldsymbol{\gamma}+\theta\boldsymbol{\gamma}^T\boldsymbol{\gamma}}{\boldsymbol{\gamma}^T\boldsymbol{\gamma}} \end{bmatrix} \begin{bmatrix} \alpha_\eta \\ \alpha_\gamma \end{bmatrix} = \begin{bmatrix} \frac{\boldsymbol{\eta}^T\boldsymbol{\gamma}}{M\|\boldsymbol{\eta}\|} \\ \frac{\boldsymbol{\gamma}^T\boldsymbol{\gamma}}{M\|\boldsymbol{\gamma}\|} \end{bmatrix} \tag{4.32}$$

$$\alpha_\eta^2 + 2\alpha_\eta\alpha_\gamma \frac{\boldsymbol{\eta}^T\boldsymbol{\gamma}}{\|\boldsymbol{\eta}\|\|\boldsymbol{\gamma}\|} + \alpha_\gamma^2 = 1 \tag{4.33}$$

In constructing the matrix of (4.32), the additional vector, $\mathbf{H}\boldsymbol{\gamma}$, is needed and is computed in the familiar form

$$\mathbf{H}\boldsymbol{\gamma} = \boldsymbol{\gamma} + \sum_{i\in Q}\left[\frac{(\mathbf{u}^i)(\mathbf{u}^i)^T\boldsymbol{\gamma}}{\beta_{\mathbf{u}}^i} + \frac{(\mathbf{v}^i)(\mathbf{v}^i)^T\boldsymbol{\gamma}}{\beta_{\mathbf{v}}^i}\right] \tag{4.34}$$

Solution values for α_η and α_γ are obtained by guessing on θ to solve the trivial 2×2 system (4.32) so that the solution, when used to compute the residual in (4.33), provides a means of adjusting θ in a trial and error fashion. When $M = \|\boldsymbol{\eta}\|$, the solution is $\theta = 0$, $\alpha_\eta = 1$, and $\alpha_\gamma = 0$ thereby yielding a step vector $\mathbf{d} = \boldsymbol{\eta}$. On the contrary, as M approaches 0, the solution will approach $\theta = \infty$, $\alpha_\eta = 0$, and $\alpha_\gamma = 1$. A modified line search is conducted as follows. First, the value of the augmented Lagrangian is computed at the current point.

$$\Phi(\mathbf{a}, \boldsymbol{\lambda}) = f(\mathbf{a}) + \mathbf{g}(\mathbf{a})^T \boldsymbol{\lambda} + \frac{\rho}{2}\mathbf{g}(\mathbf{a})^T \mathbf{g}(\mathbf{a}) \tag{4.35}$$

In order to preserve superlinear convergence as much as possible, an initial choice of $M = \|\boldsymbol{\eta}\|$ is made to obtain $\mathbf{d}$. The step is applied to the variable values by generating a vector $\mathbf{x} = \mathbf{a} + \mathbf{D_x d}$. The objective function and equation residuals are all evaluated at the new point and then scaled as above.

$$f(\mathbf{x}) \Leftarrow D_f^{-1} f(\mathbf{x}) \tag{4.36}$$

$$\mathbf{g}(\mathbf{x}) \Leftarrow \mathbf{D_g}^{-1} \mathbf{g}(\mathbf{x}) \tag{4.37}$$

Before accepting the new point, the augmented Lagrangian must be computed and compared with that evaluated at the old point in order to ensure descent.

$$\Phi(\mathbf{x}, \boldsymbol{\lambda}) = f(\mathbf{x}) + \mathbf{g}(\mathbf{x})^T \boldsymbol{\lambda} + \frac{\rho}{2}\mathbf{g}(\mathbf{x})^T \mathbf{g}(\mathbf{x}) \tag{4.38}$$

If $\Phi(\mathbf{x}, \boldsymbol{\lambda}) \geq \Phi(\mathbf{a}, \boldsymbol{\lambda})$, then the trust region radius must be reduced and a new step vector computed. Based on these evaluations as well as on the known gradient at the old point, a quadratic interpolation is generated and minimized to produce a formula for resetting M.

$$M \Leftarrow \frac{M}{2}\left(\frac{M\|\boldsymbol{\gamma}\|}{M\|\boldsymbol{\gamma}\| + \Phi(\mathbf{x}, \boldsymbol{\lambda}) - \Phi(\mathbf{a}, \boldsymbol{\lambda})}\right) \tag{4.39}$$

This new M is used to generate another step vector and the process is repeated until descent is achieved. In some instances, the quadratic interpolation may be inaccurate in which case the factor in the above equation may cause M to take on very small values. This has the detrimental effect of inhibiting progress. Typically, an upper bound on the reduction in M is set at about 90 percent in order to remedy the situation.

4.2.7 Iteration

With a new point accepted, the gradients are now evaluated and scaled.

$$\nabla_{\mathbf{x}} f(\mathbf{x}) \Leftarrow \mathbf{D}_{\mathbf{x}}^T \nabla_{\mathbf{x}} f(\mathbf{x}) D_f^{-1} \tag{4.40}$$

$$\nabla_{\mathbf{x}} \mathbf{g}(\mathbf{x}) \Leftarrow \mathbf{D}_{\mathbf{x}}^T \nabla_{\mathbf{x}} \mathbf{g}(\mathbf{x}) \mathbf{D}_{\mathbf{g}}^{-1} \tag{4.41}$$

The Hessian must be updated as well. First, the update vector $\mathbf{u}^k = \mathbf{Hd}$ is computed by taking advantage of previously calculated quantities.

$$\mathbf{u}^k = M(\alpha_\eta \frac{\boldsymbol{\gamma}}{\|\boldsymbol{\eta}\|} + \alpha_\gamma \frac{\mathbf{H}\boldsymbol{\gamma}}{\|\boldsymbol{\gamma}\|}) \tag{4.42}$$

The other update vector is broken up into parts $\mathbf{v}_L$ and $\mathbf{v}_A$.

$$\mathbf{v}_L = \nabla_{\mathbf{x}} f(\mathbf{x}) + \nabla_{\mathbf{x}} \mathbf{g}(\mathbf{x}) \boldsymbol{\lambda} - \nabla_{\mathbf{x}} f(\mathbf{a}) - \nabla_{\mathbf{x}} \mathbf{g}(\mathbf{a}) \boldsymbol{\lambda} \tag{4.43}$$

$$\mathbf{v}_A = \nabla_{\mathbf{x}} \mathbf{g}(\mathbf{x}) \mathbf{g}(\mathbf{x}) - \nabla_{\mathbf{x}} \mathbf{g}(\mathbf{a}) \mathbf{g}(\mathbf{a}) \tag{4.44}$$

In order keep $\mathbf{H}$ positive definite, the parameter ρ is adjusted here if necessary so that the new set of update vectors are consistent with Powell's popular criterion for positive definiteness [43].

$$\rho = MAX(\rho, \frac{0.01(\mathbf{u}^k)^T\mathbf{d} - (\mathbf{v}_L)^T\mathbf{d}}{(\mathbf{v}_A)^T\mathbf{d}}) \tag{4.45}$$

Although Powell recommends 0.2 for use with his mechanism of altering the Hessian, a less demanding value of 0.01 is used here. It is noted that (4.45) is a mechanism for only increasing values for ρ when needed. There is always a danger of numerical ill-conditioning when ρ becomes excessively large. It is felt in this work that scaling is important in this respect. Scaling offers a means of attempting to equalize the contributions of all three terms of the augmented Lagrangian. In doing so, threshold values for ρ for achieving positive definiteness may remain in an accessible range. A more thorough approach to adjusting ρ entails re-evaluating all vectors $\mathbf{u}^k, \mathbf{v}^k, \forall_{k \in Q}$ in order to reflect the change. This, however, would require not only much additional work but also additional storage of the vectors $\mathbf{d}$, $\mathbf{v}_L$, and $\mathbf{v}_A$ at each iteration. At least for problems involving nonlinear constraints, mechanism (4.45) is sufficient for adjusting ρ along an infeasible solution path. With linear constraints, feasibility will be reached earlier than optimality and $\mathbf{v}_A$ will be zero making formula (4.45) dangerous. But, in such a case, the Lagrangian itself is sufficiently positive definite at a local minimum and a preset specification of $\rho = 0$ should suffice. Finally, the remaining update vector is computed to be $\mathbf{v}^k = \mathbf{v}_L + \rho\mathbf{v}_A$ along with the scalars $\beta^k_{\mathbf{u}} = -(\mathbf{u}^k)^T\mathbf{d}$ and $\beta^k_{\mathbf{v}} = (\mathbf{v}^k)^T\mathbf{d}$ which will always be strictly negative and positive, respectively. To record the new information, the index k is added to the set Q ($Q \Leftarrow Q \vee \{k\}$). This marks the end of an iteration. Before beginning the next iteration, the vector $\mathbf{a}$ is set to $\mathbf{x}$ and the index k is incremented by one. The algorithm then proceeds above to check for convergence which marks the beginning of all iterations.

