Real-time Observer Modelling of a Gas-phase Ethylene Polymerisation Reactor

Richard Thomason

Submitted in partial fulfilment of the requirements for the degree of Master of Science in Engineering in the School of Chemical Engineering, University of Natal, Durban.

Durban
September 2000

Abstract

The desire for precise polymer property control, minimum wastage through grade transitions, and early instrument fault detection, has led to a significant effort in the modelling and control of ethylene polymerisation world-wide. Control is difficult due to complex inter-relationships between variables and long response times from gas to solid phase.

The approach in this study involves modelling using the kinetic equations. This forms the basis of a scheme for real-time kinetic parameter identification and Kalman filtering of the reactor gas composition. The scheme was constructed off-line and tested on several industrial polymer grades using historical plant data. The scheme was also converted into a form for use on the linear low-density polyethylene plant, Poly 2, at POLIFIN Limited.

There proved to be no difficulty in the identification step, but the Kalman filter requires more tuning for reliable fault detection. The software has been commissioned on-line and results from the POLIFIN plant match the off-line model exactly.

Preface

POLIFIN initiated this project in an attempt to explore different advanced control techniques with the view of improving polyethylene product quality and throughput, as well as plant stability and operability. Commercial quality control packages are extremely costly and use published methods. By funding two postgraduate research studies, it was hoped that a novel solution would be found.

The investigation required close co-operation with the Process Control engineers at POLIFIN Ltd., Nirmal Narotam and Cillus van der Merwe. During the study, the plant in Sasolburg was visited frequently in an attempt to gain a better understanding of the Poly 2 process and, at a later stage, to commission the software on-line.

Other studies and off-line simulations were carried out in the postgraduate offices of the School of Chemical Engineering at the University of Natal, Durban under the supervision of Professor Michael Mulholland.

This research study comprises 80 credits, with the remaining 64 credits being filled by appropriate coursework. The completed coursework is listed below, with the corresponding credit weightings and results achieved:

DNEL4CN2	Automation	8.0	73
DSMA4EV2	Mathematics of Control	8.0	83
DNCH5CT1	Applied Control Theory	16.0	74
DNCH5RT1	Real Time Process Data Analysis	16.0	79
DNEL5NC1	Linear Multivariable Control	16.0	54

This was the first time that the process control group has had a coursework component within the faculty. The coursework was very beneficial in that it provided an introduction to the concepts of process control. Algorithms discussed in lectures were used in assignments as well as in this research study.

The work contained in this document, except where otherwise stated, is my own work.

RP Thomason
R.P. Thomason

2001/03/19
Date

M. Mulholland
Professor M. Mulholland
Supervisor

2001. 03. 19
Date

Acknowledgements

I would like to thank the following people, who have contributed towards this research study:

My supervisor, Professor Michael Mulholland. He suggested, checked and added to the model and computer code.

Nirmal Narotam for his vast knowledge of the RTAP database and control layout at Poly 2. His efficient programming methods helped simplify the code a lot.

Cilius van der Merwe for his organisation, added knowledge of the process and for taking over from Nirmal in the final stages of the project.

POLIFIN Ltd. for providing financial assistance and superb computers, for conceiving this project and for entrusting it with the University of Natal.

Table of Contents

List of Figures and Tables

Figures:

Tables:

List of Symbols

Lowercase letters

f_i	Mole fraction of monomer i
$k_a(j)$	Impurity desorption rate constant for a site of type j
k_d	A deactivation rate constant
$k_{dI}(j)$	Rate constant for deactivation by impurities for a site of type j
$k_{ds}(j)$	Spontaneous deactivation rate constant for a site of type j
$k_f(j)$	Rate constant for formation of a site of type j
$k_{fHi}(j)$	Transfer rate constant at a site of type j for a chain with terminal monomer Mi reacting with hydrogen
$k'_{fH}(j)$	Rate constant for spontaneous re-initiation of a site of type j with hydrogen attached
$k_{fMik}(j)$	Transfer rate for a site of type j with terminal monomer M_l, reacting with monomer M_k
$k_{fMiT}(j)$	Transfer to monomer pseudo-rate constant for a site of type j
$k_{fMTi}(j)$	Transfer to monomer pseudo-rate constant for a site of type j
$k_{fMTT}(j)$	Transfer to monomer pseudo-rate constant for a site of type j
$k_{fRi}(j)$	Transfer rate constant at a site of type j for a chain with terminal monomer M_i reacting with cocatalyst
$k_{fsi}(j)$	Spontaneous transfer rate constant for a site of type j with terminal monomer
k_H	The rate constant for hydrogen
$k_{Hi}(j)$	Rate constant for initiation of a site of type j by monomer
$k_{Ik}(j)$	Rate constant for re-initiation of a site of type j with an impurity attached by monomer M_k
$k_{pik}(j)$	Propagation rate constant at a site of type j, for a chain with terminal monomer M_l, reacting with monomer M_k
$k_{piT}(j)$	Pseudokinetic propagation rate constant for a site of type j
k_{p2}	Propagation rate constant for ethylene
k_{p4}	Propagation rate constant for butene
k_{p6}	Propagation rate constant for hexene
$k_{Ri}(j)$	Rate constant for re-initiation of a site of type j, with cocatalyst attached, by reaction with monomer Mi
$k_{pTi}(j)$	Pseudokinetic propagation rate constant for a site of type j
$k_{pTT}(j)$	Pseudokinetic propagation rate constant for a site of type j
mMMp	Mean molecular mass of polymer
mw_i	Molecular weight of monomer i
n_{AS}	The total number of moles of active sites in the reactor
s_2	Moles of C2 lost from the gas per total moles in the product
s_4	Moles of C4 lost from the gas per total moles in the product
s_6	Moles of C6 lost from the gas per total moles in the product

Uppercase letters

AS	Active Sites
Bi	Number of moles of monomer bound in the polymer
C	A selection matrix, used in the identifier, to restrict to only measurable state
C2	Monomer Ethylene (C_2H_4)
C4	Co-monomer Butene (C_4H_8)
C6	Co-monomer Hexene (C_6H_{12})
Fi	Molar flow rate of species i
F_{la}	Mole fraction of each comonomer in the polymer
$F*_{in}(j)$	Molar flow rate of potential active sites into the reactor
F_p	Flow of polymer from the reactor
F_V	Vent flow to flare

H_2	Hydrogen in the gas phase
I	An identity matrix
Im	Impurity/poison in the gas phase
K_p	Partition coefficient
$[M_i]_{gas}$	Concentration of component i in the gas phase
$[M_i]_{pol}$	Concentration of component i within the polymer particles
M_k	Monomer of type k in the gas phase
$\overline{M_n}$	Number average molecular weight
$\overline{M_w}$	Weight average molecular weight
M_T	Total monomer concentration
N	Number of moles of a species
$N^*(j)$	Potential active sites of type j
N(0,j)	Uninitiated sites of type j produced by formation reaction
$N_d(j)$	Spontaneously deactivated sites of type j
$N_{dI}(0,j)$	Sites of type j killed by impurity
$N_{dIH}(0,j)$	Sites of type j with hydrogen attached, killed by impurity
$N_{dIR}(0,j)$	Sites of type j with cocatalyst attached, killed by impurity
N_g	Number of moles in the gas phase
N_p	Total moles of polymer within the reactor
$N_H(0,j)$	Uninitiated sites of type j with hydrogen attached, produced by transfer to hydrogen reaction, or spontaneous transfer
$N_l(1,j)$	Living polymer of length ℓ, growing at an active site of type j, with terminal monomer M_l
$N_k(1,j)$	Living polymer of length ℓ, growing at an active site of type j, with terminal monomer M_k
N_p	Number of moles in the polymer phase
$N_R(0,j)$	Uninitiated sites of type j with cocatalyst attached, produced by transfer to cocatalyst
Ns	Number of sites in the catalyst
Q(ℓ,j)	Dead polymer of length ℓ produced at a site of type j
	Measurement error covariance matrix
R_i	Instantaneous consumption rate of monomer i
R_v	Polymer production rate, m^3 solid polymer
V_p	Volume of solid polymer produced in the reactor
X(n,j)	nth moment of the dead-polymer chain-length distribution produced by sites of type j
Y(n,j)	nth moment of the live-polymer chain-length distribution produced by sites of type j
Z	Polydispersity index

Acronyms

ACM	Advanced Control Module
API	Application Programming Interface
APS	APACS Process Supervisor
ASTM	American Society for Testing and Materials
CCD	Copolymer Composition Distribution
CLD	Chain Length Distribution
CSTR	Continuous Stirred Tank Reactor
CFBR	Continuous Fluidised Bed Reactor
DCS	Distributed Control System
EKF	Extended Kalman Filter
HDPE	High-density polyethylene (HDPE)
IMC	Internal Model Control
I/O	Input/Output
LDPE	Low-density Polyethylene
LLDPE	Linear Low-density Polyethylene
MATLAB	Matrix Laboratory computer program by Mathworks Inc.
MI	Melt Index

MPC	Model Predictive Control
MWD	Molecular Weight Distribution
PE	Polyethylene
PID	Proportional-Integral-Derivative
PRBS	Pseudo Random Binary Sequence
RLS	Recursive Least Squares
RPEM	Recursive Prediction Error Method
RTAP	Real-time Application Programming
SCADA	Supervisory Control and Data Acquisition
SEC	Size Exclusion Chromatography
TREF	Temperature Rising Elution Fractionation
UHMWPE	Ultra-high Molecular Weight Polyethylene

Vectors and Matrices

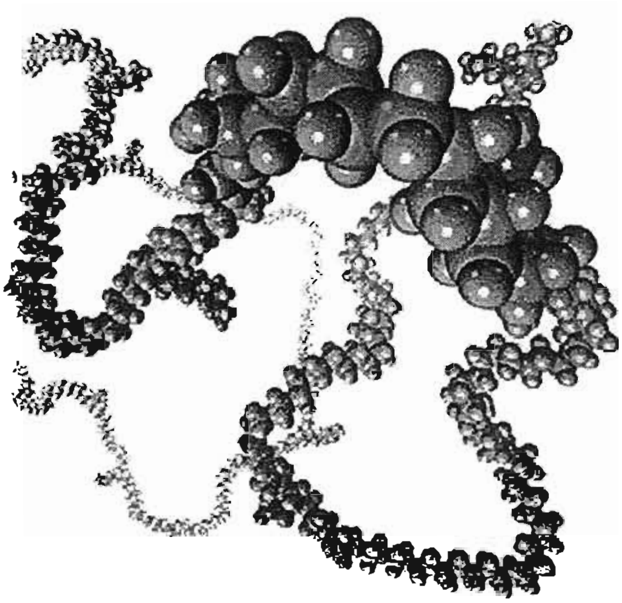
A	Coefficient of $\hat{n}$, used in the model
$A^{\bullet}$	Discrete form of A above
B	Coefficient of $\bar{F}$, used in the model
$B^{\bullet}$	Discrete form of B above
$\bar{F}$	Input vector of flows
g	Matrix forming part of A and containing entries of g_H and k_d
g_H	A mismatch factor to account for uncertainty in the hydrogen mass balance
G	Matrix forming part of A, selects gas component leaving the loop via the vent
G_i	Selection matrix to restrict to measurable states in the identifier and Kalman filter
h	Matrix forming part of A and containing entries of s_2, s_4 and s_6
H	Matrix forming part of A, selects polymer components leaving the loop via product flow
H_{AS}	Matrix forming part of A, selects active sites leaving the loop via product flow
K	Propagation matrix forming a part of A and containing entries of k_{P2}, k_{P4}, k_{P6}, k_H
K_i	Gain matrix for interval i, used in the Identifier and the Kalman filter
$\bar{n}$	The vector of predicted molar states from the Kalman filter
Q	Model error covariance matrix
R	Cocatalyst in the gas phase
$\bar{x}_i$	Vector of predicted molar states from the Kalman filter
xfbar	Vector of estimated kinetic parameters from the identifier
x_typical	A vector of component mole fractions (C2, C4, C6) in the polymer

Greek letters

ϕ_i	Fraction of sites of type j with terminal monomer M_i
δ_{i-1}	A process noise term

Subscripts

i	Present step
i+1	Present step plus 1
t	Present time step
t+1	Present time step plus 1



POLYETHYLENE (The Scientific American, May 1997)

CHAPTER 1

Introduction

The global plastic consumption for 1999 was 147 million tons. Of this consumption, 72 % was polyolefins (mainly polyethylene, polypropylene or polystyrene) and 45 % was polyethylene ((Davidovici 2000).

POLIFIN Limited is the largest producer of linear low-density polyethylene (LLDPE) in Southern Africa ahead of SAFRIPOL, its principle competitor. The Poly 2 plant in Sasolburg produces LLDPE. LLDPE has established itself as the third major member of the world polyethylene business (Schumacher 1996). Its consumption has grown substantially to exceed 10 million metric tons in 1999. Projected consumption in 2005 is 15 million tons. There are strong indications that the polyethylene market, and in particular the LLDPE market, is set to grow well into the future.

The LLDPE produced by Poly 2 is mainly used for the production of plastic bags, a commodity that should be in demand for the next few years at least. The UNIPOL gas-phase process also has the capability of producing resins with different grades that can be used in the manufacture of products such as plastic sheeting and piping.

With these factors in consideration it seems reasonable to assume that the Poly 2 plant should have a sufficient market for its LLDPE products in the near future. This warrants attempts to improve the control strategies on the plant, increasing profitability.

1.1 Layout of this Thesis

Chapter 1 introduces the reader to polyethylene, its various derivatives and the corresponding commercial polymerisation processes used in its manufacture. The chapter following this looks specifically at the Poly 2 process that was used for the investigation. It also looks closely at ethylene polymerisation with respect to the reactions that occur to allow polymer growth. Chapter 3 presents a comprehensive literature review on issues pertaining to the current problem. It also covers the complex theory behind polymerisation as well as the theory of the advanced control techniques employed in the project. Chapter 4 focuses on the current control techniques used at Poly 2 and outlines the control problem. The next chapter is used to formulate the control algorithm and outlines its objectives. The following two chapters (6 and 7) present the off-line (in Durban) and on-line (at Poly 2) applications of the observation application. Finally some conclusions are drawn in chapter 8.

1.2 Polymerisation Processes

In characterising any polymerisation process, certain factors need to be considered. The monomer used is the first factor. The reaction medium can be emulsion, solution heterogeneous, gas-phase or bulk. Also the type of catalyst used is important (heterogeneous or homogeneous). The types of reactors investigated also vary: batch, semi-batch, tubular, continuous stirred tank reactors (CSTR), continuous fluidised bed reactors (CFBR) and continuous stirred autoclave reactors. Lastly there are three types of polymerisation reactions that may occur. They are homopolymerisation, copolymerisation and terpolymerisation.

Due to the fact that the investigation is of POLIFIN's Poly 2 process, the parameters to be investigated have already been set. Poly 2 manufactures linear low-density polyethylene using a titanium-based Ziegler-Natta catalyst and an α-olefin comonomer (1-butene or 1-hexene). The catalyst itself has multiple active sites and this gives rise to broad chain length and copolymer composition distributions. These distributions affect the polymer properties that govern its end uses.

1.3 The World of Polyethylene

Polyethylene is a polymer made using the monomer ethylene. It is probably the most commonly encountered polymer in everyday life and it is used to produce grocery bags, shampoo bottles, children's toys, and even bulletproof vests. The chain structure is relatively simple, in fact the simplest of all commercial polymers. A molecule of polyethylene is simply a long chain of carbon atoms, with two hydrogen atoms attached to each carbon atom (*figure* 1.1).

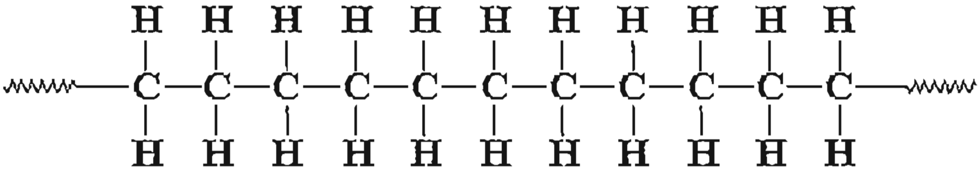


Figure 1.1: The Chain Structure of a Polyethylene Molecule (Michalovic, Anderson et al. 1996)

The main carbon chain often has long chains of polyethylene attached to it. This is called low-density polyethylene (LDPE). Hydrogen can act as a chain-terminating agent and this type of polyethylene with controlled branching is known as linear low-density polyethylene (LLDPE). When there is no branching, it is called linear polyethylene, or high-density polyethylene (HDPE). Linear polyethylene is much stronger than branched polyethylene, but branched polyethylene is cheaper and easier to make.

Linear polyethylene is normally produced with molecular weights in the range of 200 000 to 500 000, but it can be made even higher. Polyethylene with molecular weights of three to six million is referred to as ultra-high molecular weight polyethylene, or UHMWPE. This form of polyethylene has the ability to replace kevlar in bulletproof vests (Michalovic, Anderson et al. 1996).

Branched polyethylene is made by free-radical vinyl polymerisation. Linear and linear low-density polyethylene is made by Ziegler-Natta polymerisation. UHMWPE is made using metallocene catalysis polymerisation.

1.3.1 Classification, Properties and Uses

Polyethylene (PE), or polythene as it is sometimes called, is the largest synthetic commodity polymer in terms of annual production (Xie, McAuley et al. 1994). It has highly versatile chemical and mechanical properties, depending on the molecular structure of the material. It is non-toxic and enjoys a wide range of application, including supermarket bags, milk sachets and frozen food packaging, stretch film, lamination, tanks, heavy-duty sacks and roto-moulded objects. *Figure* 1.2 below from James (1986) and Foster (1991) illustrates this well.

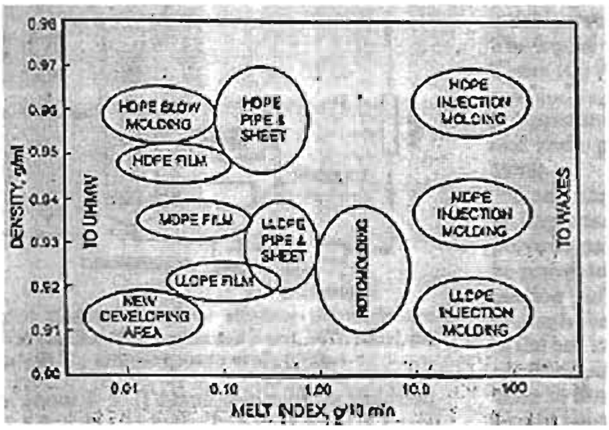


Figure 1.2: Applications of low-pressure polyethylene ((James 1986) and (Foster 1991))

PE has been classified into four groups, based on density, by the American Society for Testing and Materials (ASTM):

• Low density:	910 – 925 kg/m^3
• Low density (also):	926 – 940 kg/m^3
• High-density copolymers:	941 – 959 kg/m^3
• High-density homopolymer:	910 – 925 kg/m^3 and above.

Normally it is simply referred to as low-density polyethylene (LDPE) at 910 - 930 kg/m^3 or high-density polyethylene (HDPE) at 931 – 970 kg/m^3. LDPE can be classified as high-pressure low-density polyethylene (HP-LDPE) and linear low-density polyethylene (LLDPE). This is based on the polymer chain structure and its degree of branching. These two resins share the market and are mainly used for manufacturing films.

Referring to *figure* 1.3 below, HDPE has few or no short chain branches and as a result it is very rigid. Therefore it has structural applications like piping and sheeting.

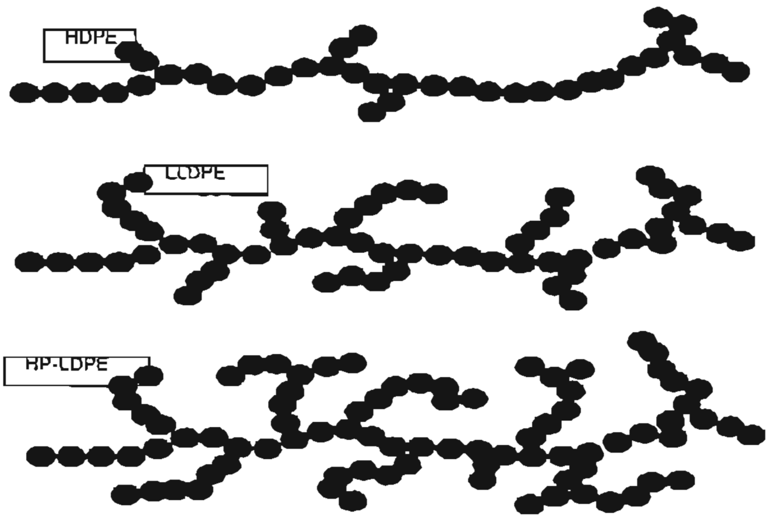


Figure 1.3: Schematic showing chain structures of commercial polyethylene types

LLDPE is formed by copolymerisation with an α-olefin (1-butene or 1-hexene). The orderly behaviour of the chain is disrupted in that copolymer molecules are incorporated into the chain and this leads to a short chain branching effect. HP-LDPE has a high degree of short and long-chain branching and this ensures good processability. It is tough whilst still transparent and it is this feature that makes it suitable for thin film applications. Interestingly, the Poly 1 plant, adjacent to Poly 2, produces this polymer.

1.3.2 Gas-phase Polymerisation

Polyethylene can be produced by at least five commercial processes: tubular, autoclave, solution, slurry and gas phase polymerisation. Of these, the gas phase and solution processes commercially produce LLDPE. The gas phase process is the most modern and is very popular due to its high versatility (it can produce PE with densities of 910 – 970 kg/m^3 and with melt indices of <0.01 – 200g/10min). Other advantages include good temperature control, low operating costs and good comonomer incorporation. With every process there are some disadvantages. Large amounts of off-spec product can be produced, fouling can occur, control is complex and a good catalyst design is required.

There are 56 technologies and processes in commercial use, many of which are available for licensing. In fact no two licensed processes have identical capabilities and no combination of catalyst and process is able to offer all things to all people (that is low capital and operating costs, product and process flexibility and low environmental impact) (Davidovici 2000). Many new polymer grades are introduced every year due to the continual development of catalysts and the new processes available. Union Carbide has patented a gas-phase process called the UNIPOL process. This process has been licensed worldwide and it is used at Poly 2.

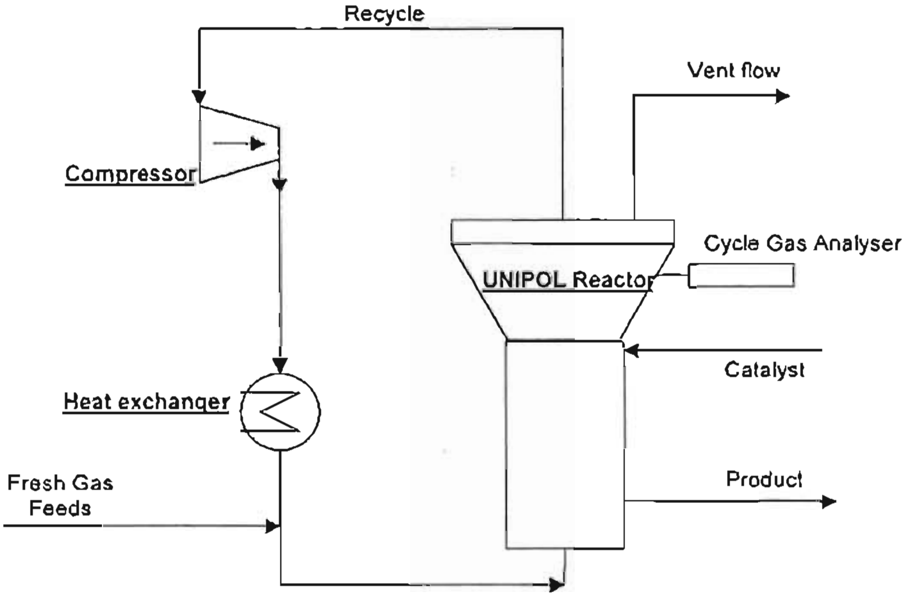


Figure 1.4: The UNIPOL (Union Carbide Gas-phase Ethylene Polymerisation) process

The reactor vessel consists of a straight section (the reaction zone) and an expanded section (the disengagement zone). The latter allows the fluidised polymer particles to disengage from the reactant gas. Unreacted gases are combined with the fresh feeds and recycled to the base of the reactor. Due to the highly exothermic nature of the reaction, heat must be removed from the cycle gas stream. This is achieved via a cycle gas cooler, which uses tempered water as a cooling medium on the shell side. Conversion in the reactor per pass is very low (2-3 %) therefore the recycle gas stream is much larger than fresh feed streams. A cycle gas analyser is located at the top of the reactor and it indicates the recycle gas composition. Fresh feed can then be added to maintain a constant gas composition within the reactor.

1.3.3 Catalyst History and Development

In Britain in the 1930's it was discovered that a 'waxy white solid' could be synthesised with ethylene at high pressure. It was subsequently discovered that oxygen was needed in the reaction and its role was believed to be the creation of free radicals for polymerisation. This type of polymer produced by free-radical polymerisation is now known to be HP-LDPE.

It was also in 1930 that the first unintentional catalysed synthesis of PE occurred in the laboratories of C.S. Marvel, a polymer chemist (Coville 1999). The reaction took place at atmospheric pressure and it involved ethylene and a lithium alkyl. The reaction was not researched further though. During the 1940's no real effort was made to pursue this polymerisation reaction under mild conditions.

The polymerisation world was revolutionised in the early 1950's when Karl Ziegler (Germany) and Guilio Natta (Italy) reported formation of polymers under mild conditions in the presence of a transition metal catalyst. This gave birth to the industrially used Ziegler-Natta catalyst. Since then the subject of catalysis in ethylene polymerisation has been the most active area of research in laboratories around the world. A good catalyst is instrumental in order to succeed at gas-phase polymerisation. It should have a high productivity, induce proper kinetic behaviour, have the correct morphology, control the polymer morphology, incorporate the comonomer; it should be reproducible at low cost and it should be easy to feed to the reactor. The catalyst should also be capable of controlling polymer properties. These include molecular weight and molecular weight distribution, density, chain branching and particle size distribution, particle size and polymer chain unsaturation (Xie, McAuley et al. 1994). Ziegler-Natta catalysts consist of a transition metal salt of metals from group IV to VIII e.g. $TiCl_3$ (the catalyst) and a metal alkyl base from groups I to III e.g. $AlEt_2Cl$, $AlEt_3$ (the cocatalyst). For industrial applications titanium salts and aluminium alkyls are normally used (Hamielec and Soares 1996). This catalyst is used to manufacture HDPE and LLDPE.

A type of catalyst currently undergoing major research is the metallocene or homogeneous catalyst. At the time of writing, $3.2 billion U.S. had already been spent in this area (Davidovici 2000). A metallocene is a positively charged metal ion incorporated between two negatively charged cyclopentadienyl anions. Without going into too much more detail, these catalysts offer numerous advantages over the heterogeneous catalysts. Among others, these are low extractables, improved physical properties, improved clarity and a better heat seal. Disadvantages include high costs, poor processability, low melt strength and melt fracture. These first two deficiencies are the major reasons why metallocenes have not penetrated the market, despite the hype and good results.

CHAPTER 2

The Poly 2 Process

POLIFIN Ltd. has four plants in three major provinces across South Africa: Sasolburg (Free State), Secunda (Mpumalanga), Umbogintwini (Kwazulu-Natal) and Witbank (Mpumalanga). The Witbank works produces calcium cyanide, which is used in gold recovery. In Secunda the plant produces polypropylene. Products produced at Umbogintwini are Chlorine, Cereclor, Sodium hypochlorite, Sodium hydroxide and Hydrochloric acid. Trichloroethylene is no longer produced there. The other products shown in *figure* 2.1 below are produced on the Midland plant in Sasolburg, with the exception of polypropylene, which is only produced in Secunda.

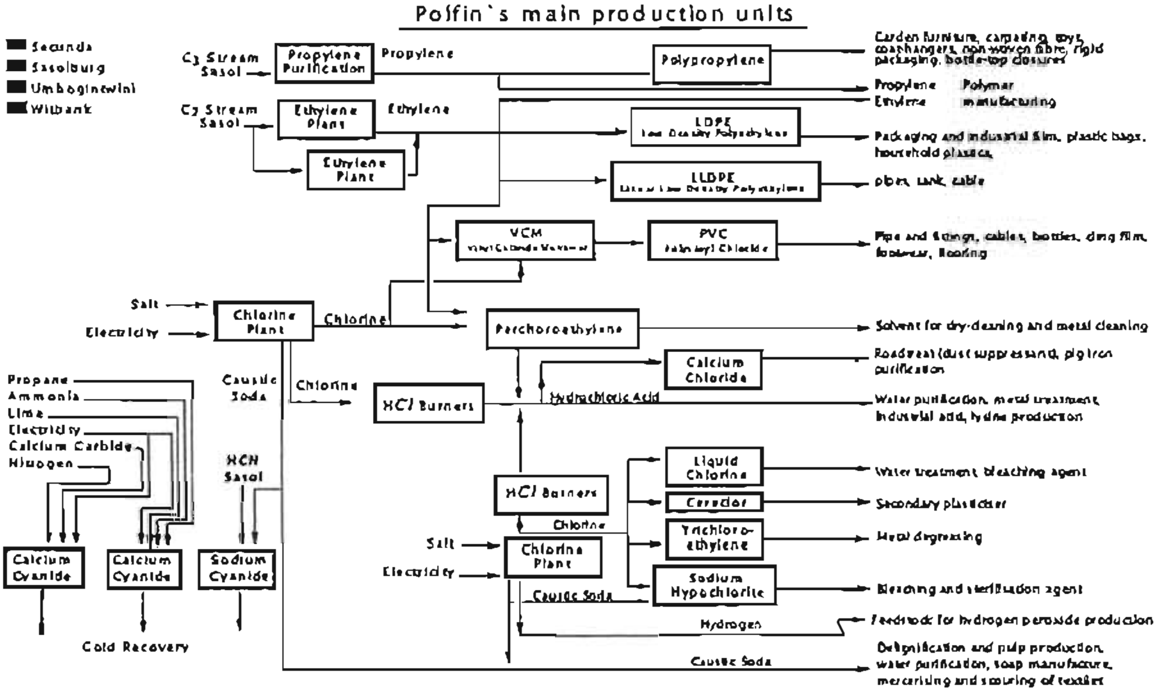


Figure 2.1: POLIFIN's Main Production Units

The main feedstock used in the process at Poly 2 is ethylene, which is supplied from the nearby SASOL plant. This supply of ethylene can vary as a result of fluctuations in SASOL's production. On the Midland site the Poly 1 plant, which produces LDPE (under extremely high pressure, using four independent reactors), has priority for the available ethylene feedstock. The remaining ethylene is then used by Poly 2. The reason for this is that the market demands for LDPE are more favourable than for LLDPE. Comonomer (1-butene or 1-hexene) is supplied from Secunda via rail cart.

Until recently only butene was used but after a R53-million upgrade project the plant can now use hexene as a comonomer. This is in line with international trend of increased demand for hexene-LLDPE. The project was commissioned in mid-year of 1998 and ensures that a wider range of polymer grades, with good mechanical properties can be produced. Production capacity should also increase from 85 000 t/a to 105 000 t/a (Royle and Rackham 1998). Elsewhere in the world, some manufacturers are starting to use octene as a comonomer.

2.1 Process Description

The feed to the reactor consists of ethylene, comonomer, hydrogen and nitrogen. These gaseous feed streams enter the cycle gas line upstream of the cycle gas cooler. The ethylene flow is manipulated to maintain reactor pressure and the comonomer and hydrogen flows are ratioed to this ethylene flow. The gases in the cycle gas line supply reactants to the growing polymer chains and provide the fluidisation and heat transfer media.

Catalyst and cocatalyst are also fed continuously into the reactor. The catalyst is a heterogeneous (titanium-based) Ziegler-Natta type and due to its high reactivity, only small feeds are required. These feeds are delivered from two feeders that operate simultaneously. The cocatalyst is triethyl aluminium (Teal, $AlEt_3$). There are two families of catalysts used, an M-catalyst and an F-catalyst. These produce M and F-resins respectively. Resin is simply a colloquial name for the product polymer.

The UNIPOL fluidised bed reactor is designed to produce 19 t/h of polymer. The reactor vessel consists of a straight section and an expanded section. The latter allows the fluidised polymer particles to disengage from the reactant gas. The typical weight of a bed of polyethylene in the reactor is 35 tons. A distributor plate is inserted at the base of the vessel to distribute gas evenly throughout the bed and to hold the bed, should fluidisation be lost.

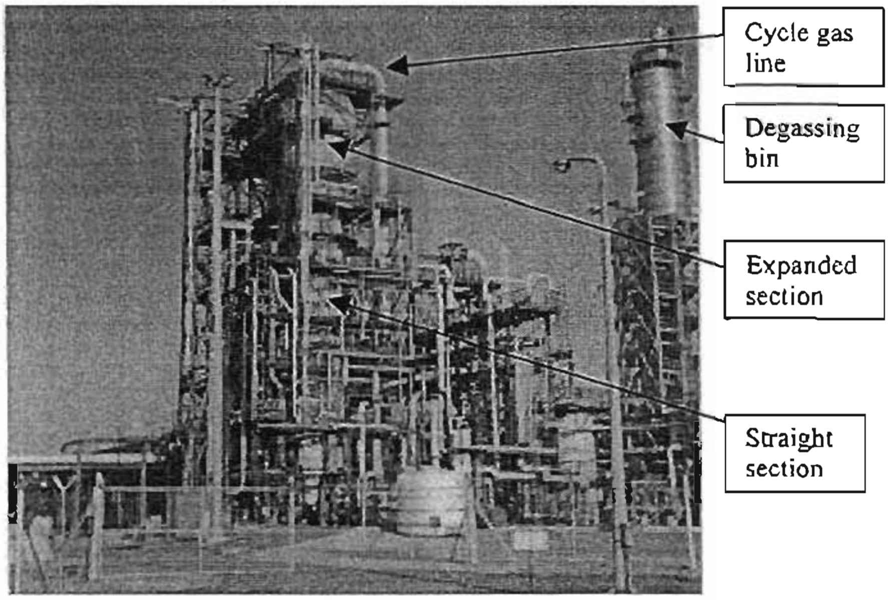


Figure 2.2: Poly 2's UNIPOL reactor

Unreacted gases are combined with the fresh feeds and recycled to the base of the reactor. A cycle gas compressor circulates these gases through a 30-inch line at a rate of 500 t/h. Due to the highly exothermic nature of the reaction, heat must be removed from the cycle gas stream. This is achieved via a cycle gas cooler, using tempered water as a cooling medium on the shell side. Conversion in the reactor per pass is very low (2-3 %) therefore the recycle gas stream is much larger than fresh feed streams. Polymer is periodically discharged from the reactor through a slide valve into one of two product discharge systems. These operate alternately. The product is degassed and proceeds to the extruder, where additives are included and pelletisation occurs. *Figure* 2.3 below illustrates this more clearly.

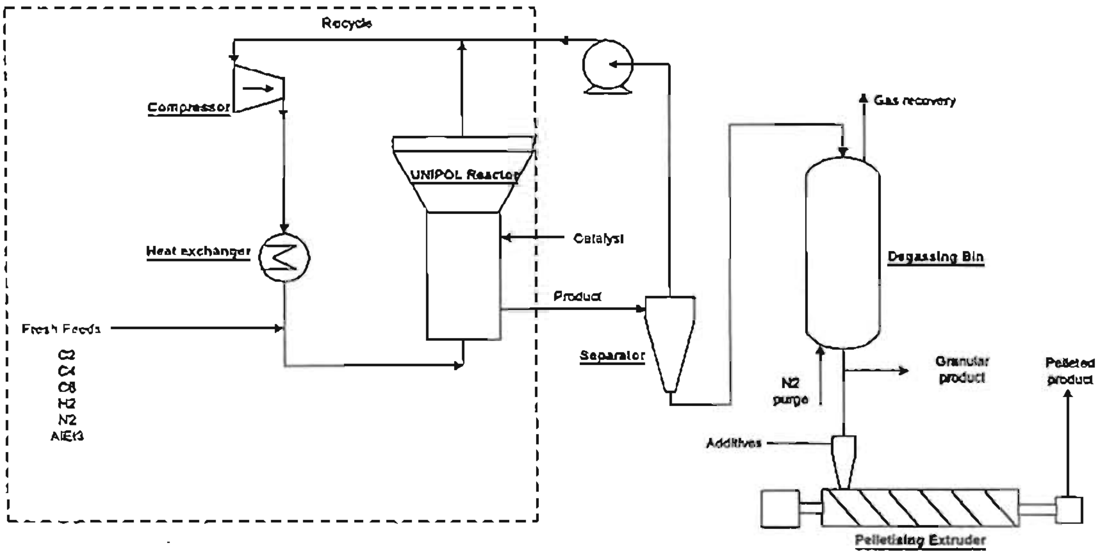


Figure 2.3: Diagram of the UNIPOL Poly 2 polymerisation process

2.2 Polymer Growth

Figure 2.4 (Xie, McAuley et al. 1994) below classifies the polymer yields at various stages of the polymer growth.

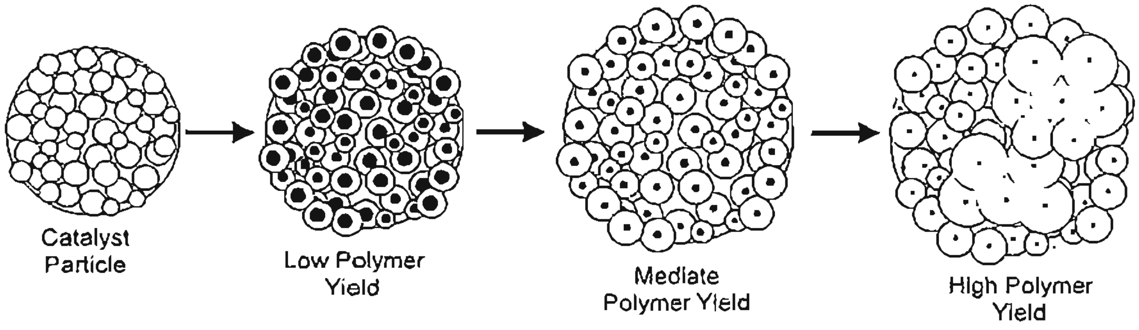


Figure 2.4: Schematic of the polymer growth on a catalyst particle (Xie, McAuley et al. 1994)

A catalyst particle is porous in nature and is composed of microscopic particles. These are called primary particles and are bonded by van der Waals forces. They provide the interstitial spaces required for a large surface area.

A monomer will diffuse through these spaces and adsorb onto the active sites. The polymer then precipitates and forms around the primary catalyst particles by the propagation reaction (explained in *section* 2.3). In this way the interstitial spaces are slowly filled up. When the stress between the particles becomes too great the primary particles, covered in polymer chains, separate.

The polymerisation process is highly exothermic and the temperature within the catalyst particle may increase rapidly, causing some of the polymer chains to fuse together. The final polymer particles are eventually 200-500 microns and consist of aggregates of primary particles. The shape is governed by the degree of annealing and distorting forces experienced during growth.

2.3 Chemistry: Polymerisation Reactions

There are a wide range of reactions that occur during the polymerisation process. There has been a great deal of research on Ziegler-Natta catalyst kinetics but to date there is no definite set of reactions that fully describe ethylene copolymerisation. Models that have been developed are based on a site balance and are very complex. The existence of multiple types of active sites makes logical sense but has yet to be proved fundamentally. It has been established that there are four main stages involved in the formation of polymer, namely formation, initiation, propagation and transfer. Deactivation and other reactions also occur. Assuming that all the sites exhibit the same mechanisms but at different reaction rates, the commonly used set of reactions is given below (Shaw, McAuley et al. 1998).

The organometallic cocatalyst, $AlEt_3$, activates the potential active sites of type j, N*(j) (2.1 below). Monomers of type k, M_k then join onto these activated sites to form a living polymer chain of length 1, N_k (1,j) (2.2). This small chain propagates, with monomers, M_k joining onto the end of the polymer chain (2.3). Transfer reactions are able to displace a chain to form a dead polymer segment of length ℓ. These take place via hydrogen, monomer, and cocatalyst and can also occur spontaneously (2.4)-(2.7). Sites with monomer, hydrogen, cocatalyst or nothing attached to their active centres can be deactivated into dead sites, N_d (j)(2.8). Reactions with impurities also affect the polymerisation process. These impurities block the respective active sites and the equations are given in equation (2.9). Impurities can be desorbed at a later stage (2.12). Sites with hydrogen or cocatalyst attached can also be displaced by monomer (2.11). Alternatively the hydrogen can be released (2.10).

$$\textit{Active Site Formation}:\ N^*(j)+R\xrightarrow{k_f(j)}N(0,j) \tag{2.1}$$

$$\textit{Initiation}:\ N(0,j)+M_k\xrightarrow{k_{ik}(j)}N_k(1,j) \tag{2.2}$$

$$\textit{Propagation}:\ N_i(l,j)+M_k\xrightarrow{k_{pik}(j)}N_k(l+1,j) \tag{2.3}$$

$$\textit{Transfer to Hydrogen}:\ N_i(l,j)+H_2\xrightarrow{k_{fHi}(j)}N_H(0,j)+Q(l,j) \tag{2.4}$$

$$\textit{Transfer to Monomer}:\ N_i(l,j)+M_k\xrightarrow{k_{fMik}(j)}N_k(1,j)+Q(l,j) \tag{2.5}$$

$$\textit{Transfer to Cocatalyst}:\ N_i(l,j)+R\xrightarrow{k_{fRi}(j)}N_R(0,j)+Q(l,j) \tag{2.6}$$

$$\textit{Spontaneous Transfer}:\ N_i(l,j)\xrightarrow{k_{fsi}(j)}N_H(0,j)+Q(l,j) \tag{2.7}$$

$$\begin{aligned}\textit{Spontaneous Deactivation}:\ N_i(l,j)&\xrightarrow{k_{ds}(j)}N_d(j)+Q(l,j)\\ N(0,j)&\xrightarrow{k_{ds}(j)}N_d(j)\\ N_H(0,j)&\xrightarrow{k_{ds}(j)}N_d(j)\\ N_R(0,j)&\xrightarrow{k_{ds}(j)}N_d(j)\end{aligned} \tag{2.8}$$

$$\begin{aligned}\textit{Reactions with Poisons}:\ N_i(l,j)+\mathrm{Im}&\xrightarrow{k_{dI}(j)}N_{dIH}(0,j)+Q(l,j)\\ N_H(0,j)+\mathrm{Im}&\xrightarrow{k_{dI}(j)}N_{dIH}(0,j)\\ N_R(0,j)+\mathrm{Im}&\xrightarrow{k_{dI}(j)}N_{dIR}(0,j)\\ N(0,j)+\mathrm{Im}&\xrightarrow{k_{dI}(j)}N_{dI}(0,j)\end{aligned} \tag{2.9}$$

$$\textit{Hydrogen Regeneration}:\ N_H(0,j)\xrightarrow{k'_{fH}(j)}N(0,j)+H_2 \tag{2.10}$$

Reinitiation by Monomer:

$$N_H(0,j) + M_i \xrightarrow{k_{Hi}(j)} N_i(1,j)$$
$$N_R(0,j) + M_i \xrightarrow{k_{Ri}(j)} N_i(1,j) \tag{2.11}$$

Desorption of Impurity:

$$N_{dIH}(0,j) \xrightarrow{k_{d1}(j)} N_H(0,j) + \mathrm{Im}$$
$$N_{dI}(0,j) \xrightarrow{k_{d}(j)} N(0,j) + \mathrm{Im} \tag{2.12}$$
$$N_{dIR}(0,j) \xrightarrow{k_{d1}(j)} N_R(0,j) + \mathrm{Im}$$

To obtain a clearer understanding of the polymerisation process, diagrams of each of the key stages of the polymerisation process are shown. These are included below (*figures* 2.5 through 2.9) and are merely illustrative. They show active site formation, initiation, propagation, transfer, deactivation and re-initiation. The large circles denote catalyst particles, which are divided into three sections (types of sites).

