NTNU – Trondheim
Norwegian University of
Science and Technology

Department of Chemical Engineering

Specialisation Project

Modeling and Optimization of Multistream Heat Exchangers with Phase Transition

Written by:

Axel Lødemel Holene

Supervisors:

Sigurd Skogestad
Johannes Jäsche
Vladimiros L. Minasidis

January 16, 2013

Abstract

This project has been carried out at the Norwegian University of Science and Technology during the autumn of 2012 and a few weeks in January 2013.

An equation oriented model for a multistream heat exchanger with phase transitions (MHEX) produced by Kamath et al. [Ref. 14]. They provide a "motivating example MHEX" solved in `GAMS`, using the mixed integer nonlinear programming solver `CONOPT`. A process scenario including inputs, outputs, disturbances and measurment error and noise has been constructed with the aim at finding possible output candidates for self-optimizing control. The optimal sensitivity, $\mathbf{F}$, output gain matrix, $\mathbf{G}^y$, and the Hessian of the cost function, $\mathbf{J}_{uu}$ has been attempted found. Improved values for the objective function was found, suggesting that more research should be done to the model. The programming languages `Python`, MATLAB and `make` has been utilized in addition to `GAMS`. The use has been documented, and the use of `GAMS` for general purposes has been exemplified.

Acknowledgments

Much appreciation is directed to Johanns Jäsche for his availability as my everyday supervisor, even though he actually has been on both vacation and "daddy leave of absence". I'm also very grateful of my head supervisor, Sigurd Skogestad, and his flexibility in relation to deadlines. General thanks are also directed to Tore Haug Warberg for being at work during New Years, for his help in programming issues and general guidance and company.

Contents

List of Figures

List of Tables

1 Introduction

Heat exchangers have existed since the dawn of life. For elephants, its ears are essential for the cooling of its blood. In humans, the nasal passages serve as a heat exchanger, warming and cooling the air that is inhaled and exhaled. In the chemical process industry, the transfer of heat to and from process fluids is essential, and is generally done in heat exchangers. In order to minimize both capital and operating expenses it is desired to utilize the heat as efficiently as possible. Scientific research aims at revealing increasingly efficient ways of designing and operating heat exchangers. Industrial sites often have huge heat exchanger networks with several utility flows. Being able to reduce the number of heat exchangers by applying multistream heat exchangers can improve both the capital and operating costs of chemical process plants.

A multistream heat exchanger (MHEX) is a single process unit in which multiple hot and cold streams exchange heat simultaneously. They are common in cryogenic applications where heat transfer equipment need to be kept compact and well insulated, while recovering heat from streams at very small temperature driving forces [Ref. 14]. Natural gas is composed of mainly methane. When cooled to approximately 111 K (−162 °C), it liquefies and has a volume of approximately 1/600 of gas at room temperature [Ref. 18]. In the production of liquefied natural gas (LNG), the gas undergoes phase transitions. As a consequence, the heat transfer coefficient will change throughout the heat exchanger and a simple logarithmic mean temperature difference approach will not be applicable.

Another important issue concerns heat integration. Pinch technology techniques are common, and are easy to implement if the stream temperatures and flow rates are known. Nevertheless, at phase transition points the nonlinear variations in heat capacity flow rates require an optimization method that can handle nonlinearities.

Kamath et al. [Ref. 14] report that a general high level targeting model that focuses on process optimization while handling phase transitions in the heat exchanger is missing. They have proposed an equation-oriented process model for MHEX's, with a special emphasis on handling phase transitions. The model is based on the work done by Duran and Grossmann [Ref. 11] on optimal heat integration of heat recovery networks.

An example process formulated in the article by Kamath et al. has been investigated. A `GAMS` model file has been acquired and used as a basis to prepare a process scenario with disturbances and measurement errors. The scenario has been used to perform an analysis of potential self-optimizing variables. The programming languages `GAMS`, `Python`, MATLAB and `make` has been used to enable the analysis.

The optimization problem is highly non-convex. The optimal solution found by Kamath et al. was reproduced, but a positive semi-definite Hessian of the cost function was not obtained. Better solutions was found for perturbations up to $2 \cdot 10^{-3}$ in the pump work applied to the nitrogen cooling flow prior to entering the MHEX. Further investigations should be made.

2 Process Description

2.1 Heat Exchanger Model

In this project the "motivating example MHEX", given in the paper by Kamath et al. [Ref. 14], is investigated. The MHEX model is written in `GAMS` (General Algebraic Modeling System). A schematic illustration of the example MHEX is given in Figure 2.1.

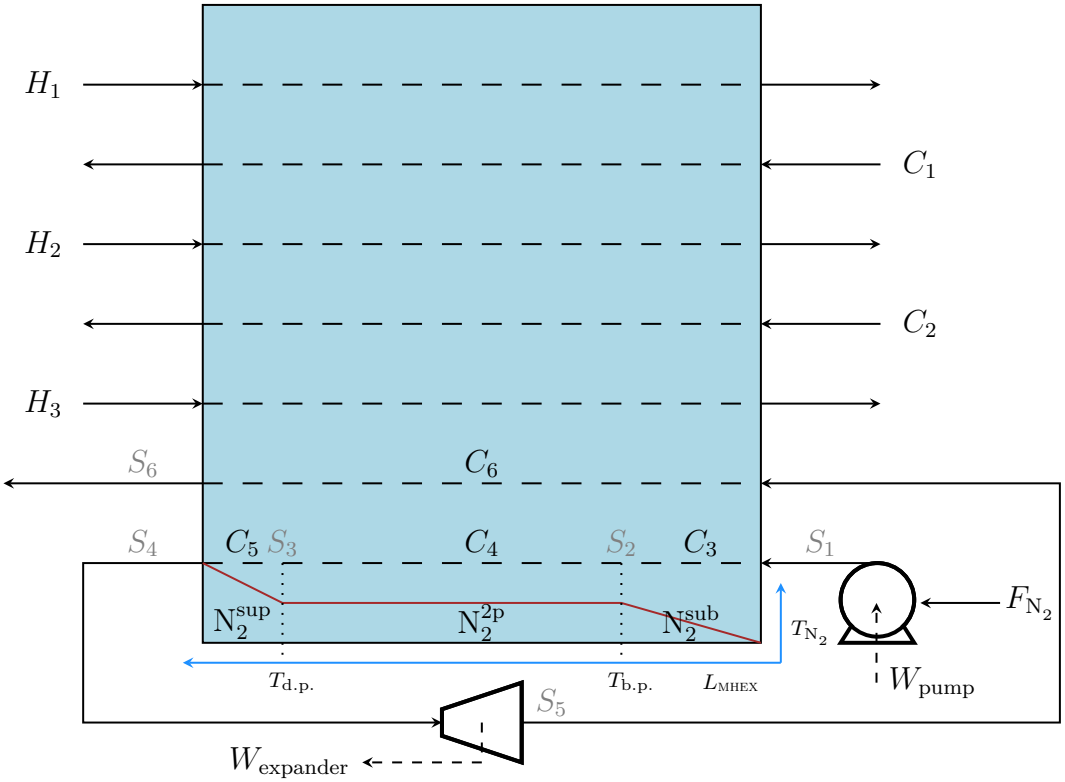

Figure 2.1: A schematic representation of the example MHEX with three hot and two cold flows, and a double loop nitrogen cooling flow.

Here, H_1, H_2, H_3, C_1 and C_2 are the hot and cold process streams, C_3, C_4, C_5 and C_6 are decompositions of the nitrogen cooling stream, $S_i, i \in [1, \ldots, 6]$ are states in the cooling cycle, as described in Table 2.1. N_2^{sub}, N_2^{2p} and N_2^{sup} are subcooled, two-phase and superheated phases of the nitrogen flow, respectively. $T_{d.p.}$ and $T_{b.p.}$ are dew point and bubble point temperatures of the nitrogen stream. The coordinate system (T_{N_2}, L_{MHEX}), displayed in blue, is introduced to illustrate the red temperature curve of the nitrogen flow in the first pass through the MHEX. The heat capacity of the flows are assumed piecewise constant.

Table 2.1: Defined states of the nitrogen cooling stream.

State	Description
S_1	After the pump, entering the MHEX.
S_2	Within the MHEX, at bubble point.
S_3	Within the MHEX, at dew point.
S_4	At the exit of the MHEX, superheated, entering the expander.
S_5	At the exit of the expander, entering the MHEX again.
S_6	Final exit from the MHEX.

Flowrates, pressures and temperatures of the process streams $H_1, \ldots, C_2$ are governed by an external process, and are assumed fixed for the problem. The data for the example process is given in Table 2.2. By pinch analysis it has been revealed that additional cooling is necessary. This is provided by the liquid nitrogen cooling flow F_{N_2}, which is assumed available at 6 bar and 95 K. The structural configuration with two passes through the heat exchanger has been determined from the concept of pressure-based energy combined with pinch analysis [Ref. 6]. Work is recovered in the expander coincidental with cooling the nitrogen gas flow.

Table 2.2: Temperature and heat transport data for the hot and cold process streams.

Stream	T_{in} [K]	T_{out} [K]	F_{Cp} [kW K^{-1}]
H_1	298	250	3.0
H_2	265	180	4.0
H_3	195	150	2.0
C_1	220	245	3.0
C_2	255	280	3.5

2.2 Process Scenario

An approximately realistic process scenario has to be simulated to produce the data necessary for control evaluation purposes. In this chapter the inputs, outputs and disturbances for the process are defined. The process scenario is based on physical relevance and the variables that are possible to display from the GAMS model (See Chapter 4.2). For instance, the temperatures T_{S_2} and T_{S_3} are available from the mathematical model, but it's not likely that temperature measurements from inside the heat exchanger are available.

2.2.1 Inputs

Possible inputs for the process are theoretically the available degrees of freedom for the optimization problem. Kamath et al. [Ref. 14] report that there are five degrees of freedom, namely the:

1. nitrogen feed flow rate (F_{N_2})
2. discharge pressure of the pump (p_{S_1}, `PresS1` in the `GAMS` model file)
3. temperature at the exit of the first pass through the MHEX (T_{S_4},`TempS4` in the `GAMS` model file)
4. pressure at the exit of the expander (p_{S_5}, `PresS5` in the `GAMS` model file)
5. temperature at the exit of the second pass through the MHEX (T_{S_6}, `TempS6` in the `GAMS` model file)

In the model file `mhexcs1v9.gms` the expander outlet pressure, `PresS5`, has been given a target value, thus removing it as a degree of freedom. The optimal bubble point and dew point are found by solving a thermodynamic minimization problem simultaneously as the heat integration and optimization is carried out[1]. The fixation of the expander pressure results in the minimization problem confiscating two more degrees of freedom. This is because the pressure drop over the expander defines the temperature T_{S_5}. As a result, the number of available degrees of freedom is reduced to two. From a process control point of view, interesting variables are degrees of freedom that can readily be controlled, such as F_{N_2} and p_{S_1}.

As the pressure p_{S_1} is a one-to-one function of the amount of work performed by the pump, using W_{pump} as a degree of freedom is equivalent to using p_{S_1}. The final input vector, $\mathbf{u}$, is thus given in Equation (2.1).

$$\mathbf{u} = \begin{bmatrix} F_{\mathrm{N}_2} & W_{\mathrm{pump}} \end{bmatrix}^T \tag{2.1}$$

2.2.2 Outputs

The output variables of the process are those that can be measured. It is assumed that temperature sensors are available at S_4, S_5 and S_6, as well as a flow meter at the feed and a pressure sensor at S_1. The pressure p_{S_5} is disregarded because of its fixed target value. Likewise with temperature T_{S_1}, as it is given from feed

[1] This is equivalent to calculating vapour-liquid equilibrium (VLE) relations, see Chapter 3.3.

specifications and the pressure p_{S_1}. The pressure at the states S_4 and S_6 are not available from the model. The output vector, $\mathbf{y}$, is given in Equation (2.2).

$$\mathbf{y} = \begin{bmatrix} F_{\mathrm{N}_2} & W_{\mathrm{pump}} & p_{S_1} & T_{S_4} & T_{S_5} & T_{S_6} \end{bmatrix}^T \tag{2.2}$$

2.2.3 Disturbances

The process streams are prone to disturbances. It is assumed that the temperatures T_i^{in}, $i \in \{H_1, \ldots, C_2\}$, the heat capacity of the streams, and thus the heat capacity flow rates $F_{Cp,i}$, $i \in \{H_1, \ldots, C_2\}$, may experience unmeasured variations. The magnitude of the disturbances is assumed to be within 10 K for temperatures and 5.0 % for heat capacity flow rates. The disturbance vector, $\mathbf{d}$, and the magnitude scaling matrix, $\mathbf{W}_d$, are given in Equation (2.3) and (2.4), respectively.

$$\begin{aligned} \mathbf{d} = \Big[& \Delta T_{H_1} \quad \Delta T_{H_2} \quad \Delta T_{H_3} \quad \Delta T_{C_1} \quad \Delta T_{H_1} \quad \ldots \\ & \ldots \quad \Delta F_{Cp,H_1} \quad \Delta F_{Cp,H_2} \quad \Delta F_{Cp,H_3} \quad \Delta F_{Cp,C_1} \quad \Delta F_{Cp,C_2} \Big]^T \end{aligned} \tag{2.3}$$

$$\mathbf{W}_d = \mathrm{diag}\left(\begin{bmatrix} 10 & \ldots & 10 & 5.0 F_{Cp,H_1} & \ldots & 5.0 F_{Cp,C2} \end{bmatrix} \right) \tag{2.4}$$

2.2.4 Measurement Error

Measurement errors have to be included. In Chapter 2.2.2, sensor locations was declared. The measurement error vector, $\mathbf{n}^y$, is given in Equation (2.5). It has been assumed that the temperature and pressure sensors experience errors of 1 K and 0.1 bar, respectively, while the flow meter measuring F_{N_2} may experience errors of $10^{-3}\,\mathrm{kmol\,s^{-1}}$. The pump work, W_{pump}, is assumed to have no error, as power indicators are very precise. The measurement magnitude diagonal matrix, $\mathbf{W}_n^y$, is given in Equation (2.6).

$$\mathbf{n}^y = \begin{bmatrix} F_{\mathrm{N}_2} & W_{\mathrm{pump}} & p_{S_1} & T_{S_4} & T_{S_5} & T_{S_6} \end{bmatrix}^T \tag{2.5}$$

$$\mathbf{W}_n^y = \mathrm{diag}\left(\begin{bmatrix} 10^{-3} & 0 & 0.1 & 1 & 1 & 1 \end{bmatrix} \right) \tag{2.6}$$

3 Theory

3.1 Constrained Optimization

Important goals in chemical process control is to meet standards of robustness and optimize performance of operation without violating neither design nor process specified constraints. Chemical process optimization usually aims at minimizing costs and maximizing throughput without violating the mentioned constraints [Ref. 1].

When optimizing a process or system, it is necessary to identify the system's objective, variables and constraints to be able to formulate an optimization problem. This process is known as modeling. A good model describes the behaviour of a system in an adequate manner, without being neither too simple nor too complex.

When a model has been established, an optimization algorithm can be used to determine an optimal solution. The optimal values for the variables has been discovered when the objective function reaches its minimum within the specified constraints.

Mathematically, an optimization problem can be formulated as given in Equation (3.1).

$$\min_{\mathbf{x}\in\mathbb{R}^n} f(\mathbf{x}) \quad \text{subject to} \quad \begin{array}{ll} c_i(\mathbf{x}) = 0, & i \in \mathcal{E} \\ c_i(\mathbf{x}) \geq 0, & i \in \mathcal{I} \end{array} \tag{3.1}$$

The vector $\mathbf{x}$ represent variables, f is the objective function, c represents constraint functions on a subset of $\mathbb{R}^n$, and $\mathcal{E}$ and $\mathcal{I}$ are sets of indices for equality and inequality constraints, respectively. Many optimization algorithms locate only some local optimum. For many optimization problems the global optimum is difficult to both recognize and locate. This is not the case for convex programming problems. However, the main optimization problems solved in this project are not convex and no more focus will be given to the properties of convex programming problems.

To ensure that a given solution, $\mathbf{x}^\star$, is a local minimizer, the Karush-Kuhn-Tucker (KKT) first order necessary conditions must be fulfilled. The KKT conditions are given in Equation (3.2):

$$\nabla_{\mathbf{x}}\mathcal{L}(\mathbf{x}^\star, \lambda^\star) = 0, \tag{3.2a}$$
$$c_i(\mathbf{x}^\star) = 0 \quad \forall\, i \in \mathcal{E}, \tag{3.2b}$$
$$c_i(\mathbf{x}^\star) \geq 0 \quad \forall\, i \in \mathcal{I}, \tag{3.2c}$$
$$\lambda_i^\star \geq 0 \quad \forall\, i \in \mathcal{I}, \tag{3.2d}$$
$$\lambda_i^\star c_i(\mathbf{x}^\star) = 0 \quad \forall\, i \in \mathcal{E} \cup \mathcal{I}, \tag{3.2e}$$

where the Lagrangian function for the general problem in Equation (3.1) is defined in Equation (3.3).

$$\mathcal{L}(\mathbf{x}, \lambda) \triangleq f(\mathbf{x}) - \sum_{i \in \mathcal{E} \cup \mathcal{I}} \lambda_i c_i(\mathbf{x}) \tag{3.3}$$

The active set of any feasible $\mathbf{x}$, denoted $\mathcal{A}(\mathbf{x})$, consists of equality constraint indices and inequality constraints indices for which $c_i(\mathbf{x}) = 0$, that is $\mathcal{A}(\mathbf{x}) \triangleq \mathcal{E} \cup \{i \in \mathcal{I} \,|\, c_i(\mathbf{x}) = 0\}$. Given a local solution $\mathbf{x}^\star$ the Mangarian-Fromovitz constraint qualification (MFCQ) is defined by linear independence of the equality constraint gradients and the existence of a search direction[2], $\mathbf{d}$, such that $\nabla c_i(\mathbf{x}^\star)^T \mathbf{d} \leq 0$, $\forall\, i \in \mathcal{A}(\mathbf{x})$. The linear inequality constraint qualification (LICQ) is said to hold if the set of active constraint gradients $\{\nabla c_i(\mathbf{x}^\star),\, i \in \mathcal{A}(\mathbf{x}^\star)\}$ is linearly independent. MFCQ is always satisfied if LICQ holds. Fulfilled MFCQ also leads to bounded, but not necessarily unique, Lagrangian multipliers. LICQ ensures that the KKT conditions are satisfied for a solution $\mathbf{x}^\star$ [Ref. 9, 20].