Problem (2.2) is used here once again to demonstrate the optimization algorithm. From the previously performed structural analysis, the variable x_2 was found to be an eligible independent variable and the problem was found to decompose into 5 separable blocks. The first four are trivially solved to give solution values $x_7 = 95$, $x_{10} = 153.535$, $x_5 = 2000$, and $x_9 = 1.56164$. The remaining subproblem requires initialization. The initial values as suggested by Hock and Schittkowski are used [23]. These values are then rounded up to the nearest power of 10 to form a set of nominal values suitable for scaling the variables. The solution path traversed by the algorithm using a tolerance of 10^{-7} is tabulated in Figure 4.3. The initial point is infeasible and so are the points of all iterations except the last. At two distinct points along the solution path, the parameter ρ is adjusted to keep the reduced Hessian (which is scalar) sufficiently positive. In advancing at each iteration, no modified line search was needed. The minimum value of -1768.81 obtained in the objective function agrees with that reported in the literature.

Iteration	x_1	x_2	x_3	x_4	x_6	x_8	ρ	Z^THZ
0	1745.00	12000.0	110.0000	3048.00	89.2000	8.0000	1.0000	57.6659
1	1726.18	12168.9	95.7775	3054.24	92.8371	8.2060	1.0158	247.0210
2	1735.40	11876.7	92.0678	3061.80	93.1498	7.9951	1.0158	237.4030
3	1732.84	11972.1	89.8242	3059.70	93.0526	8.0631	1.0158	6.3710
4	1625.55	15277.2	18.5923	2971.76	89.4601	10.4697	1.0158	1.4372
5	1697.28	15713.3	51.5075	3030.56	90.1698	10.4348	1.0158	1.6512
6	1698.02	15813.9	54.1323	3031.17	90.1172	10.4910	1.1248	11.7412
7	1698.08	15816.6	54.1099	3031.21	90.1164	10.4922	1.1248	8.6051
8	1698.08	15817.0	54.1085	3031.21	90.1162	10.4924	1.1248	1.6434
9	1698.09	15818.6	54.1027	3031.23	90.1154	10.4933	1.1248	1.6439

Figure 4.3: Solution path

4.3 Conditional models

In conditional models, the equations of each search region form a continuous and differentiable augmented Lagrangian. Each augmented Lagrangian is assumed to be convex provided the penalty parameter ρ used by all of them is large enough. If the solution to the conditional model is unique, then what is envisioned is that the equations of all but one search region will give rise to an augmented Lagrangian function which has no minimum internal to the search region in which it is defined. This implies that if conventional nonlinear programming techniques were used at some initial point internal to the correct search region, then the solution would be found. If, on the other hand, the initial point was placed inside an incorrect search region, in a finite number of iterations, one of the search region boundaries will be encountered. At a search region boundary, the only requirement for continuity in the model functionality is that the feasible region is continuous. The augmented Lagrangian functions of the equations of either side of a search region boundary need not be continuous and neither their gradients as well. This amounts to the existence of cliffs and valleys in the minimizing surface made up of a composite of all augmented

Lagrangians. The difficulty is then allowing solution progress to penetrate search region boundaries. With a mixed integer formulation, search region boundaries are jumped rather than crossed as mentioned earlier because continuity in the feasible region is not assumed nor required. This sometimes leads to the advantage of requiring only very few search regions to be visited before finding the solution.

4.3.1 Nondifferentiable optimization

Nondifferentiable optimization has been studied extensively where only the objective function was considered to contain nonsmooth functionality [24] [52] [64]. Much research has been aimed at the elaboration of algorithms such as the subgradient method for solving rather general classes of problems which do not assume in advance the knowledge of the specific structure of the minimized function, but require only the evaluation of the function and its gradients (or their analogues in the nondifferentiable case) at any given point. As a result, very slow algorithms have been developed (usually with a linear rate of convergence) in anticipation of a point where the function is nondifferentiable. The advantage that at almost all of the points in the feasible region the minimizing function will be differentiable, though, is lost.

In conditional modeling, all of the points where the functionality is nondifferentiable have been characterized by boundary hyper-planes as dictated by the physics of the model. This allows a more exact subgradient method to be employed, and, by further exploiting the continuous functionality, boundary crossing may take place. For all points not residing on boundary hyper-planes, superlinear convergence can be obtained.

4.3.2 Boundary crossing

Without differentiability, an analogue of the steepest descent vector on search region boundaries is derived from subgradient analysis, and this direction can be used to advance the solution progress into the correct search region. Once off of the boundaries, the superlinearly converging algorithm resumes using the possibly new set of equations. Assuming convexity in the individual augmented Lagrangian functions, cycling, which is the situation where a loop of connected search regions are visited continually, may be prevented by performing a gradient analysis. In addition, a gradient analysis can terminate the solution path at a point on a boundary if a local minimum to the augmented Lagrangians cannot be found internal to any of the search regions.

In solving continuous and differentiable models, the steepest descent direction at some point with values $\mathbf{a}$ for the variables and $\boldsymbol{\lambda}$ for the multipliers was mentioned to be opposite that of the gradient of the augmented Lagrangian at the point. This is true only at all nonstationary points where the gradient does not vanish. In fact, a vector $\mathbf{d}$ with a direction of steepest descent is a vector which solves the following subproblem.

$$\begin{aligned} \min \quad & \nabla_{\mathbf{x}} \Phi(\mathbf{a}, \boldsymbol{\lambda})^T \mathbf{d} \\ s.t. \quad & \mathbf{d}^T \mathbf{d} \leq 1 \end{aligned} \tag{4.46}$$

When a descent direction exists at $\Phi(\mathbf{a}, \boldsymbol{\lambda})$, the solution vector $\mathbf{d}$ will have the same direction as $-\nabla_{\mathbf{x}} \Phi(\mathbf{a}, \boldsymbol{\lambda})$ but will be of unit length. If, however, $\mathbf{a}$ is a local minimizer of $\Phi(\mathbf{x}, \boldsymbol{\lambda})$, any direction will yield the same minimal value of zero in the directional derivative. In summary, the steepest descent for a differentiable function is that direction which yields a minimum in the directional derivative.

In solving conditional models, multiple augmented Lagrangians exist at search region boundaries. Also, the multipliers will be different at the same point depending on which equations are used. Ignoring for now the fact that each augmented Lagrangian will have different values, the focus is on the gradients. All of these gradients can be used in any convex combination to yield an infinite number of what are called subgradients. These subgradients can be thought of as normals of supporting hyper-planes. For all points not on a boundary, only one supporting hyper-plane exists for the differentiable augmented Lagrangian, namely the one whose normal is given by the gradient. At search region boundaries, many directional derivatives exist for some direction $\mathbf{d}$ as given by using the gradients of all of the neighboring search regions. The directional derivative, in this situation, is defined to be the one generated with largest value. The analogy of the intersection of two inclined planes is useful here. Considering some point along the valley in the roof of a house, for example, the expression for the directional derivative along direction $\mathbf{d}$ changes depending on which side of the valley $\mathbf{d}$ points. The normals of the two inclined planes on either side of the valley are, by analogy, similar to the gradients of the augmented Lagrangian on either side of a search region boundary.

The directional derivative and, subsequently, the direction of steepest descent will now be derived at search region boundaries. The augmented Lagrangians of each search region s at some point $\mathbf{a}$ are referenced as $\Phi_s(\mathbf{a}, \boldsymbol{\lambda}_s)$ with the multipliers also being dependent on which equations are used.

$$\Phi_s(\mathbf{a}, \boldsymbol{\lambda}_s) = f(\mathbf{a}) + \mathbf{r}_s(\mathbf{a})^T \boldsymbol{\lambda}_s + \frac{\rho}{2} \mathbf{r}_s(\mathbf{a})^T \mathbf{r}_s(\mathbf{a}) \tag{4.47}$$

At the beginning of the gradient analysis, the multipliers are computed so as to zero the components of the gradient of the Lagrangian corresponding to the dependent variables $\mathbf{y}$.

$$\boldsymbol{\lambda}_s = -\nabla_{\mathbf{y}} \mathbf{r}_s(\mathbf{x})^{-1} \nabla_{\mathbf{y}} f(\mathbf{x}) \tag{4.48}$$

In doing so, the components of the gradient of the Lagrangian corresponding to the independent variables $\mathbf{z}$ when these values of the multipliers are used will make up the reduced gradient or what are sometimes referred to as constrained derivatives. These are then added to the gradient of the augmented term to produce the gradient of the augmented Lagrangian for each neighboring search region. Taking a point that resides on all of the boundary hyper-planes, the directional derivative will be given by the discrete maximum over $s \in \mathcal{R}$ of all search region directional derivatives. The minimum of this directional derivative over various directions constitutes the steepest descent.

$$\begin{aligned} &\min \quad \left(\begin{array}{ll} \max & \nabla_{\mathbf{x}} \Phi_s(\mathbf{a}, \boldsymbol{\lambda}_s)^T \mathbf{d} \\ s.t. & s \in \mathcal{R} \end{array} \right) \\ &s.t. \quad \mathbf{d}^T \mathbf{d} \leq 1 \end{aligned} \tag{4.49}$$

This is recognized as a mini-max problem which is convex, but it has a nonlinear inequality constraint. Because of convexity, the dual problem can be derived to yield the same solution and is seen to be a quadratic program whose solution is easier to understand.

