Modeling and Optimization of Multistream Heat Exchangers with Phase Transition

Written by:
Axel Lødemel Holene

Supervisors:
Sigurd Skogestad
Johannes Jäschke
Vladimiro 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 measurement error and noise has been constructed with the aim at finding possible output candidates for self-optimizing control. The optimal sensitivity, $F$, output gain matrix, $G^y$, and the Hessian of the cost function, $J_{uu}$ 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 Jä sche 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

Abstract

List of Figures

List of Tables

1 Introduction

2 Process Description
   2.1 Heat Exchanger Model
   2.2 Process Scenario
      2.2.1 Inputs
      2.2.2 Outputs
      2.2.3 Disturbances
      2.2.4 Measurement Error

3 Theory
   3.1 Constrained Optimization
   3.2 Self-optimizing Control
      3.2.1 Operational Objective
      3.2.2 Degree of Freedom Analysis
      3.2.3 Implementation of Optimal Operation
      3.2.4 Throughput Manipulator
      3.2.5 Regulatory Layer
      3.2.6 Supervisory Layer
      3.2.7 Real-time Optimization
   3.3 Heat Exchanger Model
   3.4 Finite Difference Approximations

4 Programming
   4.1 GAMS
   4.2 Modifications Done to mhexcs1v9.gms
      4.2.1 Display variables
      4.2.2 Sensitivity Matrix, $F$
      4.2.3 Gain Matrix, $G^y$
   4.3 Python
      4.3.1 Reading Data
      4.3.2 Classes
      4.3.3 Sensitivity and Gain Matrices
      4.3.4 Hessian of Cost
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>Results</td>
<td>34</td>
</tr>
<tr>
<td>6</td>
<td>Discussion</td>
<td>35</td>
</tr>
<tr>
<td></td>
<td>6.1 Simulations</td>
<td>35</td>
</tr>
<tr>
<td></td>
<td>6.2 Sensitivity</td>
<td>35</td>
</tr>
<tr>
<td></td>
<td>6.2.1 Feed Flow, Pump Work and Pressure at $S_1$</td>
<td>36</td>
</tr>
<tr>
<td></td>
<td>6.2.2 Temperatures</td>
<td>36</td>
</tr>
<tr>
<td></td>
<td>6.3 Gain</td>
<td>37</td>
</tr>
<tr>
<td></td>
<td>6.4 The Hessian of the Cost Function</td>
<td>37</td>
</tr>
<tr>
<td></td>
<td>6.4.1 Loss</td>
<td>37</td>
</tr>
<tr>
<td>7</td>
<td>Conclusion</td>
<td>39</td>
</tr>
<tr>
<td>References</td>
<td>40</td>
<td></td>
</tr>
<tr>
<td>Appendices</td>
<td>42</td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>The GAMS Model mhexcs1v9.gms</td>
<td>42</td>
</tr>
<tr>
<td>B</td>
<td>The Python Script</td>
<td>53</td>
</tr>
<tr>
<td>C</td>
<td>The Matlab Script</td>
<td>59</td>
</tr>
<tr>
<td>D</td>
<td>Nifty Trifles</td>
<td>62</td>
</tr>
<tr>
<td></td>
<td>D.1 Python: Autovivification</td>
<td>62</td>
</tr>
<tr>
<td></td>
<td>D.2 Python: Pretty Float</td>
<td>62</td>
</tr>
<tr>
<td></td>
<td>D.3 make: Makefile Script</td>
<td>62</td>
</tr>
<tr>
<td>E</td>
<td>An Optimization Example with GAMS</td>
<td>64</td>
</tr>
</tbody>
</table>
List of Figures

2.1 A schematic representation of the example MHEX with three hot and two cold flows, and a double loop nitrogen cooling flow. 4
3.1 Block diagram illustrating the principle of self-optimization. 11
3.2 A schematic illustration of how choosing different self-optimizing variables affect the loss. 12
3.3 Schematic representation of the integrated model for simultaneous optimization and heat integration with phase transitions. $H_1$ and $C_2$ are streams without phase transitions while $H_2$ and $C_1$ are streams with phase transitions. 20
6.1 Temperature variations as function of feed specifications. The illustration is given from right to left to coincide with Figure 2.1. 36
6.2 Some caption 38

List of Tables

2.1 Defined states of the nitrogen cooling stream. 5
2.2 Temperature and heat transport data for the hot and cold process streams. 5
E.1 Data for the octane number mixing problem 64
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.](image)

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. $N_{2\text{sub}}, N_{2\text{2p}}$ and $N_{2\text{sup}}$ are subcooled, two-phase and superheated phases of the nitrogen flow, respectively. $T_{\text{d.p.}}$ and $T_{\text{b.p.}}$ are dew point and bubble point temperatures of the nitrogen stream. The coordinate system $(T_{N_2}, L_{\text{mhex}})$, 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.
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 $T_{S2}$ and $T_{S3}$ are available from the mathematical model, but it’s not likely that temperature measurements from inside the heat exchanger are available.
2.2 Process Scenario

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 ($p_{S_1}$, Pres$S1$ in the GAMS model file)
3. temperature at the exit of the first pass through the MHEX ($T_{S_4}$, Temp$S4$ in the GAMS model file)
4. pressure at the exit of the expander ($p_{S_5}$, Pres$S5$ in the GAMS model file)
5. temperature at the exit of the second pass through the MHEX ($T_{S_6}$, Temp$S6$ in the GAMS model file)

In the model file mhexcs1v9.gms the expander outlet pressure, Pres$S5$, 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 $W_{\text{pump}}$ as a degree of freedom is equivalent to using $p_{S_1}$. The final input vector, $u$, is thus given in Equation (2.1).

$$u = \begin{bmatrix} F_{N_2} & W_{\text{pump}} \end{bmatrix}^T$$

(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, $y$, is given in Equation (2.2).

$$
y = \begin{bmatrix} F_{N_2} & W_{\text{pump}} & p_{S_1} & T_{S_4} & T_{S_5} & T_{S_6} \end{bmatrix}^T \tag{2.2}
$$

### 2.2.3 Disturbances

The process streams are prone to disturbances. It is assumed that the temperatures $T_{i}^{m}, i \in \{H_1, \ldots, C_2\}$, 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, $d$, and the magnitude scaling matrix, $W_d$, are given in Equation (2.3) and (2.4), respectively.

$$
d = \begin{bmatrix} \Delta T_{H_1} & \Delta T_{H_2} & \Delta T_{H_3} & \Delta T_{C_1} & \Delta T_{H_1} & \ldots \\
\ldots & \Delta F_{Cp,H_1} & \Delta F_{Cp,H_2} & \Delta F_{Cp,H_3} & \Delta F_{Cp,C_1} & \Delta F_{Cp,C_2} \end{bmatrix}^T \tag{2.3}
$$

$$
W_d = \text{diag} \left( [10 \ldots 10 \ 5.0 F_{Cp,H_1} \ldots \ 5.0 F_{Cp,C_2}] \right) \tag{2.4}
$$

### 2.2.4 Measurement Error

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

$$
n^y = \begin{bmatrix} F_{N_2} & W_{\text{pump}} & p_{S_1} & T_{S_4} & T_{S_5} & T_{S_6} \end{bmatrix}^T \tag{2.5}
$$

$$
W_{n^y} = \text{diag} \left( [10^{-3} \ 0 \ 0.1 \ 1 \ 1 \ 1] \right) \tag{2.6}
$$
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).

\[
\begin{align*}
\min_{\mathbf{x} \in \mathbb{R}^n} & \quad f(\mathbf{x}) \\
\text{subject to} & \quad c_i(\mathbf{x}) = 0, \quad i \in \mathcal{E} \\
& \quad c_i(\mathbf{x}) \geq 0, \quad i \in \mathcal{I}
\end{align*}
\]  