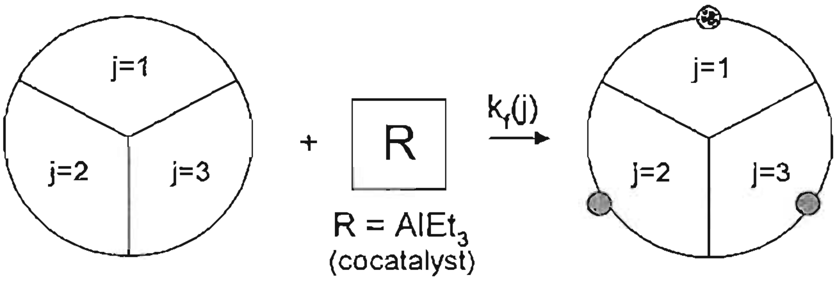


Figure 2.5: Active site formation schematic

Triethylaluminium (Teal) is a cocatalyst and its function is to activate catalyst sites on the catalyst particle. This is commonly referred to as the formation reaction.

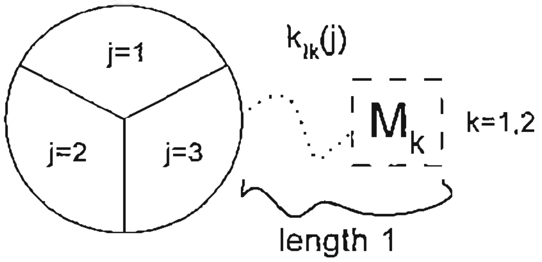


Figure 2.6: Initiation schematic

An activated site is now available for the attachment of monomers. These monomer blocks, M_k may be ethylene (k=1) or the comonomer, 1-butene or 1-hexene (k=2). The reaction is known as the initiation reaction and it results in a chain of length 1.

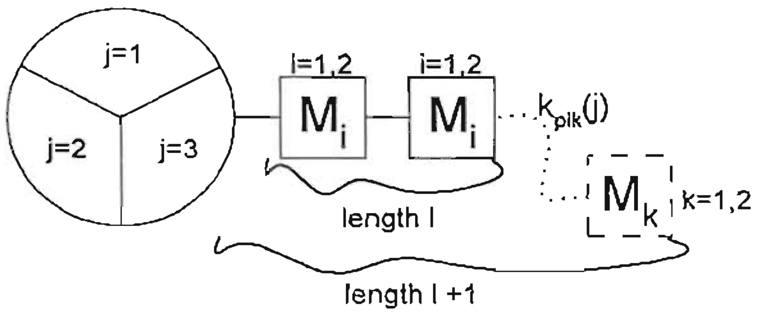


Figure 2.7: Propagation schematic

The polymer chain propagates by the addition of further monomers, as in the above initiation reaction, giving it the name of the propagation reaction.

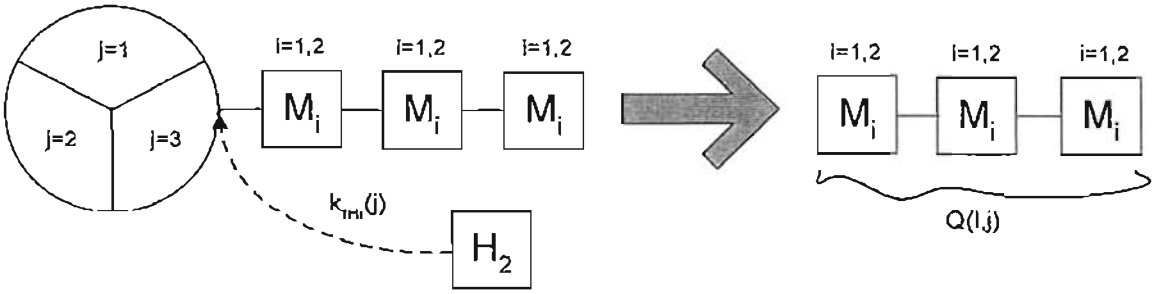


Figure 2.8: Transfer schematic

The above schematic represents a chain transfer (or transfer) reaction. The diagram shows a chain with terminal monomer M_i attached to site type j(=3), reacting with hydrogen. Other possible reactions in this category are chain transfer by reaction with monomer or cocatalyst and spontaneous transfer. These molecules attach themselves to the active centre, "cleaving" the polymer chain, which results in a dead polymer chain Q.

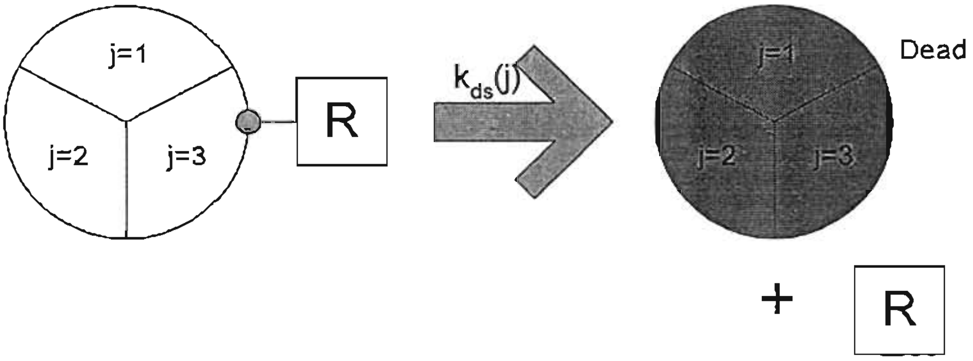


Figure 2.9: Deactivation schematic

A site can also be spontaneously deactivated. Active sites with no monomer attached or with polymer chains attached may undergo this reaction. Sites with only cocatalyst and hydrogen attached (from transfer reactions above) are also candidates for this reaction. These catalyst particles are deactivated and do not take part in any further reactions.

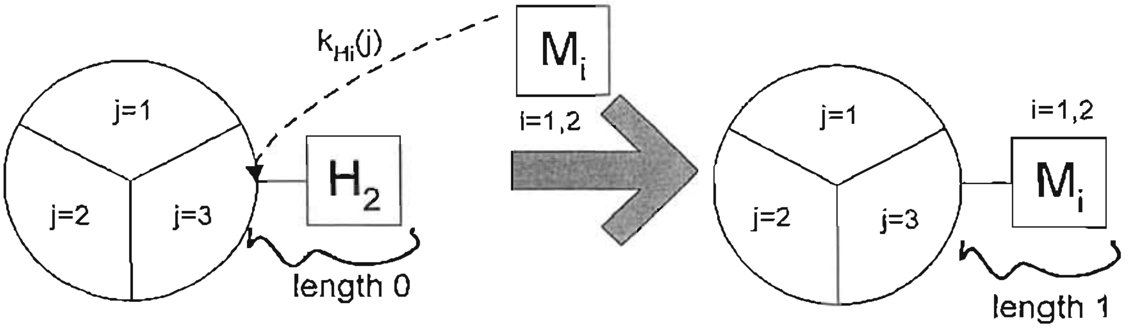


Figure 2.10: Re-initiation schematic

Sites with only cocatalyst and hydrogen attached (from transfer reactions above) can be re-initiated. This reaction is similar to the transfer reaction mechanism. The monomer attaches itself to the active centre, allowing the polymer chain to take part in polymerisation again.

Other reactions that are not detailed here also occur. These are reactions with impurities (CO (a reversible poison), H_2O, CO_2, O_2 and S-compounds) (2.9) as well as desorption of these impurities (2.12). Hydrogen regeneration is also a possiblity (2.10).

CHAPTER 3

Previous Work

3.1 Literature Review

The availability of specific articles on advanced control of polyethylene production appears to be limited. A small team of well-recognised researchers dedicated to this field often writes the few papers that are available. However, other polymerisation processes and the corresponding control techniques used were also explored. The review presented below gives details on polyethylene reviews (3.1.1), complex micro-scale modelling (3.1.2), (often too detailed for any on-line applications) and the two techniques used in the formulation of a parameter and state estimation scheme for Poly 2. The final heading (3.1.5) details combinations of these methods as well as alternative approaches.

3.1.1 Process Reviews

Xie, McAuley et al. (1994) provide an excellent summary of the gas phase polyethylene production process. Detailed summaries of the major ethylene processes are given, including the UNIPOL Union Carbide Process. Descriptions of the catalysts used, their development and production are also included. A large section of the paper deals with modelling of reactors, including both microscale (kinetic) and macroscale (dynamic) aspects.

A comprehensive literature review of advanced control of polymerisation reactors has been compiled (Embirucu, Lima et al. 1996). The paper discusses reactor modelling and control, estimation of polymer properties, optimal control and steady-state optimisation, non-linear control, linear predictive control, adaptive control, classical controllers and alternative control schemes.

3.1.2 Kinetic Models

McAuley, MacGregor et al. (1990) provide a detailed kinetic model of gas-phase olefin copolymerisation using multiple active site Ziegler-Natta catalysts. The model is able to predict production rate, molecular weight and copolymer composition changes in an industrial polyethylene reactor. Two potential uses for such a model are:

- Simulation and testing of on-line quality control schemes.
- Prediction of the effects of grade recipe transitions on molecular weight and compositional distributions.

A paper by Shaw, McAuley et al. (1998) gives extensive reaction mechanisms; material balances on active sites, polymer chains and gaseous components, moment balances, kinetic reaction data and initial conditions. All balances are for a semi-batch reactor situation but the equations are easily extended to a continuous situation. This paper is useful in that it updates some of the equations and kinetic rate constants in the earlier research (McAuley, MacGregor et al. 1990). The essence of the study is described below.

Heterogeneous Ziegler-Natta catalysts used in industry have multiple active sites, each site having its own kinetic rate constants. Broad chain length distributions (CLD's) and copolymer composition distributions (CCD's) are characteristic of these catalysts. Measuring joint CL and CC distribution can be accomplished but the methods are both costly and time-consuming. The development of a mathematical model that predicts these parameters would prove advantageous and the authors set about formulating this. The development is hampered by the need for many kinetic rate constants. These parameters were estimated by cross-fractionation of the polymer into CL and CC bins using Temperature Rising Elution Fractionation (TREF) and Size Exclusion Chromatography (SEC). Polymer produced at each site is then obtained, giving the relevant rates of reaction at each site. Stockmayer's (1945) bivariate distribution was used to develop a methodology for modelling the quantity of accumulated copolymer corresponding to each specific bin.

A comprehensive site-based kinetic model is developed by Xie, McAuley et al. (1995). The model can be used in the design of catalysts to achieve products with the desired polydispersity. Other uses include simulation of industrial α-olefin copolymerisation processes and for kinetic parameter estimation.

3.1.3 Kalman Filters

State estimation techniques are very popular in polymerisation research and one of the most widely used is the extended Kalman filter (EKF). Using this technique McAuley and MacGregor (1993) estimated rate constants and process parameters. A property inference scheme then updated the polymer properties (melt index (MI) and density). The effectiveness of the controller was demonstrated by comparing it to a linear decoupled internal model control (IMC) design. This provides a linear analogue to the non-linear control design above. The performance of the non-linear controller was far superior to that of the linear controller, due to the fact that it accounts for nonlinearities. It was found to be capable of achieving near optimal grade transitions and was able to control product properties at many different grades.

In an earlier paper by the same authors, McAuley and MacGregor (1991) use a scheme to infer MI and density in a fluidised-bed polyethylene reactor. While the above-mentioned kinetic model presented by McAuley, MacGregor et al. (1990) is able to predict these polymer properties and production rate, it is too complex for on-line use. This article attempts to provide a simpler model for on-line use. It develops instantaneous and cumulative MI and density models. The cumulative parameters require initial estimates and a series of measurements. Updating of model parameters is needed and recursive parameter estimation is ideal. Linear empirical models are not suitable since models must hold over a wide range of reactor products. The methods investigated in this paper and used for recursive parameter estimation with non-linear models are the extended Kalman filter (EKF) and recursive prediction error methods (RPEM). The potential benefits of the EKF over the RPEM are:

- The ability to estimate unmeasured model states.
- The potential for estimating several time-varying parameters simultaneously while controlling the rate of change of each parameter.
- The ability to estimate parameters that appear simultaneously in several highly coupled models.
- A means of optimally accounting for errors in the measured model inputs.

This study eventually used the RPEM method since it is easier to implement and has fewer parameters to specify. This article is almost exactly the scheme that POLIFIN wish to implement, only that the paper predicts MI and ρ and not gas ratios, as desired by POLIFIN.

de Wolf, Cuypers et al. (1996) address the problem of predicting the MI in a polypropylene slurry reactor. The investigation involved changing the controlled variable from the hydrogen concentration in the gas cap to the hydrogen concentration in the slurry phase. However because this variable is not easily measured, a linear Kalman filter was used to predict it. This was combined with a model predictive controller that was implemented to make up for the lack of feed-forward in the system. It was found that this new control system showed better performance for both grade changes and disturbances.

Valappil and Georgakis (2000) tackle the problem of selecting the model error covariance matrix, Q, for tuning an extended Kalman filter. In the article, Q is referred to as the process-noise covariance matrix. Estimates for this matrix are normally difficult to obtain and it is often viewed as a tuning parameter for trial-and-error simulations. The authors investigate two methods that estimate the matrix on-line using the current filter states. The first method uses linear approximation of the dependence of the model predictions on the model parameters. The second method is more rigorous and is for non-linear processes. It finds this non-linear dependence using Monte Carlo simulations. Although these methods seem to offer benefit over the trial-and-error technique, the authors still use a single tuning coefficient. This parameter η is a coefficient of the Q matrix and can be adjusted, should the filter diverge.

Another application of the extended Kalman filter was reported by Becerra and Roberts (2000), who used it for the state estimation of non-linear systems. A class of differential-algebraic models described the system investigated and a time-varying linearisation was derived for a semi-explicit, index one, differential-algebraic equation (DAE). A simulation study, including a single EKF, was then carried out as an example of a semi-explicit, index one DAE system. A study of a mixing process also showed that for a system model with *two* index one DAE's, it was necessary to use a bank of estimators.

Kozub and MagGregor (1992) investigated an EKF, using the semi-batch emulsion copolymerisation of styrene/butadiene rubber (SBR) as a case study. It was discovered that the filter was sometimes slow to converge from state initialisation errors, owing to its recursive nature. The solution suggested was to implement a second filter, or a recursive prediction error method. The latter would provide improved initial estimates of the unmeasurable states. Another approach studied was the use of a full non-linear optimisation procedure. However, a reiterative EKF provided better results. The importance of including non-stationary disturbances and/or model parameter mismatch in an EKF implementation was also stressed.

The batch free-radical solution polymerisation process was studied by Crowley and Choi (1998) using a bench scale setup. The aim of the study was to control directly an important resin quality control variable, the molecular weight distribution (MWD) of polymer. An EKF was an ideal candidate algorithm for the study. The filter used reactor temperature setpoints and these were recomputed and updated at each sampling point in order to meet the desired MWD.

In an older paper by Jo and Bankoff (1976), an extended Kalman filter was used in one of the first polymerisation applications. A simulation study of the free-radical polymerisation of vinyl acetate was carried out using a bench scale CSTR. During the course of polymerisation the impurity concentration was varied. Samples taken periodically from the reactor were analysed for conversion and weight-average molecular weight. It was found that adaptive and/or iterative techniques did not prove advantageous. This was attributed to the long residence time in the reactor. Another conclusion was that the initial state estimates had little effect on the filter. The authors also made the important point that simulation studies of detailed systems with complex relationships may give over-optimistic results.

Wilson, Agarwal et al. (1998) published a relevant paper that assesses the industrial feasibility of an on-line state estimator by means of tests on a 1m^3 semi-batch reactor. Many model predictive control (MPC) schemes find routine applications in the petrochemical industry but there seems to be a reluctance to use state-space ideas in industry. The reason is that these techniques require reliable models and a means of measuring states. The EKF is a simple way of predicting unmeasurable states and the authors use this algorithm to comment on the industrial feasibility of it. There have only been a handful of industrial applications of the EKF but most applications reported have been in the polymerisation and biotechnology fields. Both these have expensive measurement alternatives and state estimation techniques offer definite cost benefits. The authors presented several comments that they felt were the reasons why the EKF had not penetrated industry on a large scale, despite good simulation results. A pilot-scale application was studied in order for researchers to comment on the industrial feasibility of an EKF. It was found that the EKF gave no real improvement to the state estimation of batch end-point, already achieved through measurement techniques. This was attributed to plant-model mismatch, poor system observability at times and poor quality of state measurements. This led the authors to be doubtful of the usefulness of on-line estimators in industry. Several criteria in the form of a checklist were presented for those wishing to pursue industrial applications.

Applications of EKF's in other areas of research include Tham and Parr (1994) (data validation) and Zorzetto and Wilson (1996) (bioprocess monitoring).

3.1.4 Recursive Least Squares Parameter Estimation

Varela (2000) applied a recursive least-squares estimation algorithm with a forgetting factor as part of a self-tuning pressure control algorithm. The estimation enabled model parameters to be determined on-line. This allowed a self-tuning algorithm (along with a first order observer and state feedback) to be implemented. Its use was demonstrated for pressure control in an injection-moulding cavity for thermoplastics.

Some practical aspects of process identification are given by Isermann (1980). Areas covered include the final goal of the model application, type of process model, required accuracy and identification method (off-line, on-line etc.). Other sections given are selection of input signals, selection of sampling time, off-line and on-line identification, data filtering, model order testing and model verification. There is also a brief record of software packages available for process identification. A useful comparison of parameter estimation methods is given. These include least squares (LS), generalised least squares (GLS), instrumental variables (IVA), maximum likelihood (ML) and correlation and least squares (COR-LS).

Mulholland and Seinfeld (1995) used a recursive least squares (RLS) technique to identify source parameters from air pollution observations. A Kalman filter was used to control the extent to which emissions were allowed to deviate from a base case.

Afonso, Ferreira et al. (1998) employed a RLS algorithm with an EKF in a fault detection and identification (FDI) strategy. During normal operation on an industrial-scale pilot plant the RLS and EKF run and allow faults to be detected and identified. When a fault is detected, the RLS is halted to prevent contamination of parameters, but the EKF state estimation is continued. Such a setup could be useful for fault identification, reducing false alarms and providing redundant measurements for alternative control purposes.

3.1.5 Other

Mulholland and Fernandes (1997) present a paper that models the reactor at Poly 2. A model was developed in order to simulate the UNIPOL process in real time. Open-loop tests showed that the predictions were in advance of the GC output by 450 seconds. A Smith predictor was then introduced with the purpose of establishing the error between the predicted compositions and the measurements. The predictor then uses the old error to correct the present prediction. The Smith predictor was combined with a Kalman filter to shift the state to reasonable regions. This proved important in the reactor gas phase, which acts as a pure integrator. The Kalman filter is simply a proportional controller controlling the state to set point, which happens to be the measurements. The Smith predictor handles dead time well. The periodic and asynchronous laboratory data generated in polymerisation industries makes it an ideal candidate. The Smith predictor therefore used these periodic measurements and provided the filter with continuous updates. This arrangement, where the 'passive' Smith predictor was combined with the 'active' Kalman filter by means of a weighting factor proved to be advantageous. The disadvantage however is that the formulation of the Smith predictor is not trivial.

In an article by Lines, Hartlen et al. (1993), model predictive control (MPC) is used with the objective of reducing product variability. The dynamic model was built using pseudo random binary sequence (PRBS) testing. This was done in order to avoid modifying product specifications. All tests were done in the open-loop mode. Some step tests were also required in order to relate manipulated variables to polymer properties at the extruder. The modelling involved three stages:

- A linear multiple regression method to identify impulse and step response weights.
- A more detailed model was used to develop confidence in the model responses. The method used was based on an approach used by Box and Jenkins (1976).
- All models were configured in a matrix.

Due to the long delay between the analysers and the reactor, a linear properties estimation algorithm had to be used to infer ln(MI) and density values from the reactor operating conditions. The estimator proved reliable and was able to provide accurate estimates even when the analysers were faulty. The algorithm uses internal models of the process matrix to anticipate future process responses. It predicts future control actions to provide smooth control. PRBS testing is a good method that can be used to avoid disturbing normal plant operation. Unfortunately not much detail of the actual algorithm formulation is given. Also, there appears to be very little literature available on this topic.

McAuley and MacGregor (1992) present an article that makes use of optimal open-loop policies, using dynamic model-based optimisation, in order to achieve better grade changeovers. The goals of the paper are:

- To formulate a set of dynamic optimisation problems.
- To solve the optimisation problems.
- To determine the effect of problem formulation on the optimal solution.
- To decide which of the optimal trajectories might be desirable for on-line use.

A review of dynamic optimisation techniques and applications is also given. Some issues involved in formulation and solving of the optimal grade transition problem are addressed. Optimal trajectories are shown for a series of grade changes. Lastly, the importance of feedback control associated with the implementation of optimal off-line grade transitions is discussed. Three optimal transition policies are outlined:

- Policy 1: Uses butene and hydrogen as manipulated variables.
- Policy 2: Uses temperature set point and bleed valve position as manipulated variables.
- Policy 3: Uses catalyst feed rate and bed level set point as manipulated variables.

3.2 Complex Site-based Kinetic Modelling

McAuley, MacGregor et al. (1990) and Shaw, McAuley et al. (1998) offer comprehensive site-based models. The latter paper features some changes but it uses a semi-batch reactor for testing purposes. Thus to match the POLIFIN situation, equations were simply extended to include inflows and outflows. Also the number of monomers at POLIFIN is two and the number of sites was taken as three.

The authors used the simplifications below in the model development:

- The recycle stream is very large and the conversion per pass through the bed is very low. Thus the vertical concentration gradients in the fluidised bed are small and can be neglected.
- Since the recycle to fresh feed ratio is approximately 40:1 in a normal industrial reactor, the plug flow reactor dynamics approach that of a continuous stirred-tank reactor (CSTR).
- A rise of $< 3^0C$ from the reaction zone to the top of the bed is typical and vertical temperature gradients are small. Therefore the effects of small radial and axial temperature gradients are neglected.

Reaction rates are controlled by the concentrations of reactants dissolved in the polymer around the active site and not by bulk concentrations in the gaseous phase. This is because fresh catalyst particles injected into the reactor quickly become covered by polymer.

In order for a model to be developed where one can predict polymer quality variables from gas-phase concentrations, some assumptions need to be made (McAuley, MacGregor et al. 1990). The first is that the polymer phase is in equilibrium with the gas phase and the diffusional effects within the polymer phase are neglected. The second is that the plasticising effects of dissolved monomers on solubilities in the polymer are negligible. These allow one to conclude that the partition coefficients between polymer and gas phases remain constant

$[M_i]_{pol} = K_p[M_i]_{gas}$

where $[M_i]_{pol}$: Concentration of component i within the polymer particles
K_p : Partition coefficient
$[M_i]_{gas}$: Concentration of component i in the gas phase

Now, if the rate of reaction occurring at a site j is defined by R,

$R = k\,[M_i]_{pol}Y(0,j)$

where k : Rate constant
Y(0,j): : The number of moles of active sites of type j in the reactor

Hence:

$R = k^*[M_i]_{gas}Y(0,j)$

where $k^* = (kK_p)$ is a pseudo-rate constant. The balances in *section* 3.2.1 below use rate constants of this form.

This convention is used since gas-phase concentrations are measured on-line, whereas concentrations in the polymer phase are not as easily measured.

3.2.1 Balances

The mass balance equations are of the form:

Accumulation = Inflow + Generation – Consumption – Outflow

3.2.1.1 General balances

A molar balance on the potential active sites in the reactor is set out below:

$$\frac{dN^*(j)}{dt} = F^*_{in}(j) - k_f(j)N^*(j)[R] - N^*(j)\frac{R_v}{V_p} \tag{3.1}$$

where $F^*_{in}(j)$ is the molar flow rate of potential active sites into the reactor, $N^*(j)$ is a potential active site of type j, R_v is volumetric polymer outflow rate and V_p is the volume of polymer phase in the reactor.

Similar balances can be seen for the moles of initiation sites $N(0,j)$ and for $N_H(0,j)$ and $N_R(0,j)$:

$$\begin{aligned}\frac{dN(0,j)}{dt} = {} & k_f(j)N^*(j)[R] + k'_{fH}(j)N_H(0,j) + k_a(j)N_{dI}(0,j) - \\ & N(0,j)\left\{k_{fT}(j)[M_T] + k_{ds}(j) + k_{dI}(j)[\mathrm{Im}] + \frac{R_v}{V_p}\right\}\end{aligned} \tag{3.2}$$

$$\begin{aligned}\frac{dN_H(0,j)}{dt} = {} & Y(0,j)\{k_{fHT}(j)[H_2] + k_{fsT}(j)\} + k_a(j)N_{dIH}(0,j) - \\ & N_H(0,j)\left\{k_{HT}(j)[M_T] + k'_{fH}(j) + k_{ds}(j) + k_{dI}(j)[\mathrm{Im}] + \frac{R_v}{V_p}\right\}\end{aligned} \tag{3.3}$$

$$\begin{aligned}\frac{dN_R(0,j)}{dt} = {} & k_{fRT}(j)Y(0,j)[R] + k_a(j)N_{dIR}(0,j) - \\ & N_R(0,j)\left\{k_{RT}(j)[M_T] + k_{ds}(j) + k_{dI}(j)[\mathrm{Im}] + \frac{R_v}{V_p}\right\}\end{aligned} \tag{3.4}$$

and M_T is the total monomer concentration and is given by:

$$[M_T] = [M_1] + [M_2] \tag{3.5}$$

where $[M_1]$ and $[M_2]$ are the concentrations of monomer 1 (ethylene) and monomer 2 (1-butene or 1-hexene) respectively. [R] and [Im] are the concentrations of cocatalyst and impurities. $Y(0,j)$ is the zeroth moment of live polymer chain length distribution from site type j as defined in equation (3.10) below. Initiated polymer chains of length 1 with monomer i as the terminal monomer give a mass balance as follows:

$$\begin{aligned}\frac{dN_i(1,j)}{dt} &= k_{iI}N(0,j)[M_I]+k_{HI}(j)N_H(0,j)[M_I]+k_{RI}(j)N_R(0,j)[M_I]\\ &+Y(0,j)k_{fMTI}(j)[M_I]\\ &-Ni(1,j)\left\{\begin{aligned}&k_{piT}(j)[M_T]+k_{fHI}(j)[H_2]+k_{fMIT}(j)[M_T]\\ &+k_{fRI}(j)[R]+k_{fsI}(j)\,k_{ds}(j)+k_{dI}(j)[\mathrm{Im}]+\frac{R_v}{V_p}\end{aligned}\right\}\end{aligned}\tag{3.6}$$

Mass balances can also be written for impurity deactivated sites:

$$\frac{dNd_{IH}(0,j)}{dt}=k_{dI}(j)[\mathrm{Im}]\{Y(0,j)+N_H(0,j)\}-N_{dIH}(0,j)\left\{ka(j)+\frac{R_v}{V_p}\right\}\tag{3.7}$$

$$\frac{dNd_I(0,j)}{dt}=k_{dI}(j)[\mathrm{Im}]\,N(0,j)-N_{dI}(0,j)\left\{ka(j)+\frac{R_v}{V_p}\right\}\tag{3.8}$$

$$\frac{dNd_{IR}(0,j)}{dt}=k_{dI}(j)[\mathrm{Im}]N_R(0,j)-N_{dIR}(0,j)\left\{ka(j)+\frac{R_v}{V_p}\right\}\tag{3.9}$$

3.2.1.2 Moment balances

If one adds the balances on living polymer (of length ℓ and with terminal monomer Mi) together one obtains the moment balances.

$$\begin{aligned}\frac{dY(0,j)}{dt} &=[M_T]\{k_{iT}(j)N(0,j)+k_{HT}(j)N_H(0,j)+k_{RT}(j)N_R(0,j)\}\\ &-Y(0,j)\left\{k_{fHT}(j)[H_2]+k_{fsT}(j)+k_{fRT}(j)[R]+k_{ds}(j)+k_{dI}(j)[\mathrm{Im}]+\frac{R_v}{V_p}\right\}\end{aligned}\tag{3.10}$$

One can obtain balances for the first and second moments of living polymer chain length distribution as well:

$$\begin{aligned}\frac{dY(1,j)}{dt} &=Y(0,j)\{k_{pTT}(j)[M_T]+k_{fMTT}(j)[M_T]\}\\ &-Y(1,j)\left\{\begin{aligned}&k_{fHT}(j)[H_2]+k_{fsT}(j)+k_{fMTT}(j)[M_T]+k_{fRT}(j)[R]\\ &+k_{ds}(j)+k_{dI}(j)[\mathrm{Im}]+\frac{R_v}{V_p}\end{aligned}\right\}\\ &+[M_T]\{k_{iT}(j)N(0,j)+k_{HT}(j)N_H(0,j)+k_{RT}(j)N_R(0,j)\}\end{aligned}\tag{3.11}$$

$$\begin{aligned}\frac{dY(2,j)}{dt} &= Y(0,j)\{k_{pTT}(j)[M_T]+k_{fMTT}(j)[M_T]\} \\ &\quad -Y(2,j)\left\{\begin{array}{l} k_{fHT}(j)[H_2]+k_{fsT}(j)+k_{fMTT}(j)[M_T]+k_{fRT}(j)[R] \\ +k_{ds}(j)+k_{dI}(j)[\mathrm{Im}]+\dfrac{R_v}{V_p} \end{array}\right\} \\ &\quad +[M_T]\{k_{IT}(j)N(0,j)+k_{HT}(j)N_H(0,j)+k_{RT}(j)N_R(0,j)\} \\ &\quad +2k_{pTT}(j)[M_T]Y(1,j)\end{aligned} \tag{3.12}$$

The nth moment of the living-polymer chain-length distribution Y(n,j), is defined as:

$$Y(n,j)=\sum_{k=1}^{N_M}\sum_{l=1}^{\infty} l^n N_k(l,j) \tag{3.13}$$

where ℓ is the number of units in the polymer chain.

Chain length distributions for dead polymer chains are required for calculating molecular weight by the method of moments.

$$\begin{aligned}\frac{dX(n,j)}{dt} &= \{Y(n,j)-N_T(1,j)\}\left\{\begin{array}{l} k_{fMTT}(j)[M_T]+k_{fHT}(j)[H_2]+k_{fRT}(j)[R]+ \\ k_{fsT}(j)+k_{ds}(j)+k_{dI}(j)[\mathrm{Im}] \end{array}\right\} \\ &\quad -X(n,j)\frac{R_v}{V_p}\end{aligned} \tag{3.14}$$

The nth moment of the dead-polymer chain-length distribution, X(n,j), is defined as:

$$X(n,j)=\sum_{l=2}^{\infty} l^n Q(l,j) \tag{3.15}$$

where Q(ℓ,j) is dead polymer of length ℓ produced at a site of type j. Only chains of length longer than 1 are considered polymer, hence the summation begins at ℓ=2. Equation (3.14) is obtained by writing a mass balance for dead polymer of length ℓ and then inserting the result into the definition (3.15) above.

3.2.2 Pseudo-kinetic Rate Constants

Some of the mass balance equations given above have pseudo-kinetic rate constants. These are a function of the distribution of terminal monomers at each site as well as the gas composition in the reactor.

The first is a pseudo-rate constant for initiation of a site of type j, by monomer M_i.

$$k_{IT}(j)=f_1 k_{I1}(j)+f_2 k_{I2}(j) \tag{3.16}$$

Analogous to this is a rate constant for re-initiation of a site of type j, with hydrogen attached, by reaction with monomer M_i.

$$k_{HT}(j)=f_1 k_{H1}(j)+f_2 k_{H2}(j) \tag{3.17}$$

A further relationship exists for a rate constant for re-initiation of a site of type j, with co-catalyst attached, by reaction with monomer M_i.

$$k_{RT}(j)=f_1 k_{R1}(j)+f_2 k_{R2}(j) \tag{3.18}$$

The term f_i is simply the mole fraction of monomer i:

$$f_i=\frac{[M_i]}{[M_1]+[M_2]} \tag{3.19}$$

Following these relationships there are also pseudo-kinetic propagation rate constants:

$$k_{piT}(j)=f_1k_{pi1}(j)+f_2k_{pi2}(j) \tag{3.20}$$

$$k_{pTi}(j)=\phi_1(j)k_{p1i}(j)+\phi_2(j)k_{p2i}(j) \tag{3.21}$$

$$k_{pTT}(j)=f_1k_{pT1}(j)+f_2k_{pT2}(j) \tag{3.22}$$

Here, ϕ_i is the fraction of sites of type j with terminal monomer M_i. The equations below are for a two-monomer system. For a system with more than two, the equations are easily extended using the cross product.

$$\phi_1=\frac{f_1k_{p21}(j)}{f_1k_{p21}(j)+f_2k_{p12}(j)} \tag{3.23}$$

$$\phi_2=1-\phi_1 \tag{3.24}$$

The transfer to monomer pseudo-rate constants are defined in the same format as the propagation rate constants (3.20)-(3.22) above:

$$k_{fMiT}(j)=f_1k_{fi1}(j)+f_2k_{fi2}(j) \tag{3.25}$$

$$k_{fMTi}(j)=\phi_1(j)k_{f1i}(j)+\phi_2(j)k_{f2i}(j) \tag{3.26}$$

$$k_{fMTT}(j)=f_1k_{fMT1}(j)+f_2k_{fMT2}(j) \tag{3.27}$$

Other possible transfer reactions that may occur are transfer to hydrogen, cocatalyst and spontaneous transfer:

$$k_{fHT}(j)=\phi_1(j)k_{fH1}(j)+\phi_2(j)k_{fH2}(j) \tag{3.28}$$

$$k_{fRT}(j)=\phi_1(j)k_{fR1}(j)+\phi_2(j)k_{fR2}(j) \tag{3.29}$$

$$k_{fsT}(j)=\phi_1(j)k_{fs1}(j)+\phi_2(j)k_{fs2}(j) \tag{3.30}$$

3.2.3 Monomers bound in the Polymer

In order to give a prediction of the average composition of polymer in the reactor at a given time, a balance for Bi can be used. This parameter is the number of moles of monomer bound in the polymer:

$$\frac{dB_i}{dt}=R_i-B_i\frac{R_v}{V_p} \tag{3.31}$$

where R_i is the instantaneous consumption rate of monomer i. It is safe to assume that most of the monomer is consumed in the propagation reaction. Therefore:

$$R_I = \sum_{j=1}^{Ns} [M_I] Y(0,j) k_{pTI}(j) \qquad (3.32)$$

3.2.4 Molecular Weight Properties

The term F_{Ia} is the mole fraction of each comonomer in the polymer.

$$F_{Ia} = \frac{B_I}{B_1 + B_2} \qquad (3.33)$$

Following on from this is the definition for the mean molecular weight of monomer I:

$$\overline{m} = mw_1 F_{1a} + mw_2 F_{2a} \qquad (3.34)$$

Here mw_I is the molecular weight of monomer I. The cumulative number average and weight average molecular weights can then be evaluated using the method of moments:

$$\overline{M_n} = \frac{\overline{m} \sum_{j=1}^{Ns} \{X(1,j) + Y(1,j)\}}{\sum_{j=1}^{Ns} \{X(0,j) + Y(0,j)\}} \qquad (3.35)$$

$$\overline{M_w} = \frac{\overline{m} \sum_{j=1}^{Ns} \{X(2,j) + Y(2,j)\}}{\sum_{j=1}^{Ns} \{X(1,j) + Y(1,j)\}} \qquad (3.36)$$

Finally, the polydispersity index can be defined as the ratio of the weight average to number average molecular weights:

$$Z = \frac{\overline{M_w}}{\overline{M_n}} \qquad (3.37)$$

3.3 On-line Model Estimation by Recursive Least Squares

Recursive least squares (RLS) parameter estimation - or recursive parameter estimation (Ljung 1999) - is useful in non-linear processes, which can be approximated, for example, by allowing the coefficients in a linear representation to vary. RLS estimation is often used along with an EKF in control problems. Mulholland and Seinfeld (1995) used a RLS technique to identify source parameters from air pollution observations. A Kalman filter was used to control the extent to which emissions were allowed to deviate from a base case. Afonso, Ferreira et al. (1998) employed a RLS algorithm with an EKF in a fault detection and identification (FDI) strategy. During normal operation on an industrial-scale pilot plant the RLS and EKF run and allow faults to be detected and identified. When a fault is detected, the RLS is halted to prevent contamination of parameters, but the EKF state estimation is continued.

Consider the system, of which the order, n is known:

$$\overline{x}_i = A\overline{x}_{i-1} + B\overline{u}_{i-1} + \delta_{i-1}$$

Here, A is a n-square matrix of unknown elements and B is not necessarily square, but its elements are also unknown. The final term indicates a noise component.

That is:

$$\bar{x}_i = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ \vdots & \vdots & \vdots \\ a_{n1} & a_{n2} & a_{n3} \end{bmatrix} \bar{x}_{i-1} + \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ \vdots & \vdots & \vdots \\ b_{n1} & b_{n2} & b_{n3} \end{bmatrix} \bar{u}_{i-1} + \delta_{i-1}$$

Here the aij and bij are the effective coefficients of $\bar{x}_i$ and $\bar{u}_i$ respectively and will vary as the operating point of the process changes.

More simply:

$$\bar{x}_i = \begin{bmatrix} \bar{a}_1^T \\ \bar{a}_2^T \\ \vdots \\ \bar{a}_n^T \end{bmatrix} \bar{x}_{i-1} + \begin{bmatrix} \bar{b}_1^T \\ \bar{b}_2^T \\ \vdots \\ \bar{b}_n^T \end{bmatrix} \bar{u}_{i-1} + \delta_{i-1} \qquad (3.38)$$

where for example $\bar{a}_1 = \begin{bmatrix} a_{11} \\ a_{12} \\ a_{13} \end{bmatrix}$ and $\bar{a}_1^T = \begin{bmatrix} a_{11} & a_{12} & a_{13} \end{bmatrix}$

and since $\bar{a}_i^T \bar{x} = \bar{x}^T \bar{a}_i$

$$\bar{x}_i = \begin{bmatrix} \bar{x}_{i-1}^T & \bar{0}^T & \cdots & \bar{0}^T \\ \bar{0}^T & \bar{x}_{i-1}^T & \cdots & \bar{0}^T \\ \vdots & \vdots & \ddots & \bar{0}^T \\ \bar{0}^T & \bar{0}^T & \bar{0}^T & \bar{x}_{i-1}^T \end{bmatrix} \begin{pmatrix} \bar{a}_1 \\ \bar{a}_2 \\ \vdots \\ \bar{a}_n \end{pmatrix} + \begin{bmatrix} \bar{u}_{i-1}^T & \bar{0}^T & \cdots & \bar{0}^T \\ \bar{0}^T & \bar{u}_{i-1}^T & \cdots & \bar{0}^T \\ \vdots & \vdots & \ddots & \bar{0}^T \\ \bar{0}^T & \bar{0}^T & \bar{0}^T & \bar{u}_{i-1}^T \end{bmatrix} \begin{pmatrix} \bar{b}_1 \\ \bar{b}_2 \\ \vdots \\ \bar{b}_n \end{pmatrix} + \delta_{i-1} \qquad (3.39)$$

If one augments the vectors $\bar{a}$ and $\bar{b}$:

$$\bar{x}_i = \left[\begin{array}{cccc:cccc} \bar{x}_{i-1}^T & \bar{0}^T & \cdots & \bar{0}^T & \bar{u}_{i-1}^T & \bar{0}^T & \cdots & \bar{0}^T \\ \bar{0}^T & \bar{x}_{i-1}^T & \cdots & \bar{0}^T & \bar{0}^T & \bar{u}_{i-1}^T & \cdots & \bar{0}^T \\ \vdots & \vdots & \ddots & \bar{0}^T & \vdots & \vdots & \ddots & \bar{0}^T \\ \bar{0}^T & \bar{0}^T & \bar{0}^T & \bar{x}_{i-1}^T & \bar{0}^T & \bar{0}^T & \bar{0}^T & \bar{u}_{i-1}^T \end{array}\right] \begin{pmatrix} \bar{a}_1 \\ \bar{a}_2 \\ \vdots \\ \bar{a}_n \\ \cdots \\ \bar{b}_1 \\ \bar{b}_2 \\ \vdots \\ \bar{b}_n \end{pmatrix} + \delta_{i-1} \qquad (3.40)$$

This is now of the form $\bar{y}(t) = G(t)\bar{p}$. During the identification procedure, $\bar{x}_i, \bar{x}_{i-1}$ and $\bar{u}_{-1}$ are observed on-line and the parameters in A and B are found . Hence $\bar{p}$ (the vector of parameters to be identified) is updated recursively. This is illustrated in *figure* 3.1 below:

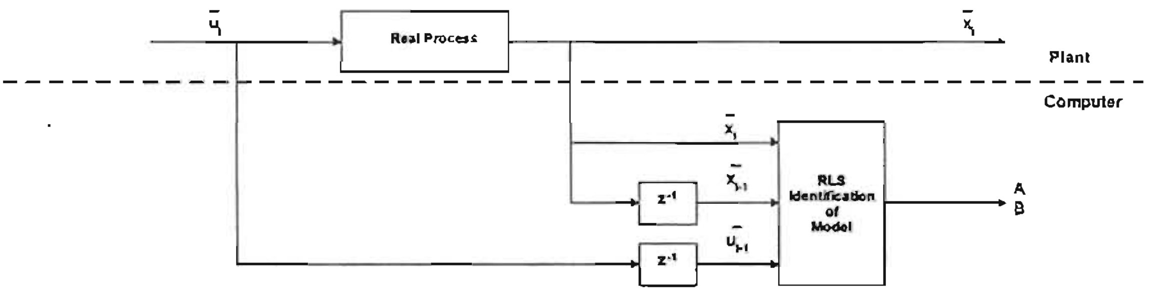


Figure 3.1: General On-line Recursive Least Squares (RLS) Identification Configuration

The algorithm is laid out below:

$$K_i = M_iG_i^T\left[G_iM_iG_i^T + R\right]^{-1} \tag{3.41}$$

$$\bar{p}_i = \bar{p}_{i-1} + K_{i-1}[\hat{y}_{i-1} - G_{i-1}\bar{p}_{i-1}] \tag{3.42}$$

$$M_{i+1} = \left[I - K_iG_i\right]M_i + Q \tag{3.43}$$

An initial error covariance matrix, M_0 is defined and is usually chosen to be small (e.g. 0.001×I) and diagonal. Also Q and R are normally distributed error covariance matrices. These allow the user to place confidence in the model or observations respectively. Here the error covariance matrice Q determines the rate at which the parameter vector changes, whilst the *R* matrix expresses the expected error covariance in the measurements.

Kalman (1960) showed that K_i gives the optimum gain for the system. Equation (3.41) then updates the parameters to be identified, using the gain evaluated above to match the measured vector, $\hat{y}$ and predicted vector, $\bar{y}$. The error covariance matrix, M is then updated and the process repeats itself.

This procedure is very useful in non-linear processes where the coefficients a_{ij} and b_{ij} vary as the operating point changes. The RLS procedure then allows on-line automatic adjustment of the control algorithm on the basis of the first order linear model:

$$\bar{x}_i = A\bar{x}_{i-1} + B\bar{u}_{-1} \tag{3.44}$$

This is known as adaptive control.