Both MFCQ and LICQ link the KKT conditions to the limiting direction $\mathbf{d}$ of a feasible sequence leading to $\mathbf{x}^\star$. In the article by Kamath et al. [Ref. 14] the temperature disjunctions caused by the handling of phase transitions are reformulated and solved as an inner minimization problem with complementarity constraints. Thus it becomes appropriate to introduce a discrete/continuous formulation known as a mixed integer nonlinear optimization problem with complementarity constraints, or more commonly a mathematical program with complementarity constraints (MPCC) [Ref. 9]. The MPCC problem formulation is given in Equation (3.4).

[2]Line search methods will not be covered in this paper, but are extensively documented, e.g. [Ref. 9, 20].

$$\begin{aligned} \min \quad & f(\mathbf{x}, \mathbf{y}, \mathbf{z}) && \text{(3.4a)} \\ \text{s.t.} \quad & h(\mathbf{x}, \mathbf{y}, \mathbf{z}) = 0, && \text{(3.4b)} \\ & g(\mathbf{x}, \mathbf{y}, \mathbf{z}) \leq 0, && \text{(3.4c)} \\ & 0 \leq \mathbf{x} \perp \mathbf{y} \geq 0, && \text{(3.4d)} \end{aligned}$$

where $\perp$ is the complementarity operator enforcing at least one of the bounds to be active, $\mathbf{x}$ and $\mathbf{y}$ are complementarity variables, while $\mathbf{z}$ holds the remaining variables. As done in the `GAMS` model given by Kamath et al., the complementarity constraints can be written in several equivalent ways, e.g. $\mathbf{x}^T\mathbf{y} = 0$, $\mathbf{x} \geq 0$, $\mathbf{y} \geq 0$, which is useful when applying nonlinear programming (NLP) solution strategies [Ref. 9].

3.2 Self-optimizing Control

The principle of self-optimizing control relies upon the existence of self-optimizing variables. Such variables possesses the property that, when kept constant, they achieve acceptable loss without having to be re-optimized every time the process experience a disturbance. The idea was first introduced by Morari in 1980 [Ref. 19], and has been developed by, amongst others, Skogestad and Postlethwaite [Ref. 25].

For the self-optimization theory presented by Skogestad and Postlethwaite some assumptions has to be presented.

1. The loss is defined as $L \triangleq J(\mathbf{u}, \mathbf{d}) - J_{\text{opt}}(\mathbf{d})$.
2. The cost function J is smooth.
3. The optimization problem is unconstrained, "active constraint control" is assumed.
4. Only steady-state control and optimization is considered.
5. The number of degrees of freedom equals the number of variables that is controlled.

A block diagram illustrating the concept is given in Figure 3.1. Here, $\mathbf{c} + \mathbf{n}$ is the linear combination selection of self-optimizing, controlled variables including affecting noise, $\mathbf{c}_s$ denotes the set points for $\mathbf{c}$ given from the optimization, $\mathbf{u}$ is the process inputs, $\mathbf{d}$ is the disturbances, $\mathbf{n}^y$ is the measurement error and noise, $\mathbf{y}$ is the process output while $\mathbf{y}_m$ represents the measured variables, and $\mathbf{H}$

is the measurement combination matrix. The consequence of choosing different self-optimizing variables is illustrated in Figure 3.2.

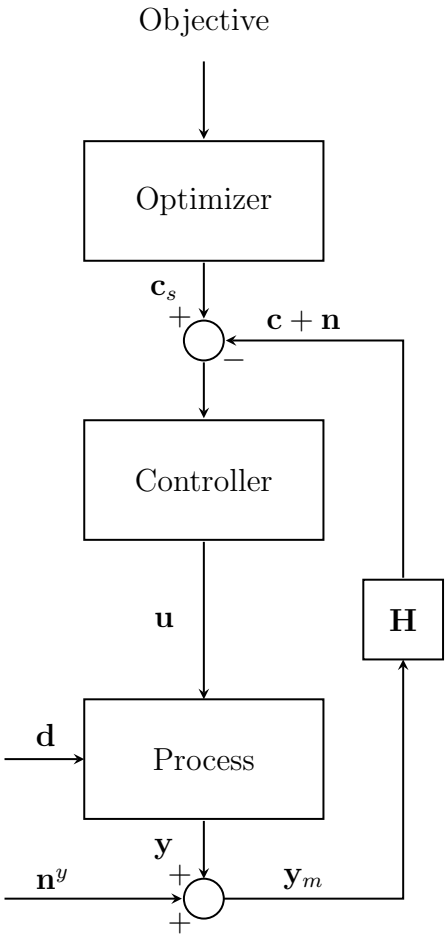

Figure 3.1: Block diagram illustrating the principle of self-optimization.

Plantwide Control In 2004, Skogestad introduced a procedure for plantwide control [Ref. 24] which summarizes to seven points:

1. Definition of the operational objective.
2. Identification of degrees of freedom and optimization of operation for expected disturbances.
3. Implementation of optimal operation.
4. Location of the throughput manipulator.
5. Implementation of regulatory control.
6. Implementation of supervisory control.
7. Implementation to real-time optimization.

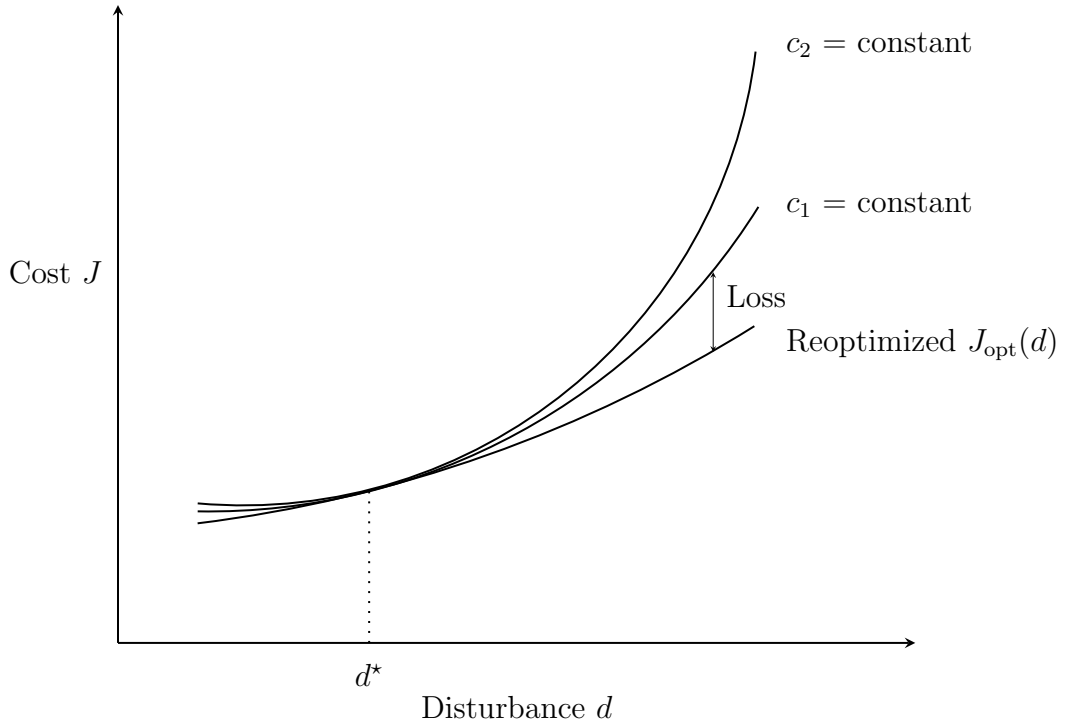

Figure 3.2: A schematic illustration of how choosing different self-optimizing variables affect the loss.

The focus of this paper is on the first three points in the procedure.

3.2.1 Operational Objective

The operational objective of the process is given by the feed cost of liquid nitrogen and the cost of electricity. The cost function, J, is given in Equation (3.5):

$$J = \frac{F_{\text{N}_2} M_{\text{N}_2} C_{\text{N}_2} \cdot 10^3}{\rho_{\text{N}_2} \cdot 10^2} + \frac{C_e (W_{\text{pump}} - W_{\text{expander}})}{3600 \cdot 10^2} \tag{3.5}$$

where F_{N_2} is the feed flow of liquid nitrogen in $\text{kmol}\,\text{s}^{-1}$, M_{N_2} is the molecular weight of nitrogen in $\text{kg}\,\text{kmol}^{-1}$, C_{N_2} is the specific cost of liquid nitrogen in ¢/L, ρ_{N_2} is the density of nitrogen in $\text{kg}\,\text{m}^{-3}$, C_e is the cost of electricity in ¢/kWh, W_{pump} is the pump work in kW and W_{expander} is the expander work in kW. The final dimension of J is \$/s.

3.2.2 Degree of Freedom Analysis

The expression for steady state degrees of freedom available for optimization, N_{opt}, is given in Equation (3.6):

$$N_{\text{opt}} = N_m - N_0 = N_m - (N_{m0} + N_{y0}) \tag{3.6}$$

where N_{m0} is the number of manipulated input variables with no effect on J, N_{y0} is the number of controlled output variables with no effect on J and N_m is the number of control degrees of freedom. J often depends on steady-state only. Active constraint degrees of freedom is disregarded as such variables are better off fixed at its constraint limits [Ref. 23].

The argumentation behind the number of degrees of freedom is found in Chapter 2.2.1. For this process $N_{\text{opt}} = 2$, being the feed flow rate of liquid nitrogen and the pressure of the feed flow before entering the heat exchanger. Thus the feed flow rate, F_{N_2}, and the pump work, W_{pump}, are the optimization degrees of freedom.

3.2.3 Implementation of Optimal Operation

The ideal solution for self-optimizing variables is the gradient of the Lagrangian of the cost function set to zero. Providing measurements that makes it possible to readily calculate the gradient is often very difficult. Figure 3.2 illustrates how the choice of controlled variables, $\mathbf{c}$, affects the loss. There has been developed approaches for selecting $\mathbf{c}$, which render to determining the measurement combination matrix $\mathbf{H}$.

The Null Space Method In 2007, Alstad and Skogestad [Ref. 3] presented an easy way to compute $\mathbf{H}$. The null space method yields locally optimal controlled variables $\mathbf{c}$. The method requires at least as many measurements as unconstrained degrees of freedom, disturbances included, and neglecting implementation error. $\mathbf{H}$ is then in the left null space of the sensitivity matrix $\mathbf{F}$, where $\mathbf{F} = \partial \mathbf{y}_{\text{opt}} / \partial \mathbf{d}^T$, such that $\mathbf{H} \in \text{Null}(\mathbf{F}^T)$.

The Exact Local Method Halvorsen et al. [Ref. 13] found the following relations by elimination of the model states using the model equations and active constraints: $\mathbf{c} = f_c(\mathbf{u}, \mathbf{d})$, $\mathbf{y} = f_y(\mathbf{u}, \mathbf{d})$ and $\mathbf{c} = h(\mathbf{y})$. These relations were linearized around a nominal point, denoted '*', given in Equation (3.7):

$$\Delta \mathbf{c} = \mathbf{G}\Delta \mathbf{u} + \mathbf{G}_d \Delta \mathbf{d} \tag{3.7a}$$
$$\Delta \mathbf{y} = \mathbf{G}^y \Delta \mathbf{u} + \mathbf{G}_d^y \Delta \mathbf{d} \tag{3.7b}$$
$$\Delta \mathbf{c} = \mathbf{H}\Delta \mathbf{y} \tag{3.7c}$$

where $\Delta \mathbf{u} \triangleq \mathbf{u} - \mathbf{u}^\star$, $\Delta \mathbf{d} \triangleq \mathbf{d} - \mathbf{d}^\star$, $\Delta \mathbf{c} \triangleq \mathbf{c} - \mathbf{c}^\star$, $\mathbf{G} \triangleq (\partial f_c / \partial \mathbf{u}^T)^\star$, $\mathbf{G}_d \triangleq (\partial f_c / \partial \mathbf{d}^T)^\star$, $\mathbf{G}^y \triangleq (\partial f_y / \partial \mathbf{u}^T)^\star$, $\mathbf{G}_d^y \triangleq (\partial f_y / \partial \mathbf{d}^T)^\star$, and $\mathbf{H} \triangleq (\partial h / \partial \mathbf{y}^T)$. By second order Taylor series expansion of the cost function around the nominal point $(\mathbf{u}^\star, \mathbf{d}^\star)$, Halvorsen et al. [Ref. 13] shows that the loss L is given from Equation (3.8).

$$L = \frac{1}{2} (\mathbf{u} - \mathbf{u}_{\text{opt}}) \mathbf{J}_{uu} (\mathbf{u} - \mathbf{u}_{\text{opt}}) \triangleq \frac{1}{2} \mathbf{z}^T \mathbf{z} \tag{3.8}$$

Here $\mathbf{z} \triangleq \mathbf{J}_{uu}^{1/2} (\mathbf{u} - \mathbf{u}_{\text{opt}})$ are the loss variables and $\mathbf{J}_{uu} = \partial^2 J / \partial \mathbf{u}^T \mathbf{u}$ is the Hessian of the cost function with respect to inputs. The average loss function is from Kariwala et al. [Ref. 15] given by Equation (3.9):

$$L_{\text{avg}} = \frac{1}{6(n_y + n_d)} \left\| \mathbf{J}_{uu}^{1/2} (\mathbf{H}\mathbf{G}^y)^{-1} \mathbf{H}\mathbf{Y} \right\|_F^2 \tag{3.9}$$

where n_y and n_d are the number of inputs and disturbances, $\mathbf{Y} = \begin{bmatrix} \mathbf{F}\mathbf{W}_d & \mathbf{W}_n^y \end{bmatrix}$ and $\mathbf{F} = \mathbf{G}^y \mathbf{J}_{uu}^{-1} \mathbf{J}_{ud} - \mathbf{G}_d^y$. The subscript F denotes the Frobenius norm of the expression. $\mathbf{W}_d$ represent the expected magnitudes of the individual disturbances and $\mathbf{W}_n^y$ represent the magnitude of the measurement noise associated with each of the candidate measurements. It is assumed that the implementation error for $\mathbf{c}$ is caused only by the measurement error such that $\mathbf{n} = \mathbf{H}\mathbf{n}^y$. The loss may also be computed as given by Umar [Ref. 26] in Equation (3.10):

$$L_{\text{avg}} = \frac{1}{6(n_y + n_d)} \text{trace}(\mathbf{P}) \tag{3.10}$$

where $\mathbf{P} = \left(\mathbf{J}_{uu}^{-0.5} (\mathbf{G}^y)^T (\mathbf{Y}\mathbf{Y}^T)^{-1} \mathbf{G}^y \mathbf{J}_{uu}^{-0.5} \right)^{-1}$.

Introducing $\Delta \mathbf{d}$ and $\mathbf{n}^y$ in Equations (3.11) and (3.12):

$$\Delta \mathbf{d} = \mathbf{W}_d \mathbf{d}' \tag{3.11}$$
$$\mathbf{n}^y = \mathbf{W}_n^y \mathbf{n}^{y\prime} \tag{3.12}$$

where $\mathbf{d}'$ and $\mathbf{n}^{y\prime}$ satisfies $\| \begin{bmatrix} \mathbf{d}' & \mathbf{n}^{y\prime} \end{bmatrix}^T \|_\infty \leq 1$. The arguments for using the chosen norm are given in Kariwala [Ref. 15].

Finding the optimal $\mathbf{H}$ implies to minimize the loss L_{avg}, that is $\min_{\mathbf{H}} L_{\text{avg}}$ [Ref. 13]. Alstad et al. [Ref. 4] showed in 2009 that this in fact is a convex optimization problem, which can be formulated as in Equation (3.13).

$$\min_{\mathbf{H}} \quad \|\mathbf{HY}\|_{\mathbf{F}} \tag{3.13a}$$

$$\text{s.t.} \quad \mathbf{HG}^y = \mathbf{J}_{uu}^{1/2} \tag{3.13b}$$

Equation (3.13) was further simplified by Yelchuru et al. [Ref. 28] yielding the mixed integer quadratic program (MIQP) in Equation (3.14).

$$\min_{\mathbf{H}} \quad \|\mathbf{HY_F}\| \tag{3.14a}$$

$$\text{s.t.} \quad \mathbf{HG}^y = \mathbf{Q} \tag{3.14b}$$

Kariwala et al. [Ref. 15] and Yelchuru et al. [Ref. 28] shows that the optimal $\mathbf{H}$-matrix is non-unique: $\mathbf{H}'_{\text{opt}} = \mathbf{QH}_{\text{opt}}$. By choosing $\mathbf{Q} = ((\mathbf{G}^y)^T(\mathbf{YY}^T)^{-1}\mathbf{G}^y)\mathbf{J}_{uu}^{1/2}$, $\mathbf{H}_{\text{opt}}$ can be expressed as given in Equation (3.15).

$$\mathbf{H}_{\text{opt}} = (\mathbf{G}^y)^T \left(\mathbf{YY}^T\right)^{-1} \tag{3.15}$$

3.2.4 Throughput Manipulator

The throughput manipulator (TPM) is defined as a degree of freedom that affects the network flow. It is not directly or indirectly determined by the control of the individual units, including their inventory control [Ref. 5]. The TMP is traditionally located at the feed of the process, but it could be set at the process bottleneck for maximum production with small back-off. It is a link between the economic objective and the stabilization of the plant.