(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, \( \mathbf{x}^* \), 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):
3.1 Constrained Optimization

\[
\nabla_x \mathcal{L}(x^*, \lambda^*) = 0, \\
c_i(x^*) = 0 \quad \forall i \in \mathcal{E}, \\
c_i(x^*) \geq 0 \quad \forall i \in \mathcal{I}, \\
\lambda_i^* \geq 0 \quad \forall i \in \mathcal{I}, \\
\lambda_i^* c_i(x^*) = 0 \quad \forall i \in \mathcal{E} \cup \mathcal{I}, \tag{3.2a-3.2e}
\]

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

\[
\mathcal{L}(x, \lambda) = f(x) - \sum_{i \in \mathcal{E} \cup \mathcal{I}} \lambda_i c_i(x) \tag{3.3}
\]

The active set of any feasible \( x \), denoted \( \mathcal{A}(x) \), consists of equality constraint indices and inequality constraints indices for which \( c_i(x) = 0 \), that is \( \mathcal{A}(x) \triangleq \mathcal{E} \cup \{ i \in \mathcal{I} | c_i(x) = 0 \} \). Given a local solution \( x^* \) the Mangarian-Fromovitz constraint qualification (MFCQ) is defined by linear independence of the equality constraint gradients and the existence of a search direction\(^2\), \( d \), such that \( \nabla c_i(x^*)^T d \leq 0 \), \( \forall i \in \mathcal{A}(x) \). The linear inequality constraint qualification (LICQ) is said to hold if the set of active constraint gradients \( \{ \nabla c_i(x^*), i \in \mathcal{A}(x^*) \} \) 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 \( x^* \) [Ref. 9,20].

Both MFCQ and LICQ link the KKT conditions to the limiting direction \( d \) of a feasible sequence leading to \( x^* \). 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].
\[
\begin{align*}
\min & \quad f(x, y, z) \\
\text{s.t.} & \quad h(x, y, z) = 0, \\
& \quad g(x, y, z) \leq 0, \\
& \quad 0 \leq x \perp y \geq 0, 
\end{align*}
\]

where \( \perp \) is the complementarity operator enforcing at least one of the bounds to be active, \( x \) and \( y \) are complementarity variables, while \( z \) 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. \( x^T y = 0, \ x \geq 0, \ y \geq 0 \), 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 \( L \triangleq J(u, d) - J_{opt}(d) \).
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, \( c + n \) is the linear combination selection of self-optimizing, controlled variables including affecting noise, \( c_s \) denotes the set points for \( c \) given from the optimization, \( u \) is the process inputs, \( d \) is the disturbances, \( n^y \) is the measurement error and noise, \( y \) is the process output while \( y_m \) represents the measured variables, and \( H \)
is the measurement combination matrix. The consequence of choosing different self-optimizing variables is illustrated in Figure 3.2.

\[ H \]

\[ y \]

\[ y_m \]

\[ n^y \]

\[ d \]

\[ c + n \]

\[ c_s \]

\[ u \]

\[ \text{Objective} \]

\[ \text{Optimizer} \]

\[ \text{Controller} \]

\[ \text{Process} \]

**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.
3.2 Self-optimizing Control

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):

$$ J = \frac{F_N N_2 C_{N_2}}{\rho N_2 \cdot 10^3} \cdot 10^3 + \frac{C_e (W_{\text{pump}} - W_{\text{expander}})}{3600 \cdot 10^2} $$  (3.5)

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

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

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

\[
N_{\text{opt}} = N_m - N_0 = N_m - (N_{m0} + N_{y0})
\]  

(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, \( W_{\text{pump}} \), 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, \( c \), affects the loss. There has been developed approaches for selecting \( c \), which render to determining the measurement combination matrix \( H \).

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

**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: \( c = f_c(u, d) \), \( y = f_y(u, d) \) and \( c = h(y) \). These relations were linearized around a nominal point, denoted ‘⋆’, given in Equation (3.7):
\[ \Delta c = G \Delta u + G_d \Delta d \]  
\[ \Delta y = G^y \Delta u + G^y_d \Delta d \]  
\[ \Delta c = H \Delta y \]  

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

\[
L = \frac{1}{2} (u - u_{opt}) J_{uu} (u - u_{opt}) \triangleq \frac{1}{2} z^T z
\]  

Here \( z \triangleq J^{1/2}_{uu} (u - u_{opt}) \) are the loss variables and \( J_{uu} = \partial^2 J / \partial u^T u \) 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):

\[
L_{avg} = \frac{1}{6(n_y + n_d)} \left\| J^{1/2}_{uu} (HG^y)^{-1} HY \right\|_F^2
\]  

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

\[
L_{avg} = \frac{1}{6(n_y + n_d)} \text{trace}(P)
\]  

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

Introducing \( \Delta d \) and \( n^y \) in Equations (3.11) and (3.12):

\[
\Delta d = W_d d'
\]
\[
n^y = W^y_n n^{y'}
\]
where \( \mathbf{d}' \) and \( \mathbf{n}^{y} \) satisfies \( \| [\mathbf{d}' \  \mathbf{n}^{y}]^{T} \|_{\infty} \leq 1 \). The arguments for using the chosen norm are given in Kariwala [Ref. 15]. Finding the optimal \( \mathbf{H} \) implies to minimize the loss \( L_{\text{avg}} \), 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).

\[
\begin{align*}
\min_{\mathbf{H}} & \quad \| \mathbf{H} \mathbf{Y} \|_F \\
\text{s.t.} & \quad \mathbf{H} \mathbf{G}^{y} = \mathbf{J}^{-1/2}_{uu}
\end{align*}
\]

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

\[
\begin{align*}
\min_{\mathbf{H}} & \quad \| \mathbf{H} \mathbf{Y} \|_F \\
\text{s.t.} & \quad \mathbf{H} \mathbf{G}^{y} = \mathbf{Q}
\end{align*}
\]

Kariwala et al. [Ref. 15] and Yelchuru et al. [Ref. 28] shows that the optimal \( \mathbf{H} \)-matrix is non-unique: \( \mathbf{H}_{\text{opt}} = \mathbf{Q} \mathbf{H}_{\text{opt}} \). By choosing \( \mathbf{Q} = ( (\mathbf{G}^{y})^{T} (\mathbf{Y}^{T})^{-1} \mathbf{G}^{y} ) \mathbf{J}^{-1/2}_{uu} \), \( \mathbf{H}_{\text{opt}} \) can be expressed as given in Equation (3.15).

\[
\mathbf{H}_{\text{opt}} = (\mathbf{G}^{y})^{T} (\mathbf{Y}^{T})^{-1}
\]

### 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, $c$, at the optimal setpoints, $c_s$. 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 Exchanger 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).

\[
\begin{align*}
\text{min} & \quad \phi(x, w) \\
\text{s.t.} & \quad h(x, w) = 0, \\
& \quad g(x, w) \leq 0,
\end{align*}
\]

\[
\Omega(x) = \sum_{i \in H} F_i (T_{in}^i - T_{out}^i) - \sum_{j \in C} F_j (T_{in}^j - T_{out}^j) = 0, \\
\text{s.t.} & \quad AP_{p}^C - AP_{p}^H \leq \epsilon, \quad p \in P,
\]

\[
AP_{p}^C = \sum_{j \in C} F_j \left[ \max\{0, (T_{out}^j - (T_p - \Delta T_{min}))\} - \max\{0, T_{in}^j - (T_p - \Delta T_{min})\} \right], \quad p \in P
\]

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

\[
\max\{0, f(x)\} = \frac{1}{2} \left[ (f(x)^2 + \beta^2)^{1/2} + f(x) \right]
\]

\(\epsilon\) (used in Equation (3.16e)) and \(\beta\) 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).