First, the convex problem is re-formulated by introducing a variable β.

$$\begin{array}{ll} \min & \beta \\ s.t. & \beta \geq \nabla_{\mathbf{x}} \Phi_s(\mathbf{a}, \boldsymbol{\lambda}_s)^T \mathbf{d}, \ \forall_{s \in \mathcal{R}} \\ & \mathbf{d}^T \mathbf{d} \leq 1 \end{array} \tag{4.50}$$

For this problem, the Lagrangian is constructed by introducing a set of nonnegative multipliers $\boldsymbol{\alpha}$ and μ.

$$L = \beta + \sum_{s \in \mathcal{R}} (\nabla_{\mathbf{x}} \Phi_s(\mathbf{a}, \boldsymbol{\lambda}_s)^T \mathbf{d} - \beta) \alpha_s + (\mathbf{d}^T \mathbf{d} - 1)\mu \tag{4.51}$$

The terms are separable in the variables β and $\mathbf{d}$ over which the Lagrangian is minimized in order to construct the proper objective function for the dual problem. First, a relevant portion of the Lagrangian is minimized over β.

$$\min \quad \beta(1 - \sum_{s \in \mathcal{R}} \alpha_s) \tag{4.52}$$

In order for the primal problem to be feasible, the dual problem must be bounded. Therefore, to avoid an indefinite solution, the constraints on $\boldsymbol{\alpha}$ are that they must be greater than or equal to zero and add to 1 which results in the minimal value for this portion of the Lagrangian to be zero. Proceeding to minimize the remainder of the Lagrangian over the variables $\mathbf{d}$, the complementarity condition of μ must be investigated. The constraints relating $\mathbf{d}$ to the multipliers

$$\sum_{s\in\mathcal{R}} \nabla_{\mathbf{x}}\Phi_s(\mathbf{a},\boldsymbol{\lambda}_s)\alpha_s + 2\mu\mathbf{d} = 0 \tag{4.53}$$

indicate that when μ is zero, so is the value of the Lagrangian. Normally, though, μ exceeds zero and when the direction $\mathbf{d}$ obtained is substituted in the Lagrangian, negative values result. It follows, then, that the dual involves maximizing the minimal value in the Lagrangian (minimizing its negative) over the variables $\boldsymbol{\alpha}$ subject to their necessary constraints.

$$\begin{aligned} \min \quad & \|\sum_{s\in\mathcal{R}} \nabla_{\mathbf{x}}\Phi_s(\mathbf{a},\boldsymbol{\lambda}_s)\alpha_s\|^2 \\ s.t. \quad & \sum_{s\in\mathcal{R}} \alpha_s = 1 \\ & \alpha_s \geq 0,\ \forall_{s\in\mathcal{R}} \end{aligned} \tag{4.54}$$

In this case, termination is detected if a minimum of zero is obtained in (4.54) which occurs if all of the gradients cannot be confined to one side of a half-plane. Otherwise, the linear combination coefficients α_s which sum to one are used to generate the direction of steepest ascent, along the negative of which movement away from the boundaries should be made. That search region which is immediately entered by traveling along this direction becomes the current search region and the superlinear convergence algorithm resumes using the equations of this new search region. To demonstrate termination, an example is generated which can be seen graphically to have an optimal solution exist on a boundary plane.

$$\begin{aligned} \min \quad & x_1^2 + x_2^2 \\ s.t. \quad & \left\{ \begin{array}{l} x_2 \geq x_1 \\ 2x_1 + x_2 = 3 \end{array} \right\} \bigvee \left\{ \begin{array}{l} x_2 < x_1 \\ x_1 + 2x_2 = 3 \end{array} \right\} \end{aligned} \tag{4.55}$$

The nonsmooth feasible region along with the contours of the objective function are shown in Figure 4.4. The solution is seen to be the point with values $x_1 = 1$ and $x_2 = 1$. This point lies on the boundary of both search regions. In order to recognize it as a local solution point, the boundary crossing algorithm is demonstrated to terminate at this point.

Because it is on the boundary, the gradient analysis must be employed. Because of continuity of the feasible region at the search region boundary, the gradient of the augmented term vanishes when using the equations of either side and it suffices to be concerned only with the gradients of the Lagrangian (or, in fact, the reduced gradients) during the analysis. This indicates that the termination criterion is independent on the value of ρ. The multipliers are computed for each search region using the different equations. The variables are partitioned into dependent (x_1) and independent (x_2) variables. The multiplier for the search region where the condition $x_2 \geq x_1$ is met is calculated to be -1 while that for the other search region is calculated to be -2. The corresponding constrained derivatives with respect to the independent variable x_2 are 1 and -2 respectively for the two search regions. Since these are in opposite direction, the solution to (4.54) when these reduced gradients are used is a zero steepest descent direction indicating termination of the algorithm. The reader may note the equivalence of this solution with that of

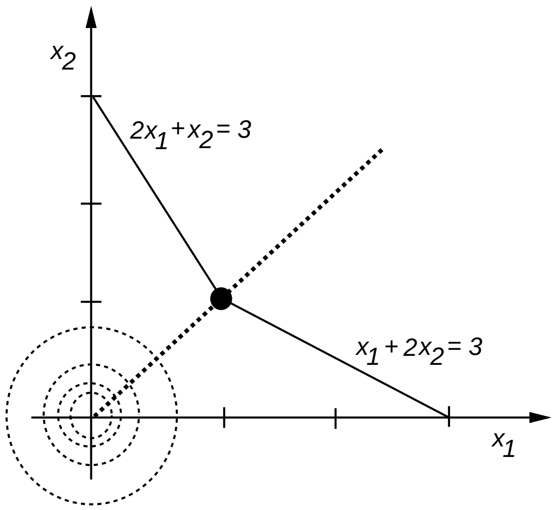

Figure 4.4: Boundary crossing termination

$$\min \quad x_1^2 + x_2^2 \tag{4.56}$$

$$\begin{aligned} s.t. \quad & x_1 + 2x_2 \geq 3 \\ & 2x_1 + x_2 \geq 3 \end{aligned}$$

4.4 Examples

All of the examples discussed in this report were solved using the algorithm prescribed above. The numerical results are given below along with a discussion about the performance of the algorithm on each example.

4.4.1 Phase equilibria

For this example, specifications of $T = 340\{K\}$ and $P = 1\{atm\}$ are made. The overall composition is set at $z_B = 0.50$, $z_E = 0.15$, and $z_W = 0.35$. An initial point is strategically chosen in the remaining variables to lie on all of the boundary planes which is one of many points neighboring all search regions. For the aqueous phase, a concentration of 2% benzene, 3% ethanol, and 95% water is assumed while for the organic phase, a concentration of 95% benzene, 3% ethanol, and 2% water is assumed. The vapor phase composition is initialized to be the same as the overall composition and the phase fractions of all phases is initialized to zero. At this initial point, all of the conditional equations are satisfied and therefore will not contribute to the gradient of the augmented Lagrangian in any of the search regions because there are no degrees of freedom and no objective function and the multipliers for all equations in this model are zero. Thus it is up to the residuals and their gradients of the unconditional equations to determine in which direction the iteration should continue. After computing the negative gradient of the augmented term using the unconditional equations, the direction suggests to increase the phase amounts of all phases by roughly the same amount. However, with regard to the composition variables, the direction indicates an increase in the mole fractions of all components in the aqueous and organic phases but a decrease in the mole fractions of all of the components in the vapor phase. As a result, the direction extends from the initial point toward the interior of search region AO. The solution path then begins using the equations of this search region with the premise that the vapor phase will not be present at the solution. The path traversed by the algorithm is shown in Figure 4.5. A solution consistent with search region AO was found. At the specified temperature and pressure, the aqueous phase is found to consist of 69% water while the organic phase consists of 81% benzene.