For this process the throughput manipulator is defined from the cold utility necessary to meet temperature targets.

3.2.5 Regulatory Layer

The regulatory layer is the control layer used to stabilize the plant, in terms of stabilizing mathematically unstable modes as well as slow modes who tend to drift due to disturbances. The layer should usually be of low complexity, i. e. using a simple, decentralized control structure of single-input single-output (SISO) PI control loops. It takes care of local disturbance rejection, active constraint control and tracks setpoint changes from the layer above. This allows for slow control in the supervisory layer [Ref. 24].

No regulatory control layer was applied in this project.

3.2.6 Supervisory Layer

The supervisory layer uses the setpoints for the regulatory layer as degrees of freedom to keep the primary controlled outputs, $\mathbf{c}$, at the optimal setpoints, $\mathbf{c}_s$. This can be done by decentralized or multivariable control [Ref. 24].

No supervisory layer was implemented in this project.

3.2.7 Real-time Optimization

The purpose of real-time optimization (RTO) is to have updated variables for all disturbance occurrences. If the active constraints doesn't change and good self-optimizing variables are obtained, the need for RTO may be reduced or eliminated [Ref. 24].

No real time optimization was used in this project.

3.3 Heat Exhanger Model

Kamath et al. [Ref. 14] states that an MHEX can be modeled the same way as heat exchanger networks. An MHEX only exchanges heat between the streams involved, and the model for an MHEX is equivalent to the following problem statement: "Given an MHEX that does not consume heating or cooling utilities, determine feasible temperatures and heat capacity flowrates for the involved streams" [Ref. 14]. The model of Duran and Grossmann [Ref. 11] was modified and applied for MHEXs, by setting the hot and cold utility loads in their heat integration constraints to zero. Thus the MHEX is treated as an adiabatic device, ensuring that the heat lost from the hot streams matches the heat gained by the

cold streams. The pinch concept enforces maximum heat recovery without violating the minimum driving force criterion. The model *without handling of phase transitions* is given in Equation (3.16).

$$\begin{aligned}
\min \quad & \phi(\mathbf{x}, \mathbf{w}) && \text{(3.16a)}\\
\text{s.t.} \quad & h(\mathbf{x}, \mathbf{w}) = 0, && \text{(3.16b)}\\
& g(\mathbf{x}, \mathbf{w}) \leq 0, && \text{(3.16c)}\\
\Omega(\mathbf{x}) = \sum_{i \in H} F_i \left(T_i^{\text{in}} - T_i^{\text{out}}\right) - \sum_{j \in C} F_j \left(T_j^{\text{in}} - T_j^{\text{out}}\right) = 0, & && \text{(3.16d)}\\
AP_C^p - AP_H^p \leq \epsilon, \quad p \in P & && \text{(3.16e)}\\
AP_H^p = \sum_{i \in H} F_i \Big[\max\{0, T_i^{\text{in}} - T^p\} - \max\{0, T_i^{\text{out}} - T^p\}\Big], \quad p \in P & && \text{(3.16f)}\\
AP_C^p = \sum_{j \in C} F_j \Big[\max\{0, (T_j^{\text{out}} - (T^p - \Delta T_{\min})\} - \max\{0, T_j^{\text{in}} - (T^p - \Delta T_{\min}\}\Big], p \in P & && \text{(3.16g)}
\end{aligned}$$

H and C are sets of indices for hot and cold streams, respectively. The vector $\mathbf{x}$ is given by $\mathbf{x} = \{F_i, T_i^{\text{in}}, T_i^{\text{out}} : \forall\, i \in H;\ F_j, T_j^{\text{in}}, T_j^{\text{out}} : \forall\, j \in C;\}$. The set $P = H \cup C$ is the index of the pinch point candidates whose temperatures are defined by $T^p = T_i^{\text{in}} : \text{for}\, p = i \in H; T^p = (T_j^{\text{in}} + \Delta T_{\min}) : \text{for}\, p = j \in C$. The vector $\mathbf{w}$ represents all the other process parameters and variables that are not associated with heat integration, while $\phi(\mathbf{x}, \mathbf{w})$, $h(\mathbf{x}, \mathbf{w})$ and $g(\mathbf{x}, \mathbf{w})$ represent the objective function, mass and energy balances, design equations and other specifications of the process. The smoothing approximation of Balakrishna and Biegler [Ref. 7], given in Equation (3.17), is used to handle the maximum function.

$$\max\{0, f(\mathbf{x})\} = \frac{1}{2}\Big[(f(\mathbf{x})^2 + \beta^2)^{1/2} + f(\mathbf{x})\Big] \tag{3.17}$$

ϵ (used in Equation (3.16e)) and β are small values used for conditioning of the smooth approximation function, and belong to a well known class of nonlinear complementarity problem (NCP) functions. The values $\beta = 10^{-4}$ and $\epsilon = 10^{-7}$ are reported to work well with the NLP solver CONOPT [Ref. 10].

Phase Transitions When applying pinch analysis it is common to assume the heat capacity to be constant. Alternatively, an average value for the temperature interval may be used [Ref. 22]. This assumption does not hold for nonlinear heat

capacity with respect to temperature, nor when a stream changes phase while exchanging heat. To apply a piecewise linear approximation of the heat capacity on temperature intervals, the dew and bubble points need to be calculated. When performing simultaneous heat integration and optimization, the dew and bubble points will change during the optimization as pressure and compositions are treated as variables. Nor is it known a priori whether phase transitions occurs as the inlet and outlet temperatures of the flows are variables of the optimization problem.

The streams are classified as either capable of undergoing phase transition or not. The former of the two is denoted as parent streams. They are subdivided into substreams corresponding to superheated (sup), two-phase (2p) and subcooled (sub) regions. The parent streams are disregarded and each of the substreams are treated as separate variables. The substreams have corresponding heat load, inlet and outlet temperatures and inherit the flow rate and overall composition of the parent streams. If the pressure drop across the MHEX can be neglected for the parent streams, the substreams also inherit the pressure from their parent streams. The substreams corresponding to all three phases are always present in the model. If a particular phase doesn't exist, the heat load of the substream is set to zero by the disjunctive model for phase detection. The model for phase detection consist of three components: disjunctions for phase detection at the inlet and outlet of the MHEX, flash calculations for the two-phase substreams that integrate with the disjunctions, and enthalpy calculations and heat load evaluation of the substreams. The model is given in Equations (3.18) and (3.19).

$$\begin{bmatrix} Y_{\text{in}}^{V} \\ T_{\text{in}} \geq T_{\text{d.p.}} \\ T_{\text{in}}^{\text{sup}} = T_{\text{in}} \\ T_{\text{in}}^{\text{2p}} = T_{\text{d.p.}} \\ T_{\text{in}}^{\text{sub}} = T_{\text{b.p.}} \end{bmatrix} \vee \begin{bmatrix} Y_{\text{in}}^{VL} \\ T_{\text{b.p.}} \leq T_{\text{in}} \leq T_{\text{d.p.}} \\ T_{\text{in}}^{\text{sup}} = T_{\text{d.p.}} \\ T_{\text{in}}^{\text{2p}} = T_{\text{in}} \\ T_{\text{in}}^{\text{sub}} = T_{\text{b.p.}} \end{bmatrix} \vee \begin{bmatrix} Y_{\text{in}}^{L} \\ T_{\text{in}} \leq T_{\text{b.p.}} \\ T_{\text{in}}^{\text{sup}} = T_{\text{d.p.}} \\ T_{\text{in}}^{\text{2p}} = T_{\text{b.p.}} \\ T_{\text{in}}^{\text{sub}} = T_{\text{in}} \end{bmatrix} \tag{3.18}$$

$$\begin{bmatrix} Y_{\text{out}}^{V} \\ T_{\text{out}} \geq T_{\text{d.p.}} \\ T_{\text{out}}^{\text{sup}} = T_{\text{out}} \\ T_{\text{out}}^{\text{2p}} = T_{\text{d.p.}} \\ T_{\text{out}}^{\text{sub}} = T_{\text{b.p.}} \end{bmatrix} \vee \begin{bmatrix} Y_{\text{out}}^{VL} \\ T_{\text{b.p.}} \leq T_{\text{out}} \leq T_{\text{d.p.}} \\ T_{\text{out}}^{\text{sup}} = T_{\text{d.p.}} \\ T_{\text{out}}^{\text{2p}} = T_{\text{out}} \\ T_{\text{out}}^{\text{sub}} = T_{\text{b.p.}} \end{bmatrix} \vee \begin{bmatrix} Y_{\text{out}}^{L} \\ T_{\text{out}} \leq T_{\text{b.p.}} \\ T_{\text{out}}^{\text{sup}} = T_{\text{d.p.}} \\ T_{\text{out}}^{\text{2p}} = T_{\text{b.p.}} \\ T_{\text{out}}^{\text{sub}} = T_{\text{out}} \end{bmatrix} \tag{3.19}$$

The Y's are boolean variables and V, VL and L are designated to "vapour", "vapour/liquid" and "liquid". T_{in} and T_{out} are inlet and outlet temperatures, $T_{\text{d.p.}}$ and $T_{\text{b.p.}}$ are the dew and bubble point temperatures, all of the parent stream. The

rest of the variables correspond to the temperatures of the substreams as seen in Figure 3.3. The disjunctions are exclusive, thus only one boolean variable can be true. The integration of flash calculations with the disjunctions is relying on the following formulation: "To allow vapour liquid equilibrium (VLE) equations for flash calculation to converge to a feasible solution in case of single phase (vapour or liquid), is to operate it at the boundary of the two-phase region." [Ref. 14] Thus:

a) If only vapour outlet exists, the operating temperature for flash calculations should be equal to the dew point temperature of the inlet stream.

b) If only liquid outlet exists, the operating temperature for flash calculations should be equal to the bubble point temperature of the inlet stream.

This way the flash calculations are forced to operate in single saturated phase at the boundaries of the two-phase region of the parent stream.

The model is illustrated in Figure 3.3, where H_1, H_2, C_1 and C_2 are the physical streams. The streams H_2 and C_1 undergo phase transitions while H_1 and C_2 do not.

Handling streams with small temperature changes For evaporating liquids or condensation of vapour, the streams can have a high heat load while the temperature difference approaches zero. To be able to use well-defined, bounded values for heat capacity flowrates, a number α is assigned as a fictitious temperature drop. The assignments in Equations (3.18) and (3.19) are relaxed. By defining $T_{\text{in}}^{\text{p.c.}}$ and $T_{\text{out}}^{\text{p.c.}}$ from process conditions (p.c.) and the heat integration variables $T_{\text{in}}^{\text{HI}}$ and $T_{\text{out}}^{\text{HI}}$, the disjunctions given in Equation (3.20) are introduced.

$$\begin{bmatrix} \Delta T \geq \alpha \\ T_{\text{in}}^{\text{HI}} = T_{\text{in}}^{\text{p.c.}} \\ T_{\text{out}}^{\text{HI}} = T_{\text{out}}^{\text{p.c.}} \end{bmatrix} \vee \begin{bmatrix} \Delta T \leq \alpha \\ T_{\text{in}}^{\text{HI}} = \dfrac{T_{\text{in}}^{\text{p.c.}} - T_{\text{out}}^{\text{p.c.}}}{2} \pm \dfrac{\alpha}{2} \\ T_{\text{out}}^{\text{HI}} = \dfrac{T_{\text{in}}^{\text{p.c.}} - T_{\text{out}}^{\text{p.c.}}}{2} \pm \dfrac{\alpha}{2} \end{bmatrix} \tag{3.20}$$

The value $\alpha/2$ is added in $T_{\text{in}}^{\text{HI}}$ and subtracted in $T_{\text{out}}^{\text{HI}}$ for hot streams, and opposite for cold streams. To avoid the boolean variables introduced by the use of the disjunction in Equation (3.20), the maximum operator is applied, as given in Equation (3.21), in combination with the smoothing approximation of Balakrishna and Biegler [Ref. 7].

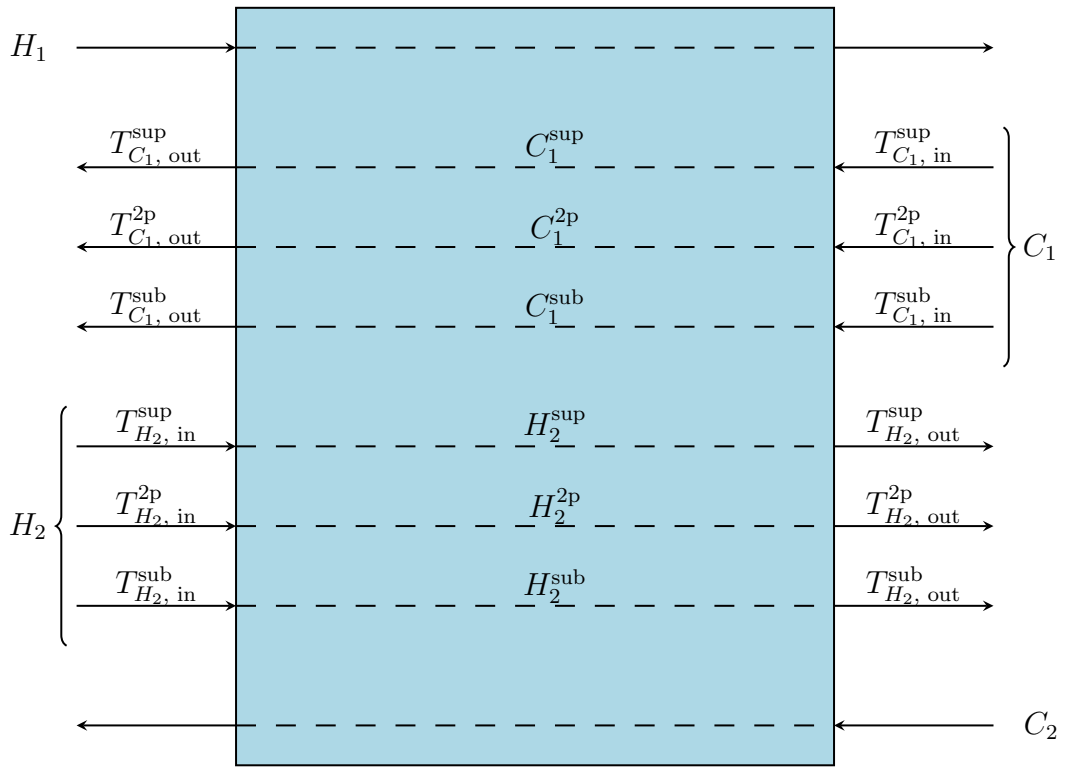

Figure 3.3: Schematic representation of the integrated model for simultaneous optimization and heat integration with phase transitions. H_1 and C_2 are streams wihtout phase transitions while H_2 and C_1 are streams with phase transitions.

$$
\begin{aligned}
T_{\text{in}}^{\text{HI}} &= \max\left(T_{\text{in}}^{\text{p.c.}}, \frac{T_{\text{in}}^{\text{p.c.}} + T_{\text{out}}^{\text{p.c.}} + \alpha}{2}\right) \\
T_{\text{out}}^{\text{HI}} &= \max\left(T_{\text{in}}^{\text{p.c.}}, \frac{T_{\text{in}}^{\text{p.c.}} + T_{\text{out}}^{\text{p.c.}} - \alpha}{2}\right)
\end{aligned}
\tag{3.21}
$$

For the process in this project, the temperature drop is small only for the isothermal stream C_4. Thus only C_4 is handled using the disjunctions in Equation (3.21).

Reformulation of Disjunctions The disjunctions for phase detection are piecewise smooth functions, and can be reformulated as a mixed-integer nonlinear programming (MINLP). By picking the correct function in piecewise smooth domains the disjunctions can be formulated as an NLP problem, avoiding the binary variables. The associated equations are given in Equation (3.22) [Ref. 8].

$$
\min_{y} \quad \sum_{i=1}^{N} (x - a_i)(x - a_{i-1}) y_i \tag{3.22a}
$$

$$
\text{s.t.} \quad \sum_{i=1}^{N} y_i = 1 \tag{3.22b}
$$

$$
y_i \geq 0 \tag{3.22c}
$$

$$
z = \sum_{i=1}^{N} f_i(x) y_i \tag{3.22d}
$$

N is the number of piecewise segments, $f_i(x)$ is the function over the interval $x \in [a_{i-1}, a_i]$, and z represents the value of the piecewise function. This is a linear programming (LP) problem, and can be embedded as an inner problem within an outer optimization problem from the KKT-conditions.

$$
\sum_{i=1}^{N} y_i = 1 \tag{3.23a}
$$

$$
(x - a_i)(x - a_{i-1}) - \mu_i + \lambda = 0 \tag{3.23b}
$$

$$
0 \leq y_i \perp \mu_i \geq 0 \tag{3.23c}
$$

where λ and μ_i are multipliers corresponding to Equations (3.22c) and (3.22d).