3.4 Kalman Filtering

State estimation techniques for the polymerisation process have been considered by a number of researchers, including McAuley and MacGregor (1993), who used an extended Kalman filter (EKF). The idea of Kalman filtering came about some 40 years ago (Kalman 1960). The EKF was created for dealing with non-linear systems and is discussed in a text by Jazwinski (1970). Not many industrial applications of the EKF are reported, but a fair number of those reported on are in the polymerisation and biotechnology fields. Both of these fields require expensive measurement alternatives and state estimation techniques offer definite cost benefits (Wilson, Agarwal et al. 1998). However, it appears difficult to find the correct formulation to make it useful in a real process environment (Kozub and MagGregor 1992). Wilson, Agarwal et al. (1998) go on to assess the industrial feasibility of an on-line state estimator. One of the earliest bench-scale applications to polymerisation was by Jo and Bankoff (1976). Kozub and MagGregor (1992) used an EKF as one of three methods for state estimation in the semi-batch emulsion copolymerisation of styrene/butadiene rubber (SBR) as a case study. Mulholland and Fernandes (1997) use a Smith predictor combined with a Kalman filter to account for dead time when modelling the same Poly 2 reactor considered in this study. Applications of EKF's in other areas of research include Tham and Parr (1994) and Zorzetto and Wilson (1996).

The Kalman filtering algorithm provides a means of estimating the values of all state variables when perhaps not all of them are measurable. It uses the linear model described by equation (3.44) above. G can be defined as a selection matrix, containing 1's to select the measurable elements and 0's so that the unmeasurable elements are not selected.

$$\bar{y}_{i-1} = G\bar{x}_{i-1}$$

Hence, the predicted states, $\bar{y}$ can be compared with the measured variables, $\hat{y}$. If one includes the filter with a gain matrix giving the necessary increments in $\bar{x}$, then the differences between $\bar{y}_i$ and $\hat{y}$ can be reduced. The arrangement Is shown below:

$$\bar{x}_i = A\bar{x}_{i-1} + B\bar{u}_{i-1} + K\left[\hat{y}_{i-1} - G\bar{x}_{i-1}\right] \tag{3.45}$$

With good tuning of the error covariance matrices, Q and R (defined below), the filter tends to follow observations smoothly, dictated by its attempt to simulate the dynamics of equation (3.44).

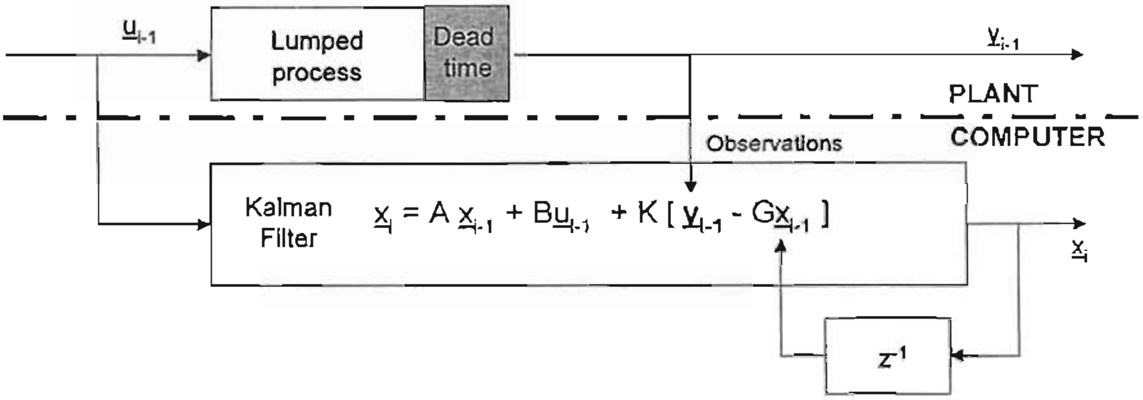


Figure 3.2: General Discrete Kalman Filter Configuration

The algorithm (equations 3.46 to 3.48 below) is similar to the RLS above. The equation that evaluates the gain (3.46) is identical but the other two differ slightly. Again there are 'tuneable' diagonal error covariance matrices, Q and R. Specification of high values in the model error covariance matrix Q forces the filter to follow observations more closely, whilst high measurement errors in *R* force it to follow the model more closely.

$$K_i = M_i G_i^T \left[G_i M_i G_i^T + R\right]^{-1} \tag{3.46}$$

$$\bar{x}_i = A\bar{x}_{i-1} + B\bar{u}_{i-1} + K_{i-1}[\hat{y}_{i-1} - G_{i-1}\bar{x}_{i-1}] \tag{3.47}$$

$$M_{i+1} = A_i\left[I - K_i G_i\right] M_i A_i^T + Q \tag{3.48}$$

There are numerous advantages and disadvantages of the EKF and the following have been obtained from the research of various workers:

Advantages:

- Its recursive nature.
- Accounts for model uncertainties.
- Able to obtain reliable state estimates from a few measurable measurements.
- Handles unstable and Integrating processes.
- Comparably low computational effort.
- Its simplicity.
- Robust and theoretically suitable for industrial applications.
- The theory is well understood.

Disadvantages:

- Local linearisation of model is required.
- Assumption of Gaussian and white noise processes.
- Requires good knowledge of the process in order to formulate a model suitable for an industrial application.
- Tuning can be arduous and is usually done by trial-and-error.
- May diverge under certain process conditions.
- Difficult to implement and maintain industrially.

CHAPTER 4

Control at Poly 2

4.1 Current Control Scheme

There are two main layers of control used on the plant: base-layer and advanced layer control.

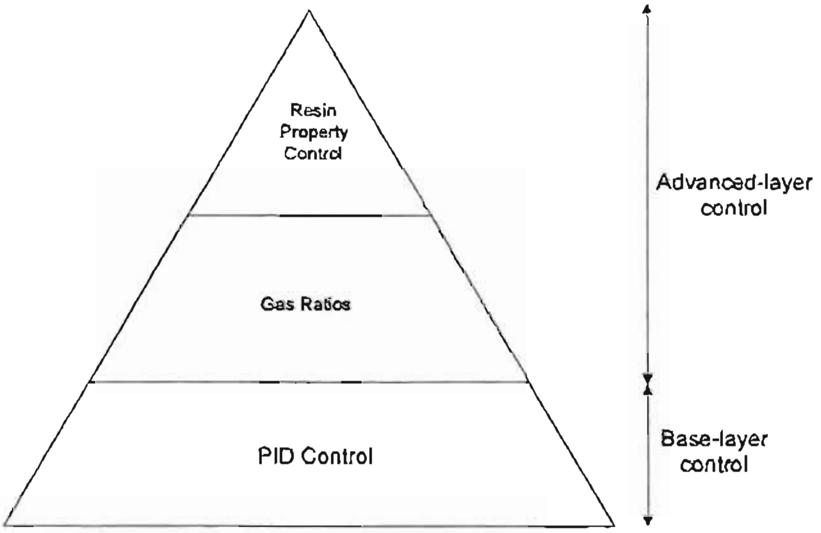


Figure 4.1: Control layers used at Poly 2

The base-layer is used for control over the reactor environment, that is temperatures, pressures, flows, concentrations, etc. This is very well handled by multi-loop PID control systems.

The advanced-layer controls intermediate variables, such as gas ratios, and the polymer properties, such as density and MI. This is obviously more difficult since these properties cannot be measured on-line. At the top of the advanced-layer is the resin property control (RPC). This is operating well at Poly 2 under license from Union Carbide. There are two gas ratios controlled, namely H2/C2 and C4/C2 or C6/C2. These in turn affect the resin properties. More detail is given in *section* 4.1.2 below.

The following figure and description is taken from Narotam (1999).

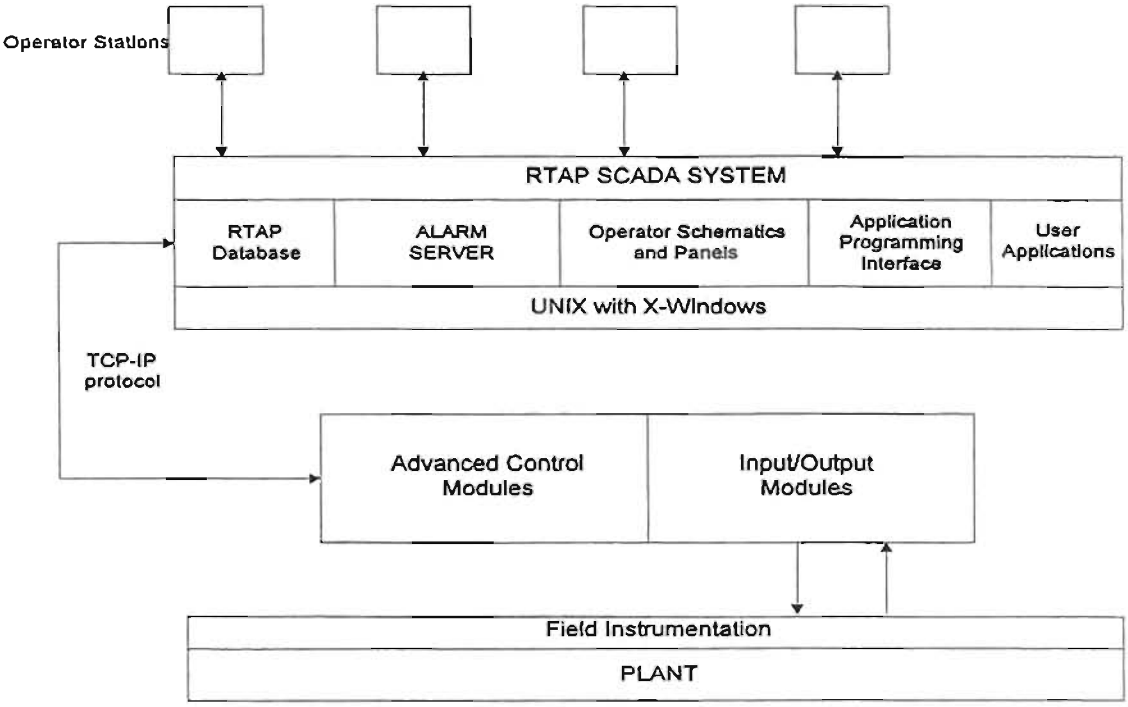


Figure 4.2: POLIFIN's Control Setup (Narotam 1999)

Polifin uses a Distributed Control System (DCS) supplied by Moore Controls to control the reactor. Referring to *figure* 4.2 above, signals to and from field instrumentation are connected to Input/Output (I/O) modules. These I/O modules are attached to Advanced Control Modules (ACM's). It is here that the standard PID, sequencing and logic control are performed. Scan cycle times of the ACM's normally range from 200-500 milliseconds. Ethernet cables provide the connection with the supervisory control and data acquisition (SCADA) system (the human-machine interface) and communication takes place via the TCP-IP protocol.

Hewlett Packard's RTAP (Real-time Application Programming) setup is used as the SCADA system. The system executes on the UNIX operating system on Digital Alpha workstations (APS1 for development work and APS100 / APS101 on the plant). Data from the ACM's are scanned into the RTAP database every second through Network Interface Modules (NIM's). This information can be read by an operator and displayed on schematics if required. Operator actions can in turn be sent from the SCADA system to the ACM's. The alarm server allows alarming functions to be performed in the database and displayed on the operator's screen.

The Application Programming Interface (API) allows data acquisition from the database. It consists of a set of C/C++ header and library files where objects are provided to access the RTAP database and the scheduling functions. In order to obtain the requested data for the study, a custom C/C++ database interface was created by the control engineers at Polifin Ltd. This allowed data to be sampled from the RTAP database at a user-specified interval (20 seconds in this case). The logged output is a comma-delimited text file. More detail of this is given in *section* 5.3. The full set of data logged is given in *table* A2 and only the data used for the study in *table* 5.2.

The following operator control strategies are implemented on the plant:

4.1.1 Pressure and Partial Pressure Control

This type of control is vital for controlling catalyst activity in M-resins. In F-resins, although catalyst activity is unaffected, good control is necessary for economic reasons.

Reactor pressure is kept as high as possible since it increases productivity. Low pressures may also lead to loss of fluidisation or compressor surging. Pressure is therefore kept constant and a pressure controller regulates it. This controller regulates the ethylene feed rate to the reactor.

The reactor pressure tends to fall as the reaction proceeds and additional gas is added to maintain operating pressure. Ethylene partial pressure control is achieved by split-range control mechanism: addition or venting. If the partial pressure is too low, reactor gas is vented to flare. Conversely, if it is too high, Nitrogen is injected to lower the partial pressure.

4.1.2 Gas Ratio Control

Two gas ratios, comonomer/ethylene (C4/C2 or C6/C2) and hydrogen/ethylene (H2/C2) are controlled. These ratios affect the polymer grade that is being produced. Comonomer ratio predominantly affects density whereas hydrogen ratio mostly affects melt index. However, the two are inter-linked. In fact it is not possible to make a single change to reactor operating conditions without affecting both properties.

The incorporation of comonomer into polymer chains causes more short side branches. The result is that the chains cannot pack too tightly together and this leads to a decrease in density.

An increase in hydrogen in the polymer chain causes a lower average molecular weight of polymer and a higher melt index.

Control is very difficult since the polymerisation equations (chapter 3) are coupled and the system is a multivariable one. Furthermore, the system has large time constants and responds slowly to changes. At the moment the cycle gas is analysed every 3 minutes by two on-line gas chromatographs. The results are fed to the DCS, which calculates the current ratios. These ratios are compared with the ratio set points by ratio controllers. The output of these ratio controllers is multiplied with the current measured ethylene flow rate to achieve a setpoint for the butene, hexene and hydrogen flow control loops.

4.1.3 Temperature Control

Optimal reactor temperature is vital as it affects resin density and melt index for both M and F-resins. The reactor temperature must be kept optimal; increasing it raises reaction rate but if it is too high the polymer tends to stick together and catalyst productivity decreases.

Temperature affects melt index and density to a lesser extent for M-resins. Thus, when controlling these properties for M-resins temperature is kept constant and the cycle gas composition is altered. For F-resins the effect of temperature is much greater, hence temperature is adjusted to control melt index and density.

The reactor bed is essentially isothermal due to the good mixing characteristics of fluidised beds. The reactor temperature is achieved by altering the cycle gas temperature (inlet to the reactor). This gas stream is manipulated via the cycle gas cooler. This cooler has a tempered water flow passing through it. The closed-loop tempered water system allows the water to pass either through a heater or a plate cooler (uses cooling tower water). During normal operation the water passes through the heater but if more cooling is required, the water is redirected through the plate cooler. A temperature controller compares bed temperature (measured by a thermocouple located a third of the way up the reactor) with set point. This signal is then sent to the reactor inlet temperature controller to determine if the tempered water should pass through or bypass the tempered water cooler. This form of cascaded control is intended to give a speedy response to temperature variations.

4.1.4 Catalyst Feed Control

Catalyst feed rate is the primary control variable for production rate. An increase in feed rate leads to an increase in production rate. This increase in polymerisation requires ethylene feed rate to be increased and in turn the other feeds are raised. The co-catalyst triethylaluminium (Teal) is used in the production of M-resin only. It activates the catalyst and also acts as a poison scavenger. Teal is fed in proportion to the flow of ethylene by a flow controller.

The production rate is calculated by the simple heat balance:

$$\text{Production rate (t/h)} = \frac{Cycle\ gas\ rate\,(t/h) \times \left[C_pT_{bed} - C_pT_{in}\right](kJ/t)}{Heat\ of\ reaction\,(kJ/t)} \qquad (4.1)$$

The denominator is actually a "fudge factor" used to match the actual with the calculated production rate as large amounts of heat are lost from the reactor.

There are two catalyst feeders located on opposite sides of the straight section of the reactor. Due to the high activity of the catalyst, only small quantities of catalyst are fed (roughly 3 kg/h). Metering discs with holes in them deliver catalyst to a pick-up block where nitrogen flow can distribute it into the reactor. The control variable is the speed of the catalyst feeder motor. This is normally increased very gradually when increasing production rate in order to avoid chunk formation and hot spots.

4.1.5 Cycle Gas Flow Control

The minimum fluidisation rate is the rate at which bed particles become supported by the cycle gas flow. This gas flow is however controlled at a point higher than the minimum fluidisation rate to ensure an expanded bed and a good degree of mixing and heat transfer. It is not set too high though to ensure that there is no solids carryover. The cycle gas line features a venturi flow meter for measurement and a butterfly valve on the same line for control.

4.1.6 Bed Level Control

Bed level does not affect resin properties but it does have an impact on production rate and catalyst productivity, both of which will increase with an increase in bed level at a constant catalyst feed.

The bed level is kept constant and the best level for operation is a few feet below the bottom of the expanded section. Therefore bubbles bursting at the top of the bed can scrub off resin dust particles which would otherwise accumulate into sheets and slide off the walls, clogging the reactor. A bed weight controller maintains a constant bed weight by regulating the product discharge.

4.1.7 Reactor Computer Control

Licensed software ensures that computer control can be implemented on the plant. The reason that this control is used is twofold:

- The process responds very slowly to settings and an operator may not see the results of a grade change (this may take up to 3 bed turnovers) during a shift. This may cause over-tampering.

- The resin properties are largely affected by the gas ratios. As mentioned above in *section* 4.1.2 these ratios are inter-linked and it is these complex inter-relationships that make control of resin properties so difficult.

4.1.8 Melt Index and Density Control

The actual instrumentation diagram for resin property control on Poly 2 cannot be reproduced, as it is confidential information of Union Carbide Corporation. However, *figure* 4.3 below (McAuley and MacGregor (1993)) illustrates a similar setup to the scheme at POLIFIN Ltd.

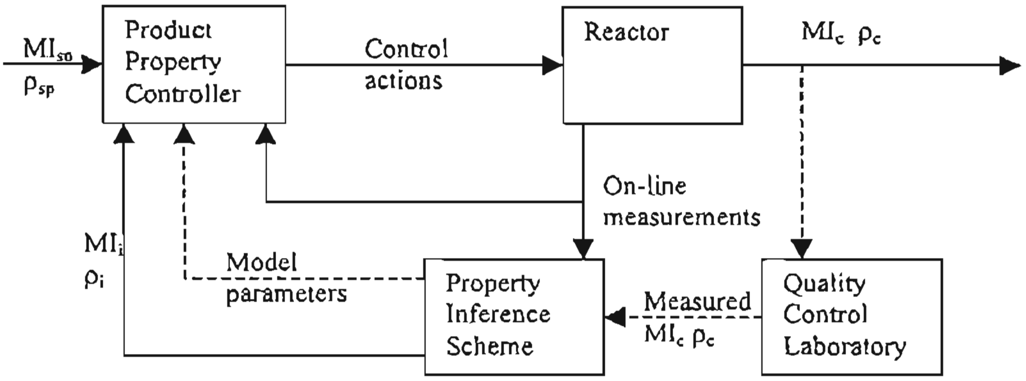


Figure 4.3: Product property control scheme, similar to POLIFIN Ltd

Due to the fact that MI and ρ cannot be measured directly, MI_i and ρ_i, used in feedback control, are inferred from measurements and theoretical models. These parameters are updated every few hours by means of laboratory measurements of cumulative melt index and density, MI_c and ρ_c. The property inference scheme predicts values for instantaneous melt index and density, MI_i and ρ_i between laboratory measurements, by using parameter estimates from the previous update step. The product property controller then uses these estimates, setpoints, model parameters and on-line measurements to calculate control actions for the reactor.

Table 4.1 summarises the control variables and the corresponding effects on the resin properties.

Increasing variable	F-resins		M-resins	
	MI	Density	MI	Density
Temperature	Increase	Decrease	Increase	Increase
C4/C2 ratio	Increase	Decrease	Increase	Decrease
C6/C2 ratio	Increase	Decrease	Increase	Decrease
H2/C2 ratio	Little effect	Little effect	Increase	Increase
O2 concentration	Increase	Increase	N/A	N/A
Poisons	Increase	Increase	Little effect	Little effect

Table 4.1: Summary of variables affecting resin properties

The computer calculates the instantaneous and bed average properties and suggests set points required to make a particular grade of resin. The gas ratios required to achieve these resin properties are given by the computer and are fed to the DCS. Laboratory analysis is supposedly carried out every 2 hours (although records analysed show new results from 55 minutes to 255 minutes apart) during operation. These actual results are compared with the computer's predicted results. The computer model is then updated to take these variations into account.

There is an on-line continuous melt index unit, which is also read by the computer, but density uses only the laboratory analysis results.

4.2 The Control Problem

There are a few problems with the current control scheme:

- The gas space acts as a pure integrator, which means that if a certain gas flow is increased, its composition will continue to ramp until something is done to prevent the rise. The gas space is also openloop unstable. This prohibits the use of algorithms such as dynamic matrix control (DMC) on the plant. Interestingly though, a paper by Gupta (1998) presents a method for modifying the DMC algorithm for use in integrating processes, such as at Poly 2. The modification allows the steady-state offset, present during sustained load changes to be eliminated. This warrants further investigation by researchers working in this area.

- The gas flow ratios controlled are C4/C2, (or C6/C2) and H2/C2. The problem is that there is no accommodation of imbalances. That is, if the butene ratio needs to be raised, more butene will be added to the system. However, this will cause the other ratios to be compromised in an attempt to accommodate the change. That is there is this nesting effect, and the controllers continuously work against each other.

- The polymer properties depend heavily on the gas compositions in the reactor. These compositions are measured on-line by one of two gas chromatographs (GC's). Polymer properties respond slowly to settings, making this a difficult control problem. The plant laboratory experimentally determines key properties like melt index (MI) and density off-line and intermittently (new results are entered every 55-255 minutes). These results are used to update the setpoints for proprietary on-line resin property control software. Therefore a large amount of off-spec product can be produced before an incorrect gas composition measurement is detected.

CHAPTER 5

Control Algorithm Formulation and Operation

5.1 The Control Objectives

One way of overcoming some of the problems mentioned in *section* 4.2 is to implement real-time fault detection and property prediction, using a model. The optimal prediction and control of the gas ratios can reduce the amount of off-spec product.

The objectives of the proposed control algorithm are threefold:

- The primary objective of the proposed control scheme is to improve product quality by reducing the quantity of off-spec product produced during grade changeovers. This would arise from better control over the gas ratios, which are used to accomplish some of these grade changes.
- A secondary objective is better control of the gas ratio control scheme, which would impact on the primary objective, and result in steadier properties in normal operation.
- A fault detection system can be implemented. This will alert operators to erroneous composition measurements and will support both present and proposed control of the plant.

5 1.1 A Different Modelling Approach

A colleague, Ryan Dunwoodie, also approached the control of ethylene polymerisation at Poly 2, but from a different angle. The method used was a "black-box" description of the process by regression of an artificial neural network (ANN). It was the intention that the ANN would eventually be used in conjunction with the model used in this research study (Thomason, Dunwoodie et al. 2000). The ANN provides a simple means of inversion to obtain a gas composition controller and will cover gaps in the interpretation of the kinetics.

5.2 Proposed Simpler Model: Algorithm Formulation

McAuley (2000) stresses that different types of models are appropriate for different applications. The temptation is to design a rigorous model to fulfil every possible need. Such a model would be too computationally intensive for on-line use. Therefore when proposing a model for on-line use in industry, it is necessary to make simplifications where appropriate. A complex site-based kinetic modelling scheme (based on sections 3.2 and 6.1.1) has its strengths in off-line use. It can be used for prediction of MWD and CCD, given a set of rate constants.

One of the reasons that kinetic models do not find use in industry is not only the complexity of polymer reactions but also the fact that these models use a large number of kinetic parameters. These kinetic parameters can be determined for a given set of reaction conditions via expensive, time-consuming techniques (TREF, SEC).

The plant flow scheme can be simplified for modelling purposes and is shown in *figure* 5.1 below. This is a simplified version of *figure* 2.3, showing only the dashed lines around the reactor input and output streams.

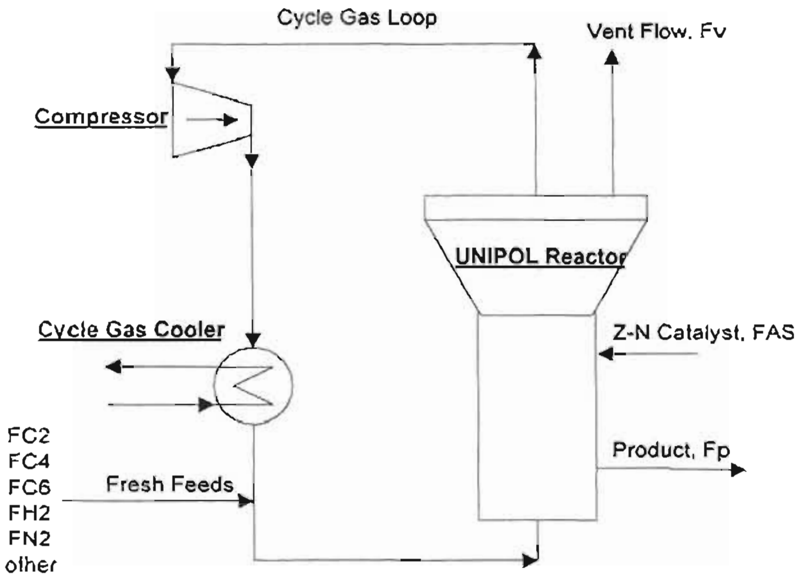


Figure 5.1: Poly 2 process simplified for the purpose of modelling

POLIFIN has proposed that a non-linear feedforward/feedback controller be designed. This would be able to account for changes in reactor temperature, vent flow rate, catalyst feed rate and bed level (the main parameters affecting grade changeovers). The ultimate goal would be to run a filter/estimator in a feedforward situation on the plant. One would like to know how much to step a flow (e.g. H_2) and for how long (in other words the ramp rate) in order to get from one steady state to the next.

The complex site-based model has been included in its entirety in *section* 3.2 for reference. Due to the complexity of this model, it was necessary to formulate a reduced model. This is in line with POLIFIN's final goal of an on-line application.

The goal of this chapter is to present the formulation of an alternative, simpler model of the polymerisation kinetics, suitable for on-line use. Whenever a grade change takes place, new kinetic parameters are required for good predictions of plant behaviour. This problem is addressed by using an on-line recursive least squares (RLS) parameter estimator to identify these kinetic parameters. The values are then used in a state estimation scheme (in the form of an extended Kalman filter (EKF)) in order to obtain useful state estimates for the reactor. The plant model is first formulated and linearised into a system of first order differential equations. It is then discretised and the vector of kinetic parameters, $\bar{p}$ for the RLS is isolated. Thereafter the algorithm sequence for the RLS and EKF are presented respectively.

Using McAuley and MacGregor (1993) as a basis, simplified balances were written for ethylene, comonomer, hydrogen, nitrogen, catalyst sites, monomer fractions in the polymer and total gas and polymer in the reactor. These balances ((5.1) through (5.11)) can be used to predict gas and solid molar inventories in the reactor, given flow rates of ethylene, comonomer, hydrogen, nitrogen, active catalyst sites, vent and product.

$$\frac{dn_{C2}}{dt} = F_{C2} - k_{P2} n_{C2} n_{AS} - \left(\frac{F_v}{N_g}\right) n_{C2} - F_p S_2 n_{C2} \tag{5.1}$$

$$\frac{dn_{C4}}{dt} = F_{C4} - k_{P4} n_{C4} n_{AS} - \left(\frac{F_v}{N_g}\right) n_{C4} - F_p S_4 n_{C4} \tag{5.2}$$

$$\frac{dn_{C6}}{dt} = F_{C6} - k_{P6} n_{C6} n_{AS} - \left(\frac{F_v}{N_g}\right) n_{C6} - F_p S_6 n_{C6} \tag{5.3}$$

$$\frac{dn_{H2}}{dt} = F_{H2} - k_H n_{H2} n_{AS} - \left(\frac{F_v}{N_g}\right) n_{H2} - g_H n_{H2} \tag{5.4}$$

$$\frac{dn_{N2}}{dt} = F_{N2} - \left(\frac{F_v}{N_g}\right) n_{C2} \tag{5.5}$$

$$\frac{dn_{AS}}{dt} = F_{AS} - \left(\frac{F_P}{N_P}\right) n_{AS} - k_d n_{AS} \tag{5.6}$$

$$\frac{dn_{P2}}{dt} = k_{P2} n_{P2} n_{AS} - \left(\frac{F_P}{N_P}\right) n_{P2} \tag{5.7}$$

$$\frac{dn_{P4}}{dt} = k_{P4} n_{P4} n_{AS} - \left(\frac{F_P}{N_P}\right) n_{P4} \tag{5.8}$$

$$\frac{dn_{P6}}{dt} = k_{P6} n_{P6} n_{AS} - \left(\frac{F_P}{N_P}\right) n_{P6} \tag{5.9}$$

$$\frac{dN_g}{dt} = F_{C2} + F_{C4} + F_{C6} + F_{H2} + F_{N2} - F_v - \left(k_{P2} n_{C2} + k_{P4} n_{C4} + k_{P6} n_{C6} + k_H n_{H2}\right) n_{AS} - g_H N_g \tag{5.10}$$

$$\frac{dN_p}{dt} = \left(k_{P2} n_{C2} + k_{P4} n_{C4} + k_{P6} n_{C6}\right) n_{AS} - F_P \tag{5.11}$$

n_{AS} is the number of active catalyst sites in the reactor
F_v is the vent flow of gas
F_P is the polymer outflow rate [as (kmol C_2, C_4, C_6) s^{-1}]
n_{Cx} is the moles of monomer/comonomer in the gas
n_{H2} and n_{N2} are the number of moles of hydrogen and nitrogen in the gas
n_{Px} is the number of moles of monomer/comonomer present in the polymer.

k_{P2} is the propagation rate constant for ethylene
k_{P4} is the propagation rate constant for butene
k_{P6} is the propagation rate constant for hexene
k_H is the rate constant for hydrogen
g_H is a mismatch factor to account for uncertainty in the hydrogen mass balance
k_d is a deactivation rate constant
n_{AS} is the total number of moles of active sites in the reactor

These equations are simplified into a manageable model later.

The total number of moles in the gas phase is estimated using the ideal gas law:

$$N_g = \frac{PV_g}{RT} \tag{5.12}$$

and the total number of moles of solid polymer phase is evaluated as follows:

$$N_p = \text{bed mass / mMMp} \tag{5.13}$$

Here *mMMp* is the mean molecular weight of polymer and is calculated from the individual monomer molecular weights and their corresponding mole fractions. These are evaluated from the cycle gas analyser readings.

Then the unmeasurable fractions of monomer and comonomer in the polymer are calculated as follows:

n_{Pi} = x_typical*N_p (5.14)

x_typical is a vector containing typical mole fractions of monomer and comonomer in the polymer. The mole fraction of butene comonomer in the copolymer has been correlated in *figure* 5.2 below (McAuley 1991). This figure was used as a guide for obtaining initial estimates of *x_typical* for the various grades tested. This parameter was initially estimated at 0.98 for ethylene and 0.02 for comonomer. The situation was later improved in that the Kalman filter was able to provide updated estimates.

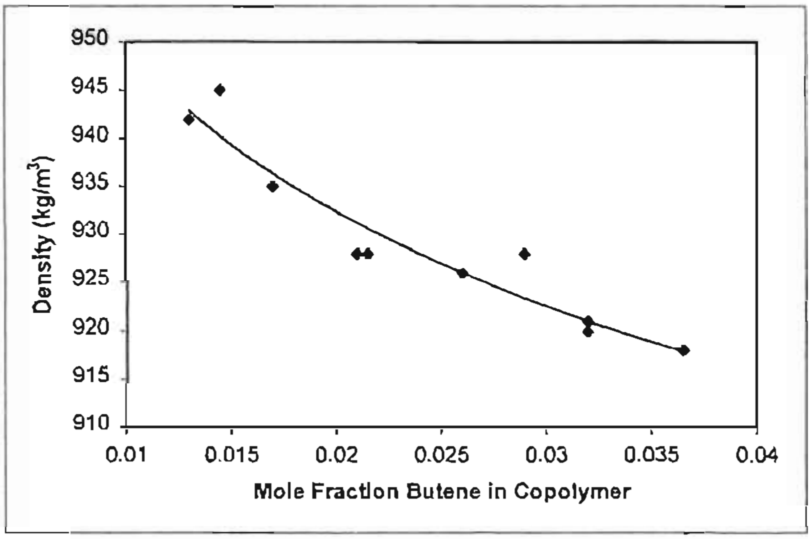


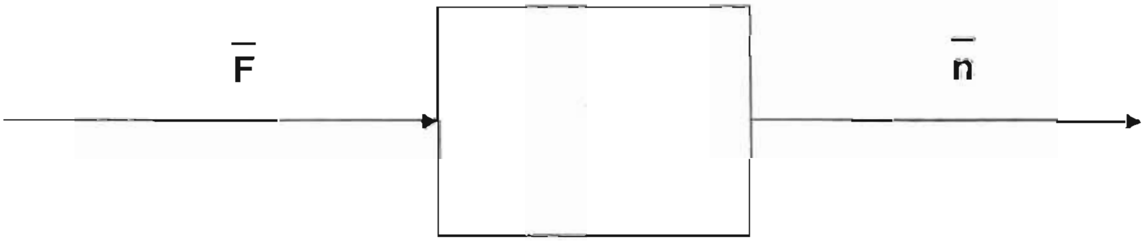
Figure 5.2: Mole Fraction Butene In Coploymer

A basic linear open-loop multivariable system can be represented as a system of first-order differential equations as follows:

$$\frac{d\bar{x}}{dt} = A\bar{x} + B\bar{u} \tag{5.15}$$

where u are the inputs and x are the outputs.

The system under investigation takes the form:



Hence, writing equations (5.1) through (5.11) in vector form:

$$\frac{d\bar{n}}{dt} = A\bar{n} + B\bar{F} \tag{5.16}$$

where

$$\bar{n} = \begin{bmatrix} n_{C2} \\ n_{C4} \\ n_{C6} \\ n_{H2} \\ n_{N2} \\ n_{AS} \\ n_{P2} \\ n_{P4} \\ n_{P6} \\ N_g \\ N_p \end{bmatrix}$$

$$A = n_{AS}K + \frac{F_r}{N_g}G + g + \frac{F_p}{N_p}(H + H_{AS}) + F_p h \qquad (5.17)$$

The matrices used in parameter A above are of dimension 11×11 and are K, G, g, H, H_{AS} and h.

$$K = \begin{array}{c} \\ n_{C2} \\ n_{C4} \\ n_{C6} \\ n_{H2} \\ n_{N2} \\ n_{AS} \\ n_{P2} \\ n_{P4} \\ n_{P6} \\ N_g \\ N_p \end{array} \begin{array}{c} \begin{array}{ccccccccccc} n_{C2} & n_{C4} & n_{C6} & n_{H2} & n_{N2} & n_{AS} & n_{P2} & n_{P4} & n_{P6} & N_g & N_p \end{array} \\ \begin{bmatrix} -k_{p2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -k_{p4} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -k_{p6} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -k_H & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ k_{p2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & k_{p4} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & k_{p6} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -k_{p2} & -k_{p4} & -k_{p6} & -k_H & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ k_{p2} & k_{p4} & k_{p6} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

K is the propagation matrix. It contains the kinetic rate constants and selects the gaseous species taking part in the formation of polymer from monomer.

$$G = \begin{array}{c|ccccccccccc}
 & n_{C2} & n_{C4} & n_{C6} & n_{H2} & n_{N2} & n_{AS} & n_{P2} & n_{P4} & n_{P6} & N_g & N_P \\
\hline
n_{C2} & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
n_{C4} & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
n_{C6} & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
n_{H2} & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
n_{N2} & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\
n_{AS} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
n_{P2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
n_{P4} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
n_{P6} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
N_g & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
N_P & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{array}$$

The G matrix selects the fraction of each gaseous species leaving the reactor via the vent.

$$H = \begin{array}{c|ccccccccccc}
 & n_{C2} & n_{C4} & n_{C6} & n_{H2} & n_{N2} & n_{AS} & n_{P2} & n_{P4} & n_{P6} & N_g & N_P \\
\hline
n_{C2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
n_{C4} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
n_{C6} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
n_{H2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
n_{N2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
n_{AS} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
n_{P2} & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\
n_{P4} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\
n_{P6} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 \\
N_g & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
N_P & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{array}$$

H selects the mole fractions of each monomer in the polymer leaving the reactor with the product.

$$H_{AS} = \begin{array}{c|ccccccccccc}
 & n_{C2} & n_{C4} & n_{C6} & n_{H2} & n_{N2} & n_{AS} & n_{P2} & n_{P4} & n_{P6} & N_g & N_P \\
\hline
n_{C2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
n_{C4} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
n_{C6} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
n_{H2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
n_{N2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
n_{AS} & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\
n_{P2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
n_{P4} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
n_{P6} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
N_g & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
N_P & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{array}$$

H_{AS} is similar to H but selects the fraction of active sites leaving the reactor with the product.

$$g = \begin{array}{c|ccccccccccc}
 & n_{C2} & n_{C4} & n_{C6} & n_{H2} & n_{N2} & n_{AS} & n_{P2} & n_{P4} & n_{P6} & N_t & N_r \\
\hline
n_{C2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
n_{C4} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
n_{C6} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
n_{H2} & 0 & 0 & 0 & -g_H & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
n_{N2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
n_{AS} & 0 & 0 & 0 & 0 & 0 & -k_d & 0 & 0 & 0 & 0 & 0 \\
n_{P2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
n_{P4} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
n_{P6} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
N_t & 0 & 0 & 0 & -g_H & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
N_r & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{array}$$

This matrix accounts for site deactivation (k_d) and for uncertainties in the hydrogen mass balance (g_H).

$$h = \begin{array}{c|ccccccccccc}
 & n_{C2} & n_{C4} & n_{C6} & n_{H2} & n_{N2} & n_{AS} & n_{P2} & n_{P4} & n_{P6} & N_t & N_r \\
\hline
n_{C2} & -s_2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
n_{C4} & 0 & -s_4 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
n_{C6} & 0 & 0 & -s_6 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
n_{H2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
n_{N2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
n_{AS} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
n_{P2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
n_{P4} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
n_{P6} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
N_t & -s_2 & -s_4 & -s_6 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
N_r & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{array}$$

h selects the moles of monomer and comonomer dissolved in the product copolymer leaving the reactor.

$$B = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 \end{bmatrix}$$

$$\bar{F} = \begin{bmatrix} F_{C2} \\ F_{C4} \\ F_{C6} \\ F_{H2} \\ F_{N2} \\ F_{AS} \\ F_{v} \\ F_{r} \end{bmatrix}$$

$N_g = n_{C2} + n_{C4} + n_{C6} + n_{H2} + n_{N2}$

$N_p = n_{P2} + n_{P4} + n_{P6}$

The matrices K and g contain the parameters k_{P2}, k_{P4}, k_{P6}, k_H, g_H, k_d that will be identified by recursive least squares. To do this, first consider that the matrices K and g and the scalar parameter n_{AS} consist of some "original" part K_o, g_o, n_{ASo} and a small deviation from this, ΔK, Δg, Δn_{AS}. Then

$$A_o = n_{ASo}K_o + \frac{Fv}{Ng}G + g_o + \frac{Fp}{Np}\left(H + H_{AS}\right) + F_P h \tag{5.18}$$

$$A \approx A_o + n_{ASo}\Delta K + K_o\Delta n_{AS} + \Delta g \tag{5.19}$$

$$\begin{aligned} \frac{d\bar{n}}{dt} &= \left\{A_o + n_{ASo}\Delta K + K_o\Delta n_{AS} + \Delta g\right\}\bar{n} + B\bar{F} \\ &= A_o\bar{n} + B\bar{F} + \left\{n_{ASo}\Delta K + K_o\Delta n_{AS} + \Delta g\right\}\bar{n} \end{aligned} \tag{5.20}$$

Assuming that the deviation portion makes only a small contribution to the integral, use it to obtain an "Euler" contribution only:

$$\bar{n}_{t+\Delta t} = e^{A_o\Delta t}\bar{n}_t + \left[I - e^{A_o\Delta t}\right]A_o^{-1}B\bar{F} + \left\{n_{ASo}\Delta K + K_o\Delta n_{AS} + \Delta g\right\}\bar{n}_t\,\Delta t \tag{5.21}$$

Now

$\{ n_{ASo}\Delta K + K_o \Delta n_{AS} + \Delta g\}\bar{n}\Delta t =$

$$n_{ASo}\Delta t\begin{pmatrix} -\Delta k_{P2}n_{C2t} \\ -\Delta k_{P4}n_{C4t} \\ -\Delta k_{P6}n_{C6t} \\ -\Delta k_{H}n_{H2t} \\ 0 \\ 0 \\ \Delta k_{P2}n_{C2t} \\ \Delta k_{P4}n_{C4t} \\ \Delta k_{P6}n_{C6t} \\ -\Delta k_{P2}n_{C2t} - \Delta k_{P4}n_{C4t} - \Delta k_{P6}n_{C6t} - \Delta k_{H}n_{H2t} \\ \Delta k_{P2}n_{C2t} + \Delta k_{P4}n_{C4t} + \Delta k_{P6}n_{C6t} \end{pmatrix} + \Delta n_{ASo}\Delta t\begin{pmatrix} -k_{P2o}n_{C2t} \\ -k_{P4o}n_{C4t} \\ -k_{P6o}n_{C6t} \\ -k_{Ho}n_{H2t} \\ 0 \\ 0 \\ k_{P2o}n_{C2t} \\ k_{P4o}n_{C4t} \\ k_{P6o}n_{C6t} \\ -k_{P2o}n_{C2t} - k_{P4o}n_{C4t} - k_{P6o}n_{C6t} - k_{Ho}n_{H2t} \\ k_{P2o}n_{C2t} + k_{P4o}n_{C4t} + k_{P6o}n_{C6t} \end{pmatrix} + \Delta t\begin{pmatrix} 0 \\ 0 \\ 0 \\ -\Delta g_{H}n_{H2t} \\ 0 \\ -k_{d}n_{AS} \\ 0 \\ 0 \\ 0 \\ -\Delta g_{H}n_{H2t} \\ 0 \end{pmatrix}$$

(5.22)

$$= \Delta t\begin{bmatrix} -n_{ASo}n_{C2t} & 0 & 0 & 0 & 0 & 0 & -k_{P2o}n_{C2t} \\ 0 & -n_{ASo}n_{C4t} & 0 & 0 & 0 & 0 & -k_{P4o}n_{C4t} \\ 0 & 0 & -n_{ASo}n_{C6t} & 0 & 0 & 0 & -k_{P6o}n_{C6t} \\ 0 & 0 & 0 & -n_{ASo}n_{H2t} & -n_{H2t} & 0 & -k_{Ho}n_{H2t} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -n_{ASt} & 0 \\ n_{ASo}n_{C2t} & 0 & 0 & 0 & 0 & 0 & k_{P2o}n_{C2t} \\ 0 & n_{ASo}n_{C4t} & 0 & 0 & 0 & 0 & k_{P4o}n_{C4t} \\ 0 & 0 & n_{ASo}n_{C6t} & 0 & 0 & 0 & k_{P6o}n_{Cot} \\ -n_{ASo}n_{C2t} & -n_{ASo}n_{C4t} & -n_{ASo}n_{C6t} & -n_{ASo}n_{H2t} & -n_{H2t} & 0 & -k_{P2o}n_{C2t} - k_{P4o}n_{C4t} - k_{P6o}n_{C6t} - k_{Ho}n_{H2t} \\ n_{ASo}n_{C2t} & n_{ASo}n_{C4t} & n_{ASo}n_{C6t} & 0 & 0 & 0 & k_{P2o}n_{C2t} + k_{P4o}n_{C4t} + k_{P6o}n_{C6t} \end{bmatrix}\begin{pmatrix} \Delta k_{P2} \\ \Delta k_{P4} \\ \Delta k_{P6} \\ \Delta k_{H} \\ \Delta g_{H} \\ \Delta k_{d} \\ \Delta n_{AS} \end{pmatrix}$$

(5.23)

The large matrix is G$_{ID}$ and the vector following it is $\overline{p}$ (as seen in equation (3.42)). Thus

$$\bar{n}_{t+\Delta t} = A.\bar{n}_t + B.\bar{F} + G_{ID}\begin{pmatrix} \Delta k_{P2} \\ \Delta k_{P4} \\ \Delta k_{P6} \\ \Delta k_{H} \\ \Delta g_{H} \\ \Delta k_{d} \\ \Delta n_{AS} \end{pmatrix} \qquad (5.24)$$

i.e. $\bar{y} = G_{ID}\,\overline{p}$

with $\quad \bar{y} = \bar{n}_{t+\Delta t} - A.\bar{n}_t - B.\bar{F}$

The polymer molar inventories cannot be observed, so an 8×11 selection matrix is used to restrict to measurable states:

$$C = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Setting

$$\bar{y}' = C\,\bar{n}_{t+\Delta t} - CA.\bar{n}_t - CB.\,\bar{F}$$

$$= \bar{n}'_{t+\Delta t} - A.^{'}\bar{n}'_t - B.^{'}\,\bar{F}$$

with $A.^{'} = C\ A_s\ C$

Then

$$\bar{y}' = G'_{ID}\,\bar{p} \tag{5.25}$$

Now the right-hand-side above can be evaluated on each step to give a "measured" $\bar{y}$. The calculation sequence, in which an optimal gain K is calculated on each time-step, is as follows:

$$K_{i-1} = M_{i-1}\,G_{ID_{i-1}}^{T}\left[G_{ID_{i-1}}\,M_{i-1}\,G_{ID_{i-1}}^{T} + R\right]^{-1} \tag{5.26}$$

$$\bar{p}_i = \bar{p}_{i-1} + K_{i-1}\left[\bar{y}'_i - G_{ID_{i-1}}\,\bar{p}_{i-1}\right] \tag{5.27}$$

$$M_i = \left[I - K_{i-1}\,G_{ID_{i-1}}\right]M_{i-1} + Q \tag{5.28}$$

Here the error covariance matrix Q determines the rate at which the parameter vector changes, whilst the R matrix expresses the expected error covariance in the measurements of $\bar{y}$. The filter covariance matix M_0 is initialised with small diagonal terms. In this case the parameters found on each step are incremental adjustments to the kinetic rate constants. The RLS identification of the kinetic parameters k_{P2}, k_{P4}, k_{P6}, k_H, g_H, k_d and n_{AS} is represented graphically below in *figure* 5.3.