4.4.2 Heat exchange

To initiate solution of the heat exchange example, a feasible point was first generated using conservatively small values of $A = 250\{ft^2\}$ and $F_c = 250\{lbmole/hour\}$ for the independent variables. This starts the algorithm off with a temperature profile

Iteration	$\sum_{i \in C} y_i^A$	$\sum_{i \in C} y_i^O$	$\sum_{i \in C} y_i^V$	Φ^A	Φ^O	Φ^V
0	1.0000	1.0000	1.0000	0.0000	0.0000	0.0000
1	1.0000	1.0000	0.9030	4.5699	0.4301	0.0000
2	1.0000	1.0000	0.9212	0.5456	0.4544	0.0000
3	1.0000	1.0000	0.9563	0.4542	0.5458	0.0000
4	1.0000	1.0000	0.9808	0.4421	0.5579	0.0000
5	1.0000	1.0000	0.9807	0.4427	0.5573	0.0000
6	1.0000	1.0000	0.9807	0.4427	0.5573	0.0000

Figure 4.5: Solution path for the phase equilibria example

consisting of no condensation. Using the equations of this initial search region, the algorithm is carried out in Figure 4.6. At iteration 5, a boundary which neighbors the search region allowing condensation to occur near the outlet of the hot side of the heat exchanger is crossed. The last step vector is adjusted so as to lie on the boundary in order to conduct a gradient analysis. Since only one of the boundaries is encountered, only two gradients representing the two neighboring search regions are needed. They are used in a convex combination to generate a steepest descent vector which directs itself from the current point into the new search region not yet visited by the solution path. It is suggested that the solution is likely to be found by continuing onto the other side of the boundary. The equations to allow condensation near the outlet of the hot side of the heat exchanger are then incorporated and the solution path continues slowly but steadily. In this work, the Hessian matrix is re-initialized to the identity matrix when a boundary is crossed since it is not obvious that continuing to update the Hessian developed from another search region will improve convergence. In the remaining iterations of Figure 4.6, the variable η^1 grows to track the movement of the new condensation point. The optimal value of the objective function achieved is 15.501 and the corresponding temperature profile is shown in Figure 4.7. A sensitivity analysis was also performed on this example to observe the behavior of the objective function in a local neighborhood of the feasible region around the optimal solution.

Iteration	$\sum_{i \in C} x_i^0$	Φ^0	η^1	F_c	A	$z^T H z$
0	0.9584	0.0000	0.0000	250.00	250.00	[6.3601, −2.9359, −2.9359, 5.2016]
1	0.9544	0.0000	0.0000	275.27	228.80	[1.2187, 0.5592, 0.5592, 3.5734]
2	0.9578	0.0000	0.0000	441.18	166.66	[0.6761, 0.4050, 0.4050, 7.6304]
3	0.9614	0.0000	0.0000	426.46	198.99	[0.8809, −0.0225, −0.0225, 5.4091]
4	0.9675	0.0000	0.0000	477.68	202.28	[0.2582, −0.3510, −0.3510, 5.9880]
5	1.0000	0.0000	0.0000	607.83	213.01	[0.2288, −0.5982, −0.5982, 6.1331]
15	1.0000	0.0001	0.0010	672.89	213.21	[0.1079, 1.8790, 1.8790, 228.7360]
20	1.0000	0.0137	0.2400	1049.99	269.31	[1.8175, −8.8203, −8.8203, 43.8733]
25	1.0000	0.0366	0.3904	1401.47	357.09	[0.3407, −0.7645, −0.7645, 3.0426]
30	1.0000	0.0427	0.4233	1179.39	396.06	[0.1195, −0.2233, −0.2233, 1.6262]
35	1.0000	0.0383	0.3888	1104.13	379.12	[0.0525, −0.0926, −0.0926, 1.0375]
36	1.0000	0.0383	0.3888	1104.13	379.12	[0.0530, −0.1006, −0.1006, 1.1537]

Figure 4.6: Solution path for the heat exchange example

The plot of data shown in Figure 4.8 confirms the obtained solution as being an optimal one.

4.4.3 Adiabatic compressible flow

This model is initialized to be in the plug flow state by using the values shown in the first row of Figure 4.9. At iteration 5, the outlet Mach number reaches 1 and a subsequent gradient analysis dictates to continue onto the other side of the boundary. From there, it is observed that P_f in fact rises above the discharge pressure thereby causing the shock waves which are associated with sonic flow. An optimal diameter of $8.6\{cm\}$ is obtained with a minimal objective value of -1281.46.