\[
\begin{bmatrix}
Y^V_{\text{in}} \\
T_{\text{in}} \geq T_{\text{d.p.}} \\
T_{\text{sup}}^{\text{in}} = T_{\text{in}} \\
T_{\text{2p}}^{\text{in}} = T_{\text{d.p.}} \\
T_{\text{sub}}^{\text{in}} = T_{\text{b.p.}} \\
\end{bmatrix} \lor \begin{bmatrix}
Y^{VL}_{\text{in}} \\
T_{\text{b.p.}} \leq T_{\text{in}} \leq T_{\text{d.p.}} \\
T_{\text{sup}}^{\text{in}} = T_{\text{d.p.}} \\
T_{\text{2p}}^{\text{in}} = T_{\text{in}} \\
T_{\text{sub}}^{\text{in}} = T_{\text{b.p.}} \\
\end{bmatrix} \lor \begin{bmatrix}
Y^L_{\text{in}} \\
T_{\text{in}} \leq T_{\text{b.p.}} \\
T_{\text{sup}}^{\text{in}} = T_{\text{d.p.}} \\
T_{\text{2p}}^{\text{in}} = T_{\text{b.p.}} \\
T_{\text{sub}}^{\text{in}} = T_{\text{in}} \\
\end{bmatrix}
\]

(3.18)

\[
\begin{bmatrix}
Y^V_{\text{out}} \\
T_{\text{out}} \geq T_{\text{d.p.}} \\
T_{\text{sup}}^{\text{out}} = T_{\text{out}} \\
T_{\text{2p}}^{\text{out}} = T_{\text{d.p.}} \\
T_{\text{sub}}^{\text{out}} = T_{\text{b.p.}} \\
\end{bmatrix} \lor \begin{bmatrix}
Y^{VL}_{\text{out}} \\
T_{\text{b.p.}} \leq T_{\text{out}} \leq T_{\text{d.p.}} \\
T_{\text{sup}}^{\text{out}} = T_{\text{d.p.}} \\
T_{\text{2p}}^{\text{out}} = T_{\text{out}} \\
T_{\text{sub}}^{\text{out}} = T_{\text{b.p.}} \\
\end{bmatrix} \lor \begin{bmatrix}
Y^L_{\text{out}} \\
T_{\text{out}} \leq T_{\text{b.p.}} \\
T_{\text{sup}}^{\text{out}} = T_{\text{d.p.}} \\
T_{\text{2p}}^{\text{out}} = T_{\text{b.p.}} \\
T_{\text{sub}}^{\text{out}} = T_{\text{out}} \\
\end{bmatrix}
\]

(3.19)

The \( Y \)'s are boolean variables and \( V, VL \) and \( L \) are designated to “vapour”, “vapour/liquid” and “liquid”. \( T_{\text{in}} \) and \( T_{\text{out}} \) 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
3.3 Heat Exchanger Model

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 $\alpha$ is assigned as a fictitious temperature drop. The assignments in Equations (3.18) and (3.19) are relaxed. By defining $T^{\text{p.c.}}_{\text{in}}$ and $T^{\text{p.c.}}_{\text{out}}$ from process conditions (p.c.) and the heat integration variables $T^{\text{HI}}_{\text{in}}$ and $T^{\text{HI}}_{\text{out}}$, the disjunctions given in Equation (3.20) are introduced.

$$
\begin{align*}
\Delta T & \geq \alpha \\
T^{\text{HI}}_{\text{in}} & = T^{\text{p.c.}}_{\text{in}} \\
T^{\text{HI}}_{\text{out}} & = T^{\text{p.c.}}_{\text{out}}
\end{align*}
\lor
\begin{align*}
\Delta T & \leq \alpha \\
T^{\text{HI}}_{\text{in}} & = \frac{T^{\text{p.c.}}_{\text{in}} - T^{\text{p.c.}}_{\text{out}}}{2} + \frac{\alpha}{2} \\
T^{\text{HI}}_{\text{out}} & = \frac{T^{\text{p.c.}}_{\text{in}} - T^{\text{p.c.}}_{\text{out}}}{2} - \frac{\alpha}{2}
\end{align*}
$$

(3.20)

The value $\alpha/2$ is added in $T^{\text{HI}}_{\text{in}}$ and subtracted in $T^{\text{HI}}_{\text{out}}$ 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].
3.3 Heat Exchanger Model

**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 without phase transitions while $H_2$ and $C_1$ are streams with phase transitions.
\[ T_{\text{HI}}^{\text{in}} = \max \left( T_{\text{in}}^{\text{p.c.}}, \frac{\frac{T_{\text{in}}^{\text{p.c.}}}{2} + \frac{T_{\text{out}}^{\text{p.c.}}}{2}}{2}, \frac{T_{\text{out}}^{\text{p.c.}}}{2} \right) \]
\[ T_{\text{HI}}^{\text{out}} = \max \left( T_{\text{in}}^{\text{p.c.}}, \frac{\frac{T_{\text{in}}^{\text{p.c.}}}{2} + \frac{T_{\text{out}}^{\text{p.c.}}}{2}}{2}, \frac{T_{\text{out}}^{\text{p.c.}}}{2} \right) \] (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].

\[
\begin{align*}
\min_y & \quad \sum_{i=1}^{N} (x - a_i)(x - a_{i-1})y_i \\
\text{s.t.} & \quad \sum_{i=1}^{N} y_i = 1 \quad (3.22a) \\
& \quad y_i \geq 0 \quad (3.22b) \\
& \quad z = \sum_{i=1}^{N} f_i(x)y_i \quad (3.22c)
\end{align*}
\]

\( N \) 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.

\[
\begin{align*}
\sum_{i=1}^{N} y_i = 1 \quad (3.23a) \\
(x - a_i)(x - a_{i-1}) - \mu_i + \lambda = 0 \quad (3.23b) \\
0 \leq y_i \perp \mu_i \geq 0 \quad (3.23c)
\end{align*}
\]

where \( \lambda \) and \( \mu_i \) 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.

$$\min \left[ -Y^V(T - T_{d.p.}) + Y^{VL}(T - T_{b.p.}) + Y^L(T - T_{b.p.}) \right]$$ (3.24a)

subject to

$$Y^V + Y^{LV} + Y^L = 1$$ (3.24b)

$$Y^V \geq 0, Y^{LV} \geq 0, Y^L \geq 0$$ (3.24c)

With the optimality conditions in Equation (3.25)

$$Y^V + Y^{LV} + Y^L = 1$$ (3.25a)

$$-(T - T_{d.p.}) - \mu^V + \lambda = 0$$ (3.25b)

$$-(T_{d.p.} - T)(T - T_{b.p.}) - \mu^{VL} + \lambda = 0$$ (3.25c)

$$-(T_{b.p.} - T) - \mu^L + \lambda = 0$$ (3.25d)

$$0 \leq Y^V \perp \mu^V \geq$$ (3.25e)

$$0 \leq Y^{LV} \perp \mu^{VL} \geq$$ (3.25f)

$$0 \leq Y^L \perp \mu^L \geq$$ (3.25g)

### 3.4 Finite Difference Approximations

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

$$\frac{\partial^2 f}{\partial x_i^2} = \frac{-f(x + 2h_i e_i) + 16 f(x + h_i e_i) - 30 f(x) + 16 f(x - h_i e_i) - f(x - 2h_i e_i)}{12h_i^2}$$ (3.26a)

$$\frac{\partial^2 f}{\partial x_i \partial x_j} = \frac{f(x + h_i e_i + h_j e_j) - f(x + h_i e_i - h_j e_j) - f(x - h_i e_i + h_j e_j) + f(x - h_i e_i - h_j e_j)}{4h_i h_j}$$ (3.26b)

Here, $f$ is a function, $x$ the function variables, $h$ represent step lengths and $e$ 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.

```gams
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 (−∞ to +∞), positive (0 to +∞), negative (−∞ 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.

```gams
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.

\begin{lstlisting}
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/
;
\end{lstlisting}

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

Using tables for the assignment of data is very efficient for two-dimensional struc-
tures. 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

\begin{lstlisting}
Table d(i,j) "distance in thousands of miles"
seattle 2.5 1.7 1.8
san-diego 2.5 1.8 1.4;
\end{lstlisting}

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.

\begin{lstlisting}
SCALARS
HRAT  "Heat recovery approach temperature" /4/
epsi  "epsilon for smooth approximation" /0.000005/
Could also be written 5e-6
;
\end{lstlisting}

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

Equations  Equations in GAMS must first be declared and then defined. Declara-
tion 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.
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.