The disjunctions given in Equation (3.18) and (3.19) can then be reformulated as given in Equation (3.24). The subscripts "in" and "out" are omitted for convenience.

$$\begin{aligned} \min \quad & -\Big[Y^V(T - T_{\text{d.p.}}) + Y^{VL}(T_{\text{d.p.}} - T)(T - T_{\text{b.p.}}) + Y^L(T_{\text{b.p.}} - T)\Big] && \text{(3.24a)} \\ \text{s.t.} \quad & Y^V + Y^{LV} + Y^L = 1 && \text{(3.24b)} \\ & Y^V \geq 0, Y^{LV} \geq 0, Y^L \geq 0 && \text{(3.24c)} \end{aligned}$$

With the optimality conditions in Equation (3.25)

$$\begin{aligned} Y^V + Y^{LV} + Y^L = 1 && \text{(3.25a)} \\ -(T - T_{\text{d.p.}}) - \mu^V + \lambda = 0 && \text{(3.25b)} \\ -(T_{\text{d.p.}} - T)(T - T_{\text{b.p.}}) - \mu^{VL} + \lambda = 0 && \text{(3.25c)} \\ -(T_{\text{b.p.}} - T) - \mu^L + \lambda = 0 && \text{(3.25d)} \\ 0 \leq Y^V \perp \mu^V \geq && \text{(3.25e)} \\ 0 \leq Y^{VL} \perp \mu^{VL} \geq && \text{(3.25f)} \\ 0 \leq Y^L \perp \mu^L \geq && \text{(3.25g)} \end{aligned}$$

3.4 Finite Difference Approximations

To determine the Hessian matrix, $\mathbf{J}_{uu}$, a central finite difference approximation method was used. The difference equations are given in Equation (3.26).

$$\frac{\partial^2 f}{\partial \mathbf{x}_i^2} = \frac{-f(\mathbf{x} + 2h_i\mathbf{e}_i) + 16f(\mathbf{x} + h_i\mathbf{e}_i) - 30f(\mathbf{x}) + 16f(\mathbf{x} - h_i\mathbf{e}_i) - f(\mathbf{x} - 2h_i\mathbf{e}_i)}{12h_i^2} \tag{3.26a}$$

$$\frac{\partial^2 f}{\partial \mathbf{x}_i \partial \mathbf{x}_j} = \frac{f(\mathbf{x} + h_i\mathbf{e}_i + h_j\mathbf{e}_j) - f(\mathbf{x} + h_i\mathbf{e}_i - h_j\mathbf{e}_j) - f(\mathbf{x} - h_i\mathbf{e}_i + h_j\mathbf{e}_j) + f(\mathbf{x} - h_i\mathbf{e}_i - h_j\mathbf{e}_j)}{4h_ih_j} \tag{3.26b}$$

Here, f is a function, $\mathbf{x}$ the function variables, h represent step lengths and $\mathbf{e}$ are unit vectors [Ref. 2].

4 Programming

4.1 GAMS

The General Algebraic Modeling System (GAMS) is a high-level modeling system for mathematical programming problems. It is designed to provide an algebra based language for the compact representation of large and complex models. It allows unambiguous statements of algebraic relationships, and permits model descriptions that are independent of solution algorithm. GAMS provides in total 41 solvers for linear, nonlinear, mixed integer, mixed integer nonlinear problems and mixed complementarity optimization problems. The program is called from the command line, accepting .gms-files as input, and returns .lst-files as output [Ref. 17, 21].

The GAMS model file used for this project is the "motivating example" given in the paper by Kamath et al. [Ref. 14]. The file mhexcs1v9.gms is provided in Appendix A.

A GAMS program can be extensive and complex. There are, however, good documentation to how a GAMS model can be prepared and solved. Some general remarks should be made [Ref. 21]:

a) A GAMS model is a collection of statements in the GAMS language. The only rule governing the ordering of statements is that the model cannot be referenced before it is declared to exist.

b) GAMS statements may be laid out typographically in almost any style, including multiple lines per statement, embedded blank lines, and multiple statements per line. In mhexcs1v9.gms there are for instance multiple EQUATION environment declarations.

c) The GAMS compiler does not distinguish between uppercase and lowercase letters.

d) Comments may be declared by an asterisk to the beginning of a line, or by the onText/offText commands.

An introduction to some of the scopes used in mhexcs1v9.gms follows.

Input, output and command shell print GAMS accepts .gms input files. The resulting output is written to a .lst-file with filename corresponding to that of the input file. The command shell print is the progress documentation given from the solver. In this case, the solver used was CONOPT, so the printout in the command shell is given by CONOPT, not GAMS itself.

Sets Sets in GAMS correspond exactly to the indices in the algebraic representations of models. Declaration of sets follow the `Set` command. The sets in `mhexcs1v9.gms` are declared as indicated in Listing 1.

```
SETS
Str          "Streams"       /H1,H2,H3,C1,C2,C3,C4,C5,C6/
HtStr(Str)   "Hot streams"   /H1,H2,H3/
CdStr(Str)   "Cold streams"  /C1,C2,C3,C4,C5,C6/

CC1 "Constants for CP of N2" /C1,C2,C3,C4,C5,C6,C7/
;

ALIAS
(Str,Str2,Str3)
;
```

Listing 1: Some of the sets defined in `mhexcs1v9.gms`.

It should be noted that the sets `HtStr` and `CdStr` are subsets of `Str`. `Streams`, `Hot streams` and `Cold streams` describes the sets, and are optional. The elements of the set are defined using slashes '`/.../`'. The `ALIAS` statement defines `Str2` and `Str3` as aliases of `Str`. The semicolons '`;`' close the environments.

Variables Decision variables are declared with a `VARIABLES` statement, and is given a name, a domain (if appropriate) and an optional description. There are different variable types, limited to default free ($-\infty$ to $+\infty$), positive (0 to $+\infty$), negative ($-\infty$ to 0), binary (0 or 1) and integer variables. Examples of the optimization variables in `mhexcs1v9.gms` are given in Listing 2, with the variable `MxInHt` having a domain of two `Str`-sets.

```
VARIABLES
MxInHt(Str,Str2)
FN2              "in kmol/s"
ExpdrWork
;
VARIABLES
Z
;
BINARY VARIABLES
Yloc(Str,Str2) "1 indicates it is above the pinch and contributes,
                while 0 means no contribution"
;
```

Listing 2: Examples of variables defined in `mhexcs1v9.gms`.

Assigning Data Data may be introduced to the model as parameters (lists), tables or by direct assignment (scalars). When declaring parameters, names and domain has to be set. Documentation is optional. Listing 3 shows how the PARAMETERS-environment is used to define heat capacities in the CC1 domain.

```
PARAMETERS
Cpconst(CC1)   "for Cp = kJ/kmol K, T = K"
/C1 = 29.105, C2 = 8.6149, C3 = 1701.6, C4 = 0.10347, C5 = 909.79/
;
```

Listing 3: Example of parameter assignment used in mhexcs1v9.gms.

Using tables for the assignment of data is very efficient for two-dimensional structures. Blank entries in a table are interpreted as zeroes. Data assignment by table has not been used in mhexcs1v9.gms. An example from the transport problem given in Rosenthal's GAMS User Guide [Ref. 21] is presented in Listing 4.

```
Table d(i,j) "distance in thousands of miles"
             new-york    chicago     topeka
seattle      2.5         1.7         1.8
san-diego    2.5         1.8         1.4     ;
```

Listing 4: Illustration of the use of tables in data assignment in a GAMS model

Scalars are used for direct value assignment, that is, a single data entry. Examples of scalars used in mhexcs1v9.gms are given in Listing 5.

```
SCALARS
HRAT   "Heat recovery approach temperature" /4/
epsi   "epsilon for smooth approximation"   /0.000005/
                                            Could also be written 5e-6
;
```

Listing 5: Examples of single data entries, scalars, used in mhexcs1v9.gms.

Equations Equations in GAMS must first be declared and then defined. Declaration of equations is exemplified in Listing 6. Definitions are exemplified in Listing 7. The syntax for defining the content of equations is to write the equation name, including its domain if necessary, followed by the '..'-operator, and then define the equation content. Each equation declaration must be closed with a semicolon, as with the equation environment.

```
EQUATIONS
EqFF2(Str)
EqOmega
EqMxInHt(Str,Str2)
;
```

Listing 6: Example of declaration of equations in mhexcs1v9.gms.

```
EqFF1(Str)$(HtStr(Str)*Nonisoth(Str))..
  FCp(Str)*(Tin(Str) - Tout(Str)) =E= HetLd(Str) ;
EqOmega..
  Omega =E= SUM(Str$(HtStr(Str)), HetLd(Str)) - SUM(Str$(CdStr(Str)),
      HetLd(Str)) ;
EqMxInHt(Str,Str2)$((HtStr(Str)*Nonisoth(Str)))..
  MxInHt(Str,Str2) =E= 0.5*sqrt(Power(Tin(Str)- TPch(Str2),2) + Power
     (epsi,2)) + 0.5*(Tin(Str)-TPch(Str2)) ;
```

Listing 7: Example of equation content definitions in mhexcs1v9.gms.

When defining equations, the operators '=e=', '=g=' and '=l=' must be used to indicate "equals", "greater than" and "less than", respectively. The dollar operator resembles a condensed way of writing an if-statement. For line 1 in Listing 7 the dollar operator statement reads: "EqFF1(Str) is, for all Str in the sets HtStr *and* Nonisoth, given by the following equation: ...". Thus, if the condition in the brackets gives a boolean true-value, the statement is executed, resulting that HetLd (heat load) is only assigned to hot, nonisothermal streams.

Bounds and Starting Points Bounds on variables are defined by the .lo and .up operators. Variables may also be fixed by the .fx operator, which enforces the relations var.fx = var.lo = var.up. The level operator, .l, defines the activity level of the variable. var.l receives new values when a model is solved, and will in a GAMS model indicate the starting value of a variable. Activity levels can also be given from equations. The use of boundary and level statements is illustrated in Listing 8.

```
HetLd.lo(Str) = 0.0 ;
HetLd.up(Str) = 1000.0 ;
Tin.fx('H1')  = 298 ;
Tout.fx('H1') = 250 ;
TempS4.l      = 180 ;
TempS5.l      = TempS4.l*(PresS5/PresS1.l)**((gamma-1)/gamma) ;
```

Listing 8: Example of the use of bounds and activity level operators from mhexcs1v9.gms.

Model and Solve Statement The model statement specifies the model and assigns a name to it. One can pick equations for a model, or make GAMS include all equations in a model. The `solve` statement defines the model that is to be solved, which solver should be used and what the objective of the solver should be. The type of the model (linear, nonlinear, etc.) must be known prior to solving the model. The heat exchanger modeled in `mhexcs1v9.gms` is a mixed integer nonlinear problem (MINLP) where the objective function is named `Z`. The `model` and `solve` statements are given in Listing 9.

```
Model Ravi /ALL/ ;
Solve Ravi using MINLP minizing Z ;
```

Listing 9: The `model` and `solve` statement used in `mhexcs1v9.gms`.

Global Options The GAMS language provides a set of global options which control compiler directives to the input file to control the preferences of the output. Some of the options used in `mhexcs1v9.gms` are displayed in Listing 10.

```
* global options
$offlisting              DCD[3] stops the echo print of the input file
$eolcom //               Defining end of line comments delimiter
$inlinecom /* */         Defining inline comments delimiters
option iterlim = 5e5 ;   Iteration limit (= 5·10^5)
option optcr = 0.0 ;     Relative termination tolerance [Ref. 17]
option decimals = 8 ;    Number of decimals to be printed by the
                         display option
```

Listing 10: Some global options used in `mhexcs1v9.gms`.

4.2 Modifications Done to `mhexcs1v9.gms`

In order to perform the necessary calculations with GAMS some modifications were made to the original script.

4.2.1 Display variables

The `.lst`-output file reports the variables with a limited amount of decimals, namely 4. This proved insufficient for the calculations. A `Display` statement was

[3]Dollar Control Directive [Ref. 21].

included in all GAMS model files after the solve statement. The activity level of each variable at the solution was printed using the command listed in Listing 11.

```
Display FN2.l, PumpWork.l ,PresS1.l, TempS4.l, TempS5.l, TempS6.l,
        Tboil.l, Z.l, YLoc.l, contri.l, FCp.l, Tin.l, Tout.l ;
```

Listing 11: The display statement added to mhexcs1v9.gms.

4.2.2 Sensitivity Matrix, F

To calculate the sensitivity matrix, $\mathbf{F} = \partial \mathbf{y}_{\text{opt}} / \partial \mathbf{d}^T$, disturbances on the inlet temperatures and heat flowrates was simulated by changing the temperature targets separately and re-optimizing. This was done by simply increasing the temperatures individually by 1 K. For heat flowrates, relative disturbances was performed individually on each stream. This was solved as illustrated in Listing 12.

```
SCALAR
dcp     /1.05/
FCp.fx('H1') = 3*dcp ;
FCp.fx('H2') = 4 ;
```

Listing 12: Disturbance simulation on heat flowrates.

4.2.3 Gain Matrix, $\mathbf{G}^y$

To produce the output gain matrix, $\mathbf{G}^y = \partial \mathbf{y} / \partial \mathbf{u}^T$, and the Hessian of the cost function, $\mathbf{J}_{uu}$, perturbations were made around the optimal input, $u^\star$, that is, u_1 was perturbed while u_2 was kept constant and vica verca. To keep the remaining degrees of freedom constant, control equations was introduced. The equations are given in Listing 13.

```
EQUATIONS
EqControl
EqControl2
;
SCALARS
perp1   /1.1e-3/
perp2   /1.1e-3/
;
EqControl..
 FN2 =E= 0.02959005;
* FN2 =E= 0.02959005*(1+perp1);
* FN2 =E= 0.02959005*(1-perp1);
```

```
EqControl2..
 PumpWork =E= 0.17132506;
* PumpWork =E= 0.17132506*(1+perp2);
* PumpWork =E= 0.17132506*(1-perp2);
```

Listing 13: Control equations for keeping degrees of freedom constant.

Removing degrees of freedom resulted in large objective function values, $Z \approx 5 \cdot 10^5 Z_{\text{opt}}$. This problem was solved by changing the intitial point of the calculation, as given in Listing 14.

```
* Optimal initial values
FN2.l    = 0.02959005 ;
PresS1.l = 7.25345316 ;
TempS4.l = 265.7347   ;
TempS6.l = 293.9971   ;
Tboil.l  = 98.999995  ;
YLoc.l('C4','C3') = 0 ;  Changed from 1 in mhexcs1v9.gms
YLoc.l('C4','C6') = 0 ;  Changed from 1 in mhexcs1v9.gms
```

Listing 14: Optimal initial values.

4.3 Python

The input and output files of a GAMS program are text files. The necessary calculations made from the input and output data of the program was carried out utilizing the objective oriented programming language Python version 2.7 [Ref. 12], with software acquired through Macports [Ref. 16].

Some key elements of the script `mhex.py` written to perform the calculations are given in the following section. The complete code is presented in Appendix B.

4.3.1 Reading Data

The input and output file formats of a GAMS model is clean text. The function `readFile` was constructed to read a file and return its content as a string. The function is given in Listing 15.

```
def readFile(folder,filename):
    try:
        with open(folder+'/'+filename, 'r') as f:
            res = f.read()
    except IOError:
```

```
6           print "Error: Can't find file or read data named '%s'" \
7               % filename
8       else:
9           print "File named '%s' from the folder '%s' successfully
                loaded" % (filename, folder)
10          f.close()
11      return res
```

Listing 15: A python function for reading `.gms` and `.lst` files.

4.3.2 Classes

To handle the different problems and solutions in a convenient way, a model class was introduced. `solvedModel` is given in Listing 16. `solvedModel` objects are populated with values for temperatures, heat flowrates, cost function values and so on, from the `.lst` output files from `GAMS`. `solvedModel` calls `readFile` which opens and reads the `.lst` files. The function `findData`, given in Listing 17, uses regular expressions to traverse the `.lst` files and extract values from them. Instantiation of a `solvedModel` object requires two arguments: which subfolder the `.lst`-file is located in and the filename.

```
1   class solvedModel:
2       '''
3       Class specifying the values retrieved from the .list file of
4       the solved .gms model.
5       '''
6       def __init__(self, folder, filename):
7           '''
8           Assigns a name to the model (typically file name),
9           then loads the values from "data".
10          '''
11          data = findData(readFile(folder, filename))
12          self.name = filename
13          self.inputfile = data.get('Inputfile')
14          self.outputfile = data.get('Outputfile')
15          self.FN2 = data.get('FN2')
16          self.PumpWork = data.get('PumpWork')
17          self.Z = data.get('Z')
18          self.Tin = data.get('Tin')
19          self.Tout = data.get('Tout')
20          self.FCp = data.get('FCp')
21          self.PresS1 = data.get('PresS1')
22          self.TempS4 = data.get('TempS4')
23          self.TempS5 = data.get('TempS5')
24          self.TempS6 = data.get('TempS6')
```

```
25      def __repr__(self):
26          return "Class for solved gamsModel"
```

Listing 16: The python class `solvedModel`.

```
1  def findData(aString):
2      oneVar = ['FN2', 'PumpWork', 'PresS1', 'TempS4', 'TempS5', \
3              'TempS6', 'Z']
4      multiVar = ['FCp', 'Tin']
5      streams = ['H1', 'H2', 'H3', 'C1', 'C2']
6      nameList = ['Input', 'Output']
7      ans = AutoVivification()
8      for var in oneVar:
9          try:
10             ans[var] = re.match(r'^(.|\n)*?$\n.*VARIABLE '+var+\
11             '\.L\s*=\s*([0-9.E+-]+)', aString, re.MULTILINE).group(2)
12         except:
13             pass
14     for var in multiVar:
15         for stream in streams:
16             try:
17                 ans[var][stream] = \
18                 re.match(r'^(.|\n)*?$\n.*VARIABLE '+var+\
19                 '\.L\s*\n.*\n.*'+stream+' ([0-9.E+-]+)', aString, \
20                 re.MULTILINE).group(2)
21             except:
22                 pass
23     for aName in nameList:
24         try:
25             ans[aName+'file'] = re.match(r'^(.|\n)*?$\n.*'+aName+\
26             '\s+(\/\S+)+\/(\S+)$', aString, re.MULTILINE).group(3)
27         except:
28             pass
29     return ans
```

Listing 17: Python function for finding model data from `.lst` files.