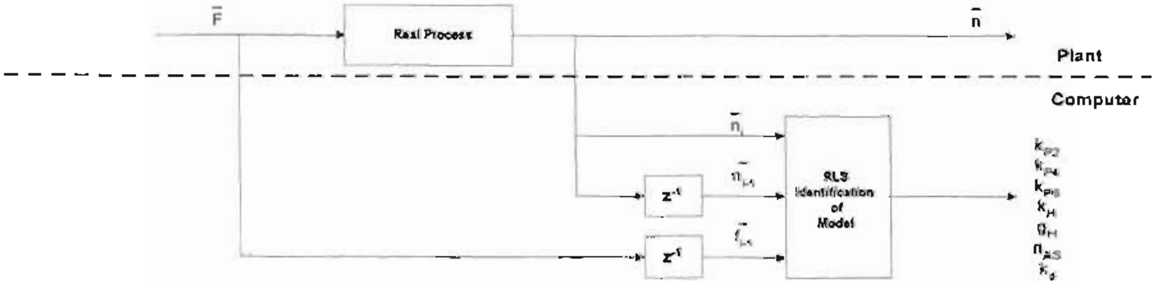


Figure 5.3: RLS Identification procedure for parameters k_{P2}, k_{P4}, k_{P6}, k_H, g_H, n_{AS} and k_d

The kinetic parameters identified from the above procedure are now used in a state estimation scheme in the form of an extended Kalman filter. The filter is based on the full state equations, which include the three polymer composition terms (molar inventories), as well as the gas molar inventories.

$$\overline{n}_{l+1} = e^{A_o\Delta t}\,\overline{n}_l + \left[I - e^{A_o\Delta t}\right]A_o^{-1}B\,\overline{F}_l$$

$$ie. \quad \overline{n}_{l+1} = A_l^{\bullet}\overline{n}_l + B_l^{\bullet}\overline{F}_l$$

The calculation sequence is

$$K_l = M_l G_{KF}^{T}\left[G_{KF}\, M_l G_{KF}^{T} + R\right]^{-1} \tag{5.29}$$

$$\overline{n}_{i+1} = A_l^{\bullet}\overline{n}_l + B_i^{\bullet}\overline{F}_l + K_l\left[\hat{y}_l - G_{KF}\overline{n}_l\right] \tag{5.30}$$

$$M_{l+1} = A_l^{\bullet}\left[I - K_l G_{KF_l}\right]M_l A_l^{\bullet T} + Q \tag{5.31}$$

The G_{KF} matrix, consisting of 0's and 1's in this case, selects the measured variables from the full state $\tilde{n}$ for comparison with the actual measurements $^{-}$. Notice that varying A* and B* matrices have been specified (EKF) to accommodate the RLS parameter updates. Specification of high values in the model error covariance matrix Q forces the filter to follow observations more closely, whilst high measurement errors in R force it to follow the model more closely.

This filter runs on a smaller time step (20s) than the parameter estimator, and asynchronously with it. It takes the present flows and measured gas inventories (composition × N_g) as inputs (selected by G_{KF} from the full state) and uses these to obtain a full state description that recognises to some extent the mass-balance. The filter acts as an observer and is useful for fault detection. An instrumentation fault will be revealed as an excessive difference between original and filtered signals. The identifier runs in the background, providing new parameters for the A* and B* matrices of the model. These kinetic parameters are only updated every time a GC update is detected (approximately every 180 s). This overall scheme is illustrated in *figure* 5.4 below.

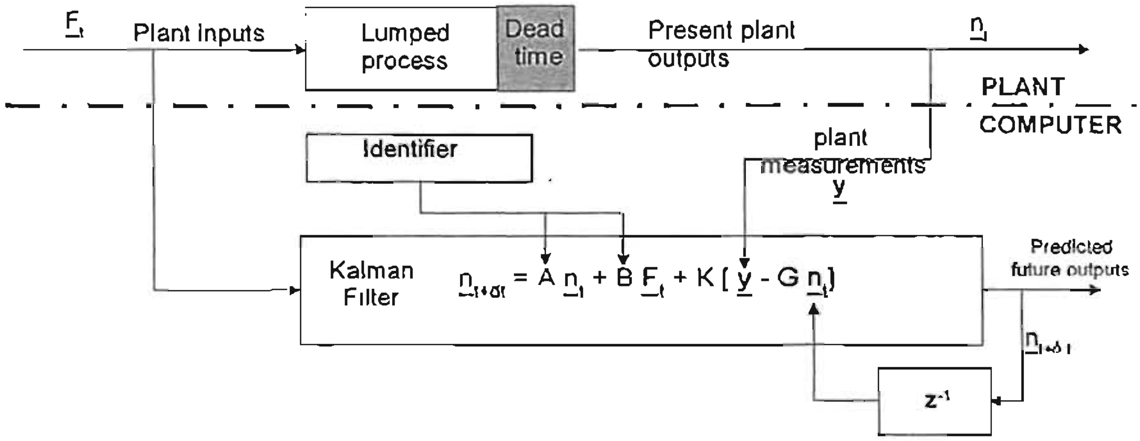


Figure 5.4: The State Estimator configuration – discrete Kalman Filter with the RLS Identifier

A^{*} and B^{*} as seen in the state estimator and A_{*} and B_{*} and as seen in the RLS parameter estimator are discrete forms of A and B. In order to get the discrete system for integration the matrix exponential is used. This is in effect a zero-order hold (ZOH) that allows samples to be taken at discrete sampling time intervals 'Δt'. The sampling interval for the filter will be 20s whereas for the identifier it will be governed by the interval between successive GC update detections (mentioned above and in section 6.2.2).

For example, for the Kalman filter, the system goes from:

$$\frac{d\bar{n}}{dt} = A\bar{n} + B\bar{F} \qquad (5.32)$$

to:

$$\overline{n_{i+1}} = A^{*}\overline{n_i} + B^{*}\overline{F_i} \qquad (5.33)$$

$$A^{*} = e^{A\Delta t}$$

$$B^{*} = [e^{A\Delta t} - I]A^{-1}B$$

Therefore:

$$\overline{n_{i+1}} = e^{A\Delta t}\overline{n_i} + [e^{A\Delta t} - I]A^{-1}B\overline{F_i} \qquad (5.34)$$

5.3 Analysis of Historical Data

The purpose of this procedure is to arrange historical plant data in a format in which it can be visually inspected with the intention of gaining a 'feel' for the plant characteristics. The procedure used in formatting raw plant data is as follows. The section of data to be analysed was selected, all the series adjusted to a similar scale and all the relevant data series plotted on the same set of axes. From the adjusted plot the following important aspects were noted.

- The relationships between the plots illustrate the basic dependencies between the variables. This assists one in understanding the dynamics of the system.
- The rates of change of the curves give a very important indication of the time dynamics of the process. This is important for choosing the data-sampling rate to be used for the model. The required sampling rate should be short enough to properly define changes in the curves, but not too short so as to cause unnecessary calculations.

For the purposes of running the original site-based off-line model it was necessary to determine a set of average operating conditions, during which the plant behaviour was stable and representative of standard plant conditions. This serves as a base condition from which to run the models and test the effect of disturbances. These average conditions were chosen from the plots during a period when all the series on the plots showed little change, indicating that there were steady conditions on the plant.

After formulating the simpler non site-based model, actual plant data was used. During the first visit to Sasolburg, the tags (e.g. a flow process variable FC_5080.PV) that were thought to be relevant were noted. A logger was then set up to log these tags at the desired interval. These were periodically sent to Durban via e-mail. A complete description of all the raw plant signals logged is given, in the order logged, in *table* A.2.

Parameter	Description	Units Logged	Units used in software
ETHFLO	R1 ethylene flow control	kg/h	kmol/s
BUTFLO	R1 butene flow control	kg/h	kmol/s
HEXFLO	R1 hexene flow control	kg/h	kmol/s
H2FLO	R1 hydrogen flow control	kg/h	kmol/s
N2_FLO	Cat feeder support flow	Nm^3/h	kmol/s
CAT_FDR1	Catalyst flow rate - feeder 1	kg/h	kg/s
CAT_FDR2	Catalyst flow rate - feeder 2	kg/h	kg/s
R1VENT_FLO	R1 reactor vent	Nm^3/h	kmol/s
Prod Rate	Production Rate	t/h	kg/s
ReacTemp	R1 Bed Temp. Control	°C	°C
R1PRESSURE	R1 pressure control	kPa-g	bar-abs
ANAL_1_on	A flag for which analyser is on	-	-
ETH_1	Ethylene analyser 1	%	%
ETH_0	Ethylene analyser 2	%	%
BUT	Butene analyser 1	%	%
BUT	Butene analyser 2	%	%
HEX	Hexene analyser 1	%	%
HEX	Hexene analyser 2	%	%
HYD	Hydrogen analyser 1	%	%
HYD	Hydrogen analyser 2	%	%
NIT	Nitrogen analyser 1	%	%
NIT	Nitrogen analyser 2	%	%
ETHANE	Ethane analyser 1	%	%
ETHANE	Ethane analyser 2	%	%
ISO	i-C5 analyser 1	%	%
ISO	i-C5 analyser 2	%	%
C4_INERTS	C4 inerts analyser 1	%	%
C4_INERTS	C4 inerts analyser 2	%	%
C6_INERTS	C6 inerts analyser 1	%	%
C6_INERTS	C6 inerts analyser 2	%	%
BED_WEIGHT	R1 bed weight	t	kg

Table 5.2: Plant data used in the software

Some of the points logged were not required for use in the software. A list of the tags that were however used is given above in *table* 5.2. The logger was eventually amended to reflect only the above variables.

Original plant data were sent in a comma-delimited text format. Then data were manipulated in a spreadsheet, extracting only the 31 relevant columns of data listed in *table* 5.2 above. The off-line software requires that this data be stored in a text format for it to be read during a run of a program.

CHAPTER 6

Off-line Applications

MATLAB® is a program by MathWorks Inc. The code was first written in this language since it is easy to use and code is easy to read and understand. It also handles matrices extremely well (MATLAB is short for MATrix LABoratory) and these are present throughout the proposed control algorithm. Another feature of the package is that graph plotting and annotating is extremely simple. This was vital for both debugging and analysis of results.

6.1 The Site-based Model

Two researchers, Guillard and Mulholland had already done some programming work in this area in 1998. The code followed the paper by McAuley, MacGregor et al. (1990). The system modelled was a 2-site, 3-monomer system.

The Poly 2 plant has a 3-site Ziegler-Natta catalyst, a monomer (ethylene) and a comonomer (either 1-butene or 1-hexene – only one is used exclusively). The work by Gulllard and Mulholland was thus altered to match this. Furthermore, it was found that the kinetic parameters used by McAuley, MacGregor et al. (1990) above had since been updated in a later paper ((Shaw, McAuley et al. 1998). These were incorporated as well to update the code. The rate constants used can be found in *table* A.1. Note that the model used by Shaw, McAuley et al. (1998) only caters for a 2-site catalyst. Hence, as no further kinetic data were available, the same parameters as for site 2 were used for site 3.

Parameter	Value	Units
$MW_{ethylene}$	28	kg/kmol
MW_{butene}	56	kg/kmol
MW_{hexene}	84	kg/kmol
Vp_{max}	40	m^3
R	0.08314	bar.m^3/kmol.K
ρ	920	kg/m^3

Table 6.1: Fixed parameters used in the sited-based model simulation

Table 6.1 gives some parameters that were used in the simulation model. The normal operating temperature and pressure of the Poly 2 reactor have been withheld for confidentiality reasons. The volume of the reactor vessel itself was found among equipment specifications. Due to the fact that there is so much cycle gas in the recycle loop, the volume of the loop was added to this initial volume. An estimate of the lengths, heights and diameters of the lines was obtained by pacing out the loop on-site. The density of polymer was set at 920 kg/m^3.

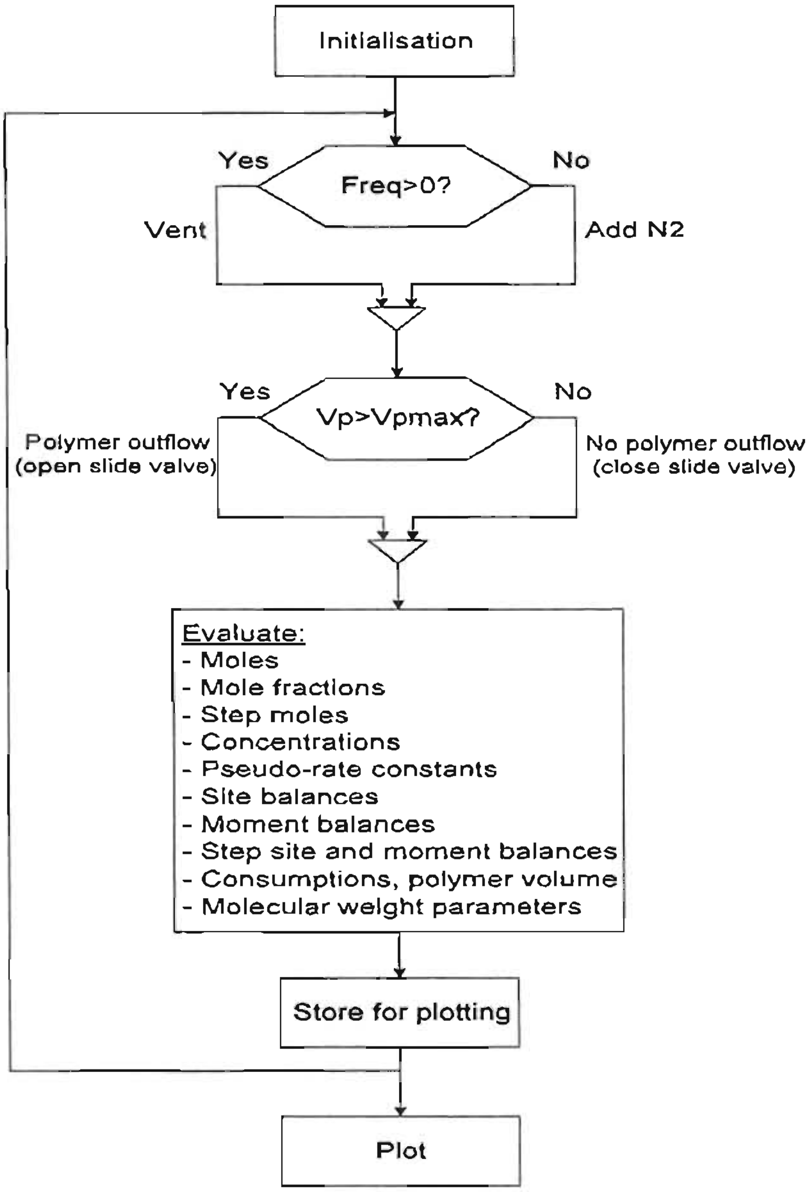


Figure 6.1: Organogram of the Site-based Model's Software

Figure 6.1 shows the basic flow of information in the site-based model. No historical plant data were used in the running of this software. Steady plant flows were used as inputs. After initialisation, 'Freq' (the flow required) was calculated using a pressure set point and the ideal gas law. If the flow was positive, the vent was opened and if negative, nitrogen was added. This was an attempt to simulate the split-range control mechanism: addition or venting. A 'Vpmax' was set and if the volume of polymer in the reactor exceeded this maximum, product was allowed out. Following these steps, many parameters and balances were updated.

6.2 The Simpler Model

Figure 6.2 below illustrates the logic used in the simpler software developed. It is analogous to the control diagram in *figure 5.4*. After initialisation, new readings are scanned from the logged historical plant data stored in text files. Data is logged at 20-second intervals from the plant. These new data are then compared with old data to test whether the GC has been updated. If it has (usually detected every 180 seconds), the identifier uses these new readings in order to update the kinetic parameters. The Kalman filter then runs, using these new parameters. Should there not be a GC update, the Kalman filter will still run, using the last set of kinetic data.

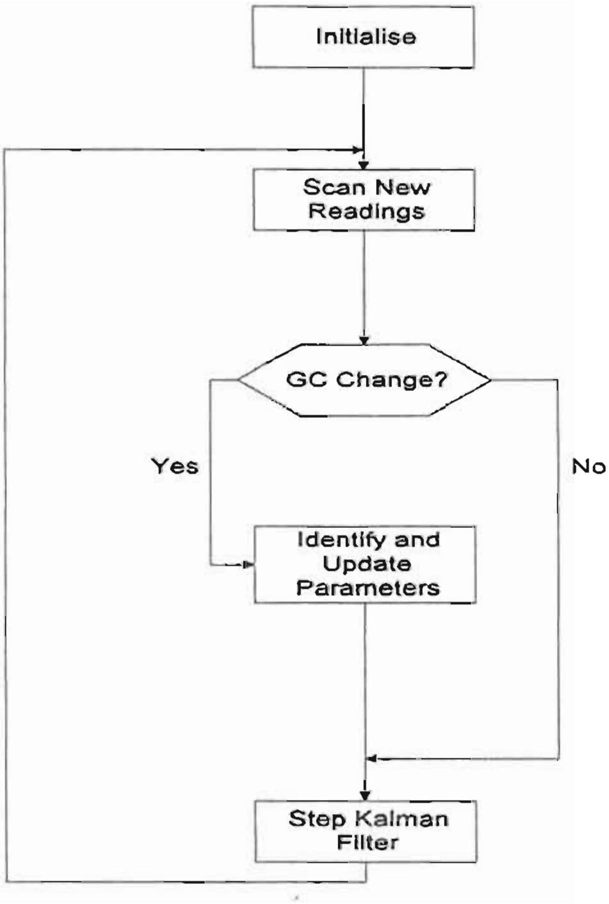


Figure 6.2: Organogram of the Simpler Model's Software

6.2.1 Project Stages

Below is a schematic that shows the various blocks of code that were written for the project. A block of code is denoted by the inner rectangles.

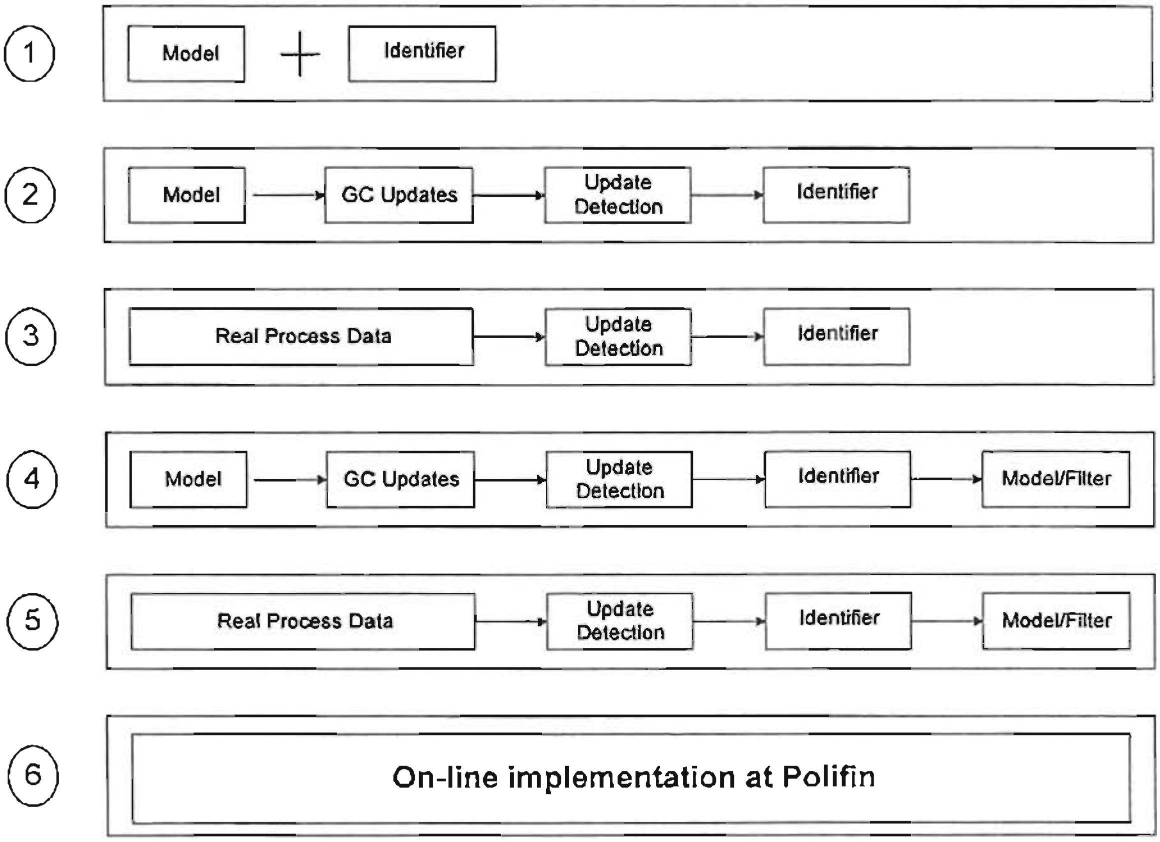


Figure 6.3: Diagram of the blocks of code used in building the on-line algorithm

Stage 2 simulates a GC update on the plant so that the measured output variables can be synchronously compared with the model predictions at the time of update. Thus the 'GC Updates' block and the 'Update Detection' block were built into the code in stage 1 to simulate this timing aspect.

Stage 3 is merely the same code as in stage 2 but with the Polifin process data being passed through it instead of fixed flows and arbitrary mole fractions.

The next step (stage 4) required the attachment of the state estimator (extended Kalman Filter). This uses the parameters estimated in the RLS algorithm above, and predicts the gas compositions. It also has the ability to predict additional properties, namely n_{P2}, n_{P4} and n_{P6}.

The fifth stage involved the passing of real process data through the model in order to test whether the algorithm was ready for on-line implementation.

In an on-line situation, fault detection would be simple: deviations from the identifier kinetics will be easily observed if for example an impurity causes n_{AS} to decrease. Faulty gaseous measurements will also be detected since there will be predicted trajectories corresponding to these measurements. Significant deviations from this trajectory will alert the operator to the possible source of a fault.

6.2.2 Why are all results scaled?

A secrecy agreement with POLIFIN Ltd. was signed at the beginning of the project. This was so that all plant operating data is protected from the public. This is also common practice by most polymerisation researchers. For this reason, all results were scaled from 0 to 100. Dividing a data point by the maximum of the range of data and converting the result to a percentage did this. The scaling may cause some confusion in certain areas but, understandably, the agreement must be honoured.

6.2.3 GC Update Detection

An array called 'buffer' was created, 'Nbuffer' rows, (100) long by 'Nm' columns, (20) wide. 'Nm' is the number of measurements obtained from the plant (see *table* 5.2).

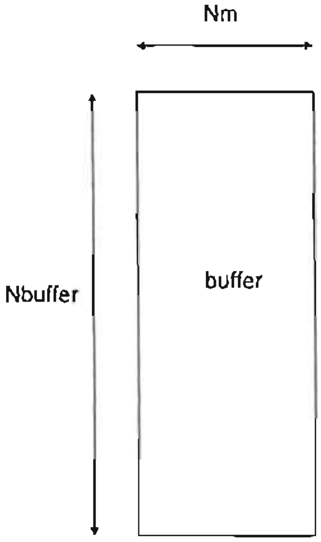


Figure 6.4: Illustration of the 'buffer' array, showing dimensions

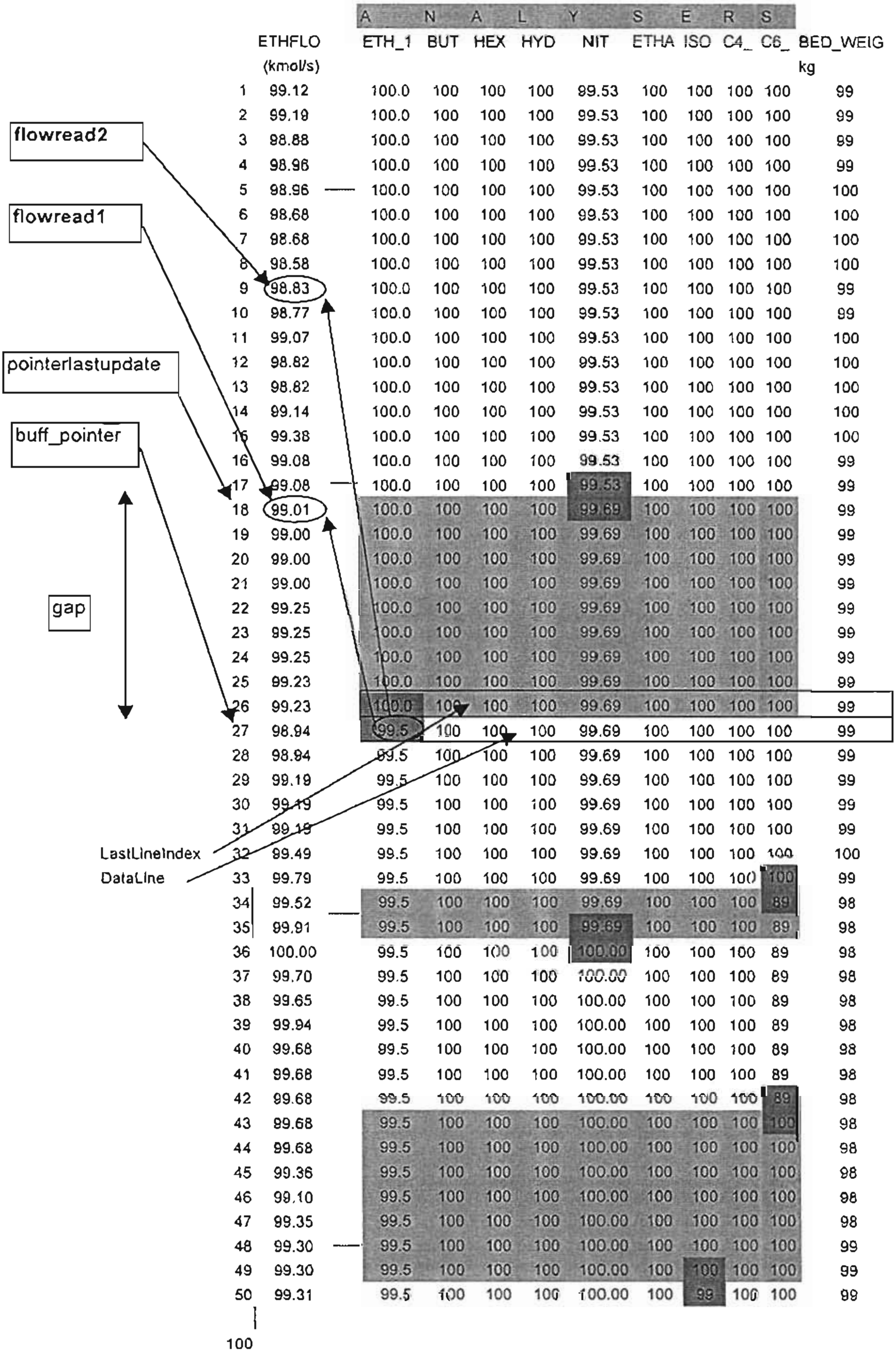
This array is loaded with historical plant readings on each time step. The first five entries are process inputs and are flows. The sixth entry is the total fresh catalyst feed rate and is used for the active site flow. The next two entries are vent flow and production rate, which are followed by operating temperature and pressure. The following nine entries are outputs from the GC analysers, used for update detection. The final column required is the bed weight, which is used in calculating the volume of polymer in the reactor.

A line of data is written into the buffer array with every time step (20s). Once the buffer array is sufficiently full, computation begins. When data have filled the entire array, the oldest data are overwritten (i.e. in position 1) with the current data.

For detecting updates, the current and previous rows, 'DataLine' and 'LastLineIndex' are compared. If any of the entries from the two rows differs, a GC update is signalled. A tolerance can be set on this though.

The gas phase compositions from the GC are only updated every 180 seconds at POLIFIN. Thus updated gas compositions are always as a result of the corresponding input flows 180 seconds ago. Therefore it is necessary to search back 180 seconds in the buffer array for the flows that correspond to the current GC compositions. The flow corresponding to the last GC update is also read and the two are combined to form an average. This is more clearly illustrated in the cyclic file extract and XL graph (*table* 6.2 and *figure* 6.5 respectively). In order to calculate the GC update interval, the difference between 'buff_pointer' and the last GC update, 'pointerlastupdate' is calculated. If this gap is less than 1, 'Nbuffer' is added to 'gap'. The interval, 'dtGC' is then this gap multiplied by the logging interval, 'dt'. The parameter 'pointerlastupdate' is made equal to 'buff_pointer' for use the next time an update is detected.

The interval between GC updates at Poly 2 is 180s. The updates detected by the software are generally between 140-180 seconds but it is not important to get the exact time of update, provided that the correct corresponding flow is used. When a GC update is not detected for 1000s, one is forced, simply to get an update on kinetic data.


flowread2
flowread1
pointerlastupdate
buff_pointer
gap
LastLineIndex
DataLine

		ANALYSERS									
	ETHFLO (kmol/s)	ETH_1	BUT	HEX	HYD	NIT	ETHA	ISO	C4_	C6_	BED_WEIG kg
1	99.12	100.0	100	100	100	99.53	100	100	100	100	99
2	99.19	100.0	100	100	100	99.53	100	100	100	100	99
3	98.88	100.0	100	100	100	99.53	100	100	100	100	99
4	98.96	100.0	100	100	100	99.53	100	100	100	100	99
5	98.96	100.0	100	100	100	99.53	100	100	100	100	100
6	98.68	100.0	100	100	100	99.53	100	100	100	100	100
7	98.68	100.0	100	100	100	99.53	100	100	100	100	100
8	98.58	100.0	100	100	100	99.53	100	100	100	100	100
9	98.83	100.0	100	100	100	99.53	100	100	100	100	99
10	98.77	100.0	100	100	100	99.53	100	100	100	100	99
11	99.07	100.0	100	100	100	99.53	100	100	100	100	100
12	98.82	100.0	100	100	100	99.53	100	100	100	100	100
13	98.82	100.0	100	100	100	99.53	100	100	100	100	100
14	99.14	100.0	100	100	100	99.53	100	100	100	100	100
15	99.38	100.0	100	100	100	99.53	100	100	100	100	100
16	99.08	100.0	100	100	100	99.53	100	100	100	100	99
17	99.08	100.0	100	100	100	99.53	100	100	100	100	99
18	99.01	100.0	100	100	100	99.69	100	100	100	100	99
19	99.00	100.0	100	100	100	99.69	100	100	100	100	99
20	99.00	100.0	100	100	100	99.69	100	100	100	100	99
21	99.00	100.0	100	100	100	99.69	100	100	100	100	99
22	99.25	100.0	100	100	100	99.69	100	100	100	100	99
23	99.25	100.0	100	100	100	99.69	100	100	100	100	99
24	99.25	100.0	100	100	100	99.69	100	100	100	100	99
25	99.23	100.0	100	100	100	99.69	100	100	100	100	99
26	99.23	100.0	100	100	100	99.69	100	100	100	100	99
27	98.94	99.5	100	100	100	99.69	100	100	100	100	99
28	98.94	99.5	100	100	100	99.69	100	100	100	100	99
29	99.19	99.5	100	100	100	99.69	100	100	100	100	99
30	99.19	99.5	100	100	100	99.69	100	100	100	100	99
31	99.19	99.5	100	100	100	99.69	100	100	100	100	99
32	99.49	99.5	100	100	100	99.69	100	100	100	100	100
33	99.79	99.5	100	100	100	99.69	100	100	100	100	99
34	99.52	99.5	100	100	100	99.69	100	100	100	89	98
35	99.91	99.5	100	100	100	99.69	100	100	100	89	98
36	100.00	99.5	100	100	100	100.00	100	100	100	89	98
37	99.70	99.5	100	100	100	100.00	100	100	100	89	98
38	99.65	99.5	100	100	100	100.00	100	100	100	89	98
39	99.94	99.5	100	100	100	100.00	100	100	100	89	98
40	99.68	99.5	100	100	100	100.00	100	100	100	89	98
41	99.68	99.5	100	100	100	100.00	100	100	100	89	98
42	99.68	99.5	100	100	100	100.00	100	100	100	89	98
43	99.68	99.5	100	100	100	100.00	100	100	100	100	98
44	99.68	99.5	100	100	100	100.00	100	100	100	100	98
45	99.36	99.5	100	100	100	100.00	100	100	100	100	98
46	99.10	99.5	100	100	100	100.00	100	100	100	100	98
47	99.35	99.5	100	100	100	100.00	100	100	100	100	98
48	99.30	99.5	100	100	100	100.00	100	100	100	100	99
49	99.30	99.5	100	100	100	100.00	100	100	100	100	99
50	99.31	99.5	100	100	100	100.00	100	99	100	100	99
100											

Table 6.2: An example of the buffer and its role in update detection

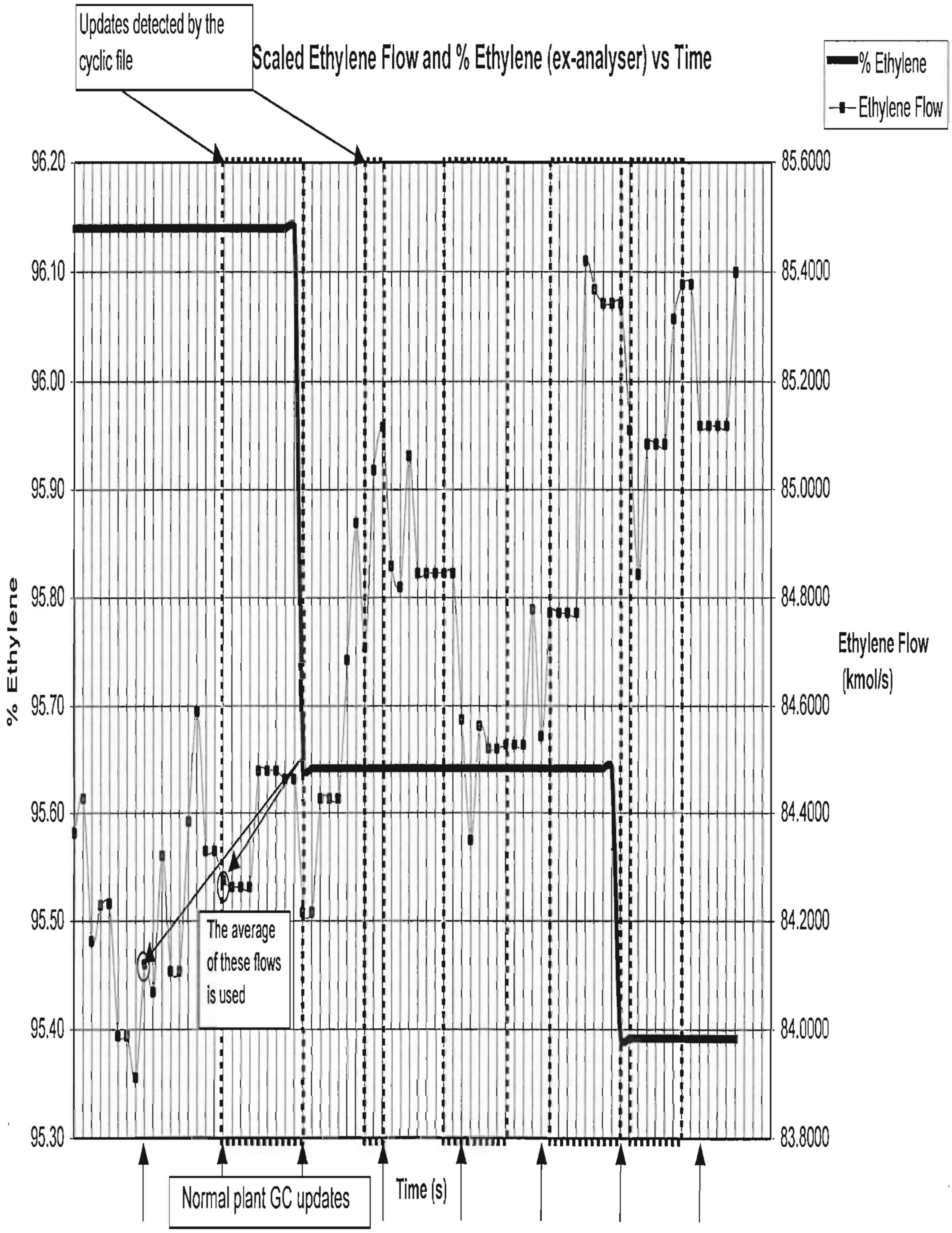


Figure 6.5: Illustration of the buffer function

6.2.4 Results

The code was tested off-line on numerous historical data sets. These included six different industrial grades, both butene and hexene-based, and a wide range of MI and ρ. Trial-and-error tuning (*section* 6.2.5) of the model error covariance matrix, Q was carried out to achieve a standard set of values that was used for all grades.

All results presented in this section have been given a code. These differ from the codes used on the plant and are:

A: A hexene film grade, low density, low MI
B: A hexene film grade, low density, lower MI
C: A hexene rota-moulding grade, high density, medium MI
D: A butene grade, medium density, high MI

The keywords low, medium and high are simply comparisons made between the grades and do not relate to those mentioned in *section* 1.3.1 in any way.

6.2.4.1 RLS Identifier

The RLS algorithm has been given earlier in the previous chapter and is laid out using equations (5.26) to (5.28). The first equation evaluates an optimal gain, K for use in determining the new parameter vector, $\overline{p}_i$ (equation (5.27)). This vector contains the kinetic parameters to be identified, k_{P2}, k_{P4}, k_{P6}, k_H, g_H, k_d and n_{AS} and is evaluated by using measured input flows and output compositions. Equation (5.28) evaluates a new M matrix for use in calculating another updated K matrix.

A real strength of the algorithm lies in the RLS parameter estimation. Published values of the pseudo-kinetic rate constants are of limited use as they are specific to catalyst and conditions. *Figure* 6.6 illustrates the ability of the identifier to reach a relatively constant value when no initial estimates are available.

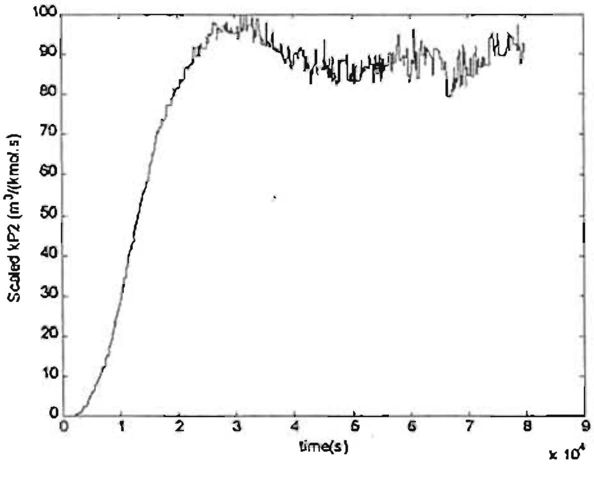


Figure 6.6: k_{P2} as estimated by the RLS identification

Once approximate values for each grade had been obtained, better initial estimates were included. The identifier was tested on various grades and graphs 6.7 through 6.10 show the results. Grades A, B and C are hexene grades whereas grade D is a butene grade. In all the cases it was demonstrated that it is possible to obtain a value for pseudo-kinetic rate constants.

Grade A:

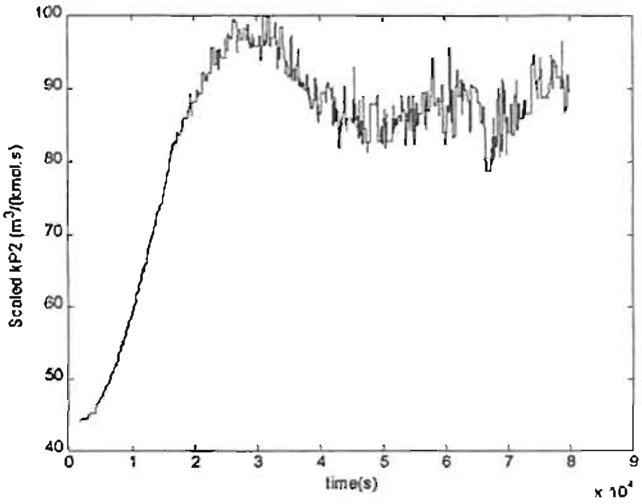


Figure 6.7: k_{P2} for 'Grade A' as estimated by the RLS identification

The above results for grade A are perhaps not conclusive. The graph does not remain as constant as in the case of grades B, C and D. This probably relates to the fact that the inputs (flows) or other plant variables were frequently altered.