---

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

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

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

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

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

```gams
1 HetLd.lo(Str) = 0.0 ;
2 HetLd.up(Str) = 1000.0 ;
3 Tin.fx(‘H1’) = 298 ;
4 Tout.fx(‘H1’) = 250 ;
5 TempS4.l = 180 ;
6 TempS5.l = TempS4.l*(PresS5/PresS1.l)**((gamma-1)/gamma) ;
```
4.2 Modifications Done to 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 minimizing 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
$eolcom // Defining end of line comments delimiter
$inlinecom /* */ Defining inline comments delimiters
option iterlim = 5e5 ; Iteration limit (= 5 \cdot 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
4.2 Modifications Done to mhexcs1v9.gms

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.

```
```

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

4.2.2 Sensitivity Matrix, F

To calculate the sensitivity matrix, \( F = \partial y_{opt}/\partial d^T \), 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, G^y

To produce the output gain matrix, \( G^y = \partial y/\partial u^T \), and the Hessian of the cost function, \( J_{uu} \), perturbations were made around the optimal input, \( u^* \), 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
EqControl1
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);
```

28
Removing degrees of freedom resulted in large objective function values, $Z \approx 5 \cdot 10^5 Z_{opt}$. This problem was solved by changing the initial point of the calculation, as given in Listing 14.

### Listing 14: Optimal initial values.

- $\text{FN2.l} = 0.02959005$
- $\text{PresS1.l} = 7.25345316$
- $\text{TempS4.l} = 265.7347$
- $\text{TempS6.l} = 293.9971$
- $\text{Tboil.l} = 98.999995$
- $\text{YLoc.l('C4','C3')} = 0$; Changed from 1 in mhexcs1v9.gms
- $\text{YLoc.l('C4','C6')} = 0$; Changed from 1 in mhexcs1v9.gms

## 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 $\text{mhex.py}$ 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.

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

---

5.3.2 Python 4 PROGRAMMING
4.3 Python Programming

```python
print "Error: Can’t find file or read data named '%%s'" \
  % filename
else:
  print "File named '%%s' from the folder '%%s' successfully loaded" % (filename, folder)
f.close()
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.

```python
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')
      self.Tin = data.get('Tin')
      self.Tout = data.get('Tout')
      self.FCp = data.get('FCp')
      self.PresS1 = data.get('PresS1')
      self.TempS4 = data.get('TempS4')
      self.TempS5 = data.get('TempS5')
      self.TempS6 = data.get('TempS6')
```