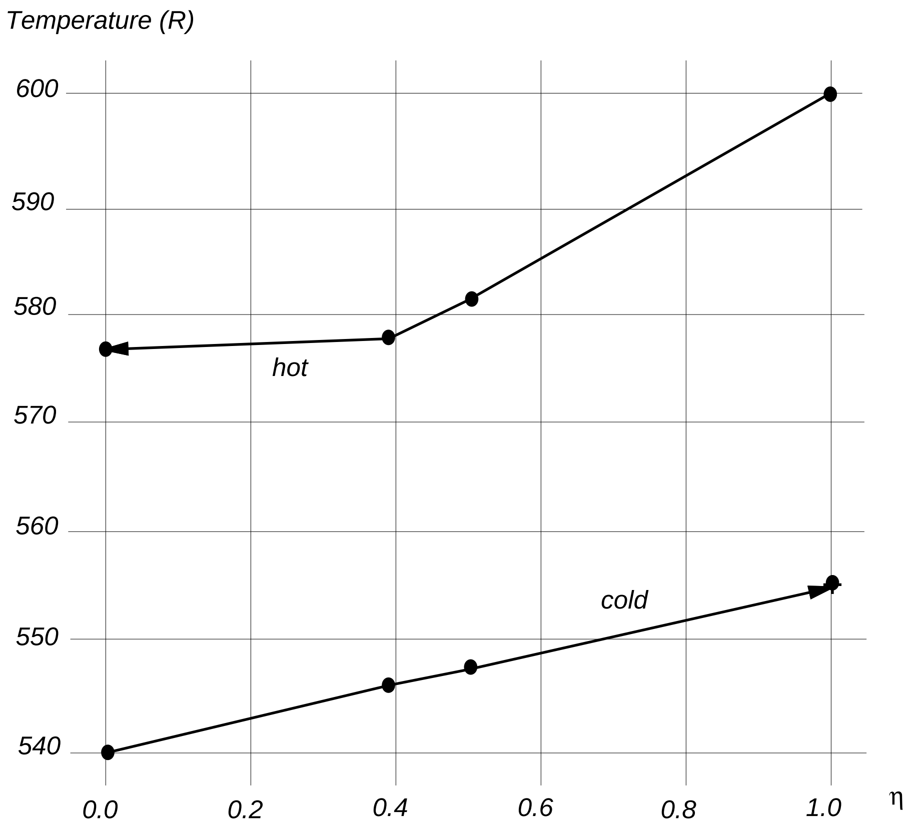

Figure 4.7: Optimal temperature profile

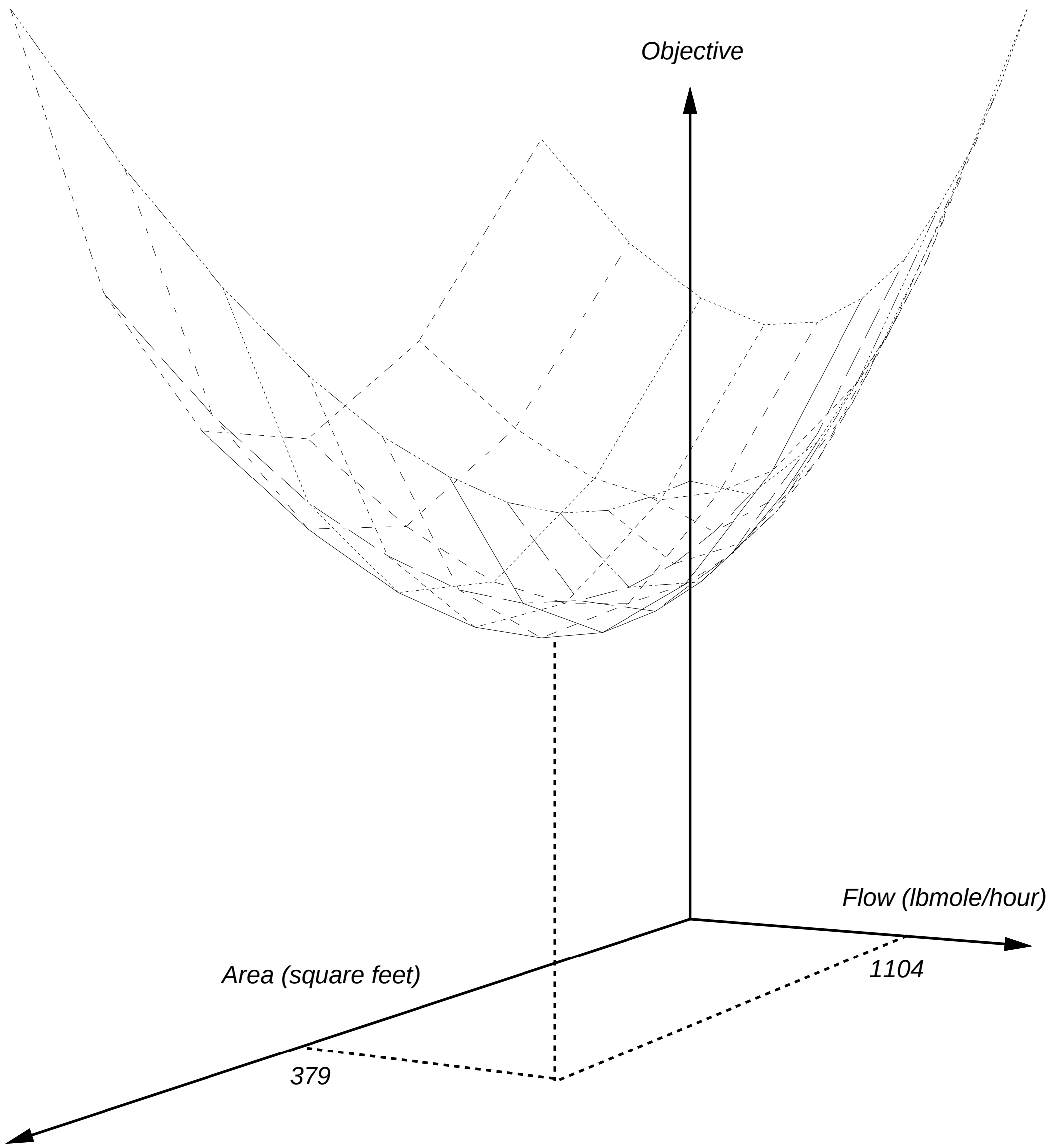

Figure 4.8: Sensitivity analysis

Iteration	M_f	P_f	T_f	M_i	F	D	ρ	$Z^T HZ$
0	0.5000	5.0000	300.00	0.5000	200.00	5.0000	1.0000	11.5061
1	0.6617	5.0000	294.73	0.5370	50.10	3.0150	1.0000	324.7510
2	0.7711	5.0000	284.02	0.4467	41.00	2.6451	1.0000	4.2682
3	0.8630	5.0000	279.01	0.4561	37.78	2.4114	1.0000	3.3564
4	0.9056	5.0000	276.58	0.4695	47.73	2.6856	16.0167	0.3572
5	1.0000	5.0000	271.62	0.5428	105.02	3.9798	16.0167	0.4993
10	0.9949	5.7084	283.77	0.5860	467.22	7.0324	16.0167	0.5658
15	1.0000	5.9189	276.33	0.6171	626.74	8.4303	16.0167	0.9485
20	1.0000	5.9537	276.48	0.6202	662.01	8.6345	16.0167	0.5213
21	1.0000	5.9537	276.48	0.6202	662.01	8.6345	16.0167	0.5140

Figure 4.9: Solution path for the adiabatic compressible flow example

Chapter 5

Contributions

5.1 Introduction

In this chapter, a summary of what are believed to be significant contributions in many aspects of conditional modeling and in equation-based modeling in general is presented. Also, contributions which are only nearly completed at present and are expected to be finished in the near future will also be discussed. It should be noted that all theoretical contributions are completed and only some implementation contributions remain.

5.2 Present

A formulation has been presented which is believed to unambigously declare what must be true at the solution of a conditional model. It is hoped that the notation of (2.3) can be used in future references to the subject in an effort to standardize continued work. For the most part, conditional models were always believed to be instances of discrete or discontinuous and nondifferentiable modeling. In this work, it has been successfully shown that there does exist a special class of problems conforming to a conditional model where, although the functionality may lose differentiability, continuity is preserved. Furthermore, this distinction has been exploited here in developing techniques for structurally analyzing and solving conditional models.

Structural analysis is seldom discussed in the literature as an intermediate step between formulating and solving equation-based models. Consistency is usually assumed. But in practice, it is very difficult to set up large models of equations without mistakingly introducing some structural inconsistencies. This is one of the main arguments utilized in the development of the ASCEND (Advanced System for Computations in ENgineering Design) modeling environment [40] [61]. The ideas discussed here for structurally analyzing continuous and differentiable models are implemented as tools within the ASCEND environment with which users can query about formulation errors. It is felt that more efficient algorithms for carrying out consistency checks are offered in this work. Much of the research conducted here has been in parallel with that of the continuing development of ASCEND and so many of the ideas developed are and have been used to update and introduce new tools.

During the course of conducting research in the area of conditional modeling, no reference to the structural analysis of this class of problems has been found throughout the literature. The extent of nondifferentiable modeling has always been limited to that which can adequately be described using inequality constraints. The burden has been on the active set strategy for deciding the binding constraints. A methodology to ensure consistency in conditional models has been presented in this report. The significance of this contribution is that tools can now be provided in ASCEND which can aid users to query about conditional models as well as conventional ones. The algorithms provided in this report are readily reproduced by the interested reader who may wish to implement and test them.

A significant contribution of this research has been made in the area of nonlinear programming. A thorough review of existing theory on the subject was given earlier in this report and at least 2 troubling discrepancies surfaced. With this knowledge, an elegant solution algorithm has evolved through the consistent use of the augmented Lagrangian which is applicable to programs with and without degrees of freedom with little or no change in efficiency. Scaling is also seldom discussed in optimization literature because formulations are assumed to be well-scaled. A finding in this work is that scaling has dramatic influence on the performance of a solution algorithm. It is felt that the scaling technique described in this report is very effective, in practice, for eliminating ill-conditioning. With proper selection of nominal values for the variables, the scaled Jacobian is expected to have all of its elements of order unity. It is readily automated and allows the user to formulate freely without regard to scaling and yet the user still has control in the scaling process by providing the nominal values for the variables at solution time. The termination criterion used in this work is again consistent with simply finding a stationary point in the augmented Lagrangian. This has the advantage of also indicating infeasibility. The elegance of having a single termination criterion for reaching a solution of a constrained nonlinear program is attractive.

With regard to zeroing the linearization of the gradient of the augmented Lagrangian in order to generate superlinear updates, it is significant to mention that in the process of using reduced space techniques, no information is lost. Through modern software engineering, full information can be used in the sequential programming algorithm even with relatively large models. It has typically been thought prior to this research that some approximations were needed so that some quantities could be neglected when projecting the system into the reduced space. When information is lost, the convergence of some decomposition techniques for reducing the space such as orthogonal decomposition prove to be less sensitive to the error introduced than the coordinate based decomposition. Without any approximation, however, it should not matter which decomposition method is used. Decomposition should be treated simply as a means of efficiently solving for the superlinear updates without ever having to construct and factorize the full Hessian matrix. With that in mind, the coordinate based method is chosen in this work.

Another discrepancy found in conventional nonlinear programming was the fact that line search was constrained to a fixed search direction whereas in nonlinear equation solving, it is well known that alternative descent directions should be used when being forced to consider reduced step lengths. The method of Westerberg and Director has already been successfully used to globalize newton's method for equation solving in ASCEND [59]. The applicability of the method has been extended to nonlinear optimization. Finally, a method of updating the penalty parameter for the augmented Lagrangian is introduced. No other reference to doing this dynamically from iteration to iteration has been found.

An initial version of the solution algorithm described above has been successfully implemented in the software package SLV based in the C programming language [63]. SLV was first developed as an efficient equation solving engine for ASCEND. One of the accomplishments of this work was supplementing SLV with the ability to perform nonlinear optimization. The version of the solution algorithm currently implemented successfully optimizes moderately sized models including a mixed differential and algebraic optimal control problem which was discretized and formulated as an equality constrained nonlinear program involving a little over 700 constraints. The algorithm required just under 25 iterations before finding a solution. Also implemented within the most current version of ASCEND is another widely used optimization software package MINOS [36]. It was found that MINOS failed to reach an optimal solution for this model.

The limited memory BFGS method has been implemented in SLV by taking advantage of linked list structures in C which allow easy accumulation and removal of update information as solution progress is made. Recently, much improvement has been made to the linear equation solver utilized by SLV. The operators offered with the sparse matrix data type were investigated when timing profiles were reported for solving the linearizations of large ASCEND models. It was found that primitive operations such as row and column traversal could be more efficiently implemented and, when they were, at least an order of magnitude reduction in the timing profiles were obtained. In addition, the existing pivoting algorithm was found to be developed primarily with preserving sparsity (*i.e.*, prevent fill-in during matrix factorization) in mind and failed to accurately solve linear systems which are only slightly ill-conditioned. A bi-partial pivoting algorithm (as it was referred to by co-workers) was implemented which expands the influence of a pre-existing pivot tolerance parameter. The pivot tolerance parameter dictates the minimum allowed in the ratio of how large in magnitude the element chosen for pivoting during matrix factorization is relative to the largest magnitude of the elements available for pivoting. Commonly, as in Gauss elimination, the search for alternative pivots when the pivot tolerance criterion is not satisfied is limited to either the current row or current column. Essentially, a contribution which proves to be extremely effective in solving linear systems which are even very ill-conditioned is to conduct the search for pivot candidates both row-wise and column-wise. With a pivot tolerance near 1, the linear solver is practically immune to ill-conditioning. As soon as it was developed, the bi-partial pivoting algorithm was ported over to ASCEND and is now widely used.