4.3.3 Sensitivity and Gain Matrices

For computing the sensitivity matrix $\mathbf{F}$, the Python function `sens` accepts a list of `solvedModel` objects, and traverses them to find the data necessary to perform the computations. $\mathbf{F}$ is returned as a list of lists, which in Python corresponds to a matrix. `sens` is given in Listing 18. The gain matrix $\mathbf{G}^y$ is found in a similar way.

```
def sens(aModelList):
    '''
    aModel = solvedModel-object
    y = [FN2, PumpWork, pS1, TS4, TS5, TS6]
    d = [H1, ... , C2, FCp(H1), ... FCp(C2)]
    F = [[dFN2/dTin(H1), dFN2/dTin(H2), ... , dFN2/dFCp(H1), .... ],
        [dPW/dTin(H1), ... , ]]
    finds dFN2 and dPW for each model in the list
    computes dTin(H1) etc. Disregards current iteration and continues
    looping if dTin(H1) is zero. If it's not, continuing to FCp,
    repeating.
    '''
    y = ['FN2', 'PumpWork', 'PresS1', 'TempS4', 'TempS5', 'TempS6']
    subs = ['Tin', 'FCp']
    streams = ['H1', 'H2', 'H3', 'C1', 'C2']
    F = [[] for _ in range(len(y))]
    for aModel in aModelList:
        for y_i in y:
            dy = float(getattr(aModel,y_i)) - float(getattr(ref,y_i))
            for sub in subs:
                for stream in streams:
                    dd = float(getattr(aModel,sub)[stream]) - \
                         float(getattr(ref,sub)[stream])
                    if dd != 0:
                        F[y.index(y_i)].append(dy/dd)
                        break
                    else:
                        continue
    return F
```

Listing 18: The function for computing the sensitivity matrix **F**.

4.3.4 Hessian of Cost

The Hessian matrix of the cost function was determined by having one main function calling two sub-functions, one for the diagonal elements and one for the nondiagonal elements. The main function accepts a nested list of `solvedModel` objects. The `printing` option is used for debugging. When assigned the value `True`, it prints extended information about each matrix element during the calculation of $\mathbf{J}_{uu}$.

4.4 MATLAB

After the computation of the sensitivity, gain and Hessian matrices from the `.lst` output files, a MATLAB script was written to perform further computations. MATLAB was used because it handles matrix calculus in a much more convenient way than Python.

5 Results

The matrices $\mathbf{F}$, $\mathbf{G}^y$ and $\mathbf{J}_{uu}$ was determined by the use of the methods described in Chapter 4.1 and 4.3. The values of the matrices are given in Equation (5.1). It should be noted that in the theory section, analytic expressions for $\mathbf{F}$ is provided. It was considered easier to measure $\mathbf{F}$ directly from simulations.

$$\mathbf{F} = \begin{bmatrix} 9.095 \cdot 10^{-5} & 0.000\,523\,94 & -1.6 \cdot 10^{-7} & 2.3572 & 1.3432 & 0.999\,96 \\ 0.000\,225\,95 & 0.001\,308\,9 & -1.6 \cdot 10^{-7} & 2.29 & 1.3049 & -4.5 \cdot 10^{-5} \\ 0.000\,148\,95 & 0.000\,861\,94 & -1.6 \cdot 10^{-7} & -0.188\,94 & -0.107\,67 & -4.2 \cdot 10^{-5} \\ 0.000\,222\,95 & 0.001\,292\,9 & -1.6 \cdot 10^{-7} & -0.282\,69 & -0.161\,09 & -4.2 \cdot 10^{-5} \\ 0.000\,483\,95 & 0.002\,801\,9 & -1.6 \cdot 10^{-7} & -0.607\,13 & -0.345\,97 & -4.2 \cdot 10^{-5} \\ 0.002\,513 & 0.014\,566 & -1.0667 \cdot 10^{-6} & 34.6638 & 19.7533 & -0.000\,28 \\ 0.006\,329\,8 & 0.036\,635 & -8 \cdot 10^{-7} & -7.738 & -4.4095 & -0.000\,21 \\ 0.003\,349\,5 & 0.019\,399 & -1.6 \cdot 10^{-6} & -4.2243 & -2.4072 & -0.000\,42 \\ -0.001\,860\,3 & -0.010\,774 & -1.0667 \cdot 10^{-6} & 2.3955 & 1.3651 & -0.000\,28 \\ -0.001\,251\,7 & -0.007\,251\,8 & -9.1429 \cdot 10^{-7} & -20.6161 & -11.7481 & -0.000\,24 \end{bmatrix} \tag{5.1a}$$

$$\mathbf{G}^y = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ -41.5667 & 7.3684 \\ 25\,000 & 58.4795 \\ 14\,666.6667 & 0 \\ -26\,666.6667 & -58.4795 \end{bmatrix} \tag{5.1b}$$

$$\mathbf{J}_{uu} = \begin{bmatrix} -3\,680\,034\,953\,700 & 2\,083\,333.3334 \\ 2\,083\,333.3334 & -11\,666.6666 \end{bmatrix} \tag{5.1c}$$

The measurement combination matrix, $\mathbf{H}$ and the average loss, L_{avg}, was calculated as described in Chapter 3.2.3. The results are given in Equation (5.2)

$$\mathbf{H} = \begin{bmatrix} 0.900\,99 & -0.153\,91 & -0.368\,86 & -6.6429 \cdot 10^{-5} & 0.000\,335\,12 & -0.000\,526\,32 \\ -0.1532 & 0.027\,171 & 0.065\,385 & 1.2842 \cdot 10^{-5} & -2.2254 \cdot 10^{-5} & -1.1793 \cdot 10^{-6} \end{bmatrix} \tag{5.2a}$$

$$\mathbf{L}_{\text{avg}} = \begin{bmatrix} 1927.6592 \end{bmatrix} \tag{5.2b}$$

$$\mathbf{L}_{\text{worst}} = \begin{bmatrix} 92\,527.6336 \end{bmatrix} \tag{5.2c}$$

6 Discussion

6.1 Simulations

The optimal solution for the unaltered file, `mhexcs1v9.gms`, was readily obtained. Changing the temperature and heat capacity flow rates yielded objective values within a reasonable range.

However, there was encountered problems with restricting the degrees of freedom. Upon the introduction of the control equations given in Listing 13 `CONOPT` reported that there were no superbasic variables. This means that the solver found no degrees of freedom, and in practice was used as a normal equation solver. Still, doubtful solutions were found. Upon fixing both `FN2` and `PumpWork` at it's optimal values, found from `mhexcs1v9.gms`, the objective function, z, yielded a *better* solution compared to the objective function value found from the "clean" optimization in `mhexcs1v9.gms`, z_{opt}.

The process of finding perturbation step size proved troublesome. The value of z increased dramatically for values of h greater than $0.5 \cdot 10^{-7}$ in positive and negative `FN2` direction. For `PumpWork`, z was further improved by performing positive perturbations. In fact, $z > z_{\text{opt}}$ for values of $h \geq 1 \cdot 10^{-4}$. Upon reduction of the step length to $h = 2 \cdot 10^{-3}$, the value of z increased drastically to a magnitude of $\approx 10^5$.

As a result, a positive semi-definite Hessian cost matrix was not obtained. Some notes on the positive elements of $\mathbf{J}_{uu}$ are discussed in Chapter 6.4.

The `GAMS display` statement is limited to displaying 8 decimals. Because of this limited ability to display highly precise numbers, using small step sizes, such as $h = 0.5 \cdot 10^{-7}$, could result in such small changes in output values that it is difficult to capture the mathematical essence of the variations. The `CONOPT` command shell print displays 10 decimal precision for the value of the objective function. It should be considered to capture the values from this print.

6.2 Sensitivity

From the resulting sensitivity matrix, $\mathbf{F}$, given in Equation (5.1a) in Chapter 5, it is clear that the temperatures T_{S_4} and T_{S_5} in general are sensitive to disturbances. High sensitivity in T_{S_4} and T_{S_5} can be observed for disturbances in $T_{H_1}, T_{H_2}, F_{Cp,H_1}$ and F_{Cp,C_2}. Both the pressure p_{S_1} and outlet temperature T_{S_6} are completely

insensitive to changes in the inlet temperatures or heat capacity flow rates. An exception can be observed for disturbances in T_{H_1}, where T_{S_6} is relatively sensitive.

6.2.1 Feed Flow, Pump Work and Pressure at S_1

The nitrogen feed flow rate shows small sensitivity to the disturbances. This is probably caused by the heat of evaporation being the main consumer of heat in the cooling process of the first pass through the MHEX. It is important to note that this is valid only as long as the dew point temperature is reached. All the nitrogen must be allowed to evaporate in the first pass through the heat exchanger. The temperature at S_1 does not change with changing F_{N_2} or p_{S_1}, as the model assumes the liquid nitrogen to be an incompressible fluid.

The pressure at S_1 affects the boiling point and dew point temperatures. The pump work will therefore change as disturbances occurs. However, based on the same arguments as for F_{N_2}, the sensitivity to disturbances is relatively small. The pressure at S_1, p_{S_1}, shows extremely small sensitivity to disturbances. Most of the values, such as $-1.6 \cdot 10^{-7}$ and $8 \cdot 10^{-7}$, are probably caused by numerical noise or `GAMS`' decimal display limit.

Figure 6.1 illustrates how moving the dew point and bubble point affect the temperature at state S_4.

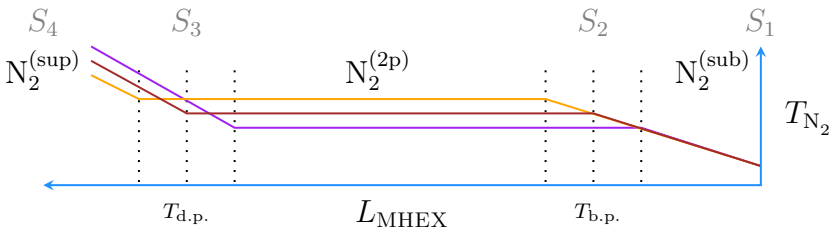

Figure 6.1: Temperature variations as function of feed specifications. The illustration is given from right to left to coincide with Figure 2.1.

6.2.2 Temperatures

The temperature at S_4, T_{S_4}, is sensitive to disturbances. Changes in F_{Cp} caused the greatest variations in T_{S_4}. This is expected, as small changes in heat capacity can result in vast changes in heat transport. The temperature after the expander, T_{S_5}, experiences a similar pattern, though of smaller magnitude.

The temperature at the exit of the nitrogen cooling flow, T_{S_6} is almost completely insensitive to disturbances. An exception is for disturbances to the temperature T_{H_1}. This sensitivity is not matched by a similar trend for disturbances in $F_{Cp,H1}$. There could be a possibility of the high temperature at $T_{H_1}^{\text{in}}$ governs this sensitivity, as the exit temperature, T_{S_6}, is close to the H_1 inlet temperature. If this is true, and the inlet temperature for H_1 is known, T_{S_6} could be a self-optimizing variable candidate.

6.3 Gain

The gain matrix shows that the input variables in general have high influence on the output variables. The value for $\Delta T_{S_6}/\Delta\texttt{FN2}$ is very high, and is thus backing up T_{S_6} as being a self-optimizing variable candidate.

6.4 The Hessian of the Cost Function

The positive elements in $\mathbf{J}_{uu}$ are of high magnitude, indicating that the starting point used for the central differences is lying in a "deep valley". For this calculation, $h_1 = 0.6 \cdot 10^{-7}$ and $h_2 = 1 \cdot 10^{-6}$, where h_1 is the step length in the `FN2` direction and h_2 is the step length in the `PumpWork` direction.

6.4.1 Loss

As $\mathbf{J}_{uu}$ is not positive semi-definite, the loss calculations does not make sense. Calculations of loss was also performed following the notation given in Equation (3.10), which yielded an unphysical loss in terms of a complex number, due to $\mathbf{J}_{uu}$.

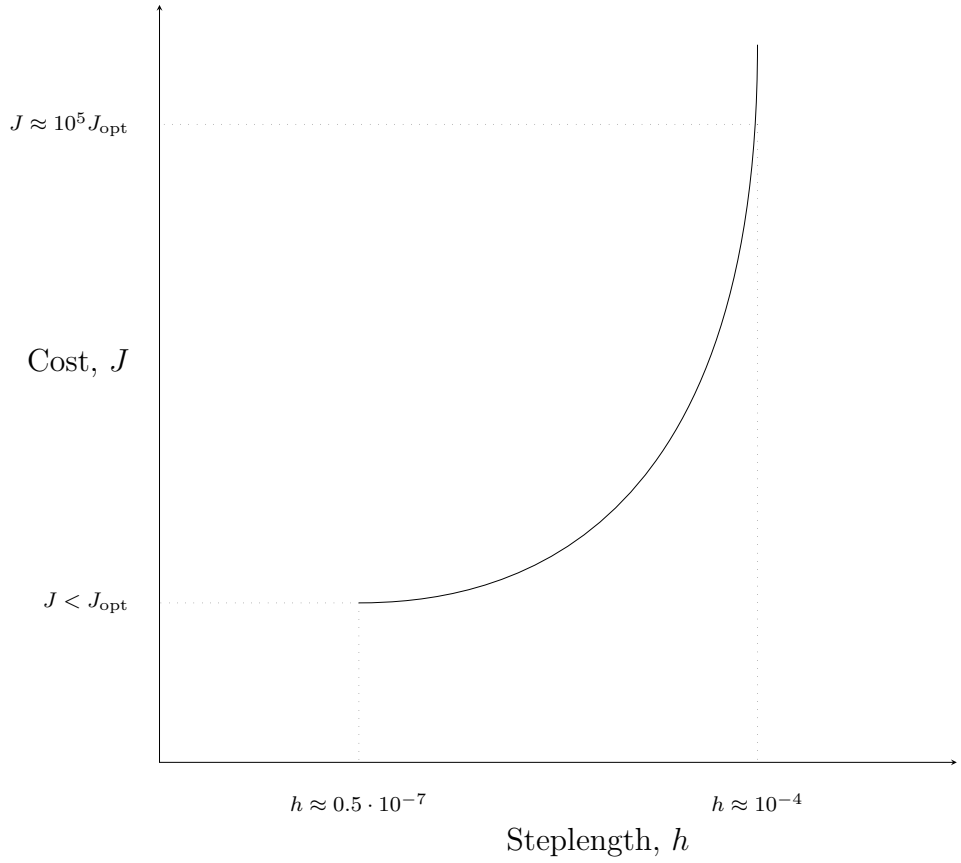

Figure 6.2: Somecaption

7 Conclusion

The results for the "motivating example MHEX" given in the paper by Kamath et al. [Ref. 14] was reproduced using `GAMS` with the solver `CONOPT`. The degrees of freedom for the example was determined to $N = 2$. A process scenario was simulated, where the degrees of freedom chosen as inputs were F_{N_2} and W_{pump}, denoted `FN2` and `PumpWork` in the `GAMS` model, respectively. When deciding data for finding $\mathbf{F}$, $\mathbf{G}^y$ and $\mathbf{J}_{uu}$ from the simulations, it was observed that a better optimum could be obtained by fixing `FN2` and `PumpWork` at it's optimal values, and giving the optimal point as a starting point for the solver. $\mathbf{J}_{uu}$ was by finite central differences found to not be positive semi-definite.

Further work should focus on further model analysis. Whether to use `GAMS` for solving the model should be re-evaluated.

References

[1] Process optimization. `http://en.wikipedia.org/wiki/Process_optimization`.

[2] M. Abramowitz and I. A. Stegun. *Handbook of Mathematical Functions*. New York: Dover Publications Inc., 1972.

[3] V. Alstad and S. Skogestad. Null space method for selecting optimal measurement combinations as controlled variables. *Ind. Eng. Chem. Res.*, 46:846, 2007.

[4] V. Alstad, S. Skogestad, and E. S. Hori. Optimal measurement combinations as controlled variables. *ScienceDirect Journal of Process Control*, 19:138, 2009.

[5] E. M. B. Aske and S. Skogestad. Consistent inventory control. *Ind. Eng. Chem. Res.*, 48:10892, 2009.

[6] A. Aspelund, D. O. Berstad, and T. Gundersen. An extended pinch analysis adn design procedure utilizing pressure based energy for subambient cooling. *Ind. Eng. Chem. Res*, 47:8724, 2008.

[7] S. Balakrishna and L. Biegler. Targeting strategies for the synthesis and energy integration of nonisothermal reactor networks. *Ind. Eng. Chem. Res.*, 31:2152.

[8] B. Baumrucker, J. Renfro, and L. Biegler. Mpec problem formulations and solution strategies with chemical engineering applications. *Computers and Chemical Engineering*, 32:2903, 2008.

[9] L. T. Biegler. *Nonlinear Programming: Consepts, Algorithms, and Applications to Chemical Processes*. Society for Industrial and Applied Mathematichs and the Mathematical Optimization Society, 2010.