30
4.3 Python

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

Listing 16: The python class solvedModel.

```python
def findData(aString):
    oneVar = ['FN2', 'PumpWork', 'PresS1', 'TempS4', 'TempS5', 'TempS6', 'Z']
multiVar = ['FCp', 'Tin']
streams = ['H1', 'H2', 'H3', 'C1', 'C2']
namelist = ['Input', 'Output']
ans = AutoVivification()
for var in oneVar:
    try:
        ans[var] = re.match(r'\n.*VARIABLE '+var+'.\n.*s*=s*([0-9.E-]+)', aString, re.MULTILINE).group(2)
    except:
        pass
for var in multiVar:
    for stream in streams:
        try:
            ans[var][stream] = re.match(r'\n.*VARIABLE '+var+'.\n.*s*\n.*s*'+stream+'.\n.*s*([0-9.E-]+)', aString, re.MULTILINE).group(2)
        except:
            pass
for aName in namelist:
    try:
        ans[aName+'file'] = re.match(r'\n.*'+aName+"\n.*s+(\n.*S)+\n.*s+\n.*S', aString, re.MULTILINE).group(3)
    except:
        pass
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 $\text{sens}$ 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. $\text{sens}$ 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 non-diagonal 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 $J_{uu}$. 

32
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 $F$, $G^y$ and $J_{uu}$ 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 $F$ is provided. It was considered easier to measure $F$ directly from simulations.

$$F = \begin{bmatrix} 9.095 \cdot 10^{-5} & 0.00052394 & -1.6 \cdot 10^{-7} & 2.3572 & 1.3432 & 0.99996 \\ 0.00022595 & 0.0013089 & -1.6 \cdot 10^{-7} & 2.29 & 1.3049 & -4.5 \cdot 10^{-5} \\ 0.00014895 & 0.00086194 & -1.6 \cdot 10^{-7} & -0.18894 & -0.10767 & -4.2 \cdot 10^{-5} \\ 0.00002295 & 0.0012929 & -1.6 \cdot 10^{-7} & -0.28269 & -0.16109 & -4.2 \cdot 10^{-5} \\ 0.000048395 & 0.0028019 & -1.6 \cdot 10^{-7} & -0.60713 & -0.34597 & -4.2 \cdot 10^{-5} \\ -0.0012517 & 0.0072518 & -9.1429 \cdot 10^{-7} & 20.6161 & -11.7481 & -0.00024 \end{bmatrix}$$

(5.1a)

$$G^y = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ -41.5667 & 7.3684 \\ 25000 & 58.4795 \\ 14666.6667 & 0 \\ -26666.6667 & -58.4795 \end{bmatrix}$$

(5.1b)

$$J_{uu} = \begin{bmatrix} -3680.034953700 & 2083333.3334 \\ 2083333.3334 & -11666.6666 \end{bmatrix}$$

(5.1c)

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

$$H = \begin{bmatrix} 0.90999 & -0.15391 & -0.36886 & -6.6429 \cdot 10^{-5} & 0.00033512 & -0.00052632 \\ -0.1532 & 0.027171 & 0.065385 & 1.2842 \cdot 10^{-5} & -2.2254 \cdot 10^{-5} & -1.1793 \cdot 10^{-6} \end{bmatrix}$$

(5.2a)

$$L_{avg} = \begin{bmatrix} 19.27.6592 \end{bmatrix}$$

(5.2b)

$$L_{worst} = \begin{bmatrix} 92.5276336 \end{bmatrix}$$

(5.2c)
6 Discussion

6.1 Simulations

The optimal solution for the unaltered file, \texttt{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 \texttt{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 \texttt{FN2} and \texttt{PumpWork} at it’s optimal values, found from \texttt{mhexcs1T9.gms}, the objective function, \(z\), yielded a better solution compared to the objective function value found from the “clean” optimization in \texttt{mhexcs1v9.gms}, \(z_{\text{opt}}\).

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 \texttt{FN2} direction. For \texttt{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 \(J_{uu}\) are discussed in Chapter 6.4.

The \texttt{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 \texttt{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, \(F\), 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](image)

*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.
6.3 Gain

The temperature at the exit of the nitrogen cooling flow, $T_{S6}$, is almost completely insensitive to disturbances. An exception is for disturbances to the temperature $T_{H1}$. 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 $T_{H1}^{in}$ governs this sensitivity, as the exit temperature, $T_{S6}$, is close to the $H_1$ inlet temperature. If this is true, and the inlet temperature for $H_1$ is known, $T_{S6}$ 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 $\Delta T_{S6}/\Delta FN2$ is very high, and is thus backing up $T_{S6}$ as being a self-optimizing variable candidate.

6.4 The Hessian of the Cost Function

The positive elements in $J_{uu}$ 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 Pump Work direction.

6.4.1 Loss

As $J_{uu}$ 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 $J_{uu}$. 

37
6.4 The Hessian of the Cost Function

Cost, $J$

$J \approx 10^5 J_{opt}$

$J < J_{opt}$

Steplength, $h$

$h \approx 0.5 \cdot 10^{-7}$

$h \approx 10^{-4}$

Figure 6.2: Some caption
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 $W_{\text{pump}}$, denoted $\text{FN2}$ and $\text{PumpWork}$ in the GAMS model, respectively. When deciding data for finding $F$, $G^y$ and $J_{uu}$ from the simulations, it was observed that a better optimum could be obtained by fixing $\text{FN2}$ and $\text{PumpWork}$ at it’s optimal values, and giving the optimal point as a starting point for the solver. $J_{uu}$ 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


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 bubble 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
Sofflisting
$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/
Tref "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/
PressS5 "pressure at state S5 in bar" /1.01325/
gamma "Cp/Cv for diatomic 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/
Simple(Str) "basic streams" /H1,H2,H3,C1,C2/
Isoth(Str) "isothermal streams" /C4/
CC1 "Constants for CP of N2" /C1,C2,C3,C4,C5,C6,C7/
NonIsoth(Str)
;
NonIsoth(Str) = Str(Str) - Isoth(Str) ;
Display Nonisoth ;
ALIAS (Str,Str2,Str3)
;
PARAMETERS
*FCp(Str) "unit is kW/K" /H1 = 3, H2 = 4, H3 = 2, C1 = 3, C2 = 3.5/
*Tin(Str) "unit is K" /H1 = 298, H2 = 265, H3 = 195, C1 = 220, C2 = 255/
*Tout(Str) "unit is K" /H1 = 250, H2 = 180, H3 = 150, C1 = 245, C2 = 280/
Cpconst(CC1) "for Cp = kJ/kmol K, T = K" /C1 = 29.105, C2 = 8.6149, C3 = 1701.6, C4 = 0.10347, C5 = 909.79/
TSatconst(CC1) "for Tsat, in K, in bar" /C1 = 46.76907454, C2 = -1084.1, C5 = -8.3144, C6 = 0.044127, C7 = 1/
LtHtconst(CC1) "in MJ/kmol, T in K" /C1 = 7.4905, C2 = 0.40406, C3 = -0.317, C4 = 0.27343/
;
EnthS1 = CpConst('C1')*(TN2Inlet - Tref) + CpConst('C2')*CpConst('C3')*(1/tanh(CpConst('C3')/TN2Inlet) - 1/tanh(CpConst('C3')/Tref)) - CpConst('C4')*CpConst('C5')*(tanh(CpConst('C5')/TN2Inlet) - tanh(CpConst('C5')/Tref)) - (LtHtconst('C1')*(1-TrS1)**(LtHtconst('C2') + LtHtconst('C3') - TrS1 + LtHtconst('C4')*(TrS1**2)))*1000 ;
Display EnthS1 ;

VARIABLES
MxInHt(Str,Str2)
MxOutHt(Str,Str2)
MxOutCd(Str,Str2)
MxInCd(Str,Str2)
HetCntHt(Str)
HetCntCd(Str)
TPch(Str)
HetLd(Str)
HtUtLd
CdUtLd
Omega  
FN2 "in kmol/s"  
PresS1 "Pressure at state S1"  
PumpWork "in kW"  
TempS4 "Temperature at state S4"  
TempS5 "Temperature at state S5"  
TempS6 "Temperature at state S6"  
Tboil "boiling point at pressure of S1"  
EnthS4 "enthalpy at S4 in kJ/kmol"  
EnthS3 "enthalpy at S3 in kJ/kmol"  
EnthS2 "enthalpy at S2 in kJ/kmol"  
EnthS5 "enthalpy at S5 in kJ/kmol"  
EnthS6 "enthalpy at S6 in kJ/kmol"  
TrS2 "reduced temperature"  
Tin(Str)  
Tout(Str)  
FCp(Str)  
ExpdrWork  
contri(Str,Str2)  
dummy1  
;  
VARIABLES  
Z ;  

BINARY VARIABLES  
YLoc(Str,Str2) "1 indicates that it is above the pinch and contributes while 0 means no contribution"  
;  
EQUATIONS  
EqFF1(Str)  
EqFF2(Str)  
EqOmega  
EqTPch1(Str)  
EqTPch2(Str)  
EqMxInHt(Str,Str2)  
EqMxOutHt(Str,Str2)  
EqMxOutCd(Str,Str2)  
EqMxInCd(Str,Str2)  
EqHetCntHt(Str)  
EqHetCntCd(Str)  
EqHtUtLd(Str)  
EqCdUtLd  
EqPumpWork  
Eqadiaexpn  
EqTboil  
EqPhasel  
EqEnthS3
EqEnths4
EqEnths5
EqEnths6
EqTrs2
EqEnths2
EqTinC3
EqToutC3
EqTinC4
EqToutC4
EqTinC5
EqToutC5
EqTinC6
EqToutC6
EqHtLdC3
EqHtLdC4
EqHtLdC5
EqHtLdC6
Eqrefri
EqExpdrWork
Eqlogic1(Str,Str2)
Eqlogic11(Str,Str2)
Eqlogic2(Str,Str2)
Eqlogic3(Str,Str2)
Eqlogic4(Str,Str2)
;
EqFF1(Str)$((HtStr(Str)*Nonisoth(Str)))..
  FCp(Str)*(Tin(Str) − Tout(Str)) =E= HetLd(Str) ;
EqFF2(Str)$((CdStr(Str)*Nonisoth(Str)))..
  FCp(Str)*(Tout(Str) − Tin(Str)) =E= HetLd(Str) ;
EqOmega..
  Omega =E= SUM(Str$(HtStr(Str)$), HetLd(Str)) − SUM(Str$(CdStr(Str)$), HetLd(Str)) ;
EqTpch1(Str)$((HtStr(Str)))..
  TPch(Str) =E= Tin(Str) ;
EqTpch2(Str)$((CdStr(Str)))..
  TPch(Str) =E= Tin(Str) + HRAT ;
EqMxInHt(Str,Str2)$((HtStr(Str)$)*Nonisoth(Str)))..
  MxInHt(Str,Str2) =E= 0.5*sqrt(Power(Tin(Str)−TPch(Str2),2) + Power(eps1,2)) + 0.5*(Tin(Str)−TPch(Str2)) ;
EqMxOutHt(Str,Str2)$((HtStr(Str)$)*Nonisoth(Str)))..
  MxOutHt(Str,Str2) =E= 0.5*sqrt(Power(Tout(Str)−TPch(Str2),2) + Power(eps1,2)) + 0.5*(Tout(Str)−TPch(Str2)) ;
EqMxOutCd(Str,Str2)$((CdStr(Str)$)*Nonisoth(Str)))..
MxOutCd(Str,Str2) =E= 0.5*sqrt(Power(Tout(Str) - TPch(Str2) + HRAT,2) + Power(epsi,2)) + 0.5*(Tout(Str) - TPch(Str2) + HRAT) ;
EqMxInCd(Str,Str2)$((CdStr(Str)*Nonisoth(Str)))..
MxInCd(Str,Str2) =E= 0.5*sqrt(Power(Tin(Str) - TPch(Str2) + HRAT,2) + Power(epsi,2)) + 0.5*(Tin(Str) - TPch(Str2) + HRAT) ;
EqHetCntHt(Str2)..
HetCntHt(Str2) =E= SUM(Str$(HtStr(Str)*Nonisoth(Str)), FCP(Str)* (MxInHt(Str,Str2) - MxOutHt(Str,Str2)));
Eqlogic1(Str,Str2)$((CdStr(Str)*Isoth(Str))).. 
contri(Str,Str2) - HetLd(Str) =L= 10000*(1 - YLoc(Str,Str2)) ;
Eqlogic11(Str,Str2)$((CdStr(Str)*Isoth(Str)))..
contri(Str,Str2) - HetLd(Str) =G= -10000*(1 - YLoc(Str,Str2)) ;
Eqlogic2(Str,Str2)$((CdStr(Str)*Isoth(Str)))..
contri(Str,Str2) =L= 10000*YLoc(Str,Str2) ;
Eqlogic3(Str,Str2)$((CdStr(Str)*Isoth(Str)))..
Tin(Str) - TPch(Str2) =G= -500*(1 - YLoc(Str,Str2)) ;
Eqlogic4(Str,Str2)$((CdStr(Str)*Isoth(Str)))..
Tin(Str) =L= TPch(Str2) - epsi + 500*YLoc(Str,Str2) ;
EqHetCntCd(Str2)...
HetCntCd(Str2) =E= SUM(Str$(CdStr(Str)*Nonisoth(Str)), FCP(Str) * (MxOutCd(Str,Str2) - MxInCd(Str,Str2)) + SUM(Str$(CdStr(Str)*Isoth(Str)),contri(Str,Str2)) ;
EqHtUtLd(Str2)...
HtUtLd =G= HetCntCd(Str2) - HetCntHt(Str2) ;
EqCdUtLd..
CdUtLd =E= Omega + HtUtLd ;
EqPumpWork..
PumpWork =E= (PresS1 - PN2Inlet)*FN2*MWN2*100/(PumpEff*DenLiqN2) ;
Eqdiaexpn..
(gamma-1)*log(PresS1) + gamma*log(TempS5) =E= (gamma-1)*log(PresS5) + gamma*log(TempS4) ;
EqTboil..
log(PresS1) =E= TSatconst(‘C1’) + TSatconst(‘C2’)/Tboil + TSatconst(‘C5’)*log(Tboil) + TSatconst(‘C6’)*Tboil ;
EqPhase1..
TempS4 =G= Tboil ;
EqEnthS3..
EnthS3 =E= CpConst(‘C1’)*(Tboil - Tref) + CpConst(‘C2’)*CpConst(‘C3’)*(1/tanh(CpConst(‘C3’)/Tboil) - 1/tanh(CpConst(‘C3’)/Tref))
- CpConst(‘C4’)*CpConst(‘C5’)*(tanh(CpConst(‘C5’)/Tboil) - tanh(CpConst(‘C5’)/Tref)) ;
EqEnthS4..
EnthS4 =E= CpConst('C1')*(TempS4 - Tref) + CpConst('C2')*CpConst('C3')*(1/tanh(CpConst('C3')/TempS4) - 1/tanh(CpConst('C3')/Tref)) - CpConst('C4')*CpConst('C5')*(tanh(CpConst('C5')/TempS4) - tanh(CpConst('C5')/Tref)) ;

EqEnthS5..
EnthS5 =E= CpConst('C1')*(TempS5 - Tref) + CpConst('C2')*CpConst('C3')*(1/tanh(CpConst('C3')/TempS5) - 1/tanh(CpConst('C3')/Tref)) - CpConst('C4')*CpConst('C5')*(tanh(CpConst('C5')/TempS5) - tanh(CpConst('C5')/Tref)) ;

EqEnthS6..
EnthS6 =E= CpConst('C1')*(TempS6 - Tref) + CpConst('C2')*CpConst('C3')*(1/tanh(CpConst('C3')/TempS6) - 1/tanh(CpConst('C3')/Tref)) - CpConst('C4')*CpConst('C5')*(tanh(CpConst('C5')/TempS6) - tanh(CpConst('C5')/Tref)) ;

EqExpdrWork..
ExpdrWork =E= (EnthS4 - EnthS5)*FN2*ExpdrEff ;

EqTrS2..
TrS2 =E= Tboil/Tc ;

EqEnthS2..
EnthS2 =E= EnthS3 - (LtHtconst('C1')*(1-TrS2)**(LtHtconst('C2') + LtHtconst('C3')*TrS2 + LtHtconst('C4')*(TrS2**2)))*1000 ;

* inlet temperature data for C3 stream
EqTinC3..
Tin('C3') =E= TN2Inlet ;

* outlet temperature data for C3 stream
EqToutC3..
Tout('C3') =E= Tboil ;

* inlet temperature data for C4 stream
EqTinC4..
Tin('C4') =E= Tboil ;

* outlet temperature data for C4 stream
EqToutC4..
Tout('C4') =E= Tboil ;

* inlet temperature data for C5 stream
EqTinC5..
Tin('C5') =E= Tboil ;

* outlet temperature data for C5 stream
EqToutC5..
Tout('C5') =E= TempS4 ;

* inlet temperature data for C6 stream
EqTinC6..
Tin('C6') =E= TempS5 ;

* outlet temperature data for C6 stream
EqToutC6..
Tout('C6') =E= TempS6 ;
EqHtLdC3.. HetLd('C3') =E= (EnthS2 - EnthS1)*FN2 ;
EqHtLdC4.. HetLd('C4') =E= (EnthS3 - EnthS2)*FN2 ;
EqHtLdC5.. HetLd('C5') =E= (EnthS4 - EnthS3)*FN2 ;
EqHtLdC6.. HetLd('C6') =E= (EnthS6 - EnthS5)*FN2 ;
Eqrefri.. dummy1 =E= HetLd('C3') + HetLd('C4') + HetLd('C5') + HetLd('C6') ;

EQUATIONS

obj ;
obj.. Z =E= 1000*HtUtLd + 1000*CdUtLd + FN2*MWN2*1000*costN2/(100*DenLiqN2) + costElec*(Pumpwork - ExpdrWork)/(3600*100) ;

* Bounds
HetLd.lo(Str) = 0.0 ;
HetLd.up(Str) = 1000.0 ;
Fcp.lo(Str) = 0.0001 ;
Fcp.up(Str) = 1000 ;
FN2.lo = 0.0 ;
FN2.up = 10.0 ;
PresS1.lo = PN2Inlet ;
PresS1.up = Pmax ;
PumpWork.lo = 0 ;
PumpWork.up = 10000 ;
TempS4.lo = 75 ;
TempS4.up = 300 ;
* to ensure that expander outlet is not in 2 phase
TempS5.lo = 77.36 ;
TempS5.up = 300 ;
Tboil.lo = 50 ;
Tboil.up = 300 ;
TempS6.lo = 77.36 ;
TempS6.up = 300 ;
TrS2.lo = 0.1 ;
TrS2.up = 1.1 ;
EnthS2.lo = -50000 ;
EnthS2.up = 0 ;
EnthS3.lo = -50000 ;
EnthS3.up = 0 ;
EnthS4.lo = -50000 ;
EnthS4.up = 0 ;
EnthS5.lo = -50000 ;

49
EnthS5.up = 0;
EnthS6.lo = -50000;
EnthS6.up = 0;
contri.lo(Str,Str2) = 0.0;
contri.up(Str,Str2) = 10000.0;
CdUtLd.lo = -0.0001;
HtUtLd.lo = -0.0001;

* Fixed

* fixing the Tin, Tout and FCp for simple streams
* unit is kW/K
FCp.fx('H1') = 3;
FCp.fx('H2') = 4;
FCp.fx('H3') = 2;
FCp.fx('C1') = 3;
FCp.fx('C2') = 3.5;

* unit is K
Tin.fx('H1') = 298;
Tin.fx('H2') = 265;
Tin.fx('H3') = 195;
Tin.fx('C1') = 220;
Tin.fx('C2') = 255;

* unit is K
Tout.fx('H1') = 250;
Tout.fx('H2') = 180;
Tout.fx('H3') = 150;
Tout.fx('C1') = 245;
Tout.fx('C2') = 280;

* temporary fixing variables
FN2.l = 0.04;
PresS1.l = 11;
TempS4.l = 180;
TempS6.l = 250;

* Initial Point

HtUtLd.l = 1000;
Tboil.l = 105;

* logic
YLoc.l('C4','H1') = 0;
YLoc.l('C4','H2') = 0;
YLoc.l('C4','H3') = 0;
YLoc.l('C4','C1') = 0;
YLoc.l('C4','C2') = 0;
YLoc.l('C4','C3') = 1;
YLoc.l('C4','C4') = 0;
YLoc.l('C4','C5') = 0;
YLoc.l('C4','C6') = 1;

PumpWork.l = (PresS1.l − PN2Inlet)*FN2.l*MWN2*100/(PumpEff*DenLiqN2);

TempS5.l = TempS4.l*(PresS5/PresS1.l)**((gamma − 1)/gamma);

EnthS3.l = CpConst('C1')*(Tboil.l − Tref) + CpConst('C2')*CpConst('C3')*(1/tanh(CpConst('C3')/Tboil.l) − 1/tanh(CpConst('C3')/Tref));

EnthS4.l = CpConst('C1')*(TempS4.l − Tref) + CpConst('C2')*CpConst('C3')*(1/tanh(CpConst('C3')/TempS4.l) − 1/tanh(CpConst('C3')/Tref));

ExpdrWork.l = (EnthS5.l − EnthS4.l)*FN2.l*ExpdrEff;

TrS2.l = Tboil.l/Tc;

EnthS2.l = EnthS3.l − (LtHtconst('C1')*(1−TrS2.l)**(LtHtconst('C2') + LtHtconst('C3')*TrS2.l + LtHtconst('C4')*(TrS2.l**2)))*1000;

Tin.l('C3') = TN2Inlet;
Tout.l('C3') = Tboil.l;
Tin.l('C4') = Tboil.l;
Tout.l('C4') = Tboil.l;
Tin.l('C5') = Tboil.l;
Tout.l('C5') = TempS4.l;
Tin.l('C6') = TempS5.l;
Tout.l('C6') = TempS6.l;

HetLd.l('C3') = (EnthS2.l − EnthS1)*FN2.l;
HetLd.l('C4') = (EnthS3.l − EnthS2.l)*FN2.l;
HetLd.l('C5') = (EnthS4.l − EnthS3.l)*FN2.l;
HetLd.l('C6') = (EnthS6.l − EnthS5.l)*FN2.l;
dummy1.l = HetLd.l('C3') + HetLd.l('C4') + HetLd.l('C5') + HetLd.l('C6');
HetLd.l(Str)$(HtStr(Str)*Nonisoth(Str)) = FCp.l(Str)*(Tin.l(Str) − Tout.l(Str));
HetLd.l(Str)$(CdStr(Str)*Nonisoth(Str)) = FCp.l(Str) *(Tout.l(Str) − Tin.l(Str)) ;

FCp.l(Str)$(HtStr(Str)*NOT Simple(Str))*Nonisoth(Str)) = HetLd.l(Str)/(Tin.l(Str) − Tout.l(Str)) ;

HetLd.l(Str)$(CdStr(Str)*NOT Simple(Str))*Nonisoth(Str)) = HetLd.l(Str)/(Tout.l(Str) − Tin.l(Str)) ;

Omega.l = SUM(Str$(HtStr(Str)*Nonisoth(Str)), HetLd.l(Str)) − SUM(Str$(CdStr(Str)*Nonisoth(Str)), HetLd.l(Str)) ;

TPch.l(Str)$(HtStr(Str)*Nonisoth(Str)) = Tin.l(Str) ;

TPch.l(Str)$(CdStr(Str)*Nonisoth(Str)) = Tin.l(Str) + HRAT ;

MxInHt.l(Str,Str2)$((HtStr(Str)*Nonisoth(Str)) AND Nonisoth(Str2)) = 0.5*sqrt(Power(Tin.l(Str)− TPch.l(Str2),2) + Power(eps1,2)) + 0.5*(Tin.l(Str)− TPch.l(Str2)) ;

MxOutHt.l(Str,Str2)$((HtStr(Str)*Nonisoth(Str)) AND Nonisoth(Str2)) = 0.5*sqrt(Power(Tout.l(Str)− TPch.l(Str2),2) + Power(eps1,2)) + 0.5*(Tout.l(Str)− TPch.l(Str2)) ;

MxOutCd.l(Str,Str2)$((CdStr(Str)*Nonisoth(Str)) AND Nonisoth(Str2)) = 0.5*sqrt(Power(Tout.l(Str)− TPch.l(Str2)+ HRAT,2) + Power(eps1,2)) + 0.5*(Tout.l(Str)− TPch.l(Str2)+HRAT) ;

HetCntHt.l(Str2)$Nonisoth(Str2) = SUM(Str$(HtStr(Str)*Nonisoth(Str)), FCp.l(Str) *(MxInHt.l(Str,Str2) − MxOutHt.l(Str,Str2));

HetCntCd.l(Str2)$Nonisoth(Str2) = SUM(Str$(CdStr(Str)*Nonisoth(Str)), FCp.l(Str) *(MxOutCd.l(Str,Str2) − MxInCd.l(Str,Str2));

CdUtLd.l = Omega.l + HtUtLd.l ;

Z.l = 10*HtUtLd.l + 120*CdUtLd.l ;

Model Ravi /All/ ;
Ravi.holdfixed = 1 ;
option minlp = sbb ;

Solve Ravi using MINLP minimizing Z ;
gms/mhex_for_print.gms
B The Python Script