With regard to solving conditional models, a new boundary crossing algorithm is presented. Prior to this research, the techniques of subgradient optimization have only been applied to models where only the objective function is known to be nondifferentiable. In this work, the theory is outlined allowing the ideas to be extended to conditional models. Boundary crossing has been found to exist in some software implementations (SPEEDUP, for example) but it is executed without theoretical backing. What has been contributed here is an algorithm that is shown to terminate and prevent cycling.

In addition to the above considerations of formulation, structural analysis, and solution, much research has been conducted on model interfacing. To make SLV by itself more readily available to engineers, a language has been designed. An equation parser is implemented, allowing users to freely express equations and introduce meaningful variable names. The equations are stored essentially as a tree of unary, binary, and operand tokens so that they can be evaluated and differentiated numerically without approximation. The language has been designed to aid in model construction. It retrieves model descriptions from an input file where constants, variables, equations, and an objective function are specified. Conditions are imposed on some of the equations using an IF construct and multiple conditions are grouped using AND. The description for the conditional phase equilibrium model is given below.

MODEL

PARAMETERS

$y_B^A := 0.02$, $y_E^A := 0.03$, $y_W^A := 0.95$,
$y_B^O := 0.95$, $y_E^O := 0.03$, $y_W^O := 0.02$,
$y_B^V := 0.50$, $y_E^V := 0.15$, $y_W^V := 0.35$,
$\phi^A := 0.0$, $\phi^O := 0.0$, $\phi^V := 0.0$;

BOUNDARIES

aqueous: $y_B^A + y_E^A + y_W^A + \phi^A \geq 1.0$,
organic: $y_B^O + y_E^O + y_W^O + \phi^O \geq 1.0$,
vapor: $y_B^V + y_E^V + y_W^V + \phi^V \geq 1.0$;

EQUATIONS

IF aqueous $y_B^A + y_E^A + y_W^A = 1.0$,
IF NOT aqueous $\phi^A = 0.0$,
IF organic $y_B^O + y_E^O + y_W^O = 1.0$,
IF NOT organic $\phi^O = 0.0$,
IF vapor $y_B^V + y_E^V + y_W^V = 1.0$,
IF NOT vapor $\phi^V = 0.0$,
$y_B^V = 0.652 * y_B^A *$
$\exp(1.695*(1-y_B^A)*y_E^A + 3.16*(1-y_B^A)*y_W^A - 1.035*y_E^A*y_W^A)$,
$y_E^V = 0.610 * y_E^A *$
$\exp(1.695*y_B^A*(1-y_E^A) - 3.16*y_B^A*y_W^A + 1.035*(1-y_E^A)*y_W^A)$,
$y_W^V = 0.267 * y_B^A *$
$\exp(-1.695*y_B^A*y_E^A + 3.16*y_B^A*(1-y_W^A) + 1.035*y_E^A*(1-y_W^A))$,

$y_B^V = 0.652*y_B^O*$
$\quad \exp(1.695*(1\text{-}y_B^O)*y_E^O + 3.16*(1\text{-}y_B^O)*y_W^O - 1.035*y_E^O*y_W^O),$
$y_E^V = 0.610*y_E^O*$
$\quad \exp(1.695*y_B^O*(1\text{-}y_E^O) - 3.16*y_B^O*y_W^O + 1.035*(1\text{-}y_E^O)*y_W^O),$
$y_W^V = 0.267*y_B^O*$
$\quad \exp(\text{-}1.695*y_B^O*y_E^O + 3.16*y_B^O*(1\text{-}y_W^O) + 1.035*y_E^O*(1\text{-}y_W^O)),$
$\phi^A*y_B^A + \phi^O*y_B^O + \phi^V*y_B^V = 0.50,$
$\phi^A*y_E^A + \phi^O*y_E^O + \phi^V*y_E^V = 0.15,$
$\phi^A*y_W^A + \phi^O*y_W^O + \phi^V*y_W^V = 0.35;$

END

The language is adequate for small to moderately sized models. For the construction of larger models, the ASCEND environment is preferable. In fact, the motivation for all research in this work stems from the desire to help in the advancement of ASCEND. The ASCEND system consists of a language to aid in model construction, an interface to aid in model management, and a solver to aid in model solution. SLV was chosen as the primary solver for the ASCEND system.

To further demonstrate the robustness of SLV, two additional examples are used. The model below has the characteristic of a quadratic program and therefore should be solved exactly in a finite number of iterations using the algorithm proposed in chapter 4.

MODEL

PARAMETERS
x_0, x_1, x_2, x_3,
x_4, x_5, x_6, x_7;

OBJECTIVE Minimize
$x_0*x_0 + x_1*x_1 + x_6*x_6 + x_7*x_7;$

EQUATIONS
$x_0 - x_2 = 1.0,$
$-x_1 + x_3 = -1.0,$
$x_2 - x_4 = 1.0,$
$-x_3 + x_5 = -1.0,$

x_4 - x_6 = 1.0,
-x_5 + x_7 = -1.0;

END

The above formulation converges exactly after 3 iterations starting with an initial value of 1.0 for all variables and using x_6 and x_7 as the decision variables. A minimal value of 9 is achieved in the objective using solution values $x_6 = -1.5$ and $x_7 = -1.5$. Another more difficult formulation exhibits 4 degrees of freedom.

MODEL

PARAMETERS
c_{11}, c_{12}, c_{21}, c_{22}, c_{23},
c_{31}, c_{32}, u_1, u_{21}, u_{22}, u_3,
q_1, q_2, q_3;

OBJECTIVE Minimize
q_1 + q_2 + q_3;

EQUATIONS
sqr(c_{11}) + sqr(c_{12}) = 1.0,
u_{21} = c_{11} - c_{12} + 2*u_1,
5*sqr(c_{21}) + 2*sqr(c_{21} + u_{21}) + 0.8*u_{21} +
sqr(c_{21} + c_{23}) = 8.0,
u_1 = c_{21} - c_{22} + u_{21} - 3*u_{22},
u_3 = 2*c_{22} - c_{23} - u_{21} + u_{22},
c_{31} + u_3 + 0.5 = 0.0,
u_{22} = c_{31} + 2.5*c_{32} - 4*u_3,
q_1 = sqr(sqr(u_1 - 1)) + 5*sqr(c_{11} + c_{12} - 2),
q_2 = 2*sqr(c_{21} - 2) + sqr(c_{22}) + 3*sqr(c_{23}) +
4*sqr(u_{21}) + sqr(u_{22}),
q_3 = sqr(c_{31} + 1) + sqr(u_3 - 1) + 2.5*sqr(c_{32});

END

This model converges in 14 iterations using u_1, u_{21}, u_{22}, and u_3 as the independent variables and starting with an inital value of 1.0 for all variables. A minimal value of

6.1008 is achieved using solution values $u_1 = 0.1729$, $u_{21} = 0.0286$, $u_{22} = 0.3313$, and $u_3 = 0.0050$.

Much contribution has been made toward the development of model libraries [66]. Graphical representations of the ASCEND language were devised to pictorialize model libraries, making it easier to construct and to convey to other users [65]. The two most significant libraries which have received overwhelming attention and have proved usefulness in numerous applications are the thermodynamics and integration libraries. At least the thermodynamics library has enabled the modeling of nonideal rigorous distillation which otherwise had not been done in ASCEND. In developing the thermodynamics library, many variable types were needed of specific dimensionality such as molar energy and temperature. There is the danger of writing a dimensionally inconsistent thermodynamic equation. For that reason, ASCEND stores dimensionality with all real quantities used in modeling. With some thought, a new tool was provided in ASCEND which can scan all the equations of any model for dimensional consistency and report any errors. This was found to be extremely useful because many times when convergence is not achieved while solving a model, an unwanted dimensional inconsistency in an equation is often the culprit. The integration library provides assistance in discretizing differential and algebraic models in ASCEND. The model structure of the libraries is pictorialized in Figures 5.1 and 5.2. The interested reader is referred to the technical report in which they are first introduced for more information [66].

