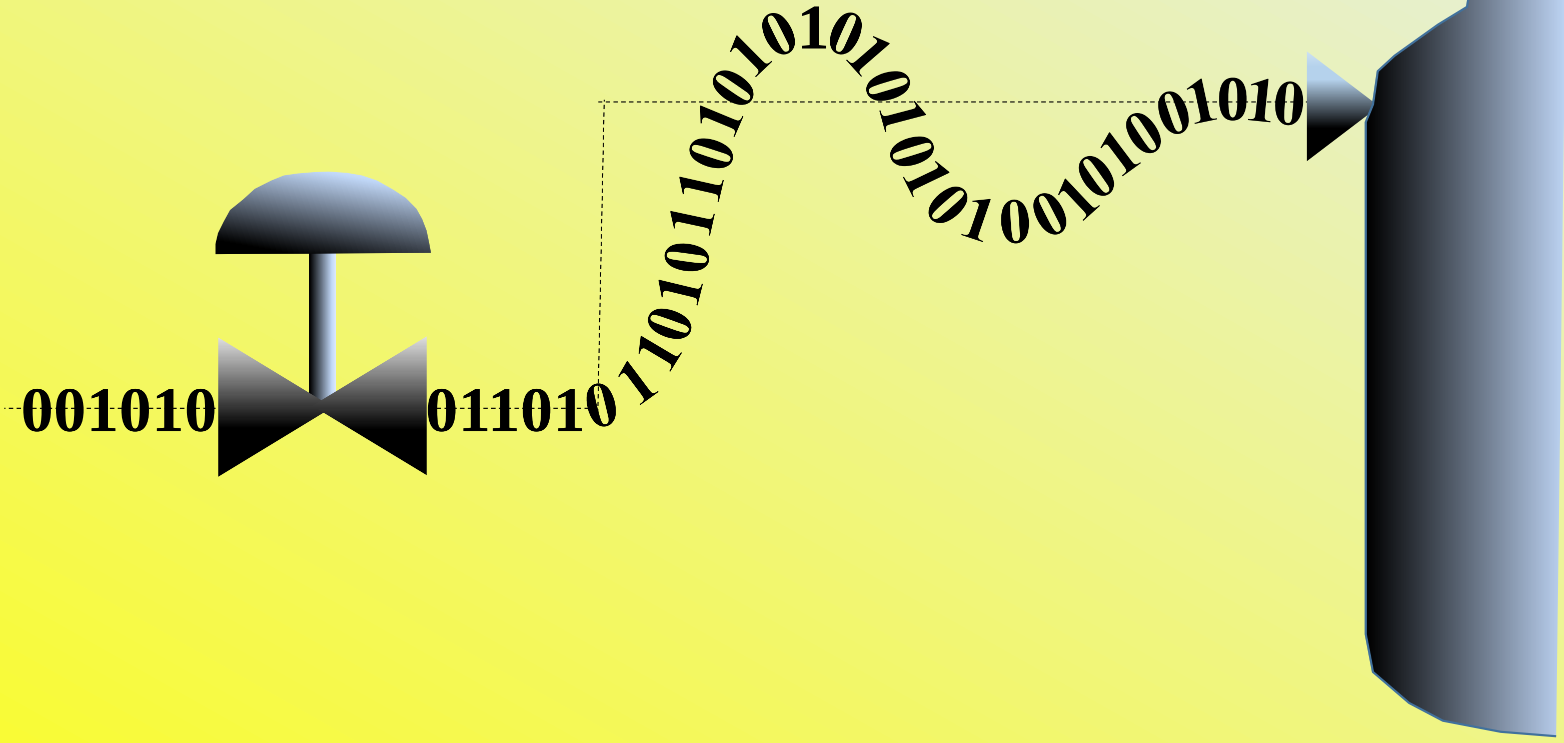
Applied Process Control

Simulations



Michael Mulholland

Applied
Process Control

Simulations

Michael Mulholland

Associated with the following books by Michael Mulholland:

Applied Process Control – Essential Methods
Applied Process Control – Efficient Problem Solving

Preface

A number of interactive simulations are provided which explore some of the techniques presented in *Applied Process Control – Essential Methods*, and *Applied Process Control – Efficient Problem Solving*. The applications arise variously from industrial research studies, undergraduate projects and student laboratory experiments.

This text is a step-by-step tutorial for the use of the simulator program, and the execution of the exercises for each application. It refers to the accompanying program *RTC* (Real-Time Control). No support can be offered for the use of the program and the conducting of the exercises. The correspondence between the simulation studies and sections of *Applied Process Control – Essential Methods*, is roughly as follows:

Chapter 2	Simulations – Openloop	*3.1, 3.2, 3.3, 3.4*
Chapter 3	Simulations – Frequency response	*4.2.6.2, 8.6*
Chapter 4	Simulations – SISO closedloop	*4.2, 4.2.6*
Chapter 5	Simulations – SISO optimisers	*4.2, 4.7, 4.10, 4.11*
Chapter 6	Simulations – Multi-loop strategies	*4.3-4.9, 5.3.1, 5.3.2, 5.3.3*
Chapter 7	Simulations – MIMO closedloop and DMC	*4.2.6, 7.8, 7.8.2, 8.6, 8.7*
Chapter 8	Simulations – Observers	*6.2, 6.4.1, 6.5.1*
Chapter 9	Simulations – Hybrid systems	*7.14.3*

Michael Mulholland
University of KwaZulu-Natal
October, 2015.