```python
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):
        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')
```

THE PYTHON SCRIPT

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

def readFile(folder, filename):
    try:
        with open(folder + '/' + filename, 'r') as f:
            res = f.read()
    except IOError:
        print "Error: Can't find file or read data named '%s'", % filename
    else:
        print "File named '%s' from the folder '%s' successfully loaded" % (filename, folder)
        f.close()
    return res

def findData(aString):
    oneVar = ['FN2', 'PumpWork', 'PresS1', 'TempS4', 'TempS5', 'TempS6', 'Z']
    multiVar = ['FCp', 'Tin']
    streams = ['H1', 'H2', 'H3', 'C1', 'C2']
    nameList = ['Input', 'Output']
    ans = AutoVivification()
    for var in oneVar:
        try:
            ans[var] = re.match(r'ˆ(\.|\n)*?\n.*VARIABLE '+var+' \'.\Ls*\s*([0-9.E+\-%]+)', aString, re.MULTILINE).group(2)
        except:
            pass
    for var in multiVar:
        for stream in streams:
            try:
                ans[var][stream] = \
                re.match(r'ˆ(\.|\n)*?\n.*VARIABLE '+var+' \'.\Ls*\n.*'+stream+' ([0-9.E+\-%]+)', aString, \
                re.MULTILINE).group(2)
            except:
                pass
    for aName in nameList:
        try:
            ans[aName+'file'] = re.match(r'ˆ(\.|\n)*?\n.*'+aName+' \'.\s+(\\S+)+(\\S+)$', aString, re.MULTILINE).group(3)
```