5.3 Future

As was mentioned above, an initial version of a nonlinear optimization algorithm has been implemented and is currently used in ASCEND. Recent advancements in the research have led the solution algorithm described in this report to be detailed

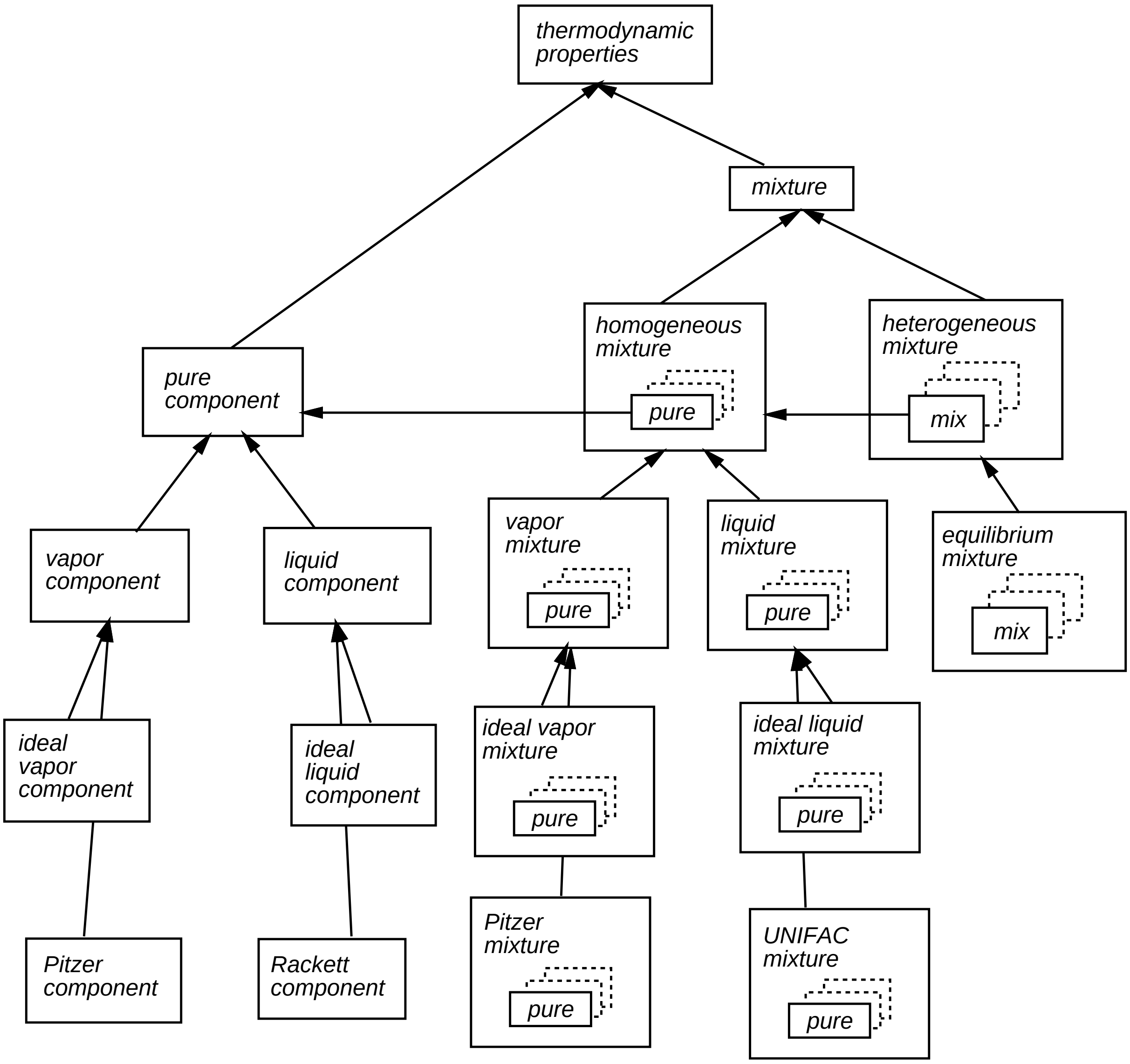

Figure 5.1: Thermodynamics library

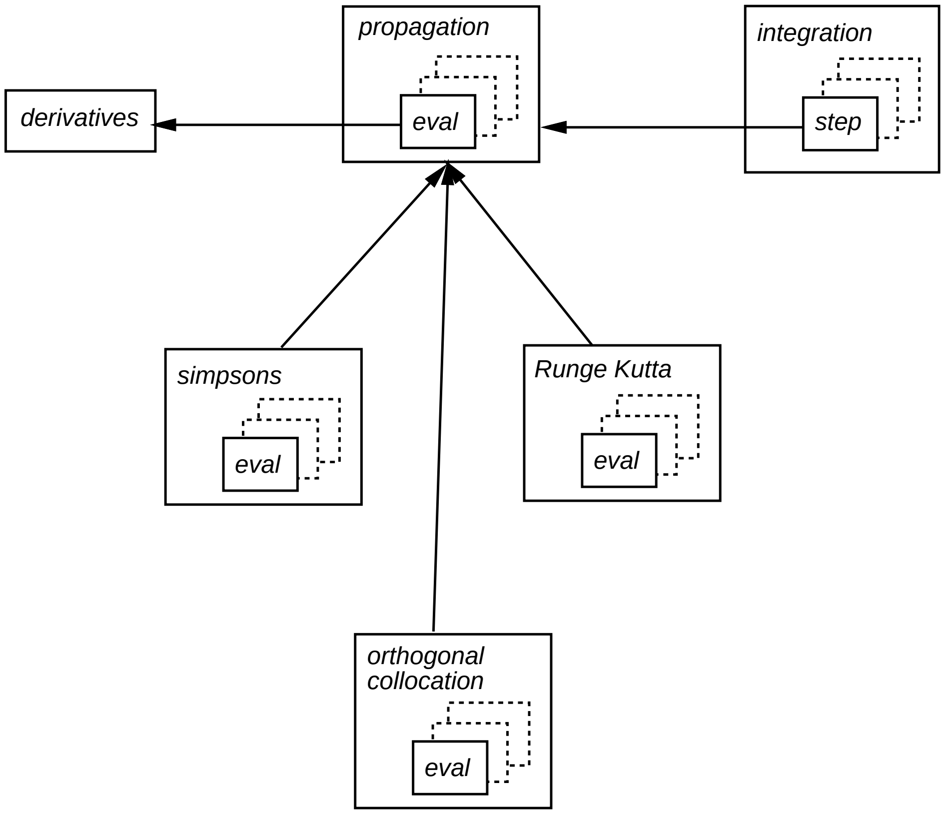

Figure 5.2: Integration library

as it is now after the initial implementation was completed. It is expected that when the initial version is updated to account for new knowledge, the solver will be significantly more robust. It is also mentioned that the boundary crossing algorithm for solving conditional models has not yet been implemented. The examples solved in this report were done by hand when investigating boundary crossing and by computer when nonlinear optimization resumed internal to the search regions.

With the setting up and solving of conditional models outlined in this work, future contributions will be to supplement the ASCEND language with the ability to associate conditions with some equations so that large conditional models can be constructed. Currently, there exists a skeleton data structure to store conditional information about equations in the form of a CASE statement similar to how it is used in the Pascal programming language. Its functionality, however, remained dormant because until now, an unambiguous formulation and solution procedure was lacking.

Bibliography

[1] E. Balas. Disjunctive programming. In P. L. Hammer, E. L. Johnson, and B. H. Korte, editors, *Discrete Optimization II*. North-Holland, 1979.

[2] M. C. Bartholomew-Biggs and M. de F. G. Hernandez. Using the kkt matrix in an augmented lagrangian sqp method for sparse constrained optimization. *Journal of Optimization Theory and Applications*, 85:201–220, 1995.

[3] P. I. Barton. *The Modeling and Simulation of Combined Discrete and Continuous Processes*. PhD thesis, Imperial College of Science, Technology and Medicine, 1992.

[4] M. S. Bazaraa and C. M. Shetty. *Nonlinear Programming, Theory and Algorithms*. John Wiley, 1979.

[5] N. Beaumont. An algorithm for disjunctive programs. *European Journal of Operational Research*, 48:362–371, 1990.

[6] T. J. Berna, M. H. Locke, and A. W. Westerberg. A new approach to optimization of chemical processes. *AIChE Journal*, 26:37–43, 1980.

[7] J. T. Betts and P. D. Frank. A sparse nonlinear optimization algorithm. *Journal of Optimization Theory and Applications*, 82:519–540, 1994.

[8] L. G. Bullard and L. T. Biegler. Iterated linear programming strategies for non-smooth simulation: a penalty based method for vapor-liquid equilibrium applications. *Computers and Chemical Engineering*, 17:95–109, 1993.

[9] B. P. Cairns and I. A. Furzer. Multicomponent three-phase azeotropic distillation. 3. modern thermodynamic models and multiple solutions. *Ind. Eng. Chem. Res.*, 29:1383–1395, 1990.

[10] V. P. Carey. *Liquid-Vapor Phase-Change Phenomena: An Introduction to the Thermophysics of Vaporization and Condensation Processes in Heat Transfer Equipment*. Hemisphere Publishing Corporation, 1992.

[11] B. Carnahan, H. A. Luther, and J. O. Wilkes. *Applied Numerical Methods*. John Wiley, 1969.

[12] R. W. Cottle and G. B. Dantzig. Complementary pivot theory of mathematical programming. *Linear Algebra and Its Applications*, 1:103–125, 1968.

[13] J. R. Jr. Dennis and J. J. Moré. Quasi-newton methods, motivation and theory. *SIAM Rev.*, 19:46–89, 1977.

[14] J. M. Douglas. *Conceptual Design of Chemical Processes*. McGraw-Hill Book Company, 1988.

[15] I. S. Duff. On algorithms for obtaining maximum transversal. *ACM Trans. Math. Software*, 7:315–330, 1981.

[16] R. Fletcher. *Practical Methods of Optimization*. John Wiley, 1987.