Grade B:

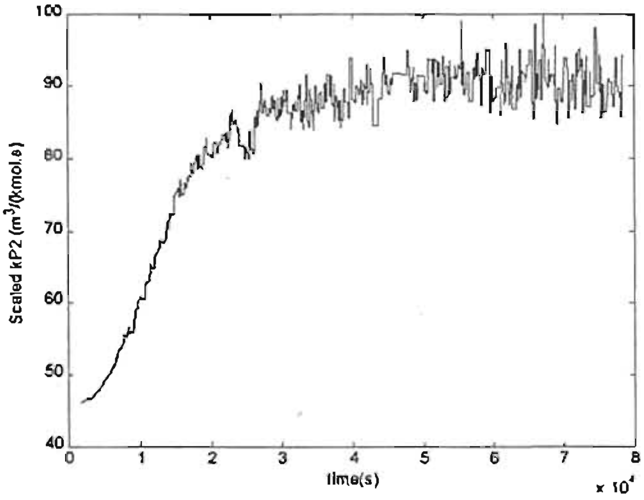


Figure 6.8: k_{P2} for 'Grade B' as estimated by the RLS identification

Grade C:

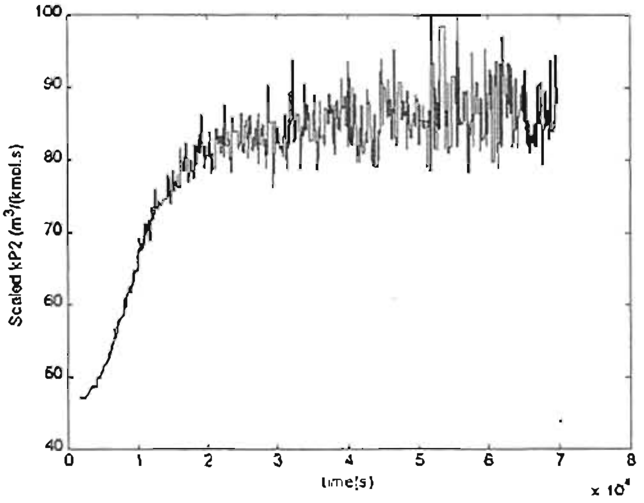


Figure 6.9: k_{P2} for 'Grade C' as estimated by the RLS identification

Grade D:

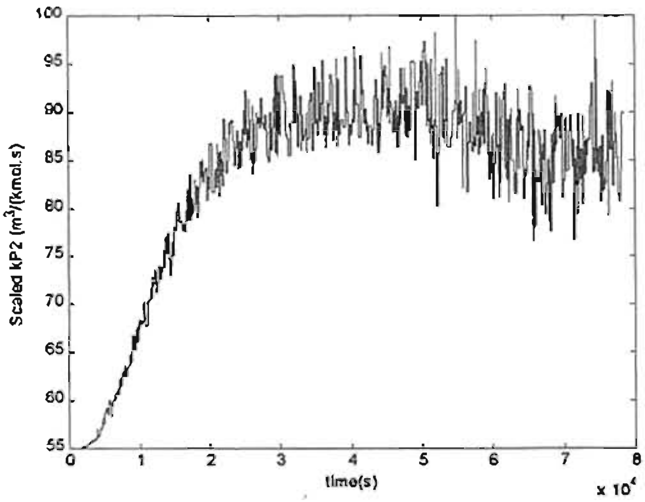


Figure 6.10: k_{P2} for 'Grade D' as estimated by the RLS identification

What should also be noted from the above plots is that different grades exhibit different rate constants. This is due to the different hydrogen and comonomer ratios in the system. The incorporation of more or less hydrogen and/or comonomer in the polymer will affect the rate at which the chain propagates.

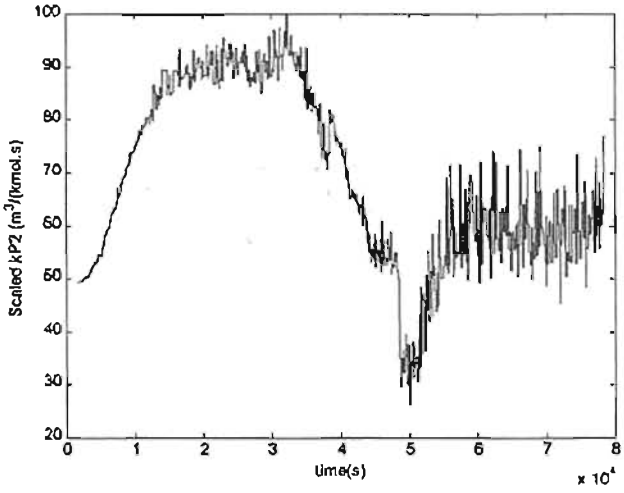
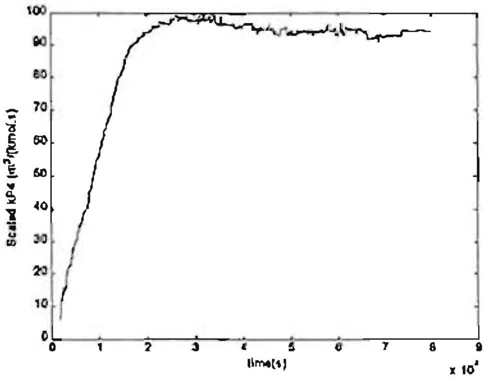


Figure 6.11: k_{P2} as estimated by the RLS identification, for a grade change from C to D

Figure 6.11 demonstrates the identifier's ability to easily find a new k_{P2} after a hexene-butene comonomer change.



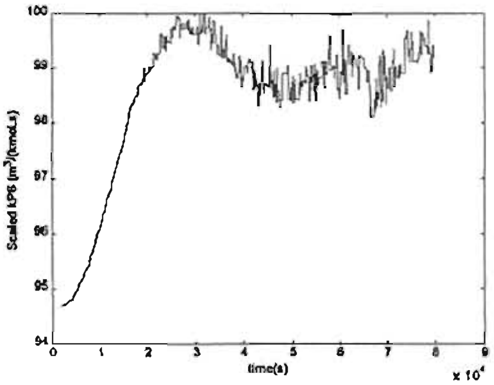
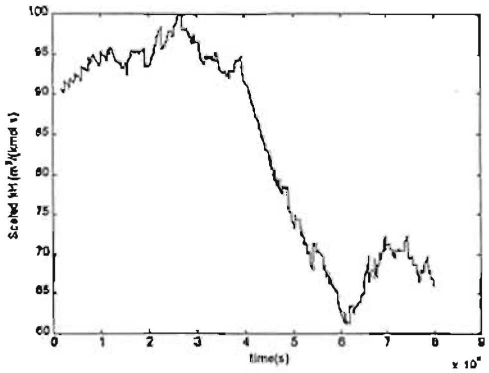


Figure 6.12: k_{P4} as estimated by the RLS identifier

Figure 6.13: k_{P6} as estimated by the RLS Identifier



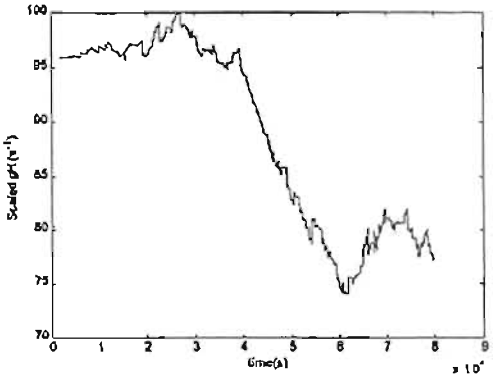


Figure 6.14: k_H as estimated by the RLS identifier

Figure 6.15: g_H as estimated by the RLS identifier

Although the above graphs are for a hexene grade (A), a very small rate constant for butene (k_{P4}) is estimated in *figure* 6.12. This is due to the fact that a small, step-like analyser reading for butene is logged when running hexene grades (*figure* 6.17). The same phenomenon does not occur when butene grades are run as the hexene analyser reading is always zero.

It can be seen that the shapes of the graphs for k_{P2} (*figure* 6.7) and k_{P6} (*figure* 6.13) are the same, but of different orders of magnitude. The same can be noticed for k_H (*figure* 6.14) and g_H (*figure* 6.15). This can be readily explained. Referring to *section* 5.2, one notices that k_{P2} and k_{P4} depend on n_{AS} and N_p. When n_{AS} was held constant over the run time, there seemed to be no effect. However, setting N_p constant, the graphs both moved with the same shape. It can be concluded that these parameters are strongly correlated with n_{AS}.

Both the parameters k_H and g_H depend on n_{AS}, n_{H2} and N_g. The graphs are also of similar shape but are not exact, as above, since g_H has no dependence on n_{AS}.

The rate constant for deactivation, k_d remains zero, which is of some concern as it is well known that deactivation of active sites occurs. A possible explanation is that there are simply not enough active sites available to cause death of the sites as well.

6.2.4.2 Kalman Filter

The results given below stem from the Kalman filter algorithm laid out in equations (5.29) to (5.31) in the previous chapter.

A full set of graphical results from the Kalman filter for grade A is given below.

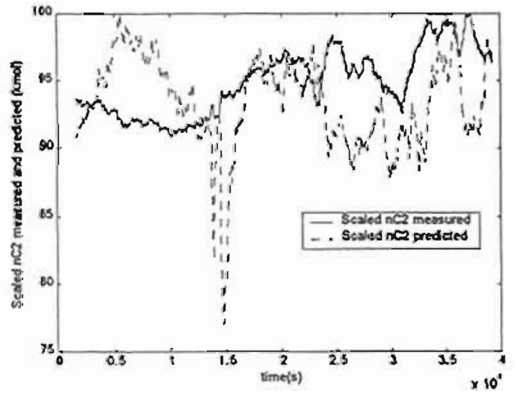


Figure 6.16: n_{C2} as estimated by the Kalman filter

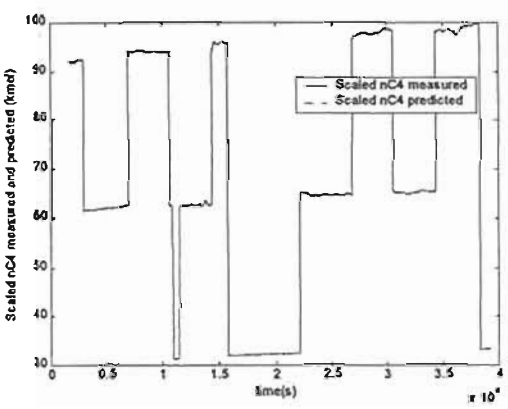


Figure 6.17: n_{C4} as estimated by the Kalman filter

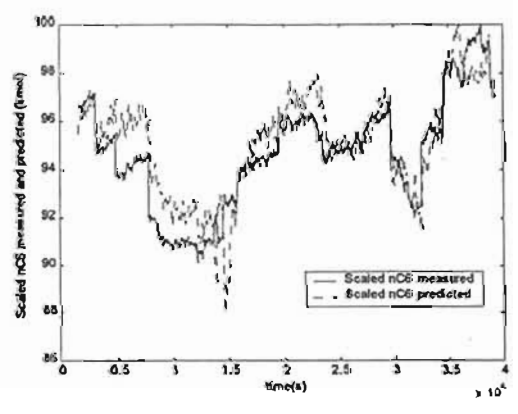


Figure 6.18: n_{C6} as estimated by the Kalman filter

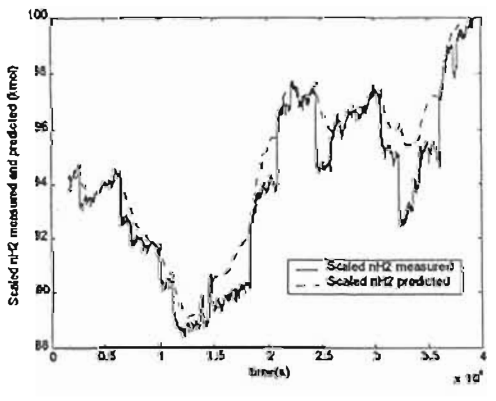


Figure 6.19: n_{H2} as estimated by the Kalman filter

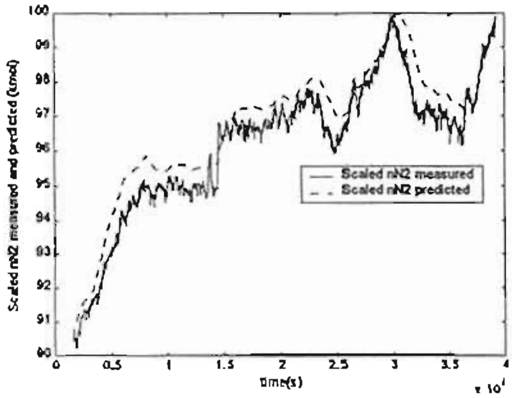


Figure 6.20: n_{N2} as estimated by the Kalman filter

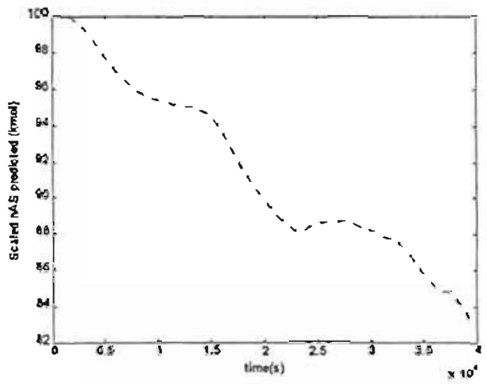


Figure 6.21: n_{AS} as estimated by the Kalman filter

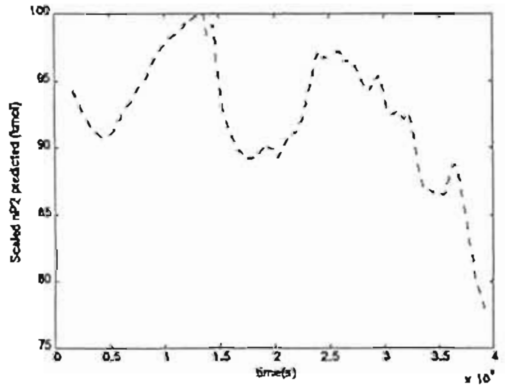


Figure 6.22: n_{P2} as estimated by the Kalman filter

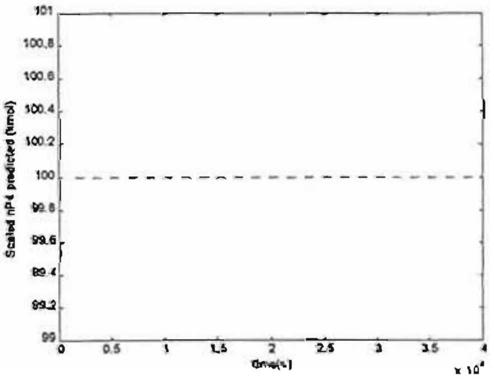


Figure 6.23: n_{P4} as estimated by the Kalman filter

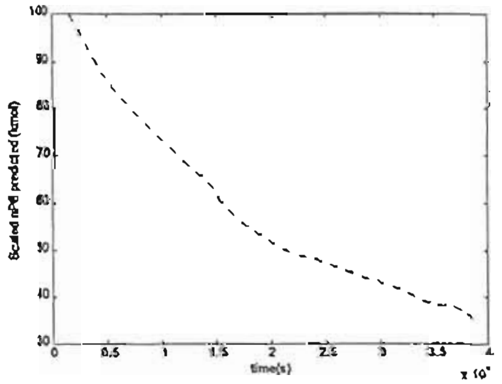


Figure 6.24: n_{P6} as estimated by the Kalman filter

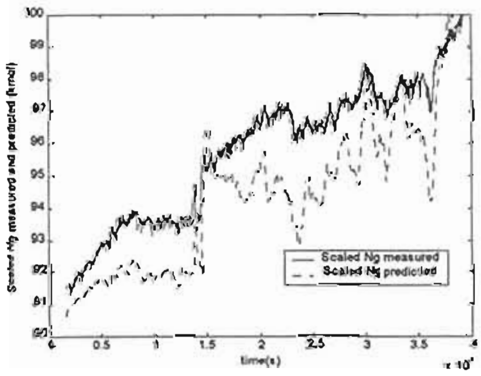


Figure 6.25: N_g as estimated by the Kalman filter

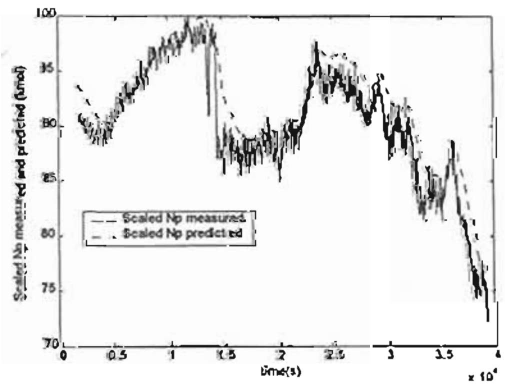


Figure 6.26: N_p as estimated by the Kalman filter

The above results are varied. One notices for *figure* 6.16 and *figure* 6.25 that the prediction trend is very noisy as opposed to filtered. The predictions are also somewhat different in shape to the measured trends. This arises from the input flows $\overline{F_i}$ that are frequently changed on the plant. The smoother, more sustained responses (*figures* 6.19, 6.20 and 6.26) arise from the fact that predicted outputs $\overline{n_i}$ are fed back in order to predict future states $\overline{n}_{i+1}$.

6.2.4.3 Spikes and NaNQ's

Suspect raw plant data may impact on the quality of the results. This commonly affected logged readings for bed weight and production rate and the way in which they were affected is described below.

Bed weight is calculated via a differential pressure over the reactor. Typical bed weights are of the order of 31 tons but both negative and uncharacteristically high or low readings are often recorded (observed in almost one in every two sets of historical data). The effect of this on n_{H2} can be seen in *figure* 6.27. Large spikes in the measured data are present. However, the prediction trend does not follow the measurement closely in this case. It was found that this was also the case for n_{N2} and N_g. However, slight disturbances (spikes) in measured n_{C2}, n_{C4}, n_{C6} and N_p resulted in the predictions following the measured values. This phenomenon is not only as a result of the tuning but also as a result of the components attempting to simultaneously satisfy the mass balance in the system.

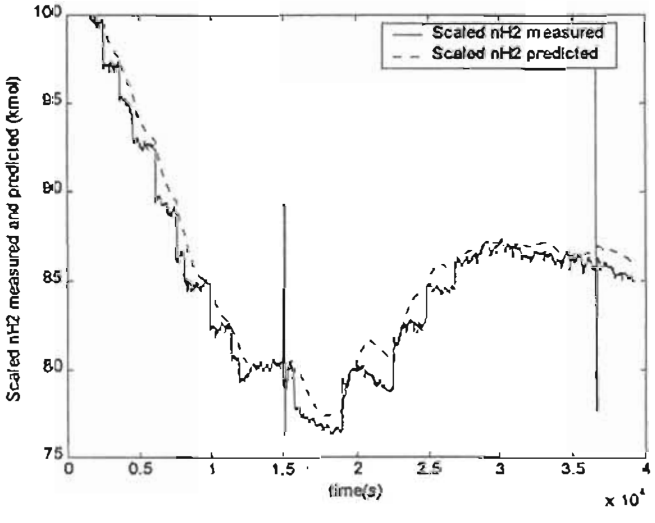


Figure 6.27: n_{H2} showing spikes as a result of faulty plant data

Production rate is calculated by equation (4.1). It is common that a code 'NaNQ' (Not a Normal Quantity) is logged for this parameter. A division by zero in the formula is normally the cause. The result during off-line runs is that data simply cannot be read in and the program aborts. During on-line testing, the effect was that matrices would not invert due to singularity problems. To remedy this, raw data protection criteria were included in the software.

6.2.4.4 Noisy Results

Figures 6.28 and 6.29 for grade D below appear very noisy compared to earlier figures presented (6.19 and 6.25 respectively). A possible reason for this is that an M-resin is being produced. For M-resins, temperature as a control variable has very little effect; thus the gas ratios are controlled. n_{C2} (not shown here) has the biggest influence on the total gaseous moles, N_g. This (and comonomer and hydrogen to a lesser extent) gives rise to the noisy effect below. The effect of the comonomer changeover is illustrated in the *figure* 6.30

following. It combines grades C and D and the effect of controlling the gas ratios for grade D is clearly illustrated after 80000 seconds.

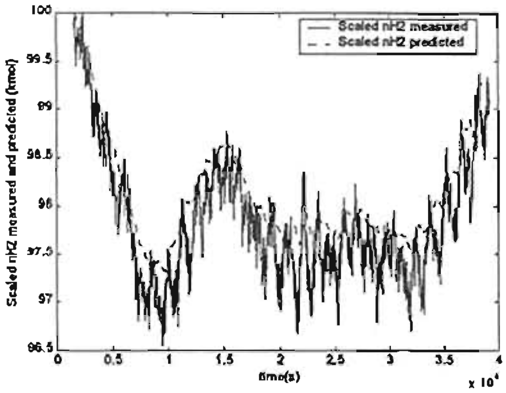


Figure 6.28: Noisy n_{H2} from Grade D

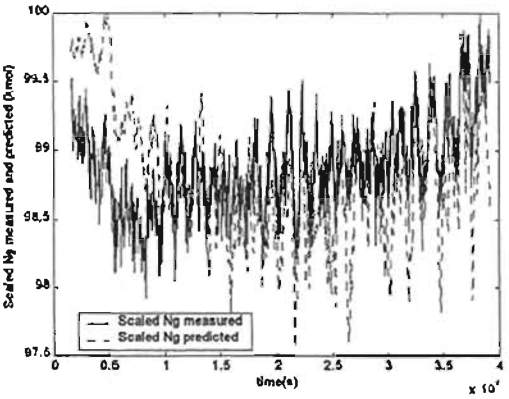


Figure 6.29: Noisy N_g from Grade D

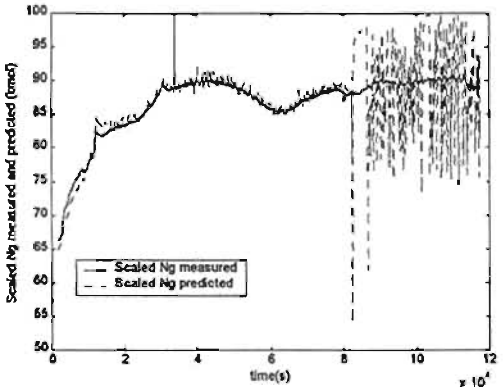


Figure 6.30: N_g from Grades C and D

6.2.4.5 Comonomer changeover

The unmeasurable states for the C to D grade change cope well with the comonomer change, as illustrated in *figures* 6.31 and 6.32 below. Soon after 80000 seconds, the moles of ethylene in the polymer are reduced by 10. This is caused by a marked increase in the butene incorporation in the polymer chain.

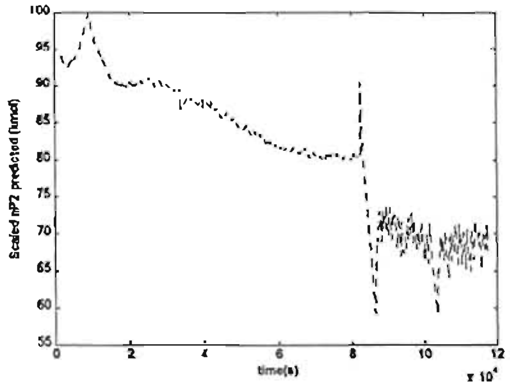


Figure 6.31: n_{P2} from Grades C and D

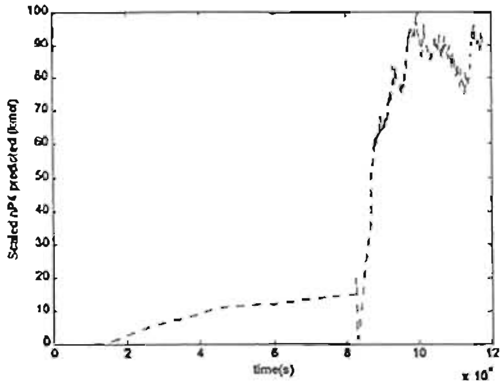


Figure 6.32: n_{P4} from Grades C and D

6.2.4.6 Response time

The lengthy response time present in the system is shown below. *Figure* 6.33 shows that after 20000 seconds the hexene gaseous inventory was zero. The result of this is no further incorporation of hexene in the polymer chain. This is only registered some 40000 seconds (11 hours) later when n_{P6} becomes zero in *figure* 6.34. This is in agreement with the earlier discussion in *section* 4.1.7.

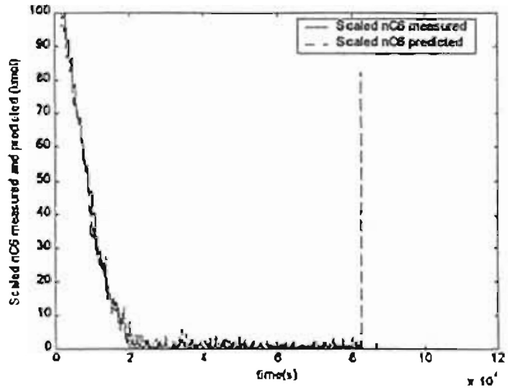


Figure 6.33: n_{C6} from Grades C and D

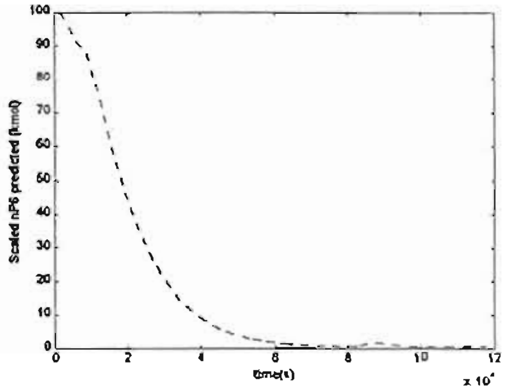


Figure 6.34: n_{P6} from Grades C and D

6.2.4.7 Fault detection possibilities

Figure 6.35 below shows moles of hydrogen as predicted by the Kalman filter. An offset error was inserted into the measurement data, and this is revealed in *Figure* 6.36. Correct tuning of the EKF will make it responsive to changes in operating conditions, yet also to follow such faults slowly enough to reveal a sustained error for fault detection.

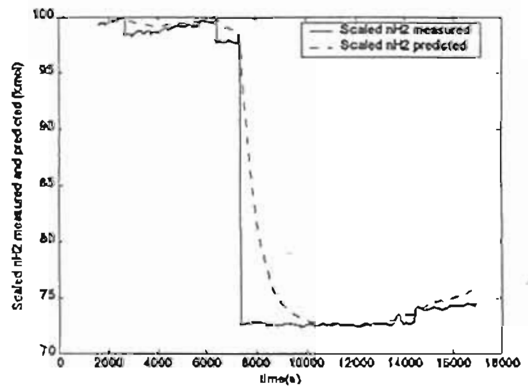


Figure 6.35: EKF n_{H2} showing a possible fault

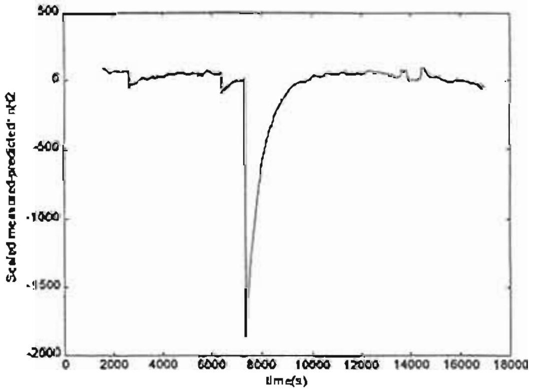


Figure 6.36: EKF measured-predicted error for n_{C2}

6.2.5 Kalman Filter Tuning

This is a very important stage in the implementation of any Kalman filter. It is also by no means simple, especially in the case of polymerisation, where complex interrelationships between control variables occur.

The matrix R represents the measurement error covariance matrix. It contains the error covariances for each of the corresponding measurable variables. Q is the model error covariance matrix. Similarly, it holds the expected error covariances corresponding to the predicted states. A high Q relative to R is equivalent to a high error in the model prediction (and good confidence in measurements) and the result is that the predictions follow the measurements very closely. On the other hand a high R relative to Q represents a large error in observations and the consequence is a smoother prediction line relative to the noisy measured output.

These matrices are usually treated as design parameters rather than measurable constants. They are changed in order to achieve better state estimates in relation to the measured states (Wilson, Agarwal et al. 1998). This causes one to be suspicious of the predicted states being generated from the filter. Crowley and Choi (1998) were able to determine values for the measured error covariances (R) for sensors from repetitive experimental measurements. For thermocouple measurements and conversion measurements, the values were estimated from experiments. The values for Q were then used as tuning parameters as these parameters are not easily quantifiable.

Due to the complexity of the system the tuning of the off-line filter was achieved by trial-and-error. Initially all elements in both Q and R were set to unity. All of the prediction trends followed the measurements very closely except for slight offsets in n_{C2} and n_{C6}. Increasing all elements in Q to 1000 caused the n_{C2} offset to be eliminated but had no effect on n_{C6}. Any further increase in Q_{33} (corresponds to n_{C6}) did not eliminate this offset.

The converse, that is Q values were left at unity and R values were set at 1000, was also tested. The result was much smoother predictions in general. Good state estimations were observed for n_{H2} and n_{N2}. However, large errors (measured-predicted) were observed for n_{C2} (8%) and n_{C6} (26%). Therefore R_{11} (the error covariance matrix element corresponding to n_{C2}), R_{22} (n_{C4}) and R_{33} (n_{C6}) were set back to 1 in order to track these measurements more closely. The predictions were brought closer to the measurements, but the important unmeasurable states, n_{P2}, n_{P4} and n_{P6}, predicted by the filter were adversely affected. These curves all seemed to tend towards zero as opposed to the more likely result shown in *figure* 6.22.

Model error covariance	Corresponding variable	Value	Measurement error covariance	Corresponding variable	Value
Q_{11}	n_{C2}	1E-02	R_{11}	n_{C2}	1
Q_{22}	n_{C4}	1E-01	R_{22}	n_{C4}	1
Q_{33}	n_{C6}	1E-01	R_{33}	n_{C6}	1
Q_{44}	n_{H2}	1E-03	R_{44}	n_{H2}	1
Q_{55}	n_{N2}	1E-03	R_{55}	n_{N2}	1
Q_{66}	n_{AS}	1E-03	R_{66}	Ng	1
Q_{77}	n_{P2}	1E-06	R_{77}	Np	1
Q_{88}	n_{P4}	1E-06			
Q_{99}	n_{P6}	1E-06			
$Q_{10/10}$	Ng	1E-02			
$Q_{11/11}$	Np	1E-06			

Table 6.3: Final off-line tuning values used in error covariance matrices

The final tuning values used are above in *table* 6.3. It was noticed that the values in each of the matrices complement each other. For example if R_{66} (N_g) has a value of 0.01 and $Q_{10/10}$ (also N_g) has a value of 1×10^{-3}, the R_{68} may be equivalently increased by 100 to 1 and the $Q_{10/10}$ may be reduced by 100 to 1×10^{-1}. For this reason, and the fact that measurement device errors are difficult to quantify, all R elements were left at unity and Q was used as a tuning parameter.

The tunings listed appear to give the best and smoothest possible predictions for n_{C2}, n_{C4}, n_{C6} and N_g. A passive offset between measured and predicted may be present. This is caused by an apparent imbalance between gas supply and consumption. Although the results are still satisfactory, this offset is difficult to reduce without compromising on other predictions. Improvements can be achieved by increasing R_{77}, but this causes some of the detail in unmeasurable states (polymer fractions) to be lost.

6.2.6 Other Important Points

6.2.6.1 Problems with Matrix Inversions

There was a problem with the discretisation of B during the testing of the software. This was due to the fact that the matrix A would not invert due to its singular nature. It was suggested that the expansion of e^{Adt} be used:

$$e^{At} = I + \frac{1}{1!}(At) + \frac{1}{2!}(At)^2 + \frac{1}{3!}(At)^3 + \dots\dots\dots\dots\dots\dots$$

$$[e^{At} - I]A^{-1} = \frac{t}{1!} + \left(\frac{t}{2!}\right)(At) + \left(\frac{t}{3!}\right)(At)^2 + \dots\dots\dots\dots\dots$$

This can be evaluated to any given tolerance.

6.2.6.2 Which Comonomer is being Used?

POLIFIN produces either a butene or a hexene grade at any given time. As a result only one comonomer is used exclusively for any particular recipe, whilst the flow of the other comonomer is kept zero. However, when examining logged flows of butene and hexene, one finds that both flows are identical because the same flow meter / control valve is shared. One of the flows must be selected to zero for use in the software. Thus one requires some knowledge of which resin recipe is in use. The first letter of the grade code stamp normally tells the user this at a glance. This method was employed initially until it was discovered that there is often a substantial lag before the new grade code appears. This is demonstrated below in *figure* 6.37. The graph shows the analyser readings during a comonomer change from hexene to butene. One notices that after 2500 seconds there is more butene present than hexene. However, the grade code stamp is only changed some 57500 seconds (16 hours) later (where labelled). Therefore the criterion in the code was changed. A check on analyser compositions is now performed – the greater of the two comonomer compositions determines the correct comonomer used.

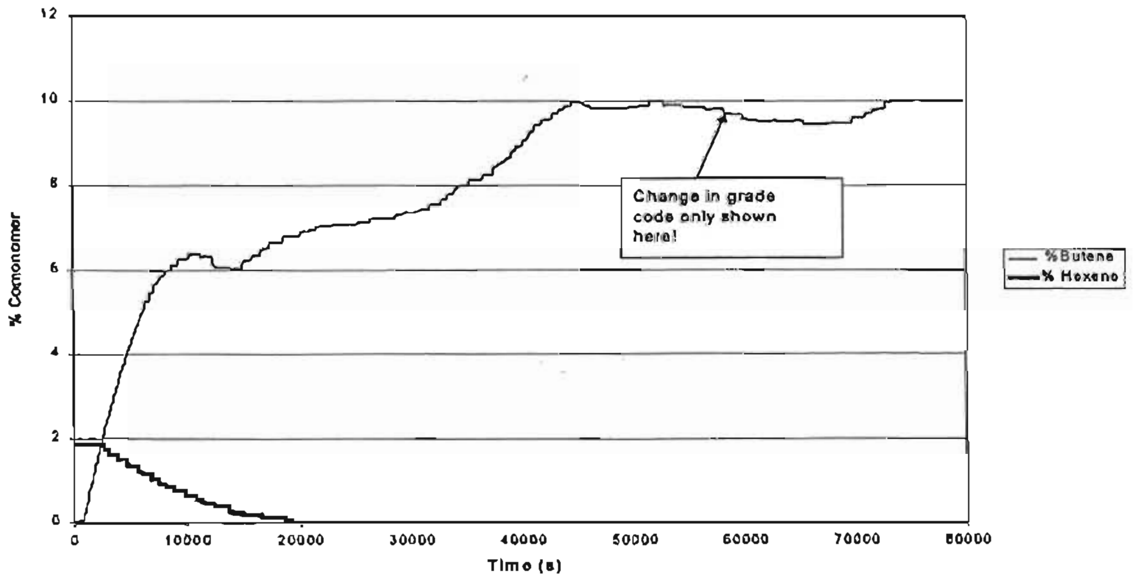


Figure 6.37: Graph showing analyser results for comonomer during a hexene-butene changeover

6.3 Conversion to C Code

Before travelling to Poly 2 to begin commissioning of the code, it was necessary to convert all of the MATLAB code to C code. The code itself is similar. Among other differences, in C the structures of a loop is slightly different and all variables used require a declaration. The main hurdle was to write routines to perform matrix arithmetic. Some of these are readily available from the Internet and the newer ones make use of overloaded operators. However, it was decided to construct new routines using ANSI C for use at Poly 2. Prof M. Mulholland wrote the routines for the program based on his knowledge from previous routines programmed in MODULA 2 for control of a multi-component distillation column.

6.3.1 In Durban: Using Visual C++

Visual C++ 4.0 using Microsoft Developer Studio was used to create the first rough version of C code. *Figures* 6.38 and 6.39 illustrate results from the identifier for k_{P2} from MATLAB and C++ respectively. The basic shape is the same but the two were matched exactly during the conversion to ANSI C during the on-line implementation (chapter 7). *Figures* 6.40 and 6.41 are n_{N2} as predicted by the Kalman filter. The shapes are very similar in this case.

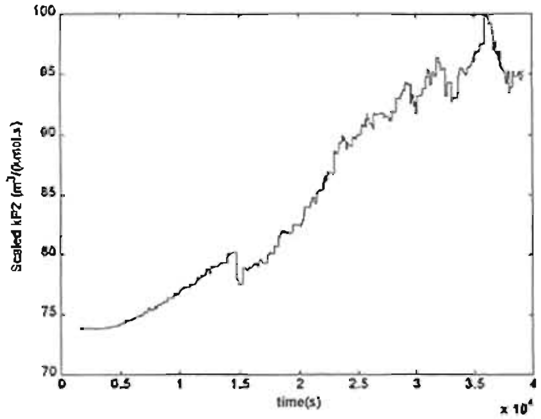


Figure 6.38: MATLAB version of k_{P2}

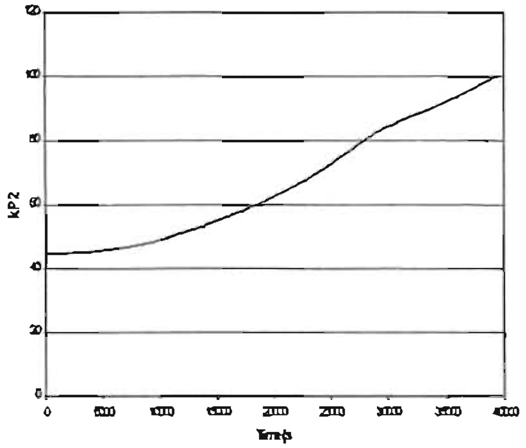


Figure 6.39: C++ version of k_{P2}

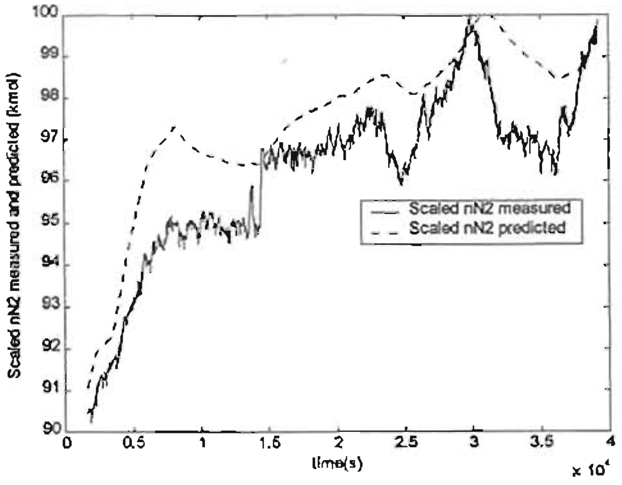


Figure 6.40: MATLAB version of n_{N2} measured and predicted

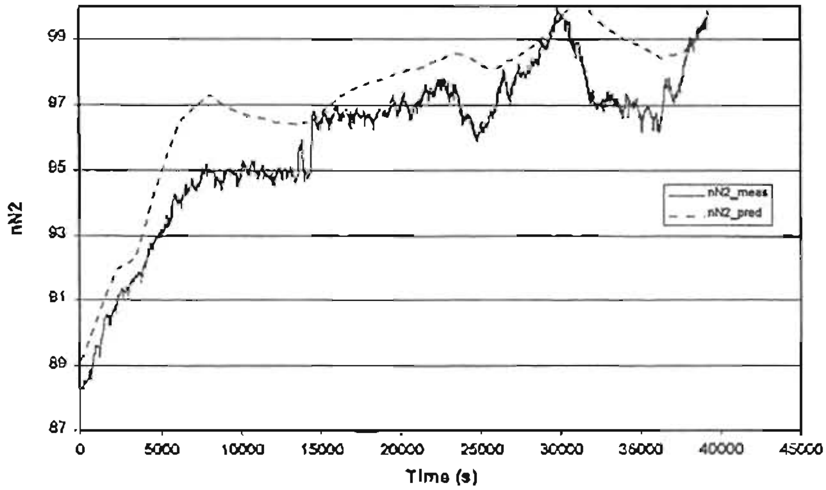


Figure 6.41: C++ version of n_{N2} measured and predicted

6.3.2 In Sasolburg: Using ANSI C

ANSI C is a version of C standardised in 1989 in the United States, through the American National Standards Institute (ANSI), and around the world through the International Standards Organisation (ISO). Full details of this conversion can be found in chapter 7.

CHAPTER 7

Real-time Application at Poly 2

7.1 Introduction

When writing the software using MSVC++, mentioned in *section* 6.3.1, care was taken so as to use ANSI C. This would minimise the effort during on-line commissioning.

The conversion to ANSI C and on-line implementation was a lengthy procedure. The code was reconstructed and re-written using object-oriented code. The structure is outlined in *section* 7.2 below. For debugging purposes, the MATLAB code was run alongside the C version. Results were compared at various key stages of the algorithm (initialisation, identifier, Kalman filter, and matrices) until the results matched.

7.2 The Program Structure

The ANSI C code is stored in five different C files along with four corresponding header files. They are listed below with a general description of the function of each module.

DbInterface.c: Software written so that one can read from and write to the database easily.

DbInterface.h: Header file for the above (RTAP database access functions and APACS access functions).

KalmanCalc.c: Main program source code.

KalmanCalc.h: Header file for controller program

KalmanControl.c: Links with RTAP database. It also sets up the timekeeper (an RTAP function) to read every 20s. Sends a message to a message handler and it is tested to see if it is a timekeeper or an event driven message. The message will execute functions if it is a timekeeper message.

KalmanControl.h: Contains general structure and function definitions that are needed by all other modules.

PointsDef.c: RTAP database points are defined in this module. Each point to be read from the database has a structure with corresponding attributes. One of these attributes is an identity number, which is all the main program needs to link with the database.

nrutil.c: Contains downloaded numerical recipe functions for vector and matrix manipulation (Trevelyan 1996).

nrutil.h: Header file for the above.

7.3 Results

Examples of on-line results from Poly 2 are shown in *figures* 7.1 to 7.6 below.

7.3.1 RLS Identifier

Figure 7.1 below again illustrates the strength of the identifier. Upon startup of the algorithm with a new butene grade, the pseudo-kinetic rate constant k_{P4} fast approaches a constant (scaled) value of 96.