[10] A. Drud. Conopt — a large scale grg code. *ORSA J. Comput*, 6:207.

[11] M. A. Duran and I. E. Grossmann. Simultaneous optimization and heat integration of chemical processes. *AIChE*, 32:123.

[12] P. S. Foundation. `http://www.python.org`.

[13] I. J. Halvorsen, S. Skogestad, J. C. Morud, and V. Alstad. Optimal selection of controlled variables. *Ind. Eng. Chem. Res.*, 42:3273, 2003.

[14] R. S. Kamath, L. T. Biegler, and I. E. Grossman. Modeling multistream heat exchangers with and wihout phase changes for simultaneous optimization and heat integration. *Wiley Online Library*, 2011.

[15] V. Kariwala, Y. Cao, and S. Janardhanan. Local self-optimizing control with average loss minimization. 47:1150, 2008.

[16] Macports. `http://www.macports.org`.

[17] B. A. McCarl, A. Meeraus, P. van der Eijk, M. Bussieck, S. Dirkse, P. Steacy, F. Nilssen, and G. D. Corporation. Mccarl gams user guide. `http://www.gams.com/dd/docs/bigdocs/gams2002/mccarlgamsuserguide.pdf`, 2012.

[18] S. Mokhatab, W. A. Poe, and J. G. Speight. *Handbook of Natural Gas Transmission and Processing*. Elsevier Inc, 2006.

[19] M. Morari, Y. Arkun, and G. Stephanopoulos. Studies in the synthesis of control structures for chemical processes. *AIChE*, 26(2):220, 1980.

[20] J. Nocedal and S. J. Wright. *Numerical Optimization*. Springer, second edition, 2006.

[21] R. E. Rosenthal. Gams – a user's guide. `http://www.gams.com/dd/docs/bigdocs/GAMSUsersGuide.pdf`, 2012.

[22] R. Sinnott and G. Towler. *Chemical Engineering Design*. Elsevier, fifth edition, 2009.

[23] S. Skogestad. Plantwide control: The search for the self-optimizing control structure. *Journal of Process Control*, 10:487, 2000.

[24] S. Skogestad. Control structure design for complete chemical plants. *Computers and Chemical Engineering*, 28:219, 2004.

[25] S. Skogestad and I. Postlethwaite. *Multivariable Feedback Control*. Wiley & Sons, Ltd, second edition, 2005.

[26] L. M. Umar, W. Hu, Y. Cao, and V. Kariwala. *Plantwide Control: Recent Developments and applications*. John Wiley and Sons, Ltd., first edition, 2012.

[27] Wikipedia. `http://en.wikipedia.org/wiki/Autovivification`.

[28] R. Yelchuru, S. Skogestad, and H. Manum. Miqp formulation fo controlled variable selection in self optimizing control. *Proceedings of the 9th Internation Symposium on Dynamics and Control of process Systems*, 2010.

A The GAMS Model mhexcs1v9.gms

```
* same as v8 except using bonmin

* same as v7 except that
* doing full optimization

* same as v6 except that
* adding isothermal stream to heat integration
* original big M by Ignacio does not occur...relaxation of
    contribution constraint?
* doing full optimization in next version

* same as mhexcs1v5 except that
* now added Fcp for C3 C5 and C6 into the heat integration
* adding above pinch contribution to heat integration in next version
    since it will affect the optimization result

* same as mhexcs1v4 except that
* adding the noisothermal streams C3, C5, C6 to the heat integration
    but one step at a time
* forgot to add expander work...adding that first
* now added Fcp for C3 C5 and C6 into the heat integration
* adding above pinch contribution to heat integration in next version
    since it will affect the optimization result

* same as mhexcs1v3 except that
* 1) Tin, Tout and FCp are converted from parameters to variables for
    simple streams

*new stream for heat integration are assigned temperature and heat
    load data
* C3 stream is from state S1 to S2
* C4 stream is from state S2 to S3
* C5 stream is from state S3 to S4
* C6 stream is from state S5 to S6

* same as mhexcs1v2 except that new streams are introduced into the
    sets but not used in the code
* basically i use new subsets so that they still work only on the
    originally specified streams

* same as mhexcs1v1 except that i am including properties of Nitrogen
* defining states for convenience
* S1 - after the pump and entering the MHEX
* S2 - within the MHEX at bubblle point
* S3 - within the MHEX at dew point
* S4 - at exit of MHEX and superheated and entering expander
```

```
* S5 - exit of expander and entering MHEX again
* S6 - final exit from MHEX

* a simple version of Duran and Grossmann for minimum utility
    calculation
* uses smooth approximation for max operators
* at bottom have information for generating composite curves

* global options
$offlisting
$eolcom //
$inlinecom /* */
option limrow = 100 ;
option limcol = 100 ;
option iterlim = 500000 ;
option reslim = 50000 ;
option optcr = 0.0 ;
option solprint = ON ;
option decimals = 6 ;
// end of global options

SCALARS
HRAT      Heat recovery approach temperature    /4/
epsi      epsilon for smooth approximation      /0.000005/
Tref      "Reference temperature for Enthalpy calculations in K"
    /298.15/
PN2Inlet  "Pressure of main N2 inlet in bar"    /6.0/
TN2Inlet  "Temperature of main N2 inlet in K"   /95/
Pmax      "Max pressure to which N2 can be pumped in bar" /15/
DenLiqN2  "density of liquid N2 in kg/m3"       /808.607/
PumpEff   "efficiency of pump"                  /0.75/
ExpdrEff  "expander efficiency"                 /0.7/
MWN2      "molecular weight"                    /28.0134/
PresS5    "pressure at state S5 in bar"         /1.01325/
gamma     "Cp/Cv for diatomatic N2"             /1.4/
Tc        "Critical temperature of N2 in K"     /126.2/
TrS1
EnthS1
costN2      "cost of liquid nitrogen in cents per litre"  /50/
costElec    "cost of electricity in cents per kW hr"      /12/
;

TrS1 = TN2Inlet/Tc ;
Display TrS1 ;

SETS
Str         Total Streams        /H1,H2,H3,C1,C2,C3,C4,C5,C6/
HtStr(Str)  Hot streams of HEX   /H1,H2,H3/
CdStr(Str)  Cold Streams of HEX  /C1,C2,C3,C4,C5,C6/
```

```
86 Simple(Str) "basic streams"      /H1,H2,H3,C1,C2/
87 Isoth(Str)  "isothermal streams" /C4/
88
89 CC1     "Constants for CP of N2"    /C1,C2,C3,C4,C5,C6,C7/
90 NonIsoth(Str)
91 ;
92
93 NonIsoth(Str) = Str(Str) - Isoth(Str) ;
94 Display Nonisoth ;
95
96 ALIAS
97 (Str,Str2,Str3)
98 ;
99
100 PARAMETERS
101 *FCp(Str)      "unit is kW/K" /H1 = 3,   H2 = 4,   H3 = 2,   C1 = 3,
       C2 = 3.5/
102 *Tin(Str)      "unit is K"    /H1 = 298, H2 = 265, H3 = 195, C1 = 220,
      C2 = 255/
103 *Tout(Str)     "unit is K"    /H1 = 250, H2 = 180, H3 = 150, C1 = 245,
      C2 = 280/
104 Cpconst(CC1)  "for Cp = kJ/kmol K, T = K" /C1 = 29.105, C2 = 8.6149,
     C3 = 1701.6, C4 = 0.10347, C5 = 909.79/
105 TSatconst(CC1)"for Tsat, in K, in bar"    /C1 = 46.76907454, C2 =
     -1084.1, C5 = -8.3144, C6 = 0.044127, C7 = 1/
106 LtHtconst(CC1)"in MJ/kmol, T in K"        /C1 = 7.4905, C2 = 0.40406,
      C3 = -0.317, C4 = 0.27343/
107 ;
108
109 EnthS1 = CpConst('C1')*(TN2Inlet - Tref) + CpConst('C2')*CpConst('C3'
     )*(1/tanh(CpConst('C3')/TN2Inlet) -  1/tanh(CpConst('C3')/Tref))
110         - CpConst('C4')*CpConst('C5')*(tanh(CpConst('C5')/TN2Inlet) -
              tanh(CpConst('C5')/Tref))
111         - (LtHtconst('C1')*(1-TrS1)**(LtHtconst('C2')+ LtHtconst('C3')
              *TrS1 + LtHtconst('C4')*(TrS1**2)))*1000 ;
112
113 Display EnthS1 ;
114
115 VARIABLES
116 MxInHt(Str,Str2)
117 MxOutHt(Str,Str2)
118 MxOutCd(Str,Str2)
119 MxInCd(Str,Str2)
120 HetCntHt(Str)
121 HetCntCd(Str)
122 TPch(Str)
123 HetLd(Str)
124 HtUtLd
125 CdUtLd
```

```
126 Omega
127 FN2         "in kmol/s"
128 PresS1      "Pressure at state S1"
129 PumpWork    "in kW"
130 TempS4      "Temperature at state S4"
131 TempS5      "Temperture at state S5"
132 TempS6      "Temperture at state S6"
133 Tboil       "boiling point at pressure of S1"
134 EnthS4      "enthalpy at S4 in kJ/kmol"
135 EnthS3      "enthalpy at S3 in kJ/kmol"
136 EnthS2      "enthalpy at S2 in kJ/kmol"
137 EnthS5      "enthalpy at S5 in kJ/kmol"
138 EnthS6      "enthalpy at S6 in kJ/kmol"
139 TrS2        "reduced temperature"
140 Tin(Str)
141 Tout(Str)
142 FCp(Str)
143 ExpdrWork
144 contri(Str,Str2)
145 dummy1
146 ;
147
148 VARIABLES
149 Z ;
150
151 BINARY VARIABLES
152 YLoc(Str,Str2) "1 indicates that it is above the pinch and
        contributes while 0 means no contribution"
153 ;
154
155 EQUATIONS
156 EqFF1(Str)
157 EqFF2(Str)
158 EqOmega
159 EqTPch1(Str)
160 EqTPch2(Str)
161 EqMxInHt(Str,Str2)
162 EqMxOutHt(Str,Str2)
163 EqMxOutCd(Str,Str2)
164 EqMxInCd(Str,Str2)
165 EqHetCntHt(Str)
166 EqHetCntCd(Str)
167 EqHtUtLd(Str)
168 EqCdUtLd
169 EqPumpWork
170 Eqadiaexpn
171 EqTboil
172 EqPhase1
173 EqEnthS3
```

```
174 EqEnthS4
175 EqEnthS5
176 EqEnthS6
177 EqTrS2
178 EqEnthS2
179
180 EqTinC3
181 EqToutC3
182 EqTinC4
183 EqToutC4
184 EqTinC5
185 EqToutC5
186 EqTinC6
187 EqToutC6
188
189 EqHtLdC3
190 EqHtLdC4
191 EqHtLdC5
192 EqHtLdC6
193
194 Eqrefri
195
196 EqExpdrWork
197
198 Eqlogic1(Str,Str2)
199 Eqlogic11(Str,Str2)
200 Eqlogic2(Str,Str2)
201 Eqlogic3(Str,Str2)
202 Eqlogic4(Str,Str2)
203 ;
204
205 EqFF1(Str)$(HtStr(Str)*Nonisoth(Str))..
206   FCp(Str)*(Tin(Str) - Tout(Str)) =E= HetLd(Str) ;
207 EqFF2(Str)$(CdStr(Str)*Nonisoth(Str))..
208   FCP(Str)*(Tout(Str) - Tin(Str)) =E= HetLd(Str) ;
209 EqOmega..
210   Omega =E= SUM(Str$(HtStr(Str)), HetLd(Str)) - SUM(Str$(CdStr(Str)),
        HetLd(Str)) ;
211 EqTPch1(Str)$(HtStr(Str))..
212   TPch(Str) =E= Tin(Str) ;
213 EqTPch2(Str)$(CdStr(Str))..
214   TPch(Str) =E= Tin(Str) + HRAT ;
215 EqMxInHt(Str,Str2)$((HtStr(Str)*Nonisoth(Str)))..
216   MxInHt(Str,Str2) =E= 0.5*sqrt(Power(Tin(Str)- TPch(Str2),2) + Power
        (epsi,2)) + 0.5*(Tin(Str)-TPch(Str2)) ;
217 EqMxOutHt(Str,Str2)$((HtStr(Str)*Nonisoth(Str)))..
218   MxOutHt(Str,Str2) =E= 0.5*sqrt(Power(Tout(Str)- TPch(Str2),2) +
       Power(epsi,2)) + 0.5*(Tout(Str)-TPch(Str2)) ;
219 EqMxOutCd(Str,Str2)$((CdStr(Str)*Nonisoth(Str)))..
```

```
220   MxOutCd(Str,Str2) =E= 0.5*sqrt(Power(Tout(Str)- TPch(Str2)+ HRAT,2)
          + Power(epsi,2)) + 0.5*(Tout(Str)-TPch(Str2)+HRAT) ;
221 EqMxInCd(Str,Str2)$((CdStr(Str)*Nonisoth(Str)))..
222   MxInCd(Str,Str2) =E= 0.5*sqrt(Power(Tin(Str)- TPch(Str2)+ HRAT,2) +
          Power(epsi,2)) + 0.5*(Tin(Str)-TPch(Str2)+HRAT) ;
223 EqHetCntHt(Str2)..
224         HetCntHt(Str2) =E= SUM(Str$(HtStr(Str)*Nonisoth(Str)), FCp(
              Str)*(MxInHt(Str,Str2) - MxOutHt(Str,Str2))) ;
225 * about the isothermal cold stream
226 Eqlogic1(Str,Str2)$((CdStr(Str)*Isoth(Str)))..