54
except:
    pass
return ans

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 in:
        dy = float(getattr(aModel,y)) - float(getattr(ref,y))
        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)].append(dy/dd)
                    break
            else:
                continue
    return F

def gain(aModelList):
    
    u = ['FN2', 'PumpWork']
    y = ['FN2', 'PumpWork', 'PresS1', 'TempS4', 'TempS5', 'TempS6']
    g = [[0 for _ in range(len(y)) for _ in range(len(u))]
    for (u, aModel) in zip(u, aModelList):
        du = float(getattr(aModel,u)) - float(getattr(ref,u))
        for y in y:
            g[u.index(u)][y.index(y)] = \
            (float(getattr(aModel,y)) - float(getattr(ref,y)))/du
    return g

def nonDiagH(aModelList, printing):
    
    nondiagonal: h12, h21
    Computing d^2/(dx_i dx_j) by central differences.

55
aModelList = ['fpp, fpm, fmp, fmm]
fpp = f((hx)_i+(hx)_j), fpm = f((hx)_i-(hx)_j), etc
eps = 1.e-8  # numerical tolerance
if printing: print "nonDiagH runs: "
hi = abs(float(aModelList[0].FN2) - float(ref.FN2))
hj = abs(float(aModelList[0].PumpWork) - float(ref.PumpWork))
if hi == 0:
    step = 4*hj**2
    if printing: print "hi = 0, step = %4.4e" % step
elif hj == 0:
    step = 4*hi**2
    if printing: print "hj = 0, step = %4.4e" % step
else:
    step = 4*hi*hj
if printing: print "step = %4.4e" % step
if printing: print "Z values:\n %.8e, %.8e, %.8e, %.8e"
  (float(aModelList[0].Z),float(aModelList[1].Z),
   float(aModelList[2].Z),float(aModelList[3].Z))
if printing: print "objective = \n %.8e" % 
  (float(aModelList[0].Z) - float(aModelList[1].Z) - 
   float(aModelList[2].Z) + float(aModelList[3].Z))
if printing: print "nonDiagH finished: "
return (float(aModelList[0].Z) - float(aModelList[1].Z) - 
   float(aModelList[2].Z) + float(aModelList[3].Z))/step
def diagH(aModelList, printing):
  ""
  Computes h11, h22. Same infrastructure as nonDiagH.
  ""
  eps = 1.e-8  # numerical tolerance
  hi = abs(float(aModelList[0].FN2) - float(ref.FN2))
  hj = abs(float(aModelList[0].PumpWork) - float(ref.PumpWork))
  if printing: print "diagH runs: "
  if hi < eps:
      step = 12*hj**2
      if printing: print "hi = 0, step = %4.4e" % step
  elif hj < eps:
      step = 12*hi**2
      if printing: print "hj = 0, step = %4.4e" % step
  else:
      step = -1
      if printing: print "h1 and h2 < eps, step = -1"
  z_pp = float(aModelList[0].Z); z_pm = float(aModelList[1].Z)
  z_p = float(aModelList[4].Z); z_m = float(aModelList[5].Z)
  z = float(ref.Z)
  if printing: print "Z values:\n %.8e, %.8e, %.8e, %.8e, %.8e",
   (z_pp, z_p, z, z_m, z_mm)
if printing: print "objective = \n %.8e" % 
(-1*z_{pp} + 16*z_p - 30*z + 16*z_{m} - z_{mm})
if printing: print "diagH finished: ***********************"
return (-1*z_{pp} + 16*z_p - 30*z + 16*z_{m} - z_{mm})/step