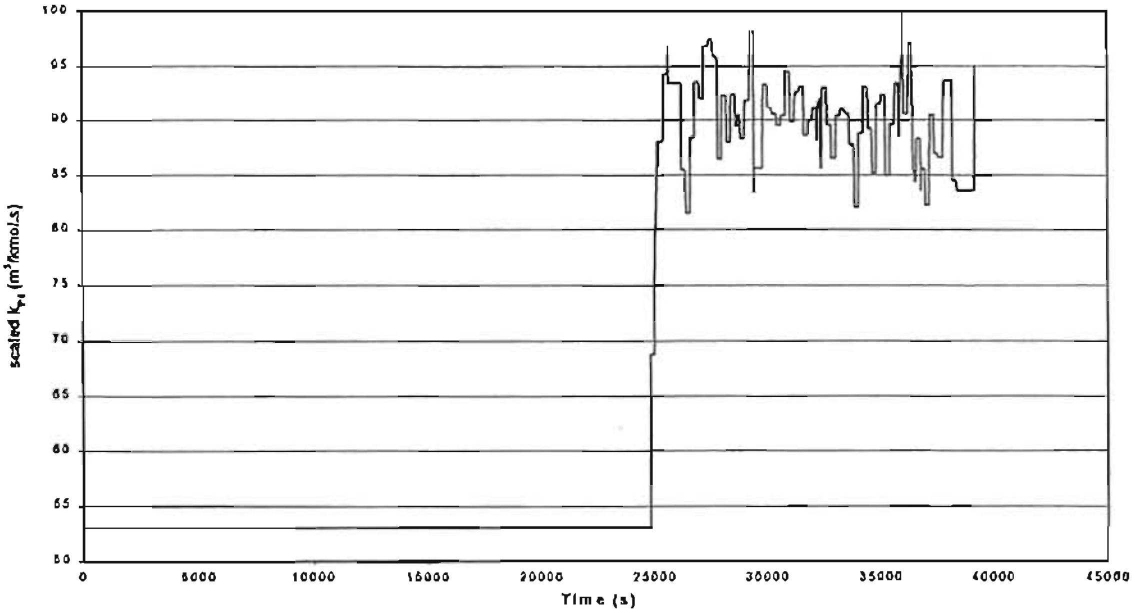


Figure 7.1: On-line results on algorithm start-up for a butene grade for k_{P4}

7.3.2 Ethylene vs Hydrogen Results

On comparison of *figure* 7.2 and 7.3, taken from the same on-line data set as *figure* 7.1 above, one finds that the hydrogen prediction (*figure* 7.3) is much slower than the ethylene (*figure* 7.4). The ethylene's response to measurement changes is quite rapid in comparison with the hydrogen, which takes almost 5000 seconds (83 minutes) to respond, and displays a large overshoot. This is in agreement with the findings of McAuley and MacGregor (1992) who also encountered slow hydrogen dynamics during large transitions in MI.

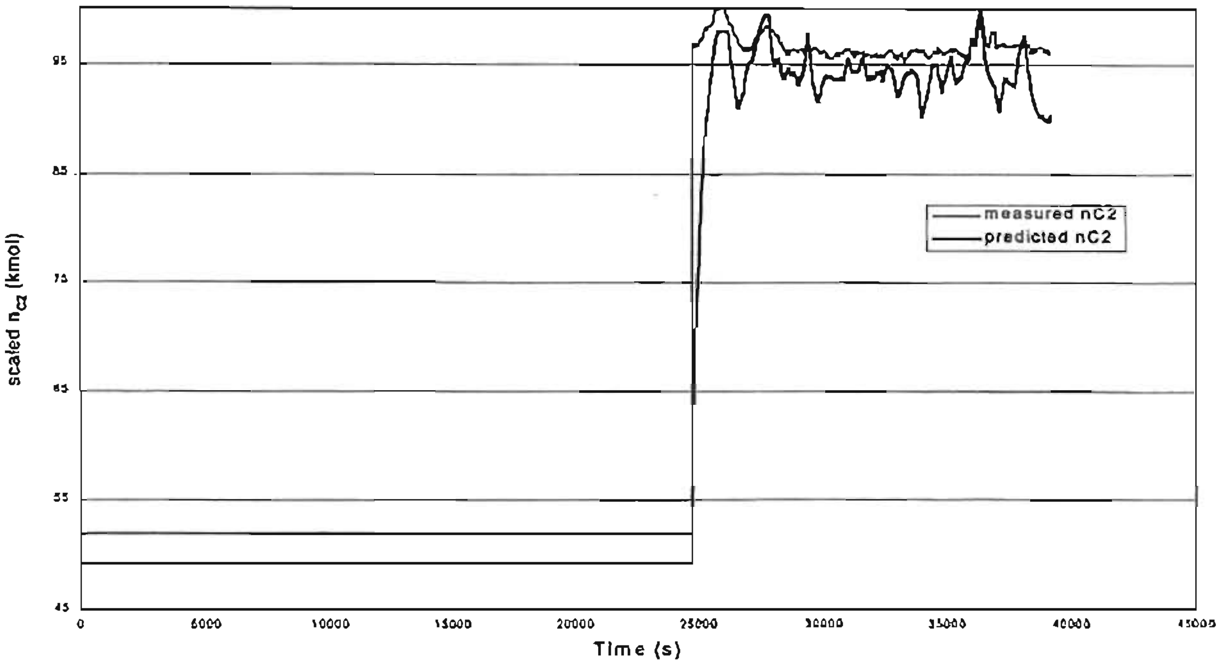


Figure 7.2: On-line results on algorithm start-up for a butene grade for n_{C2}

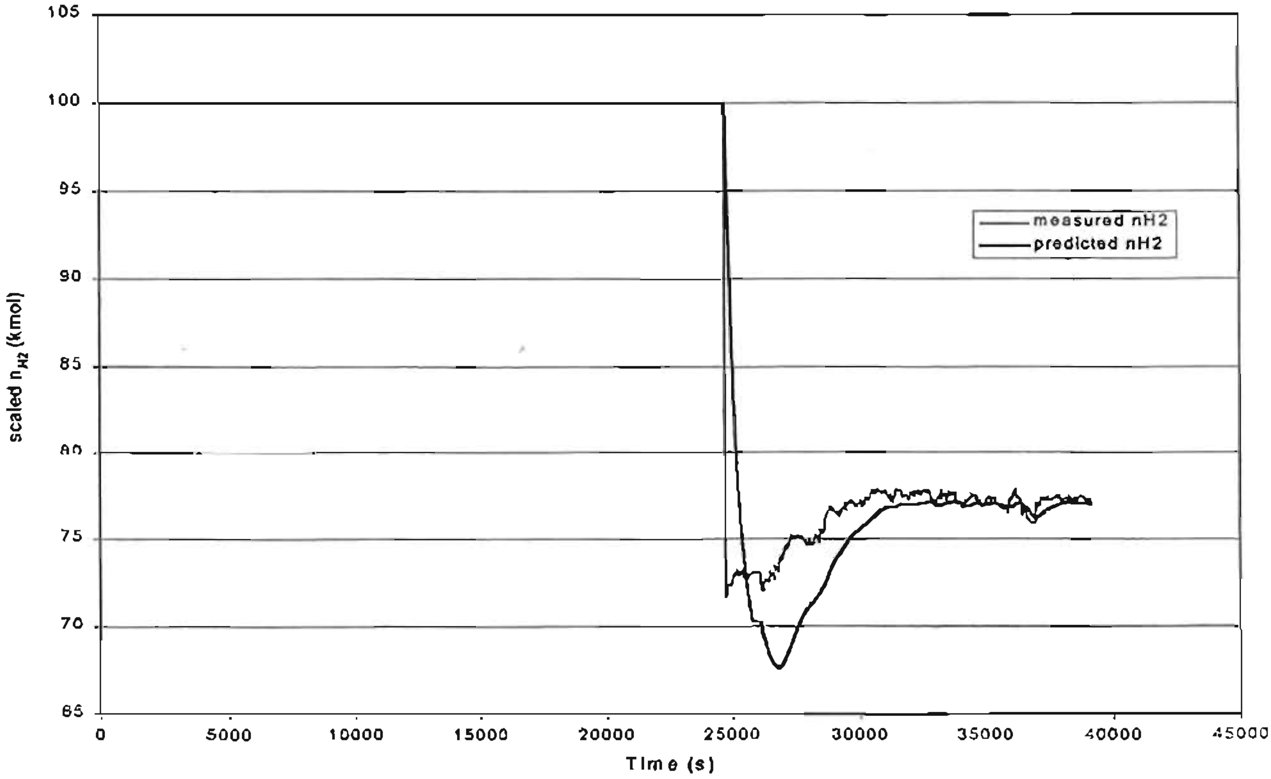


Figure 7.3: On-line results on algorithm start-up for a butene grade for n_{H2}

7.3.3 A Comparison of Hexene and Butene Runs

Below (*figures* 7.4 and 7.5) are results from the EKF for a hexene and a butene grade. One can see that the trends are similar in that they both provide a very smooth prediction in contrast to the noisy plant data. These plots were also achieved with a universal tuning set, used for both butene and hexene grades. This is encouraging. Of some concern is that the predictions are clearly following the measured values, as opposed to predicting in advance. As mentioned above in *section* 6.2.3.7, correct tuning of the EKF should improve the situation and provide some opportunities for fault detection.

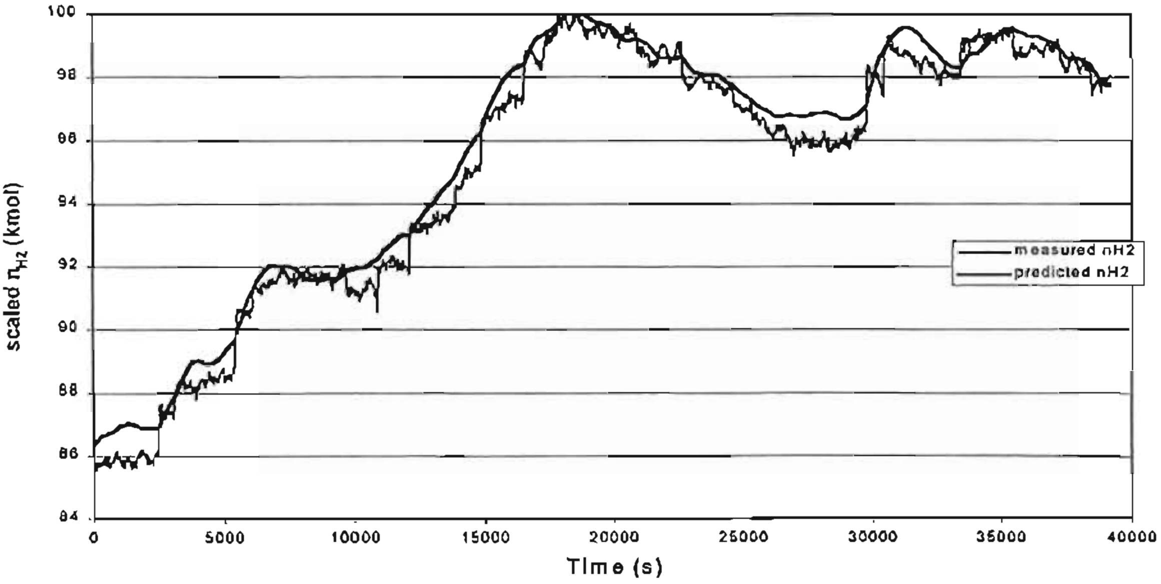


Figure 7.4: On-line results for a hexene grade for n_{H2}

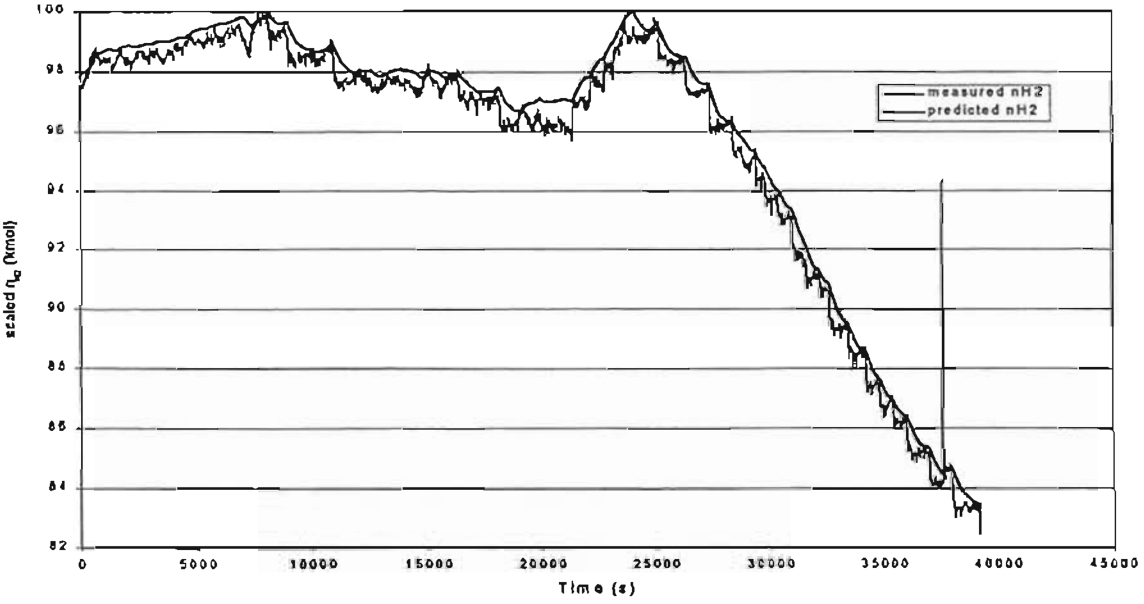


Figure 7.5: On-line results for a butene grade for n_{H2}

7.3.4 Unmeasurable States

One of the strengths of the EKF is its ability to provide unmeasurable state estimates from a few measurable measurements. The unmeasurable states in this study were n_{P2}, n_{P4} and n_{P6}. In addition to off-line results (*figures* 6.24 to 6.26) these parameters were successfully predicted *on-line* as well. *Figure* 7.6 below is an example of n_{P2}.

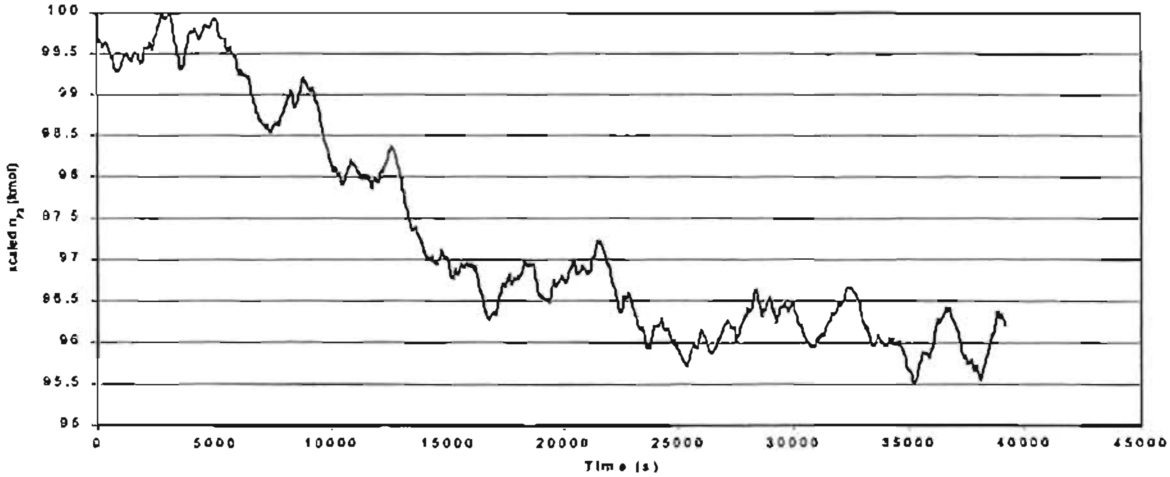
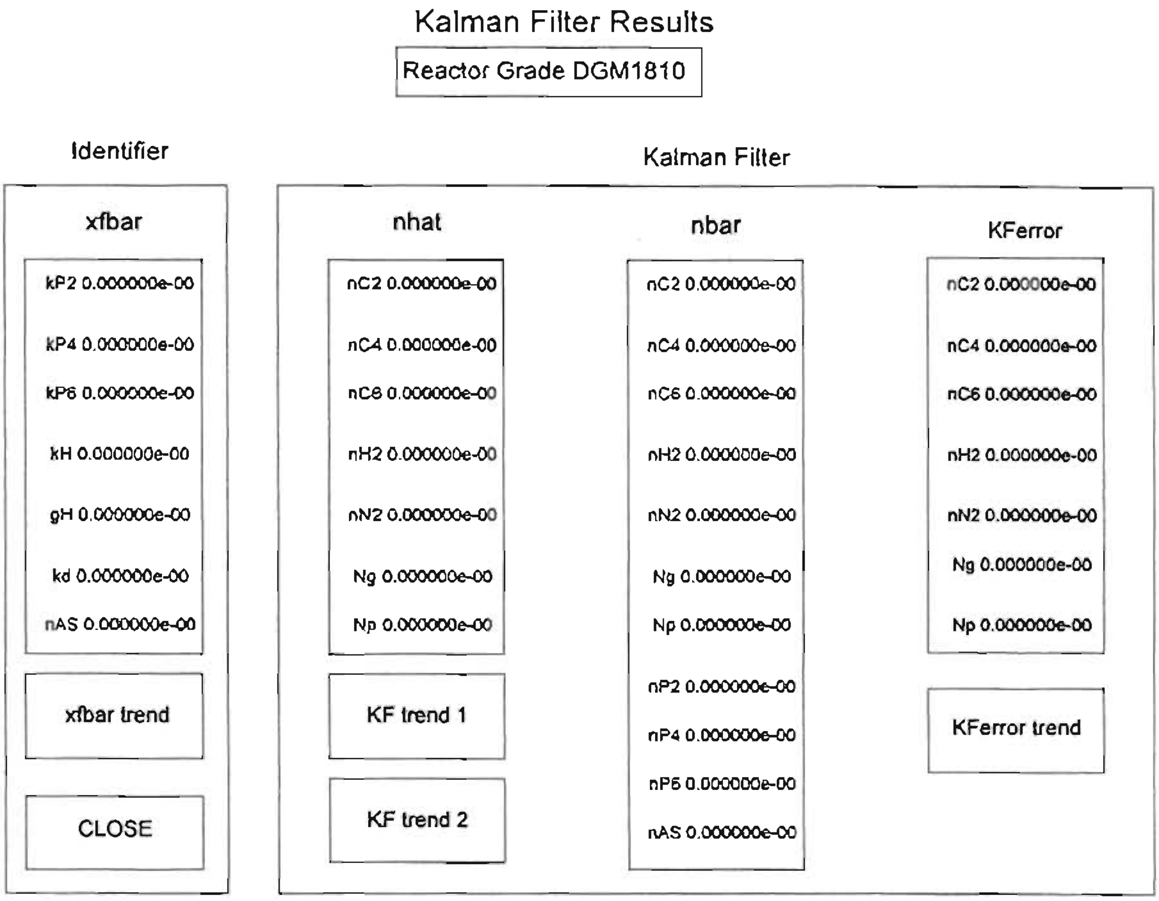


Figure 7.6: On-line results for a butene grade for unmeasurable n_{P2}

Similar graphs were obtained for n_{P4} for butene grades and for n_{P6} for hexene grades. When one compares the n_{P4} and n_{P6} plots (not shown here, since scaled values cannot be used for comparison purposes), it is interesting to note the differences in comonomer incorporation in the polymer chain for hexene grades and for butene grades. The degree of comonomer incorporation varies according to the grade that is being manufactured.

7.3.5 Operator Interfacing

A schematic of the algorithm was built, using the RTAP Schematic Builder. This allows operators easy access to the key results from the control algorithm. A copy of the schematic is given in *figure* 7.3 below. The plant version uses a range of colours though. It not only displays the measured and predicted values, it allows the user access to the identifier and Kalman filter trends. Should there be a suspected fault, (excessive difference between the predicted vector, nbar and the measured vector, nhat) the Kalman filter error (KFerror) trend can be accessed easily.



xfbar

kP2 0.000000e-00
kP4 0.000000e-00
kP6 0.000000e-00
kH 0.000000e-00
gH 0.000000e-00
kd 0.000000e-00
nAS 0.000000e-00

nhat

nC2 0.000000e-00
nC4 0.000000e-00
nC6 0.000000e-00
nH2 0.000000e-00
nN2 0.000000e-00
Ng 0.000000e-00
Np 0.000000e-00

nbar

nC2 0.000000e-00
nC4 0.000000e-00
nC6 0.000000e-00
nH2 0.000000e-00
nN2 0.000000e-00
Ng 0.000000e-00
Np 0.000000e-00
nP2 0.000000e-00
nP4 0.000000e-00
nP6 0.000000e-00
nAS 0.000000e-00

KFerror

nC2 0.000000e-00
nC4 0.000000e-00
nC6 0.000000e-00
nH2 0.000000e-00
nN2 0.000000e-00
Ng 0.000000e-00
Np 0.000000e-00

Figure 7.7: Results Schematic at Poly 2

CHAPTER 8

Conclusions and Recommendations

8.1 Conclusions

There are major gains to be made in precise polymer property control, control through grade changes to minimise off-specification product, and early detection of instrumentation faults. As a result, numerous researchers around the world have tackled these issues in ethylene polymerisation. The kinetic modelling, if it proves reliable, will offer the advantage of insight into the process, and allow monitoring of significant intermediate parameters such as reaction rate constants.

The algorithms described in this thesis were developed off-line in MATLAB, and tested on long records of plant data in this form before being translated into quite different code on a plant computer for the application. The program is running continuously at Poly 2. Basic range checking of input plant data has improved the robustness of the scheme. Results are similar to the off-line experiments. Bearing in mind the long response times of the process, the tuning of the algorithms is easier off-line, where long plant records can be processed quite rapidly. Since the on-line and off-line computations have been shown to be equivalent, there should be no difficulty in the near future in improving the tuning of the on-line algorithms.

This thesis presents an industrial application of an EKF in the polymerisation field. On-line tuning may still need some attention due to the complexity of the system. A RLS parameter estimator calculates pseudo-kinetic rate constants for POLIFIN's Ziegler-Natta catalyst system. These are updated with every detected GC update and are used to improve an EKF model that runs on a smaller time cycle (samples every 20s). This model provides smoothed estimates for some plant data and allows prediction of unmeasurable states, such as the polymer composition. There is also scope for a fault detection scheme to be implemented via the EKF.

8.2 Recommendations

Areas for possible improvement of the code are:

- For the polymer density, a constant, *ro* is used as a typical average laboratory value.
- Henry's Law can possibly improve the 'solubility factors', s1, s2 and s3. This would require some literature research.
- The use of sparse matrix methods to eliminate wasted space arising from the zero-based matrices.
- The parameters k_d and n_{AS} play an integral role in the polymerisation process yet the results achieved are not as satisfactory as was anticipated.

POLIFIN now has the required groundwork completed and has suggested that the code be used as follows:

- The schematic will allow monitoring of reaction rate constants via the RLS parameter estimation scheme. Graphical results from the EKF, in addition to providing operators with additional information, should be linked to a fault detection/alarm system, where possible instruments errors could be detected early.
- Special attention must be paid to the tuning issues and the complexity in this area. Tuning should be continued, bearing in mind that it may be different for the two comonomer grade types.

References

Afonso, P. A. F. N. A., J. M. L. Ferreira, et al. (1998). "Sensor Fault Detection and Identification in a Pilot Plant Under Process Control." Trans IChemE **76**(A): 490-498.

Becerra, V. M. and P. D. Roberts (2000). "Applying the extended Kalman filter to systems described by nonlinear differential-algebraic equations." Control Engineering Practice.

Box, G. E. P. and G. M. Jenkins (1976). Time Series Analysis: Forecasting and Control. San Francisco, California, Holden-Day.

Coville, N. J. (1999). Polyolefins: The Catalyst Connection - Part 1. Chemical World. **April 1999:** 25-27.

Crowley, T. J. and K. Y. Choi (1998). "Experimental studies on optimal molecular weight distribution control in a batch-free radical polymerization process." Chemical Engineering Science **53**(15): 2769-2790.

Davidovici, G. (2000). The World of Polyethylene and its Competitive Position. Sasolburg, Free State, South Africa, Polifin Limited.

de Wolf, S., R. L. E. Cuypers, et al. (1996). "Model Predictive Control of a Slurry Polymerization Reactor." Computers Chem. Engng. **20**(Suppl.): S955-S961.

Embirucu, M., E. L. Lima, et al. (1996). "A Survey of Advanced Control of Polymerization Reactors." Polymer Engineering and Science **36**(4): 433-447.

Foster, G. (1991). Polymer Reaction Engineering Course. Hamilton, Ontario, McMaster University.

Gupta, Y. P. (1998). "Control of Integrating Processes using Dynamic Matrix Control." Trans IChemE **76**(A).

Hamielec, A. E. and J. B. P. Soares (1996). "Polymerization Reaction Engineering - Metallocene Catalysts." Prog. Polym. Sci. **21**: 651-707.

Isermann, R. (1980). "Practical Aspects of Process Identification." Automatica **16**: 575-587.

James, D. E. (1986). Encyclopedia of Polymer Science and Engineering. New York, John Wiley and Sons.

James, D. E. (1986). Encyclopedia of Polymer Science and Engineering. J. I. Kroschwitz. New York, John Wiley & Sons. **6 p 429.**

Jazwinski, A. H. (1970). Stochastic Processes and Filtering Theory. New York, Academic Press.

Jo, J. H. and S. G. Bankoff (1976). "Digital Monitoring and Estimation of Polymerization Reactors." AIChE Journal **22**(2): 361-369.

Kalman, R. E. (1960). "A New Approach to Linear Filtering and Prediction Problems." ASME Trans. J. Basic Engng **82**(D): 35.

Kozub, D. J. and J. F. MagGregor (1992). "State Estimation for Semi-batch Polymerization Reactors." Chemical Engineering Science **47**(5): 1047-1062.

Lines, B., D. Hartlen, et al. (1993). Polyethylene reactor modeling and control design. Hydrocarbon Processing: 119-124.

Ljung, L. (1999). System Identification : Theory for the User (Prentice Hall Information and System Sciences Series).

McAuley, K. B. (1991). Modeling, Estimation and Control of Product Properties in a Gas Phase Polyethylene Reactor. Chemical Engineering. Hamilton, Ontario, McMaster University: 328.

McAuley, K. B. (2000). Modelling of Gas-phase Polyethylene Reactors - Different Models for Different Purposes. AspenWorld Conference, Orlando, Florida.

McAuley, K. B. and J. F. MacGregor (1991). "On-Line Inference of Polymer Properties in an Industrial Polyethylene Reactor." AIChE Journal **37**(6): 825-835.

McAuley, K. B. and J. F. MacGregor (1992). "Optimal Grade Transitions in a Gas Phase Polyethylene Reactor." AIChE Journal **38**(10): 1564-1576.

McAuley, K. B. and J. F. MacGregor (1993). "Nonlinear Product Property Control in Industrial Gas-Phase Polyethylene Reactors." AIChE Journal **39**(5): 855-866.

McAuley, K. B., J. F. MacGregor, et al. (1990). "A Kinetic Model for Industrial Gas-Phase Ethylene Copolymerization." AIChE Journal **36**(6): 837-850.

Michalovic, M., K. Anderson, et al. (1996). "The Macrogalleria - a cyberwonderland of polymer fun." .

Mulholland, M. and L. A. R. Fernandes (1997). "Combined Smith prediction and recursive filtering for observation of a polyethylene reactor." Computers chem. Engng **21**(11): 1283-1289.

Mulholland, M. and J. H. Seinfeld (1995). "Inverse Air Pollution Modelling of Urban-scale Carbon Monoxide Emissions." Atmospheric Environment **29**(4): 497-516.

Narotam, N. K. (1999). Development of Artificial Neural Networks for the Modelling and Control of Chemical Processes. School of Chemical Engineering. Durban, University of Natal: 239.

Royle, B. and B. Rackham (1998). Polifin's Hexene-based PE-LLD. Plastics SA. **October 1998:** 16-20.

Schumacher, J. W. (1996). "CEH Abstract, Linear Low-density Polyethylene (LLDPE) Resins." .

Shaw, B. M., K. B. McAuley, et al. (1998). "Simulating Joint Chain Length and Composition Fractions from Semi-batch Ethylene Copolymerization Experiments." Polymer Reaction Engineering **6**(2): 113-142.

Tham, M. T. and A. Parr (1994). Succeed at On-line Validation and Reconstruction of Data. Chemical Engineering Progress: 46-56.

Thomason, R. P., R. Dunwoodie, et al. (2000). Application of Kalman Filters and Artificial Neural Networks in Ethylene Polymerisation Control. SAIChE 9th National Meeting, Secunda, South Africa.

Trevelyan, J. (1996). "Package of Matrix and Vector Utility Functions for use with "Numerical Recipes"." .

Valappil, J. and C. Georgakis (2000). "Systematic Estimation of State Noise Statistics for Extended Kalman Filters." AIChE Journal **46**(2): 292-308.

Varela, A. E. (2000). "Self-Tuning Pressure Control in an Injection Moulding Cavity During Filling." Trans IChemE **78**(Part A): 79-86.

Wilson, D. I., M. Agarwal, et al. (1998). "Experiences implementing the extended Kalman filter on an industrial batch reactor." Computers Chem. Engng. **22**(11): 1653-1672.

Xie, T., K. B. McAuley, et al. (1994). "Gas Phase Ethylene Polymerization: Production Processes, Polymer Properties, and Reactor Modeling." Ind. Eng. Chem. Res. **33**: 449-479.

Xie, T., K. B. McAuley, et al. (1995). "Modeling Molecular Weight Development of Gas-Phase alpha-Olefin Copolymerization." AIChE Journal **41**(5): 1251-1265.

Zorzetto, L. F. M. and J. A. Wilson (1996). "Monitoring Bioprocesses using Hybrid Models and an Extended Kalman Filter." Computers chem. Engng **20**(Suppl.): S689-S694.

APPENDIX A

Pseudo-kinetic Rate Constants used in Site-based Modelling

Parameter	Units	Site 1	Site 2	Site 3
k_f	$m^3/(kmol.s)$	1	1	1
k_{I1}	$m^3/(kmol.s)$	1	1	1
k_{I2}	$m^3/(kmol.s)$	0.14	0.14	0.14
k_{p11}	$m^3/(kmol.s)$	85	85	85
k_{p12}	$m^3/(kmol.s)$	2	15	15
k_{p21}	$m^3/(kmol.s)$	64	64	64
k_{p22}	$m^3/(kmol.s)$	1.5	22.6	22.6
k_{fH1}	$m^3/(kmol.s)$	0.088	0.37	0.37
k_{fH2}	$m^3/(kmol.s)$	0.088	0.37	0.37
k_{fM11}	$m^3/(kmol.s)$	0.0021	0.0021	0.0021
k_{fM12}	$m^3/(kmol.s)$	0.006	0.11	0.11
k_{fM21}	$m^3/(kmol.s)$	0.0021	0.0021	0.0021
k_{fM22}	$m^3/(kmol.s)$	0.006	0.11	0.11
k_{fR1}	$m^3/(kmol.s)$	0.024	0.12	0.12
k_{fR2}	$m^3/(kmol.s)$	0.048	0.24	0.24
k_{fs1}	s^{-1}	0.0001	0.0001	0.0001
k_{fs2}	s^{-1}	0.0001	0.0001	0.0001
k_{ds}	s^{-1}	0.0001	0.0001	0.0001
k_{dI}	$m^3/(kmol.s)$	2000	2000	2000
k'_{fH}	s^{-1}	0.0088	0.037	0.037
k_{H1}	$m^3/(kmol.s)$	1	1	1
k_{H2}	$m^3/(kmol.s)$	0.1	0.1	0.1
k_{R1}	$m^3/(kmol.s)$	0.1	0.1	0.1
k_{R2}	$m^3/(kmol.s)$	0.01	0.01	0.01
k_a	s^{-1}	0.0003	0.0003	0.0003

Table A.1: List of kinetic values used in the site-based model (Shaw, McAuley et al. 1998)

APPENDIX B

Raw Signals Logged for Possible use in the Software

Parameter	Description	Units
Time	-	s
Grade	Code	-
BedAvDE	Bed average density	kg/m^3
InstDE	Instantaneous density	kg/m^3
BedAvMI	Bed average melt index	g/10min
InstMI	Instantaneous melt index	g/10min
MIlab	Laboratory melt index	g/10min
DElab	Laboratory density	kg/m^3
ReacTemp	R1 Bed Temp. Control	^{0}C
C4/C2	Butene Gas Ratio	-
C6/C2	Hexene Gas Ratio	-
H2/C2	Hydrogen Gas Ratio	-
C2PP	Ethylene partial pressure control	kPa-g
BTR	Bed turnover rate	h
Prod Rate	Production Rate	t/h
ANAL_1_on	A flag for which analyser is on	-
ETH_1	Ethylene analyser 1	%
ETH_0	Ethylene analyser 2	%
BUT	Butene analyser 1	%
BUT	Butene analyser 2	%
HEX	Hexene analyser 1	%
HEX	Hexene analyser 2	%
HYD	Hydrogen analyser 1	%
HYD	Hydrogen analyser 2	%
NIT	Nitrogen analyser 1	%
NIT	Nitrogen analyser 2	%
ETHANE	Ethane analyser 1	%
ETHANE	Ethane analyser 2	%
ISO	i-C5 analyser 1	%
ISO	i-C5 analyser 2	%
C4_INERTS	C4 inerts analyser 1	%
C4_INERTS	C4 inerts analyser 2	%
C6_INERTS	C6 inerts analyser 1	%
C6_INERTS	C6 inerts analyser 2	%
VENT_MR	DUPLICATE OF MONREC_VENT	Nm^3/h
VENTFLOW	DUPLICATE OF R1VENT_FLO	Nm^3/h
LEVELSP	SP for bed level	ft
LEVELPV	corrected bed level	ft
R1INLETT	R1 reactor inlet temp.	^{0}C
R1PRESSURE	R1 pressure control	kPa-g
UBED_DEN	R1 upper bed density	kg/m^3
LBED_DEN	R1 lower bed density	kg/m^3
ETHFLO	R1 ethylene flow control	kg/h
REC_COMON_FLO	Recovered comonomer flow	l/h
BUTFLO	R1 butene flow control	kg/h
HEXFLO	R1 hexene flow control	kg/h
TEAL_FLO	R1 Teal flow control	kg/h
H2FLO	R1 hydrogen flow control	kg/h
HP_DEOXO_FLO	R1 HP deoxo nitrogen injection	Nm^3/h
I-C5_FLO	R1 isopentane flow control	kg/h
MONREC_VENT	Monomer recovery vent	Nm^3/h
R1VENT_FLO	R1 reactor vent	Nm^3/h
CYCLE_GASFLO	R1 cycle gas flow	Nm^3/h
BED_WEIGHT	R1 bed weight	t

CAT_FDR1	Catalyst flow rate - feeder 1	kg/h
CAT_FDR2	Catalyst flow rate - feeder 2	kg/h
N2_FLO	Cat feeder support flow	Nm^3/h

Table A.2: Raw signals logged for possible use in the software

APPENDIX C

Extracts of ANSI C Source Code used at Poly 2

KalmanCalc.h

```
/*********************************************************************/
/* sources/KalmanCalc.h 2000/04/11 N Narotam                         */
/*                                                                   */
/*   Copyright (c) N Narotam 1999, 2000                              */
/*   All Rights Reserved                                             */
/*                                                                   */
/* DESCRIPTION                                                       */
/* Header file for controller program                                */
/*********************************************************************/
/* COMPILITATION CONTROL                                             */
/* 2000/04/11  N Narotam  conceived                                  */
/*                                                                   */
/*********************************************************************/

extern void InitKalmanCalc (int);           /* Sets up debug status                                    */
extern int KalmanCalcInitialize (void);     /* Main initialisation routine                             */
extern rtUInt8 ControlDataBuffer (void);    /* Routine for GC update detection in buffer               */
extern int MainIdentLoop (void);            /* Main RLS loop run whenever the GC updates               */
extern int UpdateIdentData (void);          /* Routine containing writes back to the database          */
extern int MainKalmanLoop (void);           /* Main Kalman filter loop run on every time cycle         */
extern int UpdateKalmanData (void);         /* Routine containing writes back to database              */
extern int IdentPlotData (void);            /* Routine to store identifier data in DataBase for plotting */
extern int KalmanPlotData (void);           /* Routine to store Kalman filter data in DataBase for     */
                                            /* plotting                                                */
```