227        contri(Str,Str2) - HetLd(Str) =L= 10000*(1-YLoc(Str,Str2)) ;
228 Eqlogic11(Str,Str2)$((CdStr(Str)*Isoth(Str)))..
229        contri(Str,Str2) - HetLd(Str) =G= -10000*(1-YLoc(Str,Str2)) ;
230 Eqlogic2(Str,Str2)$((CdStr(Str)*Isoth(Str)))..
231        contri(Str,Str2) =L= 10000*YLoc(Str,Str2) ;
232 Eqlogic3(Str,Str2)$((CdStr(Str)*Isoth(Str)))..
233        Tin(Str) - TPch(Str2) =G= -500*(1 -YLoc(Str,Str2)) ;
234 Eqlogic4(Str,Str2)$((CdStr(Str)*Isoth(Str)))..
235        Tin(Str) =L=  TPch(Str2) - epsi + 500*YLoc(Str,Str2) ;
236 EqHetCntCd(Str2)..
237         HetCntCd(Str2) =E= SUM(Str$(CdStr(Str)*Nonisoth(Str)), FCP(
              Str)*(MxOutCd(Str,Str2) - MxInCd(Str,Str2))) + SUM(Str$(
              CdStr(Str)*Isoth(Str)),contri(Str,Str2)) ;
238 EqHtUtLd(Str2)..
239         HtUtLd =G= HetCntCd(Str2) - HetCntHt(Str2) ;
240 EqCdUtLd..
241         CdUtLd =E= Omega + HtUtLd ;
242 EqPumpWork..
243         PumpWork =E= (PresS1 - PN2Inlet)*FN2*MWN2*100/(PumpEff*
              DenLiqN2)  ;
244 * pressure in bar...so multipled by 100 to convert to kPa...kg gets
      cancelled with m3 in numerator
245 * final result is kW
246 Eqadiaexpn..
247         (gamma-1)*log(PresS1) + gamma*log(TempS5) =E= (gamma-1)*log(
              PresS5) + gamma*log(TempS4) ;
248 * wrote the expression in log form
249 EqTboil..
250         log(PresS1) =E=  TSatconst('C1') + TSatconst('C2')/Tboil +
              TSatconst('C5')*log(Tboil) +  TSatconst('C6')*Tboil ;
251 EqPhase1..
252         TempS4 =G= Tboil ;
253 EqEnthS3..
254        EnthS3 =E= CpConst('C1')*(Tboil - Tref) + CpConst('C2')*
             CpConst('C3')*(1/tanh(CpConst('C3')/Tboil) -  1/tanh(
             CpConst('C3')/Tref))
255                    - CpConst('C4')*CpConst('C5')*(tanh(CpConst('C5')/
                         Tboil) - tanh(CpConst('C5')/Tref)) ;
256 EqEnthS4..
```

```
257         EnthS4 =E= CpConst('C1')*(TempS4 - Tref) + CpConst('C2')*
                CpConst('C3')*(1/tanh(CpConst('C3')/TempS4) -  1/tanh(
                CpConst('C3')/Tref))
258                     - CpConst('C4')*CpConst('C5')*(tanh(CpConst('C5')/
                            TempS4) - tanh(CpConst('C5')/Tref)) ;
259 EqEnthS5..
260         EnthS5 =E= CpConst('C1')*(TempS5 - Tref) + CpConst('C2')*
                CpConst('C3')*(1/tanh(CpConst('C3')/TempS5) -  1/tanh(
                CpConst('C3')/Tref))
261                     - CpConst('C4')*CpConst('C5')*(tanh(CpConst('C5')/
                            TempS5) - tanh(CpConst('C5')/Tref)) ;
262 EqEnthS6..
263         EnthS6 =E= CpConst('C1')*(TempS6 - Tref) + CpConst('C2')*
                CpConst('C3')*(1/tanh(CpConst('C3')/TempS6) -  1/tanh(
                CpConst('C3')/Tref))
264                     - CpConst('C4')*CpConst('C5')*(tanh(CpConst('C5')/
                            TempS6) - tanh(CpConst('C5')/Tref)) ;
265 EqExpdrWork..
266         ExpdrWork  =E= (EnthS4 - EnthS5)*FN2*ExpdrEff ;
267 EqTrS2..
268         TrS2 =E=  Tboil/Tc ;
269 EqEnthS2..
270         EnthS2 =E=  EnthS3 - (LtHtconst('C1')*(1-TrS2)**(LtHtconst('C2
                ')+ LtHtconst('C3')*TrS2 + LtHtconst('C4')*(TrS2**2)))*1000
                 ;
271 * inlet temperature data for C3 stream
272 EqTinC3..
273         Tin('C3') =E=  TN2Inlet ;
274 * outlet temperature data for C3 stream
275 EqToutC3..
276         Tout('C3') =E=  Tboil ;
277 * inlet temperature data for C4 stream
278 EqTinC4..
279         Tin('C4') =E=  Tboil ;
280 * outlet temperature data for C4 stream
281 EqToutC4..
282         Tout('C4') =E=  Tboil ;
283 * inlet temperature data for C5 stream
284 EqTinC5..
285         Tin('C5') =E=  Tboil ;
286 * outlet temperature data for C5 stream
287 EqToutC5..
288         Tout('C5') =E=  TempS4 ;
289 * inlet temperature data for C6 stream
290 EqTinC6..
291         Tin('C6') =E=  TempS5 ;
292 * outlet temperature data for C6 stream
293 EqToutC6..
294         Tout('C6') =E=  TempS6 ;
```

```
295 EqHtLdC3..
296         HetLd('C3') =E= (EnthS2 - EnthS1)*FN2 ;
297 EqHtLdC4..
298         HetLd('C4') =E= (EnthS3 - EnthS2)*FN2 ;
299 EqHtLdC5..
300         HetLd('C5') =E= (EnthS4 - EnthS3)*FN2 ;
301 EqHtLdC6..
302         HetLd('C6') =E= (EnthS6 - EnthS5)*FN2 ;
303 Eqrefri..
304         dummy1 =E= HetLd('C3') + HetLd('C4') + HetLd('C5') + HetLd('C6
              ')  ;
305
306 EQUATIONS
307 obj ;
308 obj..
309          Z =E= 1000*HtUtLd + 1000*CdUtLd + FN2*MWN2*1000*costN2/(100*
               DenLiqN2) + costElec*(Pumpwork - ExpdrWork)/(3600*100) ;
310
311 *---------
312 * Bounds
313 *---------
314 HetLd.lo(Str) = 0.0 ;
315 HetLd.up(Str) = 1000.0 ;
316 Fcp.lo(Str) = 0.0001 ;
317 Fcp.up(Str) = 1000 ;
318 FN2.lo = 0.0 ;
319 FN2.up = 10.0 ;
320 PresS1.lo  = PN2Inlet ;
321 PresS1.up  = Pmax ;
322 PumpWork.lo = 0 ;
323 PumpWork.up = 10000 ;
324 TempS4.lo = 75 ;
325 TempS4.up = 300 ;
326 * to ensure that expander outlet is not in 2 phase
327 TempS5.lo = 77.36 ;
328 TempS5.up = 300 ;
329 Tboil.lo = 50 ;
330 Tboil.up = 300 ;
331 TempS6.lo = 77.36 ;
332 TempS6.up = 300 ;
333 TrS2.lo = 0.1 ;
334 TrS2.up = 1.1 ;
335 EnthS2.lo = -50000 ;
336 EnthS2.up =  0 ;
337 EnthS3.lo = -50000 ;
338 EnthS3.up =  0 ;
339 EnthS4.lo = -50000 ;
340 EnthS4.up =  0 ;
341 EnthS5.lo = -50000 ;
```

```
342 EnthS5.up =  0 ;
343 EnthS6.lo = -50000 ;
344 EnthS6.up =  0 ;
345 contri.lo(Str,Str2) = 0.0 ;
346 contri.up(Str,Str2) = 10000.0 ;
347 CdUtLd.lo = -0.0001 ;
348 HtUtLd.lo = -0.0001 ;
349
350 *--------
351 * Fixed
352 *--------
353 * fixing the Tin, Tout and FCp for simple streams
354 * unit is kW/K
355 FCp.fx('H1') = 3 ;
356 FCp.fx('H2') = 4 ;
357 FCp.fx('H3') = 2 ;
358 FCp.fx('C1') = 3 ;
359 FCp.fx('C2') = 3.5 ;
360
361 * unit is K
362 Tin.fx('H1') = 298 ;
363 Tin.fx('H2') = 265 ;
364 Tin.fx('H3') = 195 ;
365 Tin.fx('C1') = 220 ;
366 Tin.fx('C2') = 255 ;
367
368 * unit is K
369 Tout.fx('H1') = 250 ;
370 Tout.fx('H2') = 180 ;
371 Tout.fx('H3') = 150 ;
372 Tout.fx('C1') = 245 ;
373 Tout.fx('C2') = 280 ;
374
375 * temporary fixing variables
376 FN2.l = 0.04 ;
377 PresS1.l  = 11 ;
378 TempS4.l =  180 ;
379 TempS6.l = 250 ;
380
381 *----------------
382 * Initial Point
383 *----------------
384 HtUtLd.l = 1000;
385 Tboil.l = 105 ;
386
387 * logic
388 YLoc.l('C4','H1') = 0 ;
389 YLoc.l('C4','H2') = 0 ;
390 YLoc.l('C4','H3') = 0 ;
```

```
391 YLoc.l('C4','C1') = 0 ;
392 YLoc.l('C4','C2') = 0 ;
393 YLoc.l('C4','C3') = 1 ;
394 YLoc.l('C4','C4') = 0 ;
395 YLoc.l('C4','C5') = 0 ;
396 YLoc.l('C4','C6') = 1 ;
397
398 PumpWork.l = (PresS1.l - PN2Inlet)*FN2.l*MWN2*100/(PumpEff*DenLiqN2)
      ;
399 TempS5.l = TempS4.l*(PresS5/PresS1.l)**((gamma-1)/gamma)    ;
400 EnthS3.l = CpConst('C1')*(Tboil.l - Tref) + CpConst('C2')*CpConst('C3
    ')*(1/tanh(CpConst('C3')/Tboil.l) -  1/tanh(CpConst('C3')/Tref))
401          - CpConst('C4')*CpConst('C5')*(tanh(CpConst('C5')/Tboil.l) -
                tanh(CpConst('C5')/Tref)) ;
402 EnthS4.l = CpConst('C1')*(TempS4.l - Tref) + CpConst('C2')*CpConst('
    C3')*(1/tanh(CpConst('C3')/TempS4.l) -  1/tanh(CpConst('C3')/Tref)
    )
403          - CpConst('C4')*CpConst('C5')*(tanh(CpConst('C5')/TempS4.l)
               - tanh(CpConst('C5')/Tref)) ;
404 EnthS5.l = CpConst('C1')*(TempS5.l - Tref) + CpConst('C2')*CpConst('
    C3')*(1/tanh(CpConst('C3')/TempS5.l) -  1/tanh(CpConst('C3')/Tref)
    )
405          - CpConst('C4')*CpConst('C5')*(tanh(CpConst('C5')/TempS5.l)
               - tanh(CpConst('C5')/Tref)) ;
406 EnthS6.l = CpConst('C1')*(TempS6.l - Tref) + CpConst('C2')*CpConst('
    C3')*(1/tanh(CpConst('C3')/TempS6.l) -  1/tanh(CpConst('C3')/Tref)
    )
407          - CpConst('C4')*CpConst('C5')*(tanh(CpConst('C5')/TempS6.l)
               - tanh(CpConst('C5')/Tref)) ;
408 ExpdrWork.l  = (EnthS5.l - EnthS4.l)*FN2.l*ExpdrEff ;
409 TrS2.l =  Tboil.l/Tc ;
410 EnthS2.l =  EnthS3.l - (LtHtconst('C1')*(1-TrS2.l)**(LtHtconst('C2')+
     LtHtconst('C3')*TrS2.l + LtHtconst('C4')*(TrS2.l**2)))*1000 ;
411 Tin.l('C3') =  TN2Inlet ;
412 Tout.l('C3') =  Tboil.l ;
413 Tin.l('C4') =  Tboil.l ;
414 Tout.l('C4') =  Tboil.l ;
415 Tin.l('C5') =  Tboil.l ;
416 Tout.l('C5') =  TempS4.l ;
417 Tin.l('C6') =  TempS5.l ;
418 Tout.l('C6') =  TempS6.l ;
419 HetLd.l('C3') = (EnthS2.l - EnthS1)*FN2.l ;
420 HetLd.l('C4') = (EnthS3.l - EnthS2.l)*FN2.l ;
421 HetLd.l('C5') = (EnthS4.l - EnthS3.l)*FN2.l ;
422 HetLd.l('C6') = (EnthS6.l - EnthS5.l)*FN2.l ;
423 dummy1.l = HetLd.l('C3') + HetLd.l('C4') + HetLd.l('C5') + HetLd.l('
    C6')  ;
424 HetLd.l(Str)$(HtStr(Str)*Nonisoth(Str)) =   FCp.l(Str)*(Tin.l(Str) -
    Tout.l(Str)) ;
```

```
425 HetLd.l(Str)$(CdStr(Str)*Nonisoth(Str)) =   FCp.l(Str)*(Tout.l(Str) -
      Tin.l(Str)) ;
426 FCp.l(Str)$(HtStr(Str)*(NOT Simple(Str))*Nonisoth(Str)) =   HetLd.l(
    Str)/(Tin.l(Str) - Tout.l(Str)) ;
427 FCp.l(Str)$(CdStr(Str)*(NOT Simple(Str))*Nonisoth(Str)) =   HetLd.l(
    Str)/(Tout.l(Str) - Tin.l(Str)) ;
428 Omega.l = SUM(Str$(HtStr(Str)*Nonisoth(Str)), HetLd.l(Str)) - SUM(
    Str$(CdStr(Str)*Nonisoth(Str)), HetLd.l(Str)) ;
429 TPch.l(Str)$(HtStr(Str)*Nonisoth(Str)) = Tin.l(Str) ;
430 TPch.l(Str)$(CdStr(Str)*Nonisoth(Str)) = Tin.l(Str) + HRAT ;
431 MxInHt.l(Str,Str2)$((HtStr(Str)*Nonisoth(Str)) AND Nonisoth(Str2)) =
    0.5*sqrt(Power(Tin.l(Str)- TPch.l(Str2),2) + Power(epsi,2)) +
    0.5*(Tin.l(Str)-TPch.l(Str2)) ;
432 MxOutHt.l(Str,Str2)$((HtStr(Str)*Nonisoth(Str)) AND Nonisoth(Str2)) =
     0.5*sqrt(Power(Tout.l(Str)- TPch.l(Str2),2) + Power(epsi,2)) +
    0.5*(Tout.l(Str)-TPch.l(Str2)) ;
433 MxOutCd.l(Str,Str2)$((CdStr(Str)*Nonisoth(Str)) AND Nonisoth(Str2)) =
     0.5*sqrt(Power(Tout.l(Str)- TPch.l(Str2)+ HRAT,2) + Power(epsi,2)
    ) + 0.5*(Tout.l(Str)-TPch.l(Str2)+HRAT) ;
434 MxInCd.l(Str,Str2)$((CdStr(Str)*Nonisoth(Str)) AND Nonisoth(Str2)) =
    0.5*sqrt(Power(Tin.l(Str)- TPch.l(Str2)+ HRAT,2) + Power(epsi,2))
    + 0.5*(Tin.l(Str)-TPch.l(Str2)+HRAT) ;
435 HetCntHt.l(Str2)$Nonisoth(Str2) = SUM(Str$(HtStr(Str)*Nonisoth(Str)),
     FCp.l(Str)*(MxInHt.l(Str,Str2) - MxOutHt.l(Str,Str2))) ;
436 HetCntCd.l(Str2)$Nonisoth(Str2) = SUM(Str$(CdStr(Str)*Nonisoth(Str)),
     FCP.l(Str)*(MxOutCd.l(Str,Str2) - MxInCd.l(Str,Str2))) ;
437 CdUtLd.l = Omega.l + HtUtLd.l ;
438 Z.l = 10*HtUtLd.l + 120*CdUtLd.l ;
439
440 Model Ravi /All/ ;
441 Ravi.holdfixed = 1 ;
442 option minlp = sbb ;
443
444 Solve Ravi using MINLP minimizing Z ;
```