def Hessian(aModelList, printing):
    '''
aModelList = [hlist11, hlist12, hlist21, hlist22]
    '''
    return [[diagH(aModelList[0], printing), \ 
            nonDiagH(aModelList[1], printing)], \ 
            [ nonDiagH(aModelList[2], printing), \ 
              diagH(aModelList[3], printing) ] ]

def printMatrix(nestedList):
    try:
        print map(pf,nestedList)
    except:
        for line in nestedList:
            print map(pf,line)

def sortPrintDict(mydict):
    for key in sorted(mydict.iterkeys()):
        print "%s: %s" % (key, mydict[key])

# Global reference object
ref = solvedModel(‘ref’, ‘mhex_ref.lst’)
sensitivity = [
    solvedModel(‘sensitivity’, ‘mhex_sens_H1.lst’), \ 
    solvedModel(‘sensitivity’, ‘mhex_sens_H2.lst’), \ 
    solvedModel(‘sensitivity’, ‘mhex_sens_H3.lst’), \ 
    solvedModel(‘sensitivity’, ‘mhex_sens_C1.lst’), \ 
    solvedModel(‘sensitivity’, ‘mhex_sens_C2.lst’), \ 
    solvedModel(‘sensitivity’, ‘mhex_sens_FCP_H1.lst’), \ 
    solvedModel(‘sensitivity’, ‘mhex_sens_FCP_H2.lst’), \ 
    solvedModel(‘sensitivity’, ‘mhex_sens_FCP_H3.lst’), \ 
    solvedModel(‘sensitivity’, ‘mhex_sens_FCP_C1.lst’), \ 
    solvedModel(‘sensitivity’, ‘mhex_sens_FCP_C2.lst’)
]

gainModels = [
    solvedModel(‘gain’, ‘gain_FW2.lst’), 
    solvedModel(‘gain’, ‘gain_PW.lst’)]

hessianModels = [
    solvedModel(‘hessian/h_{11}’, ‘hessian_pp.lst’), \ 
    solvedModel(‘hessian/h_{11}’, ‘hessian_pm.lst’), \ 
    solvedModel(‘hessian/h_{11}’, ‘hessian_mp.lst’), \ 
    solvedModel(‘hessian/h_{11}’, ‘hessian_mm.lst’), \ 
]
solvedModel('hessian/h.11', 'hessian_mm.lst'),
\solvedModel('hessian/h.11', 'hessian_p.lst'),
\solvedModel('hessian/h.11', 'hessian_m.lst'),
{solvedModel('hessian/h.12', 'hessian_pp.lst'),
\solvedModel('hessian/h.12', 'hessian_pm.lst'),
\solvedModel('hessian/h.12', 'hessian_mp.lst'),
{solvedModel('hessian/h.21', 'hessian_pp.lst'),
\solvedModel('hessian/h.21', 'hessian_pm.lst'),
\solvedModel('hessian/h.21', 'hessian_mp.lst'),
{solvedModel('hessian/h.22', 'hessian_pp.lst'),
\solvedModel('hessian/h.22', 'hessian_pm.lst'),
\solvedModel('hessian/h.22', 'hessian_mp.lst'),
\solvedModel('hessian/h.22', 'hessian_p.lst'),
\solvedModel('hessian/h.22', 'hessian_m.lst')},
\]
\printing = False

F = sens(sensitivity)
print "F = \
", printMatrix(F)
Gy = gain(gainModels)
print "Gy = \n", printMatrix(Gy)
Juu = Hessian(hessianModels, printing)
print "Juu =\n", printMatrix(Juu)

../python/mhex_print.py
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; % Absolute flow measurement 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, 2.083333334e+06; 2.083333334e+06, -1.166666596e+04];
E = eig(Juu);
Gy = [1.0000000000e+00, 0.0000000000e+00, -4.156666668e+01, ... 2.5000000000e+04, 1.466666667e+04, -2.666666666e+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, ...
C THE MATLAB SCRIPT

6.3297500000e-03, 3.3495000000e-03, -1.8603333333e-03, ...
-1.2517142857e-03
5.2394000000e-04, 1.3089400000e-03, 8.6194000000e-04, ...
1.2929400000e-03, 2.8019400000e-03, 1.4566266667e-02, ...
3.6634700000e-02, 1.9399400000e-02, -1.0773733333e-02, ...
-1.2517142857e-03;

% NOTE: transposed

%% Calculations Exact Local Method

Y = [F'*Wd Wny] ;
H = Gy'*(Y*Y')^(-1) ;
%% Loss calulations
L = (1/(6*(length(y)+length(d))))*...
(norm(Juu^0.5*(H*Gy)^(-1)*H*Y,'fro'))^2;
L_avg = (1/(6*(length(y)+length(d))))*trace(P);
L_w = 0.5*(svds(Juu^0.5*(H*Gy)^(-1)*H*Y,'L'))^2;

%% Exporting as tex
H_n = H/norm(H);  % Making H readable. Doesn’t affect solution

cd(’~/Documents/NTNU/Project/tex/matlab/’)
matrix2latexmatrix(F, ‘FVal.pdf’)
matrix2latexmatrix(Gy, ‘GyVal.pdf’)
matrix2latexmatrix(Juu, ‘JuuVal.pdf’)
matrix2latexmatrix(H_n, ‘HVal.pdf’)
matrix2latexmatrix(L_n, ‘LVal.pdf’)
matrix2latexmatrix(L_avg, ‘L_avgVal.pdf’)
matrix2latexmatrix(L_w, ‘L_wVal.pdf’)
%% Null Space Method
H_null = null(F');
L_null = (1/(6*length(y)*length(d)))*...
       (norm(Juu^0.5*(H_null*Gy)~(-1)*H_null*Y,'fro'))^2;

%% Testing
H2 = H/norm(H,'fro');
L_w2 = 0.5*(svds(Juu^0.5*(H2*Gy)~(-1)*H2*Y,1,'L'))^2;
H3 = H/norm(H);
L_w3 = 0.5*(svds(Juu^0.5*(H3*Gy)~(-1)*H3*Y,1,'L'))^2;
Nifty Trifles

D.1 Python: Autovivification

For storing and easily having available the different temperatures associated with, for instance, $T_{\text{in}}$, 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.

```python
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)()
```

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 `PrettyFloat` class, as done in Listing 20 provide the possibility of mapping a list with `PrettyPrint` producing user defined formatting.

```python
class pf(float):
    """ Class for printing readable lists."""
    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 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

<table>
<thead>
<tr>
<th>Stream</th>
<th>Octane #</th>
<th>Benzene cont. [%]</th>
<th>Price $/kg</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>99</td>
<td>0</td>
<td>0.1(\hat{n}_1)</td>
</tr>
<tr>
<td>2</td>
<td>105</td>
<td>0</td>
<td>0.2</td>
</tr>
<tr>
<td>3</td>
<td>95</td>
<td>0</td>
<td>0.12</td>
</tr>
<tr>
<td>4</td>
<td>99</td>
<td>2</td>
<td>0.185</td>
</tr>
</tbody>
</table>

```plaintext
$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,
```

64
E AN OPTIMIZATION EXAMPLE WITH GAMS

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