KalmanCalc.c

```
/****************************************************************************/
/* sources/KalmanCalc.c 2000/04/11 N Narotam,R Thomason  */
/*                    M.Mulholland, C. van der Merwe      */
/*  Copyright (c) N Narotam 1999, 2000                    */
/*  All Rights Reserved                                   */
/*                                                        */
/* DESCRIPTION                                            */
/*   Source file - Main program source code               */
/****************************************************************************/
/* COMPILITATION CONTROL                                  */
/* 2000/04/11      N Narotam Conceived                    */
/*                                                        */
/****************************************************************************/

#include <cr/crStandards.h>
#include <rtap/rtap.h>
#include <math.h>
#include <time.h>
#include <sys/time.h>

#include "KalmanControl.h"
#include "KalmanCalc.h"
#include "DbInterface.h"
#include "nrutil.h"

/* local function prototypes */
void PrintBufferLine (int);
void DataLineRead (void);
int HexeneMode (void);
int PartMatExpKF (double **,double **,double **,double **,int,int);
int PartMatExpIdent (double **,double **,double **,double **,int,int);

/* local data storage */
static int thisDebug;
static int firstIdent;

/* Data History Stack */
static BufferControl bc;
static bufferTable buffer[100];
static bufferTable DataLine;
static rtDouble *r1;
static rtDouble *r2;

/* Model Data storage area */
static rtDouble **B;
static rtDouble **G;
static rtDouble **H;
static rtDouble **HAS;
static rtDouble **h;

/* RLS (identifier) data storage area */
static rtDouble *xfbar;
static TypicalData xfTyp;
static rtDouble **Kld;
static rtDouble *nobs;
static rtDouble *nobs_pred;
static rtDouble *nobs_tm1;
static rtDouble **Q;
static rtDouble **C;
static rtDouble **M;
static rtDouble **Gir;
static rtDouble **R;
static rtDouble **Iid;
static rtDouble **Ii;
static rtDouble *xfbarmin;
static rtDouble *xfbarmax;

/* Kalman filter data storage area */
static rtDouble **Gkf;
static rtDouble **Qkf;
static rtDouble **Rkf;
static rtDouble **Mkf;
static rtDouble **Ikf;
static rtDouble **Kkf;
static rtDouble *nbarmin;
static rtDouble *nbarmax;
static rtDouble *nbar;
static rtDouble *nhat;
static rtDouble *KFerror;

/* general data storage area */
static FixedParam fp;
static rtDouble *x_typical_hex;
static rtDouble *x_typical_but;
static rtDouble *x_typical;
static rtDouble *MW;

/**********************************************************/
/*      InitKalmanCalc:  Initialisation routine for this */
/*                       function.  Currenly sets up     */
/*                       debug status only.              */
/*      type:            external                        */
/**********************************************************/
void InitKalmanCalc (int GlobDebug)
{
thisDebug = GlobDebug & 0x04;
if (thisDebug)
  printf ("KalmanCalc is in Debug mode\n");
}

/**********************************************************************/
/* KalmanCalcInitialize: Main initialization routine.                 */
/*                   Must be called at the start of                   */
/*                   application and is done once only or by          */
/*                   operator action on demand                        */
/*      type:            external                                     */
/*      Return Codes:    0 if no errors                               */
/*                       1 if there were errors                       */
/**********************************************************************/

int KalmanCalcInitialize (void) {

bufferTable zerobuffer[100] = {0};
rtDouble smallnumber = 0.001;
xfbarHist xfbarhist, xfbarzero = {0};
nhatHist nhathist, nhatzero= {0};
nhatHist KFerrorhist, KFerrorzero = {0};
nbarHist nbarhist, nbarzero = {0};
int i;

/* Allocate memory - general data*/
MW      = dvector (0,2);
B       = sdmatrix (11,8);
G       = sdmatrix (11,11);
H       = sdmatrix (11,11);
HAS     = sdmatrix (11,11);
h       = sdmatrix (11,11);

x_typical_hex    = dvector(0,2);
x_typical_but    = dvector(0,2);
x_typical= dvector(0,2);
```

```
r1 = dvector(0,8);
r2 = dvector(0,8);

/* Allocate memory - Kalman Filter */
Gkf                 = sdmatrix (7,11);
Qkf                 = sdmatrix (11,11);
Rkf                 = sdmatrix (7,7);
Mkf                 = sdmatrix (11,11);
Ikf                 = sdmatrix (11,11);
Kkf                 = sdmatrix (11,7);

nbarmin             = dvector (0,10);
nbarmax             = dvector (0,10);
nbar                = dvector (0,10);
nhat                = dvector (0,6);
KFerror  = dvector (0,6);

/* RLS Memory Alloc */
Kid                 = sdmatrix (7,8);
C                   = sdmatrix (8,11);
R                   = sdmatrix (8,8);
Q                   = sdmatrix (7,7);
M                   = sdmatrix (7,7);
Gir                 = sdmatrix (8,7);
Iid                 = sdmatrix (7,7);
Ii                  = sdmatrix (11,11);
xfbar               = dvector (0,ROWIDENT-1);
xfbarmin = dvector (0,ROWIDENT-1);
xfbarmax            = dvector (0,ROWIDENT-1);
nobs                = dvector(0,7);
nobs_pred           = dvector(0,7);
nobs_tm1            = dvector(0,7);

/* general program initialisation */
if (thisDebug) printf ("**** Start General Program Initialisations ****\n");
firstIdent = 1;

/* initialize global identity matrices */
deyes (7,7,Iid);
deyes (11,11,Ii);
deyes (11,11,Ikf);

SingleRead ( (rtUInt8 *) &fp,43);
SingleRead ( (rtUInt8 *) x_typical_hex,44);
SingleRead ( (rtUInt8 *) x_typical_but,45);
SingleRead ( (rtUInt8 *) MW,46);

fp.dt = DT;
SingleWrite ( (rtUInt8 *) &fp,43);

/* Fixed Matrices Initialisation */
if (thisDebug) printf ("**** Fixed model matrices initialisation ****\n");

SingleRead ( (rtUInt8 *) *B,42);
SingleRead ( (rtUInt8 *) *G,38);
SingleRead ( (rtUInt8 *) *H,39);
SingleRead ( (rtUInt8 *) *HAS,40);
SingleRead ( (rtUInt8 *) *h,41);

/*
if (thisDebug) {
  printf ("****MW****\n");
  show_dvector (MW,0,2);
  printf ("****B****\n");
  show_dmatrix (B,0,10,0,7);
  printf ("****G****\n");
  show_dmatrix (G,0,10,0,10);
  printf ("****H****\n");
  show_dmatrix (H,0,10,0,10);
  printf ("****HAS****\n");
  show_dmatrix (HAS,0,10,0,10);
  printf ("****h****\n");
  show_dmatrix (h,0,10,0,10);
}
*/

/* set up h (depends on the grade) */
h[0][0] = -1.0*fp.s2;
h[9][0] = -1.0*fp.s2;
if (HexeneMode()) {
  h[2][2] = -fp.s6;
  h[1][1] = 0.0;
  h[9][2] = -fp.s6;
  h[9][1] = 0.0;
} else {
  h[2][2] = 0.0;
  h[1][1] = -fp.s4;
  h[9][2] = 0.0;
  h[9][1] = -fp.s4;
}

SingleWrite ( (rtUInt8 *) *h,41);

/* Kalman Filter specific Initialization */
if (thisDebug) printf("**** Initializing Kalman filter ****\n");

SingleRead ( (rtUInt8 *) *Qkf,55);
SingleRead ( (rtUInt8 *) *Mkf,56);
SingleRead ( (rtUInt8 *) *Rkf,57);
SingleRead ( (rtUInt8 *) *Gkf,58);
SingleRead ( (rtUInt8 *) *Kkf,65);
SingleRead ( (rtUInt8 *) nbarmin,53);
SingleRead ( (rtUInt8 *) nbarmax,54);

zero_dvector (nbar,0,10);

/* Initialize Mkf */
dmsmy (Ikf,11,11,smallnumber,Mkf);
SingleWrite ( (rtUInt8 *) *Mkf,56);

/*
if (thisDebug) {
  printf ("****Qkf****\n");
  show_dmatrix (Qkf,0,10,0,10);
  printf ("****Ikf****\n");
  show_dmatrix (Ikf,0,10,0,10);
  printf ("****Mkf****\n");
  show_dmatrix (Mkf,0,10,0,10);
  printf ("****Rkf****\n");
  show_dmatrix (Rkf,0,6,0,6);
  printf ("****Gkf****\n");
  show_dmatrix (Gkf,0,6,0,10);
  printf ("****nbarmin****\n");
  show_dvector (nbarmin,0,10);
  printf ("****nbarmax****\n");
  show_dvector (nbarmax,0,10);
}
*/

/* RLS (ident) Specific Initialization */
if (thisDebug) printf ("**** Initializing RLS Identifier ****\n");

SingleRead ( (rtUInt8 *) xfbarmin,51);
SingleRead ( (rtUInt8 *) xfbarmax,52);
SingleRead ( (rtUInt8 *) *M,35);
SingleRead ( (rtUInt8 *) *Kid,34);
SingleRead ( (rtUInt8 *) &xfTyp,33);
SingleRead ( (rtUInt8 *) *C,48);

/* Initial values for kinetic parameters to be identified */
xfbar[KP2]          = xfTyp.kP2i_typical;

DataLineRead(); /* so that HexeneMode function is active */

if (HexeneMode ()){
          xfbar[KP4] = xfTyp.kP4i_typical_hex;
          xfbar[KP6] = xfTyp.kP6i_typical_hex;
}else{
          xfbar[KP4] = xfTyp.kP4i_typical_but;
          xfbar[KP6] = xfTyp.kP6i_typical_but;
}

xfbar[KH]           = xfTyp.kHi_typical;
xfbar[GH]           = xfTyp.gHi_typical;
xfbar[KD]           = xfTyp.kdi_typical;
xfbar[NASI]         = xfTyp.nAS_typical;

/* write xfbar to the DataBase */
SingleWrite ( (rtUInt8 *) xfbar,32);

/* Initialize M */
dmsmy (Ii,7,7,smallnumber,M);
SingleWrite ( (rtUInt8 *) *M,35);

zero_dmatrix (R,0,7,0,7);
R[0][0] = 1.0e15 / pow(30,2);                /* nC2   */
R[1][1] = 1.0e12 / pow(1,2);                 /* nC4   */
R[2][2] = 1.0e12 / pow(1,2);                 /* nC6   */
R[3][3] = 1.0e6     / pow(1,2);              /* nH2   */
R[4][4] = 1.0e15 / pow(50,2);                /* nN2   */
R[5][5] = 1.0e14 / pow(8,2);                 /* nAS   */
R[6][6] = 1.0e14 / pow(100,2);               /* Ng    */
R[7][7] = 1.0e14 / pow(100,2);               /* Np    */

SingleWrite ( (rtUInt8 *) *R,37);

zero_dmatrix (Q,0,6,0,6);
Q[0][0] = 1.0e-10 / pow(xfTyp.kP2i_typical,2);          /* kP2 */

if (HexeneMode ()) {
          Q[1][1]=1e-20;
          Q[2][2]=1e-13 / pow(xfTyp.kP6i_typical_hex,2);
}
else {
          Q[1][1]=1e-12 / pow(xfTyp.kP4i_typical_but,2);
          Q[2][2]=1e-20;
}

Q[3][3] = 1.0e-17 / pow(xfTyp.kHi_typical,2); /* kH was a bit jumpy */
Q[4][4] = 1.0e-14 / pow(xfTyp.gHi_typical,2); /* gH was a bit jumpy */
Q[5][5] = 1.0 / pow(xfTyp.kdi_typical,2);     /* kd : minimise kd variations to fit data until we
                                              /* understand it better */
Q[6][6] = 1.0e-10 / pow(xfTyp.nAS_typical,2);/* nAS : minimise nAS variations to fit data until we
                                              /* understand it better */

SingleWrite ( (rtUInt8 *) *Q,47);

/*
if (thisDebug) {
  printf ("*****C*****\n");
  show_dmatrix (C,0,7,0,10);
  printf ("****xfbarmin****\n");
  show_dvector (xfbarmin,0,ROWIDENT-1);
  printf ("****xfbarmax****\n");
  show_dvector (xfbarmax,0,ROWIDENT-1);
  printf ("****Q****\n");
  show_dmatrix (Q,0,6,0,6);
  printf ("****R****\n");
  show_dmatrix (R,0,7,0,7);
}
*/

/* Initialize the data history points for plotting (KalmanHist) */
if (thisDebug) printf ("**** Initializing history stacks ****\n");

SingleRead ( (rtUInt8 *) &xfbarhist,68);
SingleRead ( (rtUInt8 *) &nhathist,69);
SingleRead ( (rtUInt8 *) &nbarhist,70);
SingleRead ( (rtUInt8 *) &KFerrorhist,71);

for (i=0;i<MAXIDHIST;i++) {
        ChangePtRecord (68,i);
        SingleWrite ( (rtUInt8 *) &xfbarzero,68);
}
for (i=0;i<MAXKFHIST;i++) {
        ChangePtRecord (69,i);
        SingleWrite ( (rtUInt8 *) &nhatzero,69);
        ChangePtRecord (70,i);
        SingleWrite ( (rtUInt8 *) &nbarzero,70);
        ChangePtRecord (71,i);
        SingleWrite ( (rtUInt8 *) &KFerrorzero,71);
}

/* Data History Stack initialisation */

SingleRead ( (rtUInt8 *) &bc,59);

bc.buff_pointer                = bc.Nbuffer - 1;
bc.pointergap_GCdeadtime             = fp.GCdeadtime / fp.dt;
bc.pointerlastupdate                 = bc.pointergap_GCdeadtime;
bc.Nstored                           = 0;

SingleWrite ( (rtUInt8 *) &bc,59);

if (thisDebug) printf ("Reading buffer table... \n");
SingleRead ( (rtUInt8 *) &DataLine,60);
SingleRead ( (rtUInt8 *) &buffer,61);

/*
if (thisDebug) PrintBufferLine (0);
*/

/* Initialize buffer to zero */
SingleWrite ( (rtUInt8 *) &zerobuffer,61);

if (thisDebug) printf ("**** Finished main initialisation routine ****\n");

return (0);
} /* end of KalmanCalcInitialize */
```

```
/**************************************************************************************/
/*        PrintBufferLine: Prints the desired quantity of the current line of plant data      */
/*                         (that has been appropriately manipulated for                       */
/*                         use in the code) directly from the buffer                          */
/*        Type: Local                                                                         */
/**************************************************************************************/

void PrintBufferLine (int line)
{
printf ("DataLine: %d\n",line);
printf ("ETHFLO: %lf\n",buffer[line].ETHFLO);
printf ("BUTFLO: %lf\n",buffer[line].BUTFLO);
printf ("CATFDRTOT: %lf\n",buffer[line].CATFDRTOT);
}

/**************************************************************************************/
/*        HexeneMode: Tells the user which comonomer is being used                            */
/*        for the current recipe by checking which composition.                               */
/*        from the analysers is greater. Cannot look at the grade                             */
/*        code string as the new code often appears    only long after                        */
/*        an actual grade changeover.                                                         */
/*        Type:            Local                                                              */
/*        Return codes:    0 if 1-butene is being used                                        */
/*                         1 if 1-hexene is being used                                        */
/**************************************************************************************/

int HexeneMode (void)
{
        if (DataLine.BUT > DataLine.HEX)   /* base on composition */
        {
                return(0);
        }
        else
        {
                return(1);
        }
}


/**************************************************************************************/
/*        DataLineRead:   Reads raw plant data and manipulates it into a form suitable        */
/*        for the model.  Also checks which analyser is being used and                        */
/*        selects appropriate GC values.  Butene and hexene flows are                         */
/*        logged as identical so one is selected to zero, using Hexenemode()                  */
/*        Type:            Local           .                                                  */
/**************************************************************************************/

void DataLineRead (void)
{
ApsPV           PVPoint;
ApsAnalog       AnalogPoint01,AnalogPoint02;
UniPV           PVUni;
ApsBOOL                 BOOLPoint;

/* read the current data                              */

SingleRead ( (rtUInt8 *) &AnalogPoint01,62);
DataLine.BEDWEIGHT = AnalogPoint01.value*1000.0;

SingleRead ( (rtUInt8 *) &PVPoint,2);
DataLine.ETHFLO = PVPoint.value / (MW[0]*3600.0);
SingleRead ( (rtUInt8 *) &PVPoint,3);
DataLine.BUTFLO = PVPoint.value / (MW[1]*3600.0);
SingleRead ( (rtUInt8 *) &PVPoint,4);
DataLine.HEXFLO = PVPoint.value / (MW[2]*3600.0);   .
SingleRead ( (rtUInt8 *) &PVPoint,5);
DataLine.H2FLO = PVPoint.value / (2*3600.0);
```

```
SingleRead ( (rtUInt8 *) &PVPoint,6);
DataLine.N2FLO = (PVPoint.value *1.013)/ (0.08314 * (20.0 + 273.15) * 3600.0);

SingleRead ( (rtUInt8 *) &AnalogPoint01,7);
SingleRead ( (rtUInt8 *) &AnalogPoint02,8);
DataLine.CATFDRTOT = (AnalogPoint01.value + AnalogPoint02.value)/3600.0;

SingleRead ( (rtUInt8 *) &PVPoint,9);
DataLine.R1VENTFLO = (PVPoint.value*1.013)/(0.08314 * (20.0 + 273.15) * 3600.0);

SingleRead ( (rtUInt8 *) &PVUni,10);
DataLine.PRODRATE = PVUni.value*1000.0/3600.0;

SingleRead ( (rtUInt8 *) &PVPoint,11);
DataLine.REACTEMP = PVPoint.value;

SingleRead ( (rtUInt8 *) &PVPoint,12);
DataLine.R1PRESSURE = (PVPoint.value+101.325)*0.01;

SingleRead ( (rtUInt8 *) &BOOLPoint,13);

if (BOOLPoint.value == 1) {
  SingleRead ( (rtUInt8 *) &AnalogPoint01,14);
  DataLine.ETH = AnalogPoint01.value;
  SingleRead ( (rtUInt8 *) &AnalogPoint01,16);
  DataLine.BUT = AnalogPoint01.value;
  SingleRead ( (rtUInt8 *) &AnalogPoint01,18);
  DataLine.HEX = AnalogPoint01.value;
  SingleRead ( (rtUInt8 *) &AnalogPoint01,20);
  DataLine.HYD = AnalogPoint01.value;
  SingleRead ( (rtUInt8 *) &AnalogPoint01,22);
  DataLine.NIT = AnalogPoint01.value;
  SingleRead ( (rtUInt8 *) &AnalogPoint01,24);
  DataLine.ETHANE = AnalogPoint01.value;
  SingleRead ( (rtUInt8 *) &AnalogPoint01,26);
  DataLine.ISO = AnalogPoint01.value;
  SingleRead ( (rtUInt8 *) &AnalogPoint01,28);
  DataLine.C4INERTS = AnalogPoint01.value;
  SingleRead ( (rtUInt8 *) &AnalogPoint01,30);
  DataLine.C6INERTS = AnalogPoint01.value;

} else {
  SingleRead ( (rtUInt8 *) &AnalogPoint01,15);
  DataLine.ETH = AnalogPoint01.value;
  SingleRead ( (rtUInt8 *) &AnalogPoint01,17);
  DataLine.BUT = AnalogPoint01.value;
  SingleRead ( (rtUInt8 *) &AnalogPoint01,19);
  DataLine.HEX = AnalogPoint01.value;
  SingleRead ( (rtUInt8 *) &AnalogPoint01,21);
  DataLine.HYD = AnalogPoint01.value;
  SingleRead ( (rtUInt8 *) &AnalogPoint01,23);
  DataLine.NIT = AnalogPoint01.value;
  SingleRead ( (rtUInt8 *) &AnalogPoint01,25);
  DataLine.ETHANE = AnalogPoint01.value;
  SingleRead ( (rtUInt8 *) &AnalogPoint01,27);
  DataLine.ISO = AnalogPoint01.value;
  SingleRead ( (rtUInt8 *) &AnalogPoint01,29);
  DataLine.C4INERTS = AnalogPoint01.value;
  SingleRead ( (rtUInt8 *) &AnalogPoint01,31);
  DataLine.C6INERTS = AnalogPoint01.value;
}

if (HexeneMode())
  DataLine.BUTFLO = 0.0;
else
  DataLine.HEXFLO = 0.0;
```

```
}        /* end of DataLineRead */


/*************************************************************************************************/
/*       ControlDataBuffer: Checks for GC updates by comparing current and              */
/*       previous lines of data.  Also allows one to look                                */
/*       back in historical data for input flows for the                                 */
/*       identifier and filter.  Evaluates interval that                                 */
/*       the identifier runs on.                                                          */
/*       Type:                    External                                                */
/*       Return Codes:            0 to 7 (See return statement at end of function)       */
/*************************************************************************************************/

rtUInt8 ControlDataBuffer (void)
{
        static int lastupdate_tempstore;              /* Value holder for the buff_pointer */
        static int firstloop = 1;
        rtUInt8          returncode = 0;
        rtUInt32 gap;
        rtUInt16 LastLineIndex;
        bufferTable      LastLineValues;
        int              i;

        if (thisDebug) printf ("**** ControlDataBuffer ****\n");

        if (firstloop != 1) bc.pointerlastupdate = lastupdate_tempstore;
        firstloop = 0;

        bc.buff_pointer = bc.buff_pointer + 1;
        if (bc.buff_pointer >= bc.Nbuffer)
          bc.buff_pointer = 0;

        if (thisDebug) {
                printf ("buff_pointer is %d\n",bc.buff_pointer);
                printf ("pointerlastupdate is %d\n", bc.pointerlastupdate);
        }

        DataLineRead ();

        /* do validity checks on raw data */


        /* set reactor up bit if the production rate is a valid number */
        printf("!!!DataLine.PRODRATE is %lf\n",DataLine.PRODRATE);
        if (DataLine.PRODRATE > MINPRODRATE)
                returncode += REACTORBIT; /* reactor is up */

        /* update the history stack with the latest information */
        ChangePtRecord (60,bc.buff_pointer);
        SingleWrite ( (rtUInt8 *) &DataLine,60);

        /* set Nstored bit if the buffer is full */
        bc.Nstored = bc.Nstored + 1;
        if (bc.Nstored >=bc.Nstored_criterion) {
                /* set Nstored bit if the buffer is full */
                returncode += NSTOREDBIT; /* buffer full */
                if (bc.buff_pointer == 0)
                        LastLineIndex = bc.Nbuffer - 1;
                else
                        LastLineIndex = bc.buff_pointer - 1;

                /* read the data from the buffer for the last and current readings */
                ChangePtRecord (60,LastLineIndex);
                SingleRead ( (rtUInt8 *) &LastLineValues,60);

                if (thisDebug) printf ("LastLineIndex is %d\n",LastLineIndex);
        r1[0] = LastLineValues.ETH;          /* the previous line of readings */
        r1[1] = LastLineValues.BUT;
        r1[2] = LastLineValues.HEX;
        r1[3] = LastLineValues.HYD;
        r1[4] = LastLineValues.NIT;
        r1[5] = LastLineValues.ETHANE;
        r1[6] = LastLineValues.ISO;
        r1[7] = LastLineValues.C4INERTS;
        r1[8] = LastLineValues.C6INERTS;

        r2[0] = DataLine.ETH;                /* the current line of readings */
        r2[1] = DataLine.BUT;
        r2[2] = DataLine.HEX;
        r2[3] = DataLine.HYD;
        r2[4] = DataLine.NIT;
        r2[5] = DataLine.ETHANE;
        r2[6] = DataLine.ISO;
        r2[7] = DataLine.C4INERTS;
        r2[8] = DataLine.C6INERTS;

/*
        if (thisDebug) {
            printf ("****LastLineValues****\n");
            show_dvector (r1,0,8);
            printf ("****DataLine****\n");
            show_dvector (r2,0,8);
        }
*/

        bc.changecount = 0;
        for (i = 0; i <= 8; i++)
        {
            if( fabs(r1[i] - r2[i]) > fp.tolerance_conc_change)
            {
                bc.changecount = bc.changecount + 1;
            }
        }

        if (bc.pointerlastupdate > bc.buff_pointer )
            gap = bc.Nbuffer + bc.buff_pointer - bc.pointerlastupdate;
        else
            gap = bc.buff_pointer - bc.pointerlastupdate;
        bc.dtident = (rtUInt32)(gap * fp.dt);

        if (thisDebug) {
            printf("gap is %d\n",gap);
            printf("bc.dtident is %d\n",bc.dtident);
        }

        /* set the GC update bit if an update occurred */
        if((bc.changecount >= bc.changecount_criterion) | (bc.dtident >= fp.dt_forced_ident))
        {
            lastupdate_tempstore = bc.buff_pointer;     /* Cannot update directly yet. */
                                                        /* store for future use        */
            returncode += GCUPDATEBIT;
        }
    }

    if (thisDebug) printf ("ControlDataBuffer returncode: %d\n",returncode);

    return (returncode);
    /* returncode = 0: Reactor down                          */
    /* returncode = 1: Reactor up                            */
    /* returncode = 2: > Nstored, reactor down               */
    /* returncode = 3: > Nstored, reactor up                 */
    /* returncode = 4: GC update, < Nstored, reactor down    */
```

```
        /* returncode = 5: GC update, < Nstored, reactor up          */
        /* returncode = 6: GC update, > Nstored, reactor down        */
        /* returncode = 7: GC update, > Nstored, reactor up          */

}       /* end of ControlDataBuffer */


/**************************************************************************************/
/*      MainIdentLoop:  The Recursive Least Squares (RLS) parameter estimator.        */
/*      Only runs when a GC updates (returncodes 6,7).                                */
/*      Uses 'dtident' and calculates an improved set of                              */
/*      pseudo-kinetic rate constants in 'xfbar'.                                     */
/*      Type:           External                                                      */
/*      Return Codes:   0 if no errors                                                */
/*                      1 if there were errors                                        */
/**************************************************************************************/

int MainIdentLoop(void)                 /* RLS loop */
{
int k;                                  /* Counter for clipping*/

rtDouble Fin,Fout,percent_error;
rtDouble Bed_Mass, Vp, Vg, T, P, Ng,mMMp,Np;
rtDouble *flowread1;                    /* vector of flow readings corresponding to the last GC update */
rtDouble *flowread2;                    /* vector of flow readings corresponding to the current GC update */
rtDouble *flowread;                     /* vector of average readings of flowread1 and flowread2 */
rtDouble kP2i;
rtDouble kP4i;
rtDouble kP6i;
rtDouble kHi;
rtDouble gHi;
rtDouble kdi;
rtDouble Fvi;
rtDouble Fpi;
rtDouble nC2i;
rtDouble nC4i;
rtDouble nC6i;
rtDouble nH2i;
rtDouble nN2i;
rtDouble nASi;
rtDouble Ngi;
rtDouble Npi;
rtDouble        **Kayi;
rtDouble        **gi;
rtDouble        **Ai;
rtDouble        **Bi;
rtDouble        **Gi;
rtDouble        **Aistar;
rtDouble        **Bistar;
rtDouble        **Aisr;
rtDouble        **Bisr;
rtDouble        *Fbar;

/* Temporary storage arrays (used by both identifier and filter) */
rtDouble        **tm01;         /* temporary storage matrix 1 - used in matrix computations */
rtDouble        **tm02;         /* temporary storage matrix 2 - used in matrix computations */
rtDouble        **tm03;         /* temporary storage matrix 3 - used in matrix computations */
rtDouble        **tm04;         /* temporary storage matrix 4 - used in matrix computations */

rtUInt16 DTIndex1;
rtUInt16 DTIndex2;
bufferTable     DTValues1;
bufferTable     DTValues2;

if (thisDebug) printf ("**** MainIdentLoop ****\n");
```

```
/* allocate memory for our local data */
Fbar            = dvector (0,7);
flowread        = dvector (0,7);
flowread1       = dvector (0,7);
flowread2       = dvector (0,7);
Kayi            = dmatrix (0,10,0,10);
gi              = dmatrix (0,10,0,10);
Ai              = dmatrix (0,10,0,10);
Bi              = dmatrix (0,10,0,7);
Gi              = dmatrix (0,10,0,6);
Aistar          = dmatrix (0,10,0,10);
Bistar          = sdmatrix (11,8);
Aisr            = dmatrix (0,7,0,7);
Bisr            = dmatrix (0,7,0,7);
tm01            = dmatrix (0,10,0,10);
tm02            = dmatrix (0,10,0,10);
tm03            = dmatrix (0,10,0,10);
tm04            = dmatrix (0,10,0,10);


/* Read from the DataBase */
if ((int) (bc.pointerlastupdate - bc.pointergap_GCdeadtime) < 0)
        DTIndex1 = bc.Nbuffer + bc.pointerlastupdate - bc.pointergap_GCdeadtime;
else
        DTIndex1 = bc.pointerlastupdate - bc.pointergap_GCdeadtime;

if ((Int) (bc.buff_pointer - bc.pointergap_GCdeadtime) < 0)
        DTIndex2 = bc.Nbuffer + bc.buff_pointer - bc.pointergap_GCdeadtime;
else
        DTIndex2 = bc.buff_pointer - bc.pointergap_GCdeadtime;

/* read the data from the buffer for indices DTIndex1 and DTIndex2*/
ChangePtRecord (60,DTIndex1);
SingleRead ( (rtUInt8 *) &DTValues1,60);
ChangePtRecord (60,DTIndex2);
SingleRead ( (rtUInt8 *) &DTValues2,60);

if (thisDebug) {
printf ("DTIndex1 is %d\n",DTIndex1);
printf ("DTIndex2 is %d\n",DTIndex2);
}

/* flows for the last GC update */
flowread1[FR1FC2]       = DTValues1.ETHFLO;
flowread1[FR1FC4]       = DTValues1.BUTFLO;
flowread1[FR1FC6]       = DTValues1.HEXFLO;
flowread1[FR1FH2]       = DTValues1.H2FLO;
flowread1[FR1FN2]       = DTValues1.N2FLO;
flowread1[FR1FAS]       = DTValues1.CATFDRTOT;
flowread1[FR1FV]        = DTValues1.R1VENTFLO;
flowread1[FR1FP]        = DTValues1.PRODRATE;

/* flows for the current GC update */
flowread2[FR2FC2]       = DTValues2.ETHFLO;
flowread2[FR2FC4]       = DTValues2.BUTFLO;
flowread2[FR2FC6]       = DTValues2.HEXFLO;
flowread2[FR2FH2]       = DTValues2.H2FLO;
flowread2[FR2FN2]       = DTValues2.N2FLO;
flowread2[FR2FAS]       = DTValues2.CATFDRTOT;
flowread2[FR2FV]        = DTValues2.R1VENTFLO;
flowread2[FR2FP]        = DTValues2.PRODRATE;

/* average flows for the last 2 GC update */
flowread[FRFC2] = (flowread1[FR1FC2] + flowread2[FR2FC2]) / 2.0;
flowread[FRFC4] = (flowread1[FR1FC4] + flowread2[FR2FC4]) / 2.0;
flowread[FRFC6] = (flowread1[FR1FC6] + flowread2[FR2FC6]) / 2.0;
flowread[FRFH2] = (flowread1[FR1FH2] + flowread2[FR2FH2]) / 2.0;
```

```
flowread[FRFN2] = (flowread1[FR1FN2] + flowread2[FR2FN2]) / 2.0;
flowread[FRFAS] = (flowread1[FR1FAS] + flowread2[FR2FAS]) / 2.0;
flowread[FRFV]  = (flowread1[FR1FV] + flowread2[FR2FV]) / 2.0;
flowread[FRFP]  = (flowread1[FR1FP] + flowread2[FR2FP]) / 2.0;

Bed_Mass        = DataLine.BEDWEIGHT;                         /* kg polymer */
Vp                      = Bed_Mass / fp.ro;
Vg                      = fp.Vreactor - Vp;
T                       = DataLine.REACTEMP;                  /* temperature (0C) */
P                       = DataLine.R1PRESSURE;                /* pressure (bar-abs) */
Ng                      = P * Vg / (fp.Rgasconst * (T + 273.15));   /* ideal gas law */

x_typical[0] = nbar[NP2]/(nbar[NP2]+nbar[NP4]+nbar[NP6]);
x_typical[1] = nbar[NP4]/(nbar[NP2]+nbar[NP4]+nbar[NP6]);
x_typical[2] = nbar[NP6]/(nbar[NP2]+nbar[NP4]+nbar[NP6]);
mMMp = MW[0]*x_typical[0] + MW[1]*x_typical[1] + MW[2]*x_typical[2];
Np = Bed_Mass/mMMp;

/* Gas inventories */
nobs[ONC2]      = 0.01 * DataLine.ETH * Ng;
nobs[ONC4]      = 0.01 * DataLine.BUT * Ng;
nobs[ONC6]      = 0.01 * DataLine.HEX * Ng;
nobs[ONH2]      = 0.01 * DataLine.HYD * Ng;
nobs[ONN2]      = 0.01 * DataLine.NIT * Ng;
nobs[ONAS]      = xfbar[NAS];
nobs[ONG]       = Ng;
nobs[ONP]       = Np;

/* Flow readings this time step (Set up vector Fbar) */
Fbar[FC2] = flowread[FRFC2];            /* readings from interface (kmol/s) */
Fbar[FC4] = flowread[FRFC4];            /* readings from interface (kmol/s) */
Fbar[FC6] = flowread[FRFC6];            /* readings from interface (kmol/s) */
Fbar[FH2] = flowread[FRFH2];            /* readings from interface (kmol/s) */
Fbar[FN2] = flowread[FRFN2];            /* readings from interface (kmol/s) */
Fbar[FAS] = flowread[FRFAS];            /* total catalyst flow (kg/s) - will affect k parameters */
Fbar[FV] = flowread[FRFV];              /* readings from interface (kmol/s) */
Fbar[FP] = flowread[FRFP] / mMMp;       /* readings from interface (kmol/s) */

/* Rough Mass Balance Check - will be a bit off as vent to MonRec is not included */
Fin             = Fbar[FC2] + Fbar[FC4] + Fbar[FC6] + Fbar[FH2] + Fbar[FN2];
Fout    = Fbar[FV] + Fbar[FP];

if (Fin>0.0)
        percent_error = 100 * (Fin - Fout) / Fin;
else
        percent_error = 100.0;

if (firstIdent)
{
        dvcopy (nobs,8,nobs_tm1);       /* steady state for the first step */
        firstIdent = 0;
}

nC2i    = nobs_tm1[OM1NC2];
nC4i    = nobs_tm1[OM1NC4];
nC6i    = nobs_tm1[OM1NC6];
nH2i    = nobs_tm1[OM1NH2];
nN2i    = nobs_tm1[OM1NN2];
nASi    = nobs_tm1[OM1NAS];
Ngi     = nobs_tm1[OM1NG];
Npi     = nobs_tm1[OM1NP];

kP2i    = xfbar[KP2];
kP4i    = xfbar[KP4];
kP6i    = xfbar[KP6];
kHi     = xfbar[KH];
gHi     = xfbar[GH];
```

```
kdi     = xfbar[KD];

Fvi     = Fbar[FV];
Fpi     = Fbar[FP];

zero_dmatrix (Kayi,0,10,0,10);
Kayi[0][0]=-kP2i;
Kayi[1][1]=-kP4i;
Kayi[2][2]=-kP6i;
Kayi[3][3]=-kHi;
Kayi[6][0]=kP2i;
Kayi[7][1]=kP4i;
Kayi[8][2]=kP6i;
Kayi[9][0]=-kP2i;
Kayi[9][1]=-kP4i;
Kayi[9][2]=-kP6i;
Kayi[9][3]=-kHi;
Kayi[10][0]=kP2i;
Kayi[10][1]=kP4i;
Kayi[10][2]=kP6i;

zero_dmatrix (gi,0,10,0,10);
gi[3][3] = -gHi;
gi[5][5] = -kdi;
gi[9][3] = -gHi;


/* Fpi*h = tm01*/
dmsmy (h,11,11,Fpi,tm01);
/*  H + HAS = tm02*/
dmadd (H,11,11,HAS,tm02);
/* (Fpi/Npi) * (H+HAS) = tm03*/
dmsmy (tm02,11,11,(Fpi/Npi),tm03);
/* (Fpi/Npi) * (H+HAS) + Fpi*h = tm02 */
dmadd (tm03,11,11,tm01,tm02);
/* (Fvi/Ngi) * G = tm01 */
dmsmy (G,11,11,(Fvi/Ngi),tm01);
/* (Fvi/Ngi) * G + gi = tm03 */
dmadd (tm01,11,11,gi,tm03);
/* (Fvi/Ngi) * G + gi + (Fpi/Npi) * (H+HAS) + Fpi*h = tm01 */
dmadd (tm03,11,11,tm02,tm01);
/* nASi*Kayi = tm02 */
dmsmy (Kayi,11,11,nASi,tm02);
/* nASi*Kayi + (Fvi/Ngi) * G + gi + (Fpi/Npi) * (H+HAS) + Fpi*h = Ai */
dmadd (tm01,11,11,tm02,Ai);

dmcopy (B,11,8,Bi);

PartMatExpIdent (Ai,Bi,Aistar,Bistar,11,8);

zero_dmatrix (Gi,0,10,0,6);

Gi[0][0] = -nASi * nC2i;
Gi[0][6] = -kP2i * nC2i;

Gi[1][1] = -nASi * nC4i;
Gi[1][6] = -kP4i * nC4i;

Gi[2][2] = -nASi * nC6i;
Gi[2][6] = -kP6i * nC6i;

Gi[3][3] = -nASi * nH2i;
Gi[3][4] = -nH2i;
Gi[3][6] = -kHi * nH2i;

Gi[5][5] = -nASi;
```

```
Gi[6][0] = nASi * nC2i;
Gi[6][6] = kP2i * nC2i;

Gi[7][1] = nASi * nC4i;
Gi[7][6] = kP4i * nC4i;

Gi[8][2] = nASi * nC6i;
Gi[8][6] = kP6i * nC6i;

Gi[9][0] = -nASi * nC2i;
Gi[9][1] = -nASi * nC4i;
Gi[9][2] = -nASi * nC6i;
Gi[9][3] = -nASi * nH2i;
Gi[9][4] = -nH2i;
Gi[9][6] = -kP2i * nC2i - kP4i * nC4i - kP6i * nC6i - kHi * nH2i;

Gi[10][0] = nASi * nC2i;
Gi[10][1] = nASi * nC4i;
Gi[10][2] = nASi * nC6i;
Gi[10][6] = kP2i * nC2i + kP4i * nC4i + kP6i * nC6i;

/*
if (thisDebug) {
        printf ("****Kayi****\n");
        show_dmatrix (Kayi,0,10,0,10);
        printf ("****gi****\n");
        show_dmatrix (gi,0,10,0,10);
        printf ("****Ai****\n");
        show_dmatrix (Ai,0,10,0,10);
        printf("****Aistar****\n");
        show_dmatrix (Aistar,0,10,0,10);
        printf("****Bistar****\n");
        show_dmatrix (Bistar,0,10,0,7);
        printf ("****Gi****\n");
        show_dmatrix (Gi,0,10,0,6);
}
*/

/* Set up reduced matrices */
/* Aisr */
dmtranspose (C,8,11,tm01);                  /* C' = tm01              */
dmmult (Aistar,11,11,tm01,11,8,tm02);       /* Aistar * C' = tm02     */
dmmult (C,8,11,tm02,11,8,Aisr);             /* C * Aistar * C = Aisr  */

/* Bisr */
dmmult (C,8,11,Bistar,11,8,Bisr);           /* C * Bistar = Bisr      */

/* Gir */
dmmult (C,8,11,Gi,11,7,Gir);                /* C * Gi = Gir           */

/* Prediction in identifier */
/* nobs_pred=Aisr*nobs_tm1 + Bisr*Fbar */
dmvmult (Bisr,8,8,Fbar,8,*tm01);            /* Bisr*Fbar=tm01                           */
dmvmult (Aisr,8,8,nobs_tm1,8,*tm02);        /* Aisr*nobs_tm1=tm02                       */
dvadd (*tm01,8,*tm02,nobs_pred);            /* Aisr*nobs_tm1 + Bisr*Fbar=nobs_pred      */

/* Patch in nAS "observation" */
nobs[ONAS]      = nobs_pred[ONAS];
xfbar[NASI] = nobs_pred[ONAS];

/* Kid=M*Gir'*(Gir*M*Gir'+R)^(-1) */
dmtranspose (Gir,8,7,tm01);                 /* Gir'=tm01                                */
dmmult (M,7,7,tm01,7,8,tm02);               /* M*Gir'=tm02                              */
dmmult (Gir,8,7,tm02,7,8,tm01);             /* Gir*M*Gir'=tm01                          */
dmadd (tm01,8,8,R,tm02);                    /* Gir*M*Gir'+R=tm02                        */
dinverse (tm02,8,tm01);                     /* (Gir*M*Gir'+R)^(-1)=tm01                 */
dmtranspose (Gir,8,7,tm02);                 /* Gir'=tm02                                */
```

```
dmmult (M,7,7,tm02,7,8,tm03);               /* M*Gir'=tm03                       */
dmmult (tm03,7,8,tm01,8,8,Kid);             /* M*Gir'*(Gir*M*Gir'+R)^(-1)=Kid    */

/* xfbar=xfbar+Kid*(nobs-nobs_pred) */
dvsub (nobs,8,nobs_pred,*tm01);             /* nobs-nobs_pred=tm01               */
dmvmult (Kid,7,8,*tm01,8,*tm02);            /* Kid*(nobs-nobs_pred)=tm02         */
dvadd (xfbar,7,*tm02,xfbar);                /* xfbar+Kid*(nobs-nobs_pred)=xfbar  */

/* M=(Iid-Kid*Gir)*M+Q */
dmmult (Kid,7,8,Gir,8,7,tm01);              /* Kid*Gir=tm01                      */
dmsub (Iid,7,7,tm01,tm02);                  /* Iid-Kid*Gir=tm02                  */
dmmult (tm02,7,7,M,7,7,tm01);               /* (Iid-Kid*Gir)*M                   */
dmadd (tm01,7,7,Q,M);                       /* (Iid-Kid*Gir)*M+Q=M               */

/* clip */
for (k = 0; k < ROWIDENT; k++)
{
        if(xfbar[k]<xfbarmin[k])
                xfbar[k]=xfbarmin[k];
        else
                if (xfbar[k]>xfbarmax[k])
                        xfbar[k]=xfbarmax[k];
}

if ( thisDebug ) {
/*      printf("****nobs****\n");
        show_dvector (nobs,0,7);
        printf("****nobs_pred****\n");
        show_dvector (nobs_pred,0,7);
        printf("****Kid****\n");
        show_dmatrix (Kid,0,6,0,7);
*/      printf("****xfbar****\n");
        show_dvector (xfbar,0,6);
/*      printf("****M****\n");
        show_dmatrix (M,0,6,0,6);
*/
}

dvcopy (nobs,8,nobs_tm1);

/* free up allocated local identifier memory */
free_dvector (Fbar,0,7);
free_dvector (flowread,0,7);
free_dvector (flowread1,0,7);
free_dvector (flowread2,0,7);
free_dmatrix (Kayi,0,10,0,10);
free_dmatrix (gi,0,10,0,10);
free_dmatrix (Ai,0,10,0,10);
free_dmatrix (Bi,0,10,0,7);
free_dmatrix (Gi,0,10,0,6);
free_dmatrix (Aistar,0,10,0,10);
free_dmatrix (Bistar,0,10,0,7);
free_dmatrix (Aisr,0,7,0,7);
free_dmatrix (Bisr,0,7,0,7);
free_dmatrix (tm01,0,10,0,10);
free_dmatrix (tm02,0,10,0,10);
free_dmatrix (tm03,0,10,0,10);
free_dmatrix (tm04,0,10,0,10);

return (0);
}       /* end of MainIdentLoop */
```

```
/**************************************************************************************/
/*      UpdateIdentData:        Parameters updated by the Identifier are   */
/*      written back to the DataBase.                                       */
/*      Type:                   External                                    */
/*      Return Codes:           0 if no errors                              */
/*                              1 if there were errors                      */
/**************************************************************************************/

int UpdateIdentData (void)
{

if (thisDebug) printf("**** UpdateIdentData ****\n");

/* write xfbar to the DataBase */
SingleWrite ( (rtUInt8 *) xfbar,32);

/* write nobs back to the DataBase */
SingleWrite ( (rtUInt8 *) nobs,49);

/* write nobs_tm1 back to the DataBase */
SingleWrite ( (rtUInt8 *) nobs_tm1,67);

/* write nobs_pred to the DataBase */
SingleWrite ( (rtUInt8 *) nobs_pred,50);

/* write Kid to the DataBase */
SingleWrite ( (rtUInt8 *) *Kid,34);

/* write M to the DataBase */
SingleWrite ( (rtUInt8 *) *M,35);

return (0);
}       /* end of UpdateIdentData */


/*******************************************************************************/
/*      IdentPlotData:   Creates a history table for recording the      */
/*      results of the identifier, 'xfbar' in the DataBase.             */
/*      When the maximum lines are exceeded, values in the               */
/*      table are shifted up one place.  Presently not used             */
/*      for trending.                                                    */
/* Type:            Local                                                */
/* Return Codes:  0 if no errors                                         */
/*                          1 if there were errors                       */
/*******************************************************************************/

int IdentPlotData (void)
{
xfbarHistxfbarhist, xfbarshift;
static int xfbarIndex = 0;
struct timeval      now;
int                 shiftIndex;

if (thisDebug) printf ("**** IdentPlotData ****\n");
now.tv_sec = time(NULL);

/* generate xfbar historical data */
SingleRead ( (rtUInt8 *) &xfbarhist, 68);

xfbarhist.timestamp         = now;
xfbarhist.kP2i      = xfbar[KP2];
xfbarhist.kP4i      = xfbar[KP4];
xfbarhist.kP6i      = xfbar[KP6];
xfbarhist.kHi       = xfbar[KH];
xfbarhist.gHi       = xfbar[GH];
xfbarhist.kdi       = xfbar[KD];
xfbarhist.nASi      = xfbar[NASI];

if (xfbarIndex < MAXIDHIST) {
        ChangePtRecord (68,xfbarIndex);
        SingleWrite ( (rtUInt8 *) &xfbarhist, 68);
        xfbarIndex++;
}
else {
        for (shiftIndex = 1; shiftIndex < MAXIDHIST; shiftIndex++) {
                ChangePtRecord (68,shiftIndex);
                SingleRead ( (rtUInt8 *) &xfbarshift, 68);
                ChangePtRecord (68,shiftIndex-1);
                SingleWrite ( (rtUInt8 *) &xfbarshift, 68);
        }
        ChangePtRecord (68,MAXIDHIST-1);
        SingleWrite ( (rtUInt8 *) &xfbarhist, 68);

}
return (0);
} /* end of IdentPlotData */

/****************************************************************************/
/*      MainKalmanLoop: The main Kalman filter algorithm.  Executed */
/*      every time step but only if the reactor is up               */
/*      (returncodes 1,3,5,7).  Calculates state estimates          */
/*      in the form of mole fractions (nbar) from measurable        */
/*      states nhat.                                                */
/*      Type:           External                                    */
/*      Return Codes:   0 if no errors                              */
/*                      1 if there were errors                      */
/****************************************************************************/

int MainKalmanLoop (void)
{
static firstFilt = 1;
static firstLoop = 1;
int             k;                      /* counter for clipping nbar */

rtDouble        Ngkf;
rtDouble Npkf;
rtUInt16 DTIndex3;
bufferTable     DTValues3;
rtDouble        *Fbarkf;
rtDouble        **Kay;
rtDouble        **g;
rtDouble        **A;
rtDouble        Fv;
rtDouble        Fp;

rtDouble        kP2;
rtDouble        kP4;
rtDouble        kP6;
rtDouble        kH;
rtDouble        gH;
rtDouble        kd;
rtDouble        nAS;

rtDouble P;
rtDouble T;
rtDouble Vp;
rtDouble Vg;
rtDouble Ng;
rtDouble mMMp;
rtDouble Np;
rtDouble *np;
rtDouble *ng;
rtDouble Bed_Mass;
rtDouble **Astar;
rtDouble **Bstar;

/* Temporary storage arrays (used by both identifier and filter) */
rtDouble **tm01;                  /* temporary storage matrix 1 - used in matrix computations */
rtDouble **tm02;                  /* temporary storage matrix 2 - used in matrix computations */
rtDouble **tm03;                  /* temporary storage matrix 3 - used in matrix computations */
rtDouble **tm04;                  /* temporary storage matrix 4 - used in matrix computations */

if (thisDebug) printf("**** MainKalmanLoop ****\n");

/* allocate memory for our local KF data */
Fbarkf          = dvector (0,7);
np              = dvector (0,2);
ng              = dvector (0,4);
Kay             = dmatrix (0,10,0,10);
g               = dmatrix (0,10,0,10);
A               = dmatrix (0,10,0,10);
tm01            = dmatrix (0,10,0,10);
tm02            = dmatrix (0,10,0,10);
tm03            = dmatrix (0,10,0,10);
tm04            = dmatrix (0,10,0,10);
Astar           = dmatrix (0,10,0,10);
Bstar           = sdmatrix (11,8);


/* kinetic constants initialisation (until identifier gives better estimates) */
kP2     = xfTyp.kP2i_typical;            /* propagation pseudo-rate constant for ethylene       */
/* kP4 DEPENDS ON GRADE - SEE BELOW /* propagation pseudo-rate constant for butene         */
/* kP6 DEPENDS ON GRADE - SEE BELOW /* propagation pseudo-rate constant for hexene         */
kH      = xfTyp.kHi_typical;             /* transfer rate constant for reaction with H2         */
gH      = xfTyp.gHi_typical;             /* hydrogen balance mismatch factor                    */
kd      = xfTyp.kdi_typical;             /* spontaneous deactivation (kds)                      */
nAS     = xfTyp.nAS_typical;             /* activated sites                                     */

if (HexeneMode ()) {
        kP4 = xfTyp.kP4i_typical_hex;
        kP6 = xfTyp.kP6i_typical_hex;
        }
else {
        kP4 = xfTyp.kP4i_typical_but;
        kP6 = xfTyp.kP6i_typical_but;
}

/* KALMAN FILTER IS RUN ON EVERY STEP,USING TEMPORARY KINETIC DATA AT THE
BEGINNING */
Bed_Mass                = DataLine.BEDWEIGHT;           /* kg polymer */
Vp                      = Bed_Mass / fp.ro;
Vg                      = fp.Vreactor - Vp;
T                       = DataLine.REACTEMP;            /* temperature (0C) */
P                       = DataLine.R1PRESSURE;          /* pressure[bar abs]=2070kPag */
Ng                      = P * Vg / (fp.Rgasconst * (T + 273.15));

if (firstFilt) {
  nbar[NC2] = 0.01 * DataLine.ETH * Ng;
  nbar[NC4] = 0.01 * DataLine.BUT * Ng;
  nbar[NC6] = 0.01 * DataLine.HEX * Ng;
  nbar[NH2] = 0.01 * DataLine.HYD * Ng;
  nbar[NN2] = 0.01 * DataLine.NIT * Ng;
  nbar[NAS] = xfTyp.nAS_typical;

  if (HexeneMode ())
    dvcopy (x_typical_hex,3,x_typical);
  else
    dvcopy (x_typical_but,3,x_typical);
```

```
/* mMMp = MW * x_typical */
  mMMp = MW[0]*x_typical[0] + MW[1]*x_typical[1] + MW[2]*x_typical[2];

  Np = Bed_Mass / mMMp;              /* moles of gas in polymer */

/*
   if (thisDebug) printf ("Bed_Mass: %f\tVp: %f\tVg: %f\tT: %f\tP: %f\tNg: %f\tmMMp: %f\tNp: %f\n",
          Bed_Mass,Vp,Vg,T,P,Ng,mMMp,Np);
*/

  nbar[NP2]        = x_typical[0] * Np; /* Polymer inventories as C2 */
  nbar[NP4]        = x_typical[1] * Np; /* Polymer inventories as C4 */
  nbar[NP6]        = x_typical[2] * Np; /* Polymer inventories as C6 */
  nbar[NG]         = Ng;
  nbar[NP]         = Np;

  firstFilt = 0;

} else {
  SingleRead ( (rtUInt8 *) nbar,63);
  x_typical[0] = nbar[NP2]/(nbar[NP2]+nbar[NP4]+nbar[NP6]);
  x_typical[1] = nbar[NP4]/(nbar[NP2]+nbar[NP4]+nbar[NP6]);
  x_typical[2] = nbar[NP6]/(nbar[NP2]+nbar[NP4]+nbar[NP6]);
  mMMp = MW[0]*x_typical[0] + MW[1]*x_typical[1] + MW[2]*x_typical[2];
  Np = Bed_Mass/mMMp;
/*
  if (thisDebug) printf ("Bed_Mass: %f\tVp: %f\tVg: %f\tT: %f\tP: %f\tNg: %f\tmMMp: %f\tNp: %f\n",
          Bed_Mass,Vp,Vg,T,P,Ng,mMMp,Np);
*/
}

Npkf = nbar[NP];
Ngkf = nbar[NG];

bc.pointergap_GCdeadtime = (int) fp.GCdeadtime/(fp.dt);

if ( (int)(bc.buff_pointer - bc.pointergap_GCdeadtime) <  0)
  DTIndex3 = bc.Nbuffer + (bc.buff_pointer - bc.pointergap_GCdeadtime);
else
  DTIndex3 = (bc.buff_pointer - bc.pointergap_GCdeadtime);

if (thisDebug) printf ("DTIndex3 is %d\n",DTIndex3);

/* read the data from the buffer for index DTIndex3 */
ChangePtRecord (60,DTIndex3);
SingleRead ( (rtUInt8 *) &DTValues3,60);


Fbarkf [FC2] = DTValues3.ETHFLO;
Fbarkf [FC4] = DTValues3.BUTFLO;
Fbarkf [FC6] = DTValues3.HEXFLO;
Fbarkf [FH2] = DTValues3.H2FLO;
Fbarkf [FN2] = DTValues3.N2FLO;
Fbarkf [FAS] = DTValues3.CATFDRTOT;
Fbarkf [FV]  = DTValues3.R1VENTFLO;
Fbarkf [FP]  = DTValues3.PRODRATE/mMMp;
Fv = Fbarkf[FV];
Fp = Fbarkf[FP];


/* set to identified values if identifier has run */
if (firstIdent != 1) {
   kP2 = xfbar[KP2];
   kP4 = xfbar[KP4];
   kP6 = xfbar[KP6];
   kH = xfbar[KH];
   gH = xfbar[GH];
  kd = xfbar[KD];
  nAS = xfbar[NASI];
}
zero_dmatrix (Kay,0,10,0,10);
Kay[0][0]=-kP2;
Kay[1][1]=-kP4;
Kay[2][2]=-kP6;
Kay[3][3]=-kH;
Kay[6][0]=kP2;
Kay[7][1]=kP4;
Kay[8][2]=kP6;
Kay[9][0]=-kP2;
Kay[9][1]=-kP4;
Kay[9][2]=-kP6;
Kay[9][3]=-kH;
Kay[10][0]=kP2;
Kay[10][1]=kP4;
Kay[10][2]=kP6;

zero_dmatrix (g,0,10,0,10);
g[3][3] = -gH;
g[5][5] = -kd;
g[9][3] = -gH;

/* set up h (depends on the grade) */
if (HexeneMode()) {
  h[2][2] = -fp.s6;
  h[1][1] = 0.0;
  h[9][2] = -fp.s6;
  h[9][1] = 0.0;
} else {
  h[2][2] = 0.0;
  h[1][1] = -fp.s4;
  h[9][2] = 0.0;
  h[9][1] = -fp.s4;
}

/* Fp*h = tm01*/
dmsmy (h,11,11,Fp,tm01);
/*  H + HAS = tm02*/
dmadd (H,11,11,HAS,tm02);
/* (Fp/Npkf) * (H+HAS) = tm03*/
dmsmy (tm02,11,11,(Fp/Npkf),tm03);
/* (Fp/Npkf) * (H+HAS) + Fp*h = tm02 */
dmadd (tm03,11,11,tm01,tm02);
/* (Fv/Ngkf) * G = tm01 */
dmsmy (G,11,11,(Fv/Ngkf),tm01);
/* (Fv/Ngkf) * G + g = tm03 */
dmadd (tm01,11,11,g,tm03);
/* (Fv/Ngkf) * G + g + (Fp/Npkf) * (H+HAS) + Fp*h = tm01 */
dmadd (tm03,11,11,tm02,tm01);
/* nAS*Kay = tm02 */
dmsmy (Kay,11,11,nAS,tm02);
/* nAS*Kay + (Fv/Ngkf) * G + g + (Fp/Npkf) * (H+HAS) + Fp*h = A */
dmadd (tm01,11,11,tm02,A);

PartMatExpKF (A,B,Astar,Bstar,11,8);


/* Show vectors and matrices for debug */
/*
if (thisDebug) {
            printf ("****Kay****\n");
            show_dmatrix (Kay,0,10,0,10);
            printf ("****g****\n");
            show_dmatrix (g,0,10,0,10);
            printf ("****A****\n");
```

```
          show_dmatrix (A,0,10,0,10);
          printf ("****Fbarkf****\n");
          show_dvector (Fbarkf,0,7);
          printf ("****Astar****\n");
          show_dmatrix (Astar,0,10,0,10);
          printf ("****Bstar****\n");
          show_dmatrix (Bstar,0,10,0,7);
}
*/

/* Load nhat (measurements) from DataLine for the Kalman Filter */
nhat[HNC2] = 0.01 * DataLine.ETH * Ng;
nhat[HNC4] = 0.01 * DataLine.BUT * Ng;
nhat[HNC6] = 0.01 * DataLine.HEX * Ng;
nhat[HNH2] = 0.01 * DataLine.HYD * Ng;
nhat[HNN2] = 0.01 * DataLine.NIT * Ng;
nhat[HNG]  = Ng;
nhat[HNP]  = Np;

/*  Kalman Filter Algorithm */
/* Kkf=Mkf*Gkf'*(Gkf*Mkf*Gkf'+Rkf)^(-1) */
dmtranspose (Gkf,7,11,tm01);                /* Gkf'=tm01                                   */
dmmult (Mkf,11,11,tm01,11,7,tm02);          /* Mkf*Gkf'=tm02                               */
dmmult (Gkf,7,11,tm02,11,7,tm01);           /* Gkf*Mkf*Gkf'=tm01                           */
dmadd (tm01,7,7,Rkf,tm02);                  /* Gkf*Mkf*Gkf'+Rkf=tm02                       */
dinverse (tm02,7,tm01);                     /* (Gkf*Mkf*Gkf'+Rkf)^(-1)=tm01                */
dmtranspose (Gkf,7,11,tm02);                /* Gkf'=tm02                                   */
dmmult (tm02,11,7,tm01,7,7,tm03);           /* Gkf'*(Gkf*Mkf*Gkf'+Rkf)^(-1)=tm03           */
dmmult (Mkf,11,11,tm03,11,7,Kkf);           /* Mkf*Gkf'*(Gkf*Mkf*Gkf'+Rkf)^(-1)=Kkf        */

/* KFerror=nhat-Gkf*nbar */
dmvmult (Gkf,7,11,nbar,11,*tm01);           /* Gkf*nbar=tm01                               */
dvsub (nhat,7,*tm01,KFerror);               /* nhat-Gkf*nbar=KFerror                       */

/* nbar=Astar*nbar+Bstar*Fbarkf+Kkf*KFerror */
dmvmult (Kkf,11,7,KFerror,7,*tm01);         /* Kkf*KFerror=tm01                            */
dmvmult (Bstar,11,8,Fbarkf,8,*tm02);        /* Bstar*Fbarkf=tm02                           */
dvadd (*tm01,11,*tm02,*tm03);               /* Bstar*Fbarkf+Kkf*KFerror=tm03               */
dmvmult (Astar,11,11,nbar,11,*tm01);        /* Astar*nbar=tm01                             */
dvadd (*tm01,11,*tm03,nbar);                /* Astar*nbar+Bstar*Fbarkf+Kkf*KFerror=nbar    */

/* Mkf=Astar*(Ikf-Kkf*Gkf)*Mkf*Astar'+Qkf */
dmmult (Kkf,11,7,Gkf,7,11,tm01);            /* Kkf*Gkf=tm01                                */
dmsub (Ikf,11,11,tm01,tm02);                /* Ikf-Kkf*Gkf=tm02                            */
dmmult (tm02,11,11,Mkf,11,11,tm01);         /* (Ikf-Kkf*Gkf)*Mkf=tm01                      */
dmtranspose (Astar,11,11,tm02);             /* Astar'=tm02                                 */
dmmult (tm01,11,11,tm02,11,11,tm03);        /* (Ikf-Kkf*Gkf)*Mkf*Astar'=tm03               */
dmmult (Astar,11,11,tm03,11,11,tm01);       /* Astar*(Ikf-Kkf*Gkf)*Mkf*Astar'=tm01         */
dmadd (tm01,11,11,Qkf,Mkf);                 /* Astar*(Ikf-Kkf*Gkf)*Mkf*Astar'+Qkf=Mkf      */

/* clip */
for (k = 0; k <NKF; k++)
{
          if (nbar[k]<nbarmin[k])
                    nbar[k]=nbarmin[k];
          else
                    if (nbar[k]>nbarmax[k])
                              nbar[k]=nbarmax[k];
}

if (thisDebug) {
/*        printf ("****Kkf****\n");
          show_dmatrix (Kkf,0,10,0,6);
          printf ("****KFerror****\n");
          show_dvector (KFerror,0,6);
*/        printf ("****nhat****\n");
          show_dvector (nhat,0,6);
```

```
        printf ("****nbar****\n");
        show_dvector (nbar,0,10);
/*      printf ("****Mkf****\n");
        show_dmatrix (Mkf,0,10,0,10);
*/
}

/* free up allocated local KF memory */
free_dvector (Fbarkf,0,7);
free_dvector (np,0,2);
free_dvector (ng,0,4);
free_dmatrix (Kay,0,10,0,10);
free_dmatrix (g,0,10,0,10);
free_dmatrix (A,0,10,0,10);
free_dmatrix (tm01,0,10,0,10);
free_dmatrix (tm02,0,10,0,10);
free_dmatrix (tm03,0,10,0,10);
free_dmatrix (tm04,0,10,0,10);
free_dmatrix (Astar,0,10,0,10);
free_dmatrix (Bstar,0,10,0,7);

return (0);
}       /* end of MainKalmanLoop */

/***************************************************************************************/
/*      UpdateKalmanData:       Parameters estimated by the filter are        */
/*      written back to the DataBase.                                         */
/*      Type:                   External                                      */
/*      Return Codes:           0 if no errors                                */
/*                              1 if there were errors                        */
/***************************************************************************************/

int UpdateKalmanData (void)
{
/* update the BufferControl point */
SingleWrite ( (rtUInt8 *) &bc,59);

/* write nhat to the database */
SingleWrite ( (rtUInt8 *) nhat,64);

/* write nbar to the DataBase */
SingleWrite ( (rtUInt8 *) nbar,63);

/* write h to the DataBase */
SingleWrite ( (rtUInt8 *) *h,41);

/* write Mkf to the DataBase */
SingleWrite ( (rtUInt8 *) *Mkf,56);

/* write Kkf to the DataBase */
SingleWrite ( (rtUInt8 *) *Kkf,65);

/* write KFerror to the DataBase */
SingleWrite ( (rtUInt8 *) KFerror,66);

return (0);
}       /* end of UpdateKalmanData */
```

```
/*******************************************************************************/
/*      KalmanPlotData: Creates a history table for recording the results      */
/*      of the filter, 'nbar' and 'KFerror' as well                            */
/*      as 'nhat' in the DataBase. When the maximum lines                      */
/*      are exceeded, values in the table are shifted                          */
/*      up one place. Presently not used for trending.                         */
/* Type:            Local                                                      */
/* Return Codes:  0 if no errors                                               */
/*                          1 if there were errors                             */
/*******************************************************************************/

int KalmanPlotData (void)
{
nhatHist nhathist, nhatshift, KFerrorhist, KFerrorshift;
nbarHist nbarhist, nbarshift;
static int nhatIndex = 0;
static int nbarIndex = 0;
static int KFerrorIndex = 0;
struct timeval       now;
int                  shiftIndex;

if (thisDebug) printf ("**** KalmanPlotData ****\n");
now.tv_sec = time(NULL);

/* generate nhat historical data       */
SingleRead ( (rtUInt8 *) &nhathist, 69);

nhathist.timestamp           = now;
nhathist.nC2                 = nhat[HNC2];
nhathist.nC4                 = nhat[HNC4];
nhathist.nC6                 = nhat[HNC6];
nhathist.nH2                 = nhat[HNH2];
nhathist.nN2                 = nhat[HNN2];
nhathist.Ng                  = nhat[HNG];
nhathist.Np                  = nhat[HNP];

if (nhatIndex < MAXKFHIST) {
        ChangePtRecord (69,nhatIndex);
        SingleWrite ( (rtUInt8 *) &nhathist, 69);
        nhatIndex++;
}
else {
        for (shiftIndex = 1; shiftIndex < MAXKFHIST; shiftIndex++) {
                ChangePtRecord (69,shiftIndex);
                SingleRead ( (rtUInt8 *) &nhatshift, 69);
                ChangePtRecord (69,shiftIndex-1);
                SingleWrite ( (rtUInt8 *) &nhatshift, 69);
        }
        ChangePtRecord (69,MAXKFHIST-1);
        SingleWrite ( (rtUInt8 *) &nhathist, 69);
}

/* generate nbar historical data       */
SingleRead ( (rtUInt8 *) &nbarhist, 70);

nbarhist.timestamp           = now;
nbarhist.nC2                 = nbar[NC2];
nbarhist.nC4                 = nbar[NC4];
nbarhist.nC6                 = nbar[NC6];
nbarhist.nH2                 = nbar[NH2];
nbarhist.nN2                 = nbar[NN2];
nbarhist.nAS                 = nbar[NAS];
nbarhist.nP2                 = nbar[NP2];
nbarhist.nP4                 = nbar[NP4];
nbarhist.nP6                 = nbar[NP6];
nbarhist.Ng                  = nbar[NG];
nbarhist.Np                  = nbar[NP];
```

```
if (nbarIndex < MAXKFHIST) {
        ChangePtRecord (70,nbarIndex);
        SingleWrite ( (rtUInt8 *) &nbarhist, 70);
        nbarIndex++;
}
else {
        for (shiftIndex = 1; shiftIndex < MAXKFHIST; shiftIndex++) {
                ChangePtRecord (70,shiftIndex);
                SingleRead ( (rtUInt8 *) &nbarshift, 70);
                ChangePtRecord (70,shiftIndex-1);
                SingleWrite ( (rtUInt8 *) &nbarshift, 70);
        }
        ChangePtRecord (70,MAXKFHIST-1);
        SingleWrite ( (rtUInt8 *) &nbarhist, 70);
}

/* generate KFerror historical data  */
SingleRead ( (rtUInt8 *) &KFerrorhist, 71);

KFerrorhist.timestamp      = now;
KFerrorhist.nC2  = KFerror[HNC2];
KFerrorhist.nC4  = KFerror[HNC4];
KFerrorhist.nC6  = KFerror[HNC6];
KFerrorhist.nH2  = KFerror[HNH2];
KFerrorhist.nN2  = KFerror[HNN2];
KFerrorhist.Ng             = KFerror[HNG];
KFerrorhist.Np             = KFerror[HNP];

if (KFerrorIndex < MAXKFHIST) {
        ChangePtRecord (71,KFerrorIndex);
        SingleWrite ( (rtUInt8 *) &KFerrorhist, 71);
        KFerrorIndex++;
}
else {
        for (shiftIndex = 1; shiftIndex < MAXKFHIST; shiftIndex++) {
                ChangePtRecord (71,shiftIndex);
                SingleRead ( (rtUInt8 *) &KFerrorshift, 71);
                ChangePtRecord (71,shiftIndex-1);
                SingleWrite ( (rtUInt8 *) &KFerrorshift, 71);
        }
        ChangePtRecord (71,MAXKFHIST-1);
        SingleWrite ( (rtUInt8 *) &KFerrorhist, 71);
}

return (0);
}        /* end of KalmanPlotData */



/******************************************************************************/
/*      PartMatExpKF:   Gets the discrete form of A and B (A* and B*)          */
/*      for the Kalman filter by using a matrix exponential.                   */
/*      This is in effect a zero-order hold (ZOH) that allows                  */
/*      samples to be taken at discrete time intervals                         */
/*      'dt'.  Uses a truncated Taylor series of e^t to avoid                  */
/*      singularity problems.                                                  */
/*      Type:           Local                                                  */
/*      Return Codes:   0 if no errors                                         */
/*      1 if there were errors (series gets too long and won't truncate)       */
/******************************************************************************/

int PartMatExpKF (double **a_mat,double **b_mat,double **Astar,double **Bstar,int rowB,
  int colB)         /* scan interval          */

{
double **expmAdt_IdivA;
double **changemat;
double **Adt;
double **tmKF;
double change = 10000.0;
int k = 1;
int j;
int i;                                   /* loop counter */

/* allocate memory for PartMatExpKF function */
expmAdt_IdivA    = dmatrix (0,rowB-1,0,rowB-1);
changemat                  = dmatrix (0,rowB-1,0,rowB-1);
Adt                                  = dmatrix (0,rowB-1,0,rowB-1);
tmKF                       = dmatrix (0,rowB-1,0,rowB-1);

/* dt * Ikf = expmAdt_IdivA */
dmsmy (Ikf,rowB,rowB,fp.dt,expmAdt_IdivA);

/* dt * Ikf = changemat */
dmsmy (Ikf,rowB,rowB,fp.dt,changemat);

/* A * dt */
dmsmy (a_mat,rowB,rowB,fp.dt,Adt);

while ( (change > TOL) && (k<=MAXEXPIT) ) {
  k = k+1;

/* tmKF = (changemat * Adt) */
  dmmult (changemat,rowB,rowB,Adt,rowB,rowB,tmKF);
/* changemat = (changemat * Adt)* 1/k */
  dmsmy (tmKF,rowB,rowB,(1.0/(double)k),changemat);
  change = 0.0;
  for (i = 0;i<rowB;i++)
    for (j = 0;j<rowB;j++)
      change = change + fabs (changemat[i][j]);

  dmadd (expmAdt_IdivA,rowB,rowB,changemat,expmAdt_IdivA);
}

/* Astar = expmAdt_IdivA * A + Ikf */
dmmult (expmAdt_IdivA,rowB,rowB,a_mat,rowB,rowB,tmKF);
dmadd (tmKF,rowB,rowB,Ikf,Astar);

/* Bstar = expmAdt_IdivA  * B */
dmmult (expmAdt_IdivA,rowB,rowB,b_mat,rowB,rowB,Bstar);

/* free up allocated memory for PartMatExpKF function */
free_dmatrix (expmAdt_IdivA,0,rowB-1,0,rowB-1);
free_dmatrix (changemat,0,rowB-1,0,rowB-1);
free_dmatrix (Adt,0,rowB-1,0,rowB-1);
free_dmatrix (tmKF,0,rowB-1,0,rowB-1);

if (k>MAXEXPIT)
  return (1);

return (0);

}          /* end of PartMatExpKF */
```

```
/*****************************************************************************************/
/*        PartMatExpIdent:           Gets the discrete form of A and B (A* and B*)        */
/*        for the Identifier by using a matrix exponential.                               */
/*        This is in effect a zero-order hold (ZOH) that allows                           */
/*        samples to be taken at discrete time intervals 'dtident'.                       */
/*        Uses a truncated Taylor series of e^t to avoid singularity                      */
/*        problems.                                                                       */
/* Type:          Local                                                                   */
/* Return Codes:  0 if no errors                                                          */
/*                          1 if there were errors (series gets too long and won't truncate)  */
/*****************************************************************************************/

int PartMatExpIdent (double **ai_mat,double **bi_mat,double **Aistar,double **Bistar,int rowB,
  int colB)

{
double **expmAidtident_IdivAi;
double **changemat;
double **Aidtident;
double **tmID;
double change = 10000.0;
int k = 1;
int j;
int i;                                   /* loop counter */

/* allocate memory for PartMatExpIdent function */
expmAidtident_IdivAi       = dmatrix (0,rowB-1,0,rowB-1);
changemat                  = dmatrix (0,rowB-1,0,rowB-1);
Aidtident             = dmatrix (0,rowB-1,0,rowB-1);
tmID                       = dmatrix (0,rowB-1,0,rowB-1);

/* dtident * Ii = expmAidtident_IdivAi */
dmsmy (Ii,rowB,rowB,bc.dtident,expmAidtident_IdivAi);

/* dtident * Ii = changemat */
dmsmy (Ii,rowB,rowB,bc.dtident,changemat);

/* Ai * dtident */
dmsmy (ai_mat,rowB,rowB,bc.dtident,Aidtident);

while ( (change > TOL) && (k<=MAXEXPIT) ) {
  k = k+1;

/* tmID = (changemat * Aidtident) */
  dmmult (changemat,rowB,rowB,Aidtident,rowB,rowB,tmID);
/* changemat = (changemat * Aidtident)* 1/k */
  dmsmy (tmID,rowB,rowB,(1.0/(double)k),changemat);
  change = 0.0;
  for (i = 0;i<rowB;i++)
    for (j = 0;j<rowB;j++)
      change = change + fabs (changemat[i][j]);

  dmadd (expmAidtident_IdivAi,rowB,rowB,changemat,expmAidtident_IdivAi);
}

/* Aistar = expmAidtident_IdivAi * Ai + Ii */
dmmult (expmAidtident_IdivAi,rowB,rowB,ai_mat,rowB,rowB,tmID);
dmadd (tmID,rowB,rowB,Ii,Aistar);

/* Bistar = expmAidtident_IdivAi  * Bi */
dmmult (expmAidtident_IdivAi,rowB,rowB,bi_mat,rowB,rowB,Bistar);

/* free up allocated memory for PartMatExpIdent function */
free_dmatrix (expmAidtident_IdivAi,0,rowB-1,0,rowB-1);
free_dmatrix (changemat,0,rowB-1,0,rowB-1);
free_dmatrix (Aidtident,0,rowB-1,0,rowB-1);
free_dmatrix (tmID,0,rowB-1,0,rowB-1);
if (k>MAXEXPIT)
  return (1);

return (0);

}          /* end of PartMatExpIdent */
```