gms/mhex_for_print.gms

B The Python Script

```
'''
@ summary:      Reads text files, return them as lists of text lines.
                Classes for model input (.gms-files) and
                outputs (.lst-files).
                Calculates F, G_y and J_uu.
@ author:       Axel Lodemel Holene
@ organization: Department of Chemical Engineering, NTNU, Norway
@ contact:      axel.holene@gmail.com
@ requires:     Python 2.7 or higher
@ todo:         Make "readFile" more efficient by using i.e.
                iteration with regex.
'''

import re

class AutoVivification(dict):
    """Implementation of perl's autovivification feature."""
    def __getitem__(self, item):
        try:
            return dict.__getitem__(self, item)
        except KeyError:
            value = self[item] = type(self)()
            return value

class pf(float):
    ''' Class for printing readable lists.'''
    def __repr__(self):
        return "%.10e" % self

class solvedModel:
    '''
    Class specifying the values retrieved from the .list file of
    the solved .gms model.
    '''
    def __init__(self, folder, filename):
        '''
        Assigns a name to the model (typically file name),
        then loads the values from "data".
        '''
        data = findData(readFile(folder, filename))
        self.name = filename
        self.inputfile = data.get('Inputfile')
        self.outputfile = data.get('Outputfile')
        self.FN2 = data.get('FN2')
        self.PumpWork = data.get('PumpWork')
        self.Z = data.get('Z')
```

```
47          self.Tin = data.get('Tin')
48          self.Tout = data.get('Tout')
49          self.FCp = data.get('FCp')
50          self.PresS1 = data.get('PresS1')
51          self.TempS4 = data.get('TempS4')
52          self.TempS5 = data.get('TempS5')
53          self.TempS6 = data.get('TempS6')
54      def __repr__(self):
55          return "Class for solved gamsModel"
56
57  def readFile(folder,filename):
58      try:
59          with open(folder+'/'+filename, 'r') as f:
60              res = f.read()
61      except IOError:
62          print "Error: Can't find file or read data named '%s'" \
63              % filename
64      else:
65          print "File named '%s' from the folder '%s' successfully
               loaded" % (filename, folder)
66          f.close()
67      return res
68
69  def findData(aString):
70      oneVar = ['FN2', 'PumpWork', 'PresS1', 'TempS4', 'TempS5', \
71              'TempS6', 'Z']
72      multiVar = ['FCp', 'Tin']
73      streams = ['H1', 'H2', 'H3', 'C1', 'C2']
74      nameList = ['Input', 'Output']
75      ans = AutoVivification()
76      for var in oneVar:
77          try:
78              ans[var] = re.match(r'^(.|\n)*?$\n.*VARIABLE '+var+\
79              '\.L\s*=\s*([0-9.E+-]+)', aString, re.MULTILINE).group(2)
80          except:
81              pass
82      for var in multiVar:
83          for stream in streams:
84              try:
85                  ans[var][stream] = \
86                  re.match(r'^(.|\n)*?$\n.*VARIABLE '+var+\
87                  '\.L\s*\n.*\n.*'+stream+' ([0-9.E+-]+)', aString, \
88                  re.MULTILINE).group(2)
89              except:
90                  pass
91      for aName in nameList:
92          try:
93              ans[aName+'file'] = re.match(r'^(.|\n)*?$\n.*'+aName+\
94              '\s+(\/\S+)+\/(\S+)$', aString, re.MULTILINE).group(3)
```

```
95          except:
96              pass
97      return ans
98
99  def sens(aModelList):
100     '''
101     aModel = solvedModel-object
102     y = [FN2, PumpWork, pS1, TS4, TS5, TS6]
103     d = [H1, ... , C2, FCp(H1), ... FCp(C2)]
104     F = [[dFN2/dTin(H1), dFN2/dTin(H2), ... , dFN2/dFCp(H1), .... ],
105         [dPW/dTin(H1), ... , ]]
106     finds dFN2 and dPW for each model in the list
107     computes dTin(H1) etc. Disregards current iteration and continues
108     looping if dTin(H1) is zero. If it's not, continuing to FCp,
109     repeating.
110     '''
111     y = ['FN2', 'PumpWork', 'PresS1', 'TempS4', 'TempS5', 'TempS6']
112     subs = ['Tin', 'FCp']
113     streams = ['H1', 'H2', 'H3', 'C1', 'C2']
114     F = [[] for _ in range(len(y))]
115     for aModel in aModelList:
116         for y_i in y:
117             dy = float(getattr(aModel,y_i)) - float(getattr(ref,y_i))
118             for sub in subs:
119                 for stream in streams:
120                     dd = float(getattr(aModel,sub)[stream]) - \
121                          float(getattr(ref,sub)[stream])
122                     if dd != 0:
123                         F[y.index(y_i)].append(dy/dd)
124                         break
125                     else:
126                         continue
127     return F
128
129 def gain(aModelList):
130     u = ['FN2', 'PumpWork']
131     y = ['FN2', 'PumpWork', 'PresS1', 'TempS4', 'TempS5', 'TempS6']
132     g = [[0 for _ in range(len(y))] for _ in range(len(u))]
133     for (u_i, aModel) in zip(u,aModelList):
134         du = float(getattr(aModel,u_i)) - float(getattr(ref,u_i))
135         for y_i in y:
136             g[u.index(u_i)][y.index(y_i)] = \
137             (float(getattr(aModel,y_i)) - float(getattr(ref,y_i)))/du
138     return g
139
140 def nonDiagH(aModelList, printing):
141     '''
142     nondiagonal: h12, h21
143     Computing d^2/(dx_i dx_j) by central differences.
```

```
144     aModelList = [fpp, fpm, fmp, fmm]
145     fpp = f((hx)_i+(hx)_j), fpm = f((hx)_i-(hx)_j), etc
146     '''
147     eps = 1.e-8      # numerical tolerance
148     if printing: print "nonDiagH runs: ****************************"
149     hi = abs(float(aModelList[0].FN2) - float(ref.FN2))
150     hj = abs(float(aModelList[0].PumpWork) - float(ref.PumpWork))
151     if hi == 0:
152         step = 4*hj**2
153         if printing: print "hi = 0, step = %4.4e" % step
154     elif hj == 0:
155         step = 4*hi**2
156         if printing: print "hj = 0, step = %4.4e" % step
157     else:
158         step = 4*hi*hj
159     if printing: print "step = %4.4e" % step
160     if printing: print "Z values:\n %.8e, %.8e, %.8e, %.8e" % \
161                 (float(aModelList[0].Z),float(aModelList[1].Z),\
162                 float(aModelList[2].Z),float(aModelList[3].Z))
163     if printing: print "objective = \n %.8e" % \
164                 (float(aModelList[0].Z) - float(aModelList[1].Z) - \
165                  float(aModelList[2].Z) + float(aModelList[3].Z))
166     if printing: print "nonDiagH finished: ************************"
167     return (float(aModelList[0].Z) - float(aModelList[1].Z) - \
168             float(aModelList[2].Z) + float(aModelList[3].Z))/step
169
170 def diagH(aModelList, printing):
171     '''
172     Computes h11, h22. Same infrastructure as nonDiagH.
173     '''
174     eps = 1.e-8      # numerical tolerance
175     hi = abs(float(aModelList[0].FN2) - float(ref.FN2))
176     hj = abs(float(aModelList[0].PumpWork) - float(ref.PumpWork))
177     if printing: print "diagH runs: ******************************"
178     if hi < eps:
179         step = 12*hj**2
180         if printing: print "hi = 0, step = %4.4e" % step
181     elif hj < eps:
182         step = 12*hi**2
183         if printing: print "hj = 0, step = %4.4e" % step
184     else:
185         step = -1
186         if printing: print "h1 and h2 < eps, step = -1"
187     z_pp = float(aModelList[0].Z); z_pm = float(aModelList[1].Z)
188     z_pm = float(aModelList[2].Z); z_mm = float(aModelList[3].Z)
189     z_p = float(aModelList[4].Z); z_m = float(aModelList[5].Z)
190     z = float(ref.Z)
191     if printing: print "Z values:\n %.8e, %.8e, %.8e, %.8e, %.8e" %\
192                 (z_pp, z_p, z, z_m, z_mm)
```

```
193     if printing: print "objective = \n %.8e" % \
194                 (-1*z_pp + 16*z_p - 30*z + 16*z_m - z_mm)
195     if printing: print "diagH finished: ****************************"
196     return (-1*z_pp + 16*z_p - 30*z + 16*z_m - z_mm)/step
197
198 def Hessian(aModelList, printing):
199     '''
200     aModelList = [hlist11, hlist12, hlist21, hlist22]
201     '''
202     return [[diagH(aModelList[0], printing), \
203              nonDiagH(aModelList[1], printing)], \
204             [ nonDiagH(aModelList[2], printing), \
205              diagH(aModelList[3], printing) ] ]
206
207 def printMatrix(nestedList):
208     try:
209         print map(pf,nestedList)
210     except:
211         for line in nestedList:
212             print map(pf,line)
213
214 def sortPrintDict(mydict):
215     for key in sorted(mydict.iterkeys()):
216         print "%s: %s" % (key, mydict[key])
217
218 # Global reference object
219 ref = solvedModel('ref', 'mhex_ref.lst')
220
221 sensitivity = [\
222   solvedModel('sensitivity', 'mhex_sens_H1.lst'), \
223   solvedModel('sensitivity', 'mhex_sens_H2.lst'), \
224   solvedModel('sensitivity', 'mhex_sens_H3.lst'), \
225   solvedModel('sensitivity', 'mhex_sens_C1.lst'), \
226   solvedModel('sensitivity', 'mhex_sens_C2.lst'),\
227   solvedModel('sensitivity', 'mhex_sens_FCP_H1.lst'),\
228   solvedModel('sensitivity', 'mhex_sens_FCP_H2.lst'),\
229   solvedModel('sensitivity', 'mhex_sens_FCP_H3.lst'),\
230   solvedModel('sensitivity', 'mhex_sens_FCP_C1.lst'),\
231   solvedModel('sensitivity', 'mhex_sens_FCP_C2.lst')\
232   ]
233
234 gainModels = \
235     [solvedModel('gain', 'gain_FN2.lst'),
236      solvedModel('gain', 'gain_PW.lst')]
237
238 hessianModels = [\
239     [solvedModel('hessian/h_11', 'hessian_pp.lst'), \
240      solvedModel('hessian/h_11', 'hessian_pm.lst'), \
241      solvedModel('hessian/h_11', 'hessian_mp.lst'), \
```

```
242      solvedModel('hessian/h_11', 'hessian_mm.lst'), \
243      solvedModel('hessian/h_11', 'hessian_p.lst'),  \
244      solvedModel('hessian/h_11', 'hessian_m.lst')], \
245     [solvedModel('hessian/h_12', 'hessian_pp.lst'), \
246      solvedModel('hessian/h_12', 'hessian_pm.lst'), \
247      solvedModel('hessian/h_12', 'hessian_mp.lst'), \
248      solvedModel('hessian/h_12', 'hessian_mm.lst')],\
249     [solvedModel('hessian/h_21', 'hessian_pp.lst'), \
250      solvedModel('hessian/h_21', 'hessian_pm.lst'), \
251      solvedModel('hessian/h_21', 'hessian_mp.lst'), \
252      solvedModel('hessian/h_21', 'hessian_mm.lst')],\
253     [solvedModel('hessian/h_22', 'hessian_pp.lst'), \
254      solvedModel('hessian/h_22', 'hessian_pm.lst'), \
255      solvedModel('hessian/h_22', 'hessian_mp.lst'), \
256      solvedModel('hessian/h_22', 'hessian_mm.lst'), \
257      solvedModel('hessian/h_22', 'hessian_p.lst'),  \
258      solvedModel('hessian/h_22', 'hessian_m.lst')] \
259     ]
260
261 printing = False
262
263 F = sens(sensitivity)
264 print "F = \n", printMatrix(F)
265 Gy = gain(gainModels)
266 print "Gy = \n", printMatrix(Gy)
267 Juu = Hessian(hessianModels, printing)
268 print "Juu =\n", printMatrix(Juu)
```

../python/mhex_print.py

C The Matlab Script

```
clear all
clc
cd('~/Documents/NTNU/Project/matlab/')

%% FYI
% y   = [FN2 PW pS1 TS4 TS5 TS6]'                                    (6x1)
% u   = [FN2 PW]'                                                    (2x1)
% d   = [Tin(H1) ... Tin(C2) FCp(H1) ... FCp(C2)]'                  (10x1)
% ny  = [FN2 PW pS1 TS4 TS5 TS6]'                                    (6x1)
% F   = [dFN2/dd ; dPW/dd]'                                         (10x2)
% Wd  = diag(d_values)                                             (10x10)
% wny = diag(ny_values)                                             (2x2)
% Gy  = [dy/du]                                                     (2x6)

%% Misc data
y = ones(6,1);                  % y   = [FN2 PW pS1 TS4 TS5 TS6]' (6x1)
d = ones(10,1);%d = [Tin(H1) ... Tin(C2) FCp(H1) ... FCp(C2)]' (10x1)
Tin = [298 265 195 220 255]';                % Inlet temperatures [K]
FCp = [3 4 2 3 3.5]';               % Inlet flow heat capacity [kW/K]
TS  = [2.657347e+2 1.514299e+2 2.939971e+2]';   % State temperatures,
                                             % TS = [TS4 TS5 TS6] [K]
pS1  = 7.25345316;                        % State pressure at S1 [bar]

%% Disturbance and measurement error data
d_T   = 10;                      % Absolute temperature disturbance [K]
d_FCp = 0.05;         % Relative flow heat capacity disturbance [kW/K]
n_F = 1e-3;                    % Abolute flow measurment error [kmol/s]
n_w = 0;                   % Absolute pump work measurement error [kW]
n_T = 1;                  % Absolute Temperature measurement error [K]
n_p = 0.01;                % Absolute pressure measurement error [bar]

Wd  = diag([ones(length(Tin),1)*d_T; FCp*d_FCp]);
Wny = diag([n_F  n_w  n_p  ones(1,length(TS))*n_T]');

% Hessian of costfunction. Values given from mhex.py
Juu = [-3.6800349537e+12, 2.0833333334e+06;
       2.0833333334e+06, -1.1666666596e+04];
% E = eig(Juu);
Gy = [1.0000000000e+00, 0.0000000000e+00, -4.1566666668e+01, ...
       2.5000000000e+04, 1.4666666667e+04, -2.6666666666e+04;
       0.0000000000e+00, 1.0000000000e+00, 7.3684210526e+00, ...
       5.8479532183e+01, 0.0000000000e+00, -5.8479532149e+01]';
  % NOTE: transposed

F =  [9.0950000000e-05, 2.2595000000e-04, 1.4895000000e-04, ...
       2.2295000000e-04, 4.8395000000e-04, 2.5130000000e-03, ...
```

```
47        6.3297500000e-03, 3.3495000000e-03, -1.8603333333e-03, ...
48        -1.2517142857e-03 ;
49        5.2394000000e-04, 1.3089400000e-03, 8.6194000000e-04, ...
50        1.2929400000e-03, 2.8019400000e-03, 1.4566266667e-02, ...
51        3.6634700000e-02, 1.9399400000e-02, -1.0773733333e-02, ...
52        -7.2517714286e-03 ;
53        -1.5999999992e-07, -1.5999999992e-07, -1.5999999992e-07, ...
54        -1.5999999992e-07, -1.5999999992e-07, -1.0666666661e-06, ...
55        -7.9999999958e-07, -1.5999999992e-06, -1.0666666661e-06, ...
56        -9.1428571380e-07 ;
57        2.3571710000e+00, 2.2899580000e+00, -1.8894300000e-01, ...
58        -2.8268600000e-01, -6.0713200000e-01, 3.4663793333e+01, ...
59        -7.7379850000e+00, -4.2242600000e+00, 2.3954533333e+00, ...
60        -2.0616057143e+01 ;
61        1.3432440000e+00, 1.3049430000e+00, -1.0766800000e-01, ...
62        -1.6108800000e-01, -3.4597500000e-01, 1.9753300000e+01, ...
63        -4.4095100000e+00, -2.4071900000e+00, 1.3650733333e+00, ...
64        -1.1748125714e+01 ;
65        9.9995600000e-01, -4.5000000000e-05, -4.2000000008e-05, ...
66        -4.2000000008e-05, -4.2000000008e-05, -2.8000000005e-04, ...
67        -2.1000000004e-04, -4.2000000008e-04, -2.8000000005e-04, ...
68        -2.4000000004e-04]';
69  % NOTE: transposed
70
71 %% Calculations Exact Local Method
72
73 Y = [F'*Wd Wny] ;
74 H = Gy'*(Y*Y')^-1 ;
75 %H = ((Y*Y')^(-1)*Gy)' ;
76 P = (Juu^-0.5*Gy'*(Y*Y')^-1*Gy*Juu^-0.5)^(-1);
77
78 %% Loss calulations
79
80 L = (1/(6*(length(y)+length(d))))*...
81     (norm(Juu^0.5*(H*Gy)^(-1)*H*Y,'fro'))^2;
82 L_avg = (1/(6*(length(y)+length(d))))*trace(P);
83 L_w = 0.5*(svds(Juu^0.5*(H*Gy)^(-1)*H*Y,1,'L'))^2;
84
85 %% Exporting as tex
86 H_n = H/norm(H);     % Making H readable. Doesn't affect solution
87
88 cd('~/Documents/NTNU/Project/tex/matlab/')
89 matrix2latexmatrix(F,   'F_val.tex')
90 matrix2latexmatrix(Gy,  'Gy_val.tex')
91 matrix2latexmatrix(Juu, 'Juu_val.tex')
92 matrix2latexmatrix(H_n, 'H_val.tex')
93 matrix2latexmatrix(L,   'L_val.tex')
94 matrix2latexmatrix(L_avg,'Lavg_val.tex')
95 matrix2latexmatrix(L_w, 'Lw_val.tex')
```

```
96
97  %% Null Space Method
98  %H_null = null(F');
99  %L_null = (1/(6*length(y)*length(d)))*...
100 %(norm(Juu^0.5*(H_null*Gy)^(-1)*H_null*Y,'fro'))^2;
101
102 %% Testing
103 H2   = H/norm(H,'fro') ;
104 L_w2 = 0.5*(svds(Juu^0.5*(H2*Gy)^(-1)*H2*Y,1,'L'))^2 ;
105 H3   = H/norm(H) ;
106 L_w3 = 0.5*(svds(Juu^0.5*(H3*Gy)^(-1)*H3*Y,1,'L'))^2 ;
```

../matlab/script.m

D Nifty Trifles

D.1 Python: Autovivification

For storing and easily having available the different temperatures associated with, for instance, T_{in}, using nested dictionaries come in handy. Python has no intuitive native way of nesting dictionaries. However, the programming language Perl has a nice feature called autovivification [Ref. 27]. This can be implemented in Python by adding the class `AutoVivification`, given in Listing 19.

```
1
2 class AutoVivification(dict):
3     """Implementation of perl's autovivification feature."""
4     def __getitem__(self, item):
5         try:
6             return dict.__getitem__(self, item)
7         except KeyError:
8             value = self[item] = type(self)()
```

Listing 19: Perl's autovivification feature as a python class.

D.2 Python: Pretty Float

Printing Python lists when the lists contain multiple digits can produce fairly unreadable printouts. Defining a `PrettyFrint` class, as done in Listing 20 provide the possibility of mapping a list with `PrettyPrint` producing user defined formatting.

```
1
2 class pf(float):
3     ''' Class for printing readable lists.'''
4     def __repr__(self):
```

Listing 20: A `PrettyFloat` class.

D.3 make: Makefile Script

`GAMS` is a command line program which is run through a command shell. When working with numerous files, it becomes convenient to automate the process of running `.gms` files if their corresponding `.lst` file is out of date. To do this, the `make` language comes in handy.

A `Makefile` script was written which contains information about where to look for `.gms` and `.lst` files. The command `make` then checks if the time stamp on the `.lst` file is out of date compared to the `.gms` file, and if necessary runs the `.gms` file to update its output file. The file is run by, in a command shell, standing in the folder where the `Makefile` is located and typing the command `make`.

The `Makefile` script is given in Listing 21.

```
all : *.lst
    @exit 0

%.lst : %.gms
    @tput setaf 6; \
    echo "Target = $(@)"; \
    echo "Depend = $(<)"; \
    tput sgr0; \
    gams $(<) O=$(@)
```

Listing 21: Makefile script used to automatically update `.lst` files.

E An Optimization Example with GAMS

In the process of familiarizing with the GAMS programming language, an exercise from the course "TKP10: Advanced Control" at the Norwegian University Of Science and Technology (NTNU) was solved using GAMS. The exercise text is given below. The proposed GAMS script is given in Listing 22.

Product specification: $1\,\mathrm{kg\,s^{-1}}$ *gasoline with an octane number of at least 98 and max.* 1 % *benzene. Max. production rate of stream 1 is* 0.4. *Assume linear mixing for octane numbers.*

Table E.1: Data for the octane number mixing problem

Stream	Octane #	Benzene cont. [%]	Price \$/kg
1	99	0	$0.1\hat{m}_1$
2	105	0	0.2
3	95	0	0.12
4	99	2	0.185

```
$TITLE Ex1
$OFFSYMXREF
$OFFSYMLIST

OPTION SOLPRINT = ON ;

SETS
    m    flows   / 1*4 /
    i(m) subset  / 2*4 /
;

PARAMETERS
    OCT(m)  the values of the octane numbers
        /   1   99  ,
            2   105 ,
            3   95  ,
            4   99      /
    BENZ(m) the values of benzene content in the flows
        /   1   0   ,
            2   0   ,
            3   0   ,
            4   0.02    /
    P(m)    the prices of the different flows in dollar per kg
        /   1   0.1 ,
            2   0.2 ,
            3   0.12,
```

```
27              4    0.185    /
28
29 VARIABLES
30     COST
31     X(m)
32 ;
33
34 POSITIVE VARIABLES
35     X(m)     flow rate of stream m
36 ;
37
38 EQUATIONS
39     E1,E2,I1,I2,I3
40 ;
41
42 *E1..COST =E= SUM(m, P(m)*X(m));
43 E1..COST =E= P('1')*(1+X('1'))*X('1') + SUM(m$(ORD(m) NE 1), P(m)*X(m
       ));
44 *E1..COST =E= P('1')*(1+X('1'))*X('1') + SUM(i, P(i)*X(i));
45 E2..1 =E= SUM(m, X(m));
46 I1..98 =L= SUM(m, X(m)*OCT(m));
47 I2..0.01 =G= SUM(m, X(m)*BENZ(m));
48 I3..0.4 =G= X('1');
49
50 MODEL JUPITER/ALL/;
51 SOLVE JUPITER USING NLP MINIMIZING COST;
```

Listing 22: Proposed solution with GAMS