[17] P. E. Gill, W. Murray, M. A. Saunders, and M. H. Wright. Constrained nonlinear programming. Technical Report SOL 87-13, Systems Optimization Laboratory, Stanford University, December 1987.

[18] S. M. Goldfeld, R. E. Quandt, and H. F. Trotter. Maximization by quadratic hillclimbing. *Econometrica*, 34:541–551, 1966.

[19] I. E. Grossman. Minlp optimization strategies and algorithms for process synthesis. In *The 3rd International Conference of Foundations of Computer-Aided Process Design*, Snowmass Village, July 1989.

[20] R. Guatam and W. D. Seider. Computation of phase and chemical equilibrium: Part ii. phase splitting. *AIChE Journal*, 25:999–1006, 1979.

[21] S. P. Han. A hybrid method for nonlinear programming. In O. L. Mangasarian, R. R. Meyer, and S. M. Robinson, editors, *Nonlinear Programming 3*. Academic Press, Inc., 1978.

[22] M. R. Hestenes. Multiplier and gradient methods. *Journal of Optimization Theory and Applications*, 4:303–320, 1969.

[23] W. Hock and K. Schittkowski. Test examples for nonlinear programming codes. In M. Beckmann and H. P. Künzi, editors, *Lecture Notes in Economics and Mathematical Systems, Volume 187*. Springer-Verlag, 1981.

[24] C. Lemarechal. Nondifferentiable optimization. In G. L. Nemhauser, editor, *Handbook in OR and MS, Vol. 1*. Elsevier Publishers B. V., North Holland, 1989.

[25] K. Levenberg. A method for the solution of certain nonlinear problems in least squares. *Q. Appl. Math.*, 2:164–168, 1944.

[26] D. C. Liu and J. Nocedal. On the limited memory bfgs method for large scale optimization. Technical Report NAM 03, Department of Electrical Engineering and Computer Science, Northwestern University, 1988.

[27] M. H. Locke, A. W. Westerberg, and R. H. Edahl. An improved successive quadratic programming optimization algorithm for engineering design problems. *AIChE Journal*, 29:871–874, 1983.

[28] S. Lucidi. Recursive quadratic programming algorithm that uses an exact augmented lagrangian function. *Journal of Optimization Theory and Applications*, 67:227–245, 1990.

[29] O. L. Mangasarian. Solution of symmetric linear complementarity problems by iterative methods. *Journal of Optimization Theory and Applications*, 22:465–485, 1977.

[30] D. W. Marquadt. An algorithm for least-squares estimation of non-linear parameters. *SIAM Journal*, 11:431–441, 1963.

[31] W. L. McCabe and J. C. Smith. *Unit Operations of Chemical Engineering.* McGraw-Hill Book Company, 1985.

[32] M. L. Michelsen. The isothermal flash problem. part i. stability. *Fluid Phase Equilibria*, 9:1–19, 1982.

[33] M. L. Michelsen. The isothermal flash problem. part ii. phase-split calculation. *Fluid Phase Equilibria*, 9:21–40, 1982.

[34] M. L. Michelsen. Calculation of multiphase equilibrium. *Computers and Chemical Engineering*, 18:545–550, 1994.

[35] J. J. Moré. The levenberg-marquadt algorithm: Implementation and theory. In G. A. Watson, editor, *The Dundee Conference on Numerical Analysis.* Springer-Verlag, Berlin, 1978.

[36] B. A. Murtaugh and M. A. Saunders. Minos 5.1 user's guide. Technical Report SOL 83-20R, Systems Optimization Laboratory, Stanford University, January 1987.

[37] P. Y. Papalambros and D. J. Wilde. *Principles of Optimal Design, Modeling and Computation.* Cambridge University Press, 1988.

[38] H. N. Pham and M. F. Doherty. Design and synthesis of heterogeneous azeotropic distillations - i. heterogeneous phase diagrams. *Chemical Engineering Science*, 45:1823–1836, 1990.

[39] P. C. Piela. *ASCEND: An Object-Oriented Computer Environment for Modeling and Analysis.* PhD thesis, Carnegie Mellon University, 1989.

[40] P. C. Piela, T. G. Epperly, K. M. Westerberg, and A. W. Westerberg. Ascend: An object-oriented computer environment for modeling and analysis: The modeling language. *Computers and Chemical Engineering*, 15(1):53–72, 1991.

[41] M. J. D. Powell. A hydbrid method for nonlinear equations. In P. Rabinowitz, editor, *Numerical Methods for Nonlinear Algebraic Equations*. Gordon and Breach, 1970.

[42] M. J. D. Powell. The convergence of variable metric methods for nonlinearly constrained optimization calculations. In O. L. Mangasarian, R. R. Meyer, and S. M. Robinson, editors, *Nonlinear Programming 3*. Academic Press, Inc., 1978.

[43] M. J. D. Powell. A fast algorithm for nonlinearly constrained optimization calculations. In G. A. Watson, editor, *The Dundee Conference on Numerical Analysis*. Springer-Verlag, Berlin, 1978.

[44] M. J. D. Powell. Methods for nonlinear constraints in optimization calculations. In A. Iserles and M. J. D. Powell, editors, *The State of the Art in Numerical Analysis*. Oxford University Press, New York, 1987.

[45] R. Raman. *Integration of Logic and Heuristic Knowledge in Discrete Optimization Techniques for Process Systems*. PhD thesis, Carnegie Mellon University, 1993.

[46] R. Raman. Modelling and computational techniques for logic based integer programming. *Computers and Chemical Engineering*, 18:563–578, 1994.

[47] R. C. Reid, J. M. Prausnitz, and B. E. Poling. *The Properties of Gases and Liquids, 4th edition*. McGraw-Hill Book Company, 1987.

[48] B. A. Ross and W. D. Seider. Simulation of three-phase distillation towers. *Computers and Chemical Engineering*, 5:7–20, 1980.

[49] L. E. Scales. *Introduction to Nonlinear Optimization*. Springer-Verlag, New York, 1985.

[50] K. Schittkowski. The nonlinear programming method of wilson, han, and powell with an augmented lagrangian type line search function. *Numerische Mathematik*, 38:83–114, 1981.

[51] C. Schmid. *Reduced Hessian Successive Quadratic Programming for Large-Scale Process Optimization*. PhD thesis, Carnegie Mellon University, 1994.

[52] N. Z. Shor. *Minimization Methods for Nondifferentiable Functions*. Springer-Verlag, Berlin, 1985.

[53] S. Smith and L. Lasdon. Solving large sparse nonlinear programs using grg. *ORSA Journal on Computing*, 4:2–14, 1992.

[54] R. A. Tapia. Quasi-newton methods for equality constrained optimization: Equivalence of existing methods and a new implementation. In O. L. Mangasarian, R. R. Meyer, and S. M. Robinson, editors, *Nonlinear Programming 3*. Academic Press, Inc., 1978.

[55] R. Tarjan. Depth-first search and linear graph algorithms. *SIAM J. Comput.*, 1:146–160, 1972.

[56] S. Vasantharajan and L. T. Biegler. Large-scale decomposition for successive quadratic programming. *Computers and Chemical Engineering*, 12(11):1087–1101, 1988.

[57] S. M. Walas. *Phase Equilibria in Chemical Engineering*. Butterworth Publishers, 1985.

[58] A. W. Westerberg and D. R. Benjamin. Thoughts on a future equation-oriented flowsheeting system. *Computers and Chemical Engineering*, 9(5):517–526, 1985.

[59] A. W. Westerberg and S. W. Director. A modified least squares algorithm for solving sparse nxn sets of nonlinear equations. *Computers and Chemical Engineering*, 2(2):77–81, 1978.

[60] A. W. Westerberg, H. P. Hutchinson, R. L. Motard, and P. Winter. *Process Flowsheeting*. Cambridge University Press, 1979.

[61] A. W. Westerberg, P. C. Piela, R. D. McKelvey, and T. G. Epperly. The ascend modeling environment and its implications. In *The 4th International Process Systems Engineering Symposium*, Ottawa, July 1991.

[62] K. M. Westerberg. Development of software for solving systems of linear equations. Technical Report EDRC 05-35-89, Engineering Design Research Center, Carnegie Mellon University, 1989.

[63] K. M. Westerberg. Development of software for solving systems of nonlinear equations. Technical Report EDRC 05-36-89, Engineering Design Research Center, Carnegie Mellon University, 1989.

[64] P. Wolfe. A method of conjugate subgradients for minimizing nondifferentiable functions. *Mathematical Programming Study*, 3:145–173, 1975.

[65] J. J. Zaher. Graphical representations of the ascend modeling language. In *12th Annual ChEGSA Symposium, Carnegie Mellon University*, Pittsburgh, October 1990.

[66] J. J. Zaher. Developing reusable model libraries in the ascend modeling environment. Technical Report EDRC 06-18-91, Engineering Design Research Center, Carnegie Mellon University, July 1991.