KalmanControl.h

```
/*****************************************************************/
/* sources/KalmanControl.h 2000/03/06 N Narotam   */
/*                                                */
/*  Copyright (c) N Narotam 1999, 2000            */
/*  All Rights Reserved                           */
/*                                                */
/* DESCRIPTION                                    */
/*   Header file - contains general structure and */
/*   function definitions that are needed by all the */
/*   other modules                                */
/*****************************************************************/
/* COMPILITATION CONTROL                          */
/* 2000/03/30  N Narotam  Ver 1.01 conceived      */
/*                                                */
/*****************************************************************/

#include <rtap/database.h>


/*        calculation constants */
#define DT 20.0
#define MINPRODRATE 1.4
#define MAXEXPIT 100
#define TOL 1.0e-20         /* tolerance for convergence */

/*        returncode constants        */
#define REACTORBIT 1
#define NSTOREDBIT 2
#define GCUPDATEBIT 4

/***********************************************************/
/*        define the size of the data array      */
/***********************************************************/
#define NALL 11
#define MALL 8
#define ROWIDENT 7
#define COLIDENT 8
#define NKF     11
#define MKF     8

/*****************************************************************/
/*        define the order of the data in the Identifier  */
/*****************************************************************/

/* define the order of the data in the identification vector */
#define KP2 0
#define KP4 1
#define KP6 2
#define KH 3
#define GH 4
#define KD 5
#define NASI 6

/* The input vector for the Identifier */
#define FC2 0
#define FC4 1
#define FC6 2
#define FH2 3
#define FN2 4
#define FAS 5
#define FV  6
#define FP  7

/* vector of flow readings corresponding to the last GC update  */
```

```
#define FR1FC2  0
#define FR1FC4  1
#define FR1FC6  2
#define FR1FH2  3
#define FR1FN2  4
#define FR1FAS  5
#define FR1FV   6
#define FR1FP   7

/* vector of flow readings corresponding to the current GC update */
#define FR2FC2  0
#define FR2FC4  1
#define FR2FC6  2
#define FR2FH2  3
#define FR2FN2  4
#define FR2FAS  5
#define FR2FV   6
#define FR2FP   7

/* vector of average readings of flowread1 and flowread2 */
#define FRFC2   0
#define FRFC4   1
#define FRFC6   2
#define FRFH2   3
#define FRFN2   4
#define FRFAS   5
#define FRFV    6
#define FRFP    7

/* nobs (measurable states) */
#define ONC2    0
#define ONC4    1
#define ONC6    2
#define ONH2    3
#define ONN2    4
#define ONAS    5
#define ONG     6
#define ONP     7

/* nobs_tm1 (measurable states) */
#define OM1NC2 0
#define OM1NC4 1
#define OM1NC6 2
#define OM1NH2 3
#define OM1NN2 4
#define OM1NAS 5
#define OM1NG  6
#define OM1NP  7

/*************************************************************/
/* define the order of the vectors for the Kalman Filter */
/*************************************************************/

/* nbar (Predicted states) */
#define NC2 0
#define NC4 1
#define NC6 2
#define NH2 3
#define NN2 4
#define NAS 5
#define NP2 6
#define NP4 7
#define NP6 8
#define NG  9
#define NP  10

/* nhat (measurable states) */
```

```
#define HNC2 0
#define HNC4 1
#define HNC6 2
#define HNH2 3
#define HNN2 4
#define HNG  5
#define HNP  6

/* np (Plastic vector) */
#define NNP2 0
#define NNP4 1
#define NNP6 2

/* ng (Gas vector) */
#define NNC2 0
#define NNC4 1
#define NNC6 2
#define NNH2 3
#define NNN2 4

/* y_typical (Typical gas composition vector) */
#define YC2 0
#define YC4 1
#define YC6 2
#define YH2 3
#define YN2 4

/**********************************************/
/* define the history stack sizes           */
/**********************************************/
#define MAXIDHIST 360
#define MAXKFHIST 360

/*  application status structure          */
typedef struct {
            rtBytes8 env;
            rtInt16    Debug;
            rtInt16  OnLine;
} AppStatus;

extern int OnLine;

typedef struct {
   rtFloat dt;                 /* basic sampling time Interval (20 sec.)                                */
   rtFloat GCdeadtime;         /* time gap between sample entering GC and outputs being updated        */
   rtFloat dt_forced_ident;    /* if GC update not found within this time, force an update             */
   rtFloat tolerance_conc_change;    /* for detection of GC update                                     */
   rtFloat Rgasconst;                /* universal gas constant [bar.m3 / kmol.K]                       */
   rtFloat ro;                       /* polymer density{kg/m3polymer}                                  */
   rtFloat Vreactor;                 /* volume of reactor (276m3) + cycle gas loop (48m3)              */
   rtFloat s2;                       /* moles C2 lost from gas per total moles C2+C4+C6 in product */
   rtFloat s4;                       /* none present in this case  (dissolved)                         */
   rtFloat s6;                       /* moles C6 lost from gas per total moles C2+C4+C6 in product */
} FixedParam;

typedef struct {
   rtDouble kP2i_typical;
   rtDouble kP4i_typical_but;
   rtDouble kP4i_typical_hex;
   rtDouble kP6i_typical_but;
   rtDouble kP6i_typical_hex;
   rtDouble kHi_typical;
   rtDouble gHi_typical;
   rtDouble kdi_typical;
   rtDouble nAS_typical;
}  TypicalData;
```

```
typedef struct {
  rtUInt32 buff_pointer;
  rtUInt32 pointerlastupdate;
  rtUInt32 Nstored;
  rtUInt32 Nstored_criterion;
  rtUInt32 Nbuffer;
  rtUInt32 changecount_criterion;
  rtUInt32 changecount;
  rtUInt32 dtident;
  rtUInt32 pointergap_GCdeadtime;
} BufferControl;

typedef struct {
  rtDouble ETHFLO;
  rtDouble BUTFLO;
  rtDouble HEXFLO;
  rtDouble H2FLO;
  rtDouble N2FLO;
  rtDouble CATFDRTOT;
  rtDouble R1VENTFLO;
  rtDouble PRODRATE;
  rtDouble REACTEMP;
  rtDouble R1PRESSURE;
  rtDouble ETH;
  rtDouble BUT;
  rtDouble HEX;
  rtDouble HYD;
  rtDouble NIT;
  rtDouble ETHANE;
  rtDouble ISO;
  rtDouble C4INERTS;
  rtDouble C6INERTS;
  rtDouble BEDWEIGHT;
  rtDouble SPARE_1;
  rtDouble SPARE_2;
  rtDouble SPARE_3;
} bufferTable;

typedef struct {
  struct timeval timestamp;
  rtDouble kP2i;
  rtDouble kP4i;
  rtDouble kP6i;
  rtDouble kHi;
  rtDouble gHi;
  rtDouble kdi;
  rtDouble nASi;
} xfbarHist;

typedef struct {
  struct timeval timestamp;
  rtDouble nC2;
  rtDouble nC4;
  rtDouble nC6;
  rtDouble nH2;
  rtDouble nN2;
  rtDouble Ng;
  rtDouble Np;
} nhatHist;

typedef struct {
  struct timeval timestamp;
  rtDouble nC2;
  rtDouble nC4;
  rtDouble nC6;
  rtDouble nH2;
  rtDouble nN2;
  rtDouble nAS;
  rtDouble nP2;
  rtDouble nP4;
  rtDouble nP6;
  rtDouble Ng;
  rtDouble Np;
} nbarHist;
```

nrutil.c

```
/*******************************************************************/
/*      nrutil.c file for use with KalmanControl                  */
/*      N Narotam, C van der Merwe - Polifin                      */
/*      R Thomason, M Mulholland - University of Natal            */
/*      version 1 March 2000                                      */
/*******************************************************************/

#include <malloc.h>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "nrutil.h"

void nrerror(char error_text[])
{


        fprintf(stderr,"Numerical Recipes run-time error...\n");
        fprintf(stderr,"%s\n",error_text);
        fprintf(stderr,"...now exiting to system...\n");
        exit(1);
}

float *vector(int nl,int nh)
{
        float *v;

        v=(float *)malloc((unsigned) (nh-nl+1)*sizeof(float));
        if (!v) nrerror("allocation failure in vector()");
        return v-nl;
}

int *ivector(int nl,int nh)
{
        int *v;

        v=(int *)malloc((unsigned) (nh-nl+1)*sizeof(int));
        if (!v) nrerror("allocation failure in ivector()");
        return v-nl;
}

double *dvector(int nl,int nh)
{
        double *v;

        v=(double *)malloc((unsigned) (nh-nl+1)*sizeof(double));
        if (!v) nrerror("allocation failure in dvector()");
        return v-nl;
}

float **matrix(int nrl,int nrh,int ncl,int nch)
{
        int i;
        float **m;

        m=(float **) malloc((unsigned) (nrh-nrl+1)*sizeof(float*));
        if (!m) nrerror("allocation failure 1 in matrix()");
        m -= nrl;

        for(i=nrl;i<=nrh;i++) {
                m[i]=(float *) malloc((unsigned) (nch-ncl+1)*sizeof(float));
                if (!m[i]) nrerror("allocation failure 2 in matrix()");
                m[i] -= ncl;
        }
        return m;
}

float **smatrix(int m, int n)
{
int i;
float **a;
float *p;

p = malloc(m*n*sizeof(float));
a = malloc(m*sizeof(float*));

for (i=0;i<m;i++) {
  a[i] = p + (i*n);
}

return a;
}

double **sdmatrix(int m, int n)
{
int i;
double **a;
double *p;

p = malloc(m*n*sizeof(double));
a = malloc(m*sizeof(double*));

for (i=0;i<m;i++) {
  a[i] = p + (i*n);
}

return a;
}

void zero_dmatrix (double **matr,int nrl,int nrh,int ncl,int nch)
{
int i,j;
for (i=nrl;i<=nrh;i++)
  for (j=ncl;j<=nch;j++)
    matr[i][j] = 0.0;

}

void zero_dvector (double *vec,int nrl,int nrh)
{
int i;
for (i=nrl;i<=nrh;i++)
    vec[i] = 0.0;
}

void zero_matrix (float **matr,int nrl,int nrh,int ncl,int nch)
{
int i,j;
for (i=nrl;i<=nrh;i++)
  for (j=ncl;j<=nch;j++)
    matr[i][j] = 0.0;
}

double **dmatrix(int nrl,int nrh,int ncl,int nch)

{
        int i;
        double **m;

        m=(double **) malloc((unsigned) (nrh-nrl+1)*sizeof(double*));
        if (!m) nrerror("allocation failure 1 in dmatrix()");
        m -= nrl;

        for(i=nrl;i<=nrh;i++) {
                m[i]=(double *) malloc((unsigned) (nch-ncl+1)*sizeof(double));
                if (!m[i]) nrerror("allocation failure 2 in dmatrix()");
                m[i] -= ncl;
        }
        return m;
}

int **imatrix(int nrl,int nrh,int ncl,int nch)

{
        int i,**m;

        m=(int **)malloc((unsigned) (nrh-nrl+1)*sizeof(int*));
        if (!m) nrerror("allocation failure 1 in imatrix()");
        m -= nrl;

        for(i=nrl;i<=nrh;i++) {
                m[i]=(int *)malloc((unsigned) (nch-ncl+1)*sizeof(int));
                if (!m[i]) nrerror("allocation failure 2 in imatrix()");
                m[i] -= ncl;
        }
        return m;
}

float **submatrix(float **a,int oldrl,int oldrh,int oldcl,int oldch,int newrl,int newcl)

{
        int i,j;
        float **m;

        m=(float **) malloc((unsigned) (oldrh-oldrl+1)*sizeof(float*));
        if (!m) nrerror("allocation failure in submatrix()");
        m -= newrl;

        for(i=oldrl,j=newrl;i<=oldrh;i++,j++) m[j]=a[i]+oldcl-newcl;

        return m;
}

void free_vector(float *v,int nl,int nh)

{
        free((char*) (v+nl));
}

void free_ivector(int *v,int nl,int nh)

{
        free((char*) (v+nl));
}

void free_dvector(double *v,int nl,int nh)

{
        free((char*) (v+nl));
}

void free_matrix(float **m,int nrl,int nrh,int ncl,int nch)

{
        int i;

        for(i=nrh;i>=nrl;i–) free((char*) (m[i]+ncl));
        free((char*) (m+nrl));
```

```
}

void free_dmatrix(double **m,int nrl,int nrh,int ncl,int nch)

{
        int i;

        for(i=nrh;i>=nrl;i--) free((char*) (m[i]+ncl));
        free((char*) (m+nrl));
}

void free_imatrix(int **m,int nrl,int nrh,int ncl,int nch)

{
        int i;

        for(i=nrh;i>=nrl;i--) free((char*) (m[i]+ncl));
        free((char*) (m+nrl));
}

void free_submatrix(float **b,int nrl,int nrh,int ncl,int nch)

{
        free((char*) (b+nrl));
}

float **convert_matrix(float *a,int nrl,int nrh,int ncl,int nch)

{
        int i,j,nrow,ncol;
        float **m;

        nrow=nrh-nrl+1;
        ncol=nch-ncl+1;
        m = (float **) malloc((unsigned) (nrow)*sizeof(float*));
        if (!m) nrerror("allocation failure in convert_matrix()");
        m -= nrl;
        for(i=0,j=nrl;i<=nrow-1;i++,j++) m[j]=a+ncol*i-ncl;
        return m;
}

void free_convert_matrix(float **b,int nrl,int nrh,int ncl,int nch)

{
        free((char*) (b+nrl));
}

void show_ivector (int *vect,int nl,int nh)
{
int i;
for (i=nl;i<=nh;i++)
        printf(" %d\n",vect[i]);
}

void show_dvector (double *vect,int nl,int nh)
{
int i;
for (i=nl;i<=nh;i++)
        printf(" %e\n",vect[i]);
}

void show_vector (float *vect,int nl,int nh)
{
int i;
for (i=nl;i<=nh;i++)
        printf(" %f\n",vect[i]);
}
```

```
void show_matrix (float **matr,int nrl,int nrh,int ncl,int nch)
{
int i,j;
for (i=nrl;i<=nrh;i++)
          {
          for (j=ncl;j<=nch;j++)
                    printf("%5.4f ",matr[i][j]);
          printf ("\n");
          }

}

void show_dmatrix (double **matr,int nrl,int nrh,int ncl,int nch)
{
int i,j;
for (i=nrl;i<=nrh;i++)
          {
          for (j=ncl;j<=nch;j++)
                    printf("%e ",matr[i][j]);
          printf ("\n");
          }
}

void show_imatrix (int **matr,int nrl,int nrh,int ncl,int nch)
{
int i,j;
for (i=nrl;i<=nrh;i++)
          {
          for (j=ncl;j<=nch;j++)
                    printf("%d ",matr[i][j]);
          printf ("\n");
          }

}

void dmcopy( double **a, int a_rows, int a_cols, double **b) /* copy a matrix */
{
int i, j;
for ( i=0; i<a_rows; i++ )
    for ( j=0; j<a_cols; j++ ) b[i][j] = a[i][j];
}

void dvcopy (double *a,int a_els,double *y)    /* copy a vector */
{
int i;
for (i=0;i<a_els;i++)
  y[i] = a[i];

}

void dmmult (        double **a, int a_rows, int a_cols,
                     double **b, int b_rows, int b_cols, double **y)
/* multiply two matrices a,b result in y. y must not be the same as a or b. */
{
int i,j,k;
double sum;

for (i=0; i<a_rows; i++)
  for (j=0; j<b_cols; j++){
    sum = 0.0;
    for (k=0; k<a_cols; k++) sum +=a[i][k]*b[k][j];
    y[i][j]=sum;
  }
}

void dmadd (double **a, int a_rows, int a_cols, double **b, double **y)
```

```
/* add two matrices a,b, result in y. y can be same as a or b */
{
int i,j;

for (i=0; i<a_rows; i++)
  for (j=0; j<a_cols; j++){
    y[i][j]=a[i][j]+b[i][j];
  }
}

void dmsmy ( double **a, int a_rows, int a_cols, double r, double **y)
/* multiply a by scalar r, result in y. y can be same as a */
{
int i,j;

for (i=0; i<a_rows; i++)
  for (j=0; j<a_cols; j++){
    y[i][j]=a[i][j]*r;
  }
}

void dmsub ( double **a, int a_rows, int a_cols, double **b, double **y)
/* subtract two matrices a,b, result in y. y can be same as a or b */
{
int i,j;

for (i=0; i<a_rows; i++)
  for(j=0; j<a_cols; j++){
    y[i][j]=a[i][j]-b[i][j];
  }
}

void dmtranspose ( double **a, int a_rows, int a_cols, double **y)
/* transpose a matrix a, result in y. y must not be same as a */
{
int i,j;

for (i=0; i<a_rows; i++)
  for (j=0; j<a_cols; j++){
    y[j][i]=a[i][j];
  }
}

void dmvmult( double **a, int a_rows, int a_cols, double *b, int b_els, double *y)
/* multiply a matrix a by vector b, result in y. y can be same as b */
{
  int i, k;
  double sum;

  for ( i=0; i<a_rows; i++ ) {
    sum = 0.0;
    for ( k=0; k<a_cols; k++ ) sum += a[i][k]*b[k];
    y[i] = sum;
  }
}

void dvadd( double *a, int a_els, double *b, double *y)
{
  int j;

  for ( j=0; j<a_els; j++ ) {
    y[j] = a[j] + b[j];
  }
}

void dvsub( double *a, int a_els, double *b, double *y)
{
  int j;

  for ( j=0; j<a_els; j++ ) {
    y[j] = a[j] - b[j];
  }
}

void dvsmy( double *a, int a_els, double r, double *y)
{
  int j;

  for ( j=0; j<a_els; j++ ) {
    y[j] = a[j] * r;
  }
}

void deyes ( int row, int col, double **I )
{
  int i,j;
  for (i=0;i<row;i++) {
          for (j=0;j<col;j++) {
            I[i][j]=0;
            if (i==j)
                    I[i][j]=1.0;
          }
  }
}

/* matrix inversion routines */

#define TINY 1.0e-20;

void dludcmp ( double **a, int n, int *indx, double *d)
{
        int i,imax,j,k;
        double big,dum,sum,temp;
        double *vv;

        vv = dvector (0,n-1);
        *d=1.0;
        for (i=0; i<n;i++) {
                big=0.0;
                for (j=0;j<n;j++)
                        if ((temp=fabs(a[i][j]))>big) big = temp;
                if (big == 0.0) nrerror ("Singular matrix in routine dludcmp");
                vv[i]=1.0/big;
        }
        for (j=0;j<n;j++) {
                for (i=0;i<j;i++) {
                        sum=a[i][j];
                        for (k=0;k<i;k++) sum -= a[i][k]*a[k][j];
                        a[i][j]=sum;
                }
                big=0.0;
                for (i=j;i<n;i++) {
                        sum=a[i][j];
                        for (k=0;k<j;k++) sum -= a[i][k]*a[k][j];
                        a[i][j]=sum;
                        if ((dum=vv[i]*fabs(sum))>=big) {
                                big=dum;
                                imax=i;
                        }
                }
                if (j!=imax) {
                        for (k=0;k<n;k++) {
                                dum=a[imax][k];
                                a[imax][k]=a[j][k];
                                a[j][k]=dum;
                        }
                        *d=-(*d);
                        vv[imax]=vv[j];
                }
                indx[j]=imax;
                if (a[j][j]==0.0) a[j][j]=TINY;
                if (j!=n-1) {
                        dum=1.0/(a[j][j]);
                        for ( i=j+1;i<n;i++) a[i][j] *= dum;
                }
        }
        free_dvector(vv,0,n-1);
}

void dlubksb ( double **a, int n, int *indx, double b[] )
{
        int i,ii=0,ip,j;
        double sum;

        for (i=0;i<n;i++) {
                ip=indx[i];
                sum=b[ip];
                b[ip]=b[i];
                if (ii)
                        for ( j=ii-1;j<=i-1;j++) sum -= a[i][j]*b[j];
                else if (sum) ii = i+1;
                b[i]=sum;
        }
        for (i=n-1;i>=0;i–) {
                sum=b[i];
                for (j=i+1;j<n;j++) sum -= a[i][j]*b[j];
                b[i]=sum/a[i][i];
        }
}

void dinverse( double **a, int n, double **y )
/* Find inverse of 'a' (decomposed in process!) and return as 'y' */
{
        double d, *col;
        int i, j, *indx /* integer vector */;

        indx = ivector( 0, n-1 );
        col = dvector( 0, n-1 );
        dludcmp( a, n, indx, &d);
        for ( j=0; j<n; j++ ) {
                for ( i=0; i<n; i++ ) col[i] = 0.0;
                col[j] = 1.0;
                dlubksb( a, n, indx, col );
                for ( i=0; i<n; i++ ) y[i][j] = col[i];
        }
        free_ivector( indx, 0, n-1 );
  free_dvector( col, 0, n-1 );
}
```