Contents

Abbreviations

A/D	analogue to digital (conversion)
ALC	advanced level control
BFW	boiler feed water
CV	controlled variable
CW	cooling water
D/A	digital to analogue (conversion)
DCS	distributed control system
DMC	dynamic matrix control
DP	differential pressure
DV	disturbance variable
EKF	extended Kalman filter
GM	gain margin
GUI	graphic user interface
HS	high select
I/P	current to pressure (pneumatic) converter
IMC	internal model control
I/O	input output
LDMC	linear dynamic matrix control
LLE	liquid-liquid extraction
LP	linear programming
LPSOLVE	MILP program (http://sourceforge.net/projects/lpsolve/)
LS	low select
MATLAB	MATLAB® program, distributed by the MathWorks, Inc.
MFR	mixed flow reactor
MIDO	mixed integer dynamic optimisation
MILP	mixed integer linear programming
MIMO	multi-input, multi-output
MINLP	mixed integer non-linear programming
MLD	mixed logical dynamical
MPC	model predictive control
MRT	mean residence time
MV	manipulated variable
P	proportional
P/I	pressure (pneumatic) to current converter
PI	proportional integral
PID	proportional integral derivative
PM	phase margin
PV	process variable
QDMC	quadratic dynamic matrix control

RTC	real time control (simulation program)
RTD	residence time distribution
SCADA	supervisory control and data acquisition
SD	standard deviation
SISO	single input single output
SP	setpoint
VPC	valve position control

Chapter 1 Simulations - Getting started

1.1 Background

This tutorial includes notes to be used in conjunction with the RTC (Real Time Control) program © M. Mulholland (2015). The notes give a brief background to the theory and programming of each application, plus a stepwise set of objectives to illustrate the main features. The simulations include a number of the techniques described in *Applied Process Control – Essential Methods,* and some of the tasks considered in *Applied Process Control – Efficient Problem Solving.* Actually, RTC is not merely a suite of simulations: Each application can be switched from "MODEL" to "PLANT" mode for real-time control of equipment through a somewhat basic interfacing arrangement. In "MODEL" mode, time can be accelerated or frozen. Continuous trend graphs and a data logging system allow examination and capture of the system behaviour.

1.2 Installing the software

The program is available at https://sourceforge.net/projects/rtc-simulator/ . On that page select FILES and download *RTC_exe.zip* and *Simulations.pdf*. Unpack *RTC_exe.zip* into your desired working diretory. Find RTCDEMO.exe and create a shortcut to it which you may place on your desktop. Tutorial instructions and background for each application are presented in *Simulations.pdf.* You may need to maximise your screen resolution to see the diagrams clearly and facilitate small enough fonts.

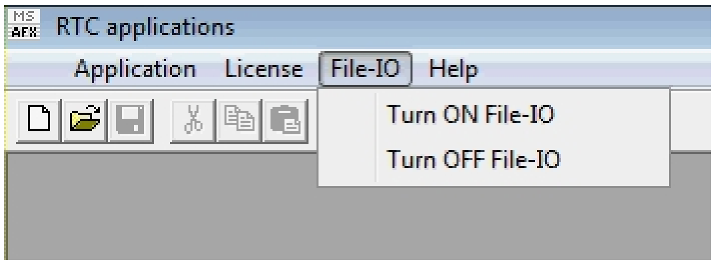
1.3 Model or Plant I/O

The RTC software has been used extensively in a university laboratory for monitoring and control of various pieces of plant equipment. In each application, a simulation model is provided of the equipment, and this can be substituted for the actual plant by selecting MODEL instead of PLANT. The *models* are of the linear "Dynamic Matrix" type, and are based on realistic step-responses.

However, the *plant* I/O has been set up specifically for the A/D, D/A equipment of a single vendor, *viz.* Eagle Technology, 24 Burg Street, Cape Town, 8001, South Africa (eagle@eagle.co.za , www.eagle.co.za , www.eagledaq.com). This includes a range of plug-in cards and external USB devices, which are found and sequenced by the software. The products of this company are available in many countries.

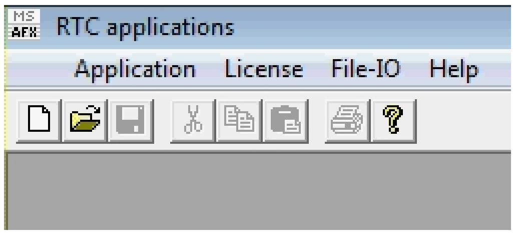
Bearing in mind this rather restricted I/O arrangement for a real plant, a somewhat crude alternative is provided in the software. This alternative relies on the typically slow responses of chemical plant equipment, allowing scan periods from 3 seconds upwards. On each “PLANT” I/O update, “reads” and “writes” can be directed to *FileInput.txt* and *FileOutput.txt* in the home directory of the software (rather than to connected hardware). These files are opened and closed on each read or write, and the read or write does not fail on (temporary) unavailability. So all that is required from the user is to simultaneously run a separate I/O scan loop of his or her own choice, and to provide the “read” information and execute the “write” information using independent writes and reads of the *FileInput.txt* and *FileOutput.txt* files. Clearly, the latter writes and reads need to open and close the files, and must also not cause software failure due to (temporary) unavailability – which should be infrequent given a big enough scan interval.

The default setting of the software, if “PLANT” is selected, is to assume *hardware* I/O for the equipment. To switch between the *hardware* and *file* I/O, press either toggle under “File-IO” before selecting the desired application (*f*_1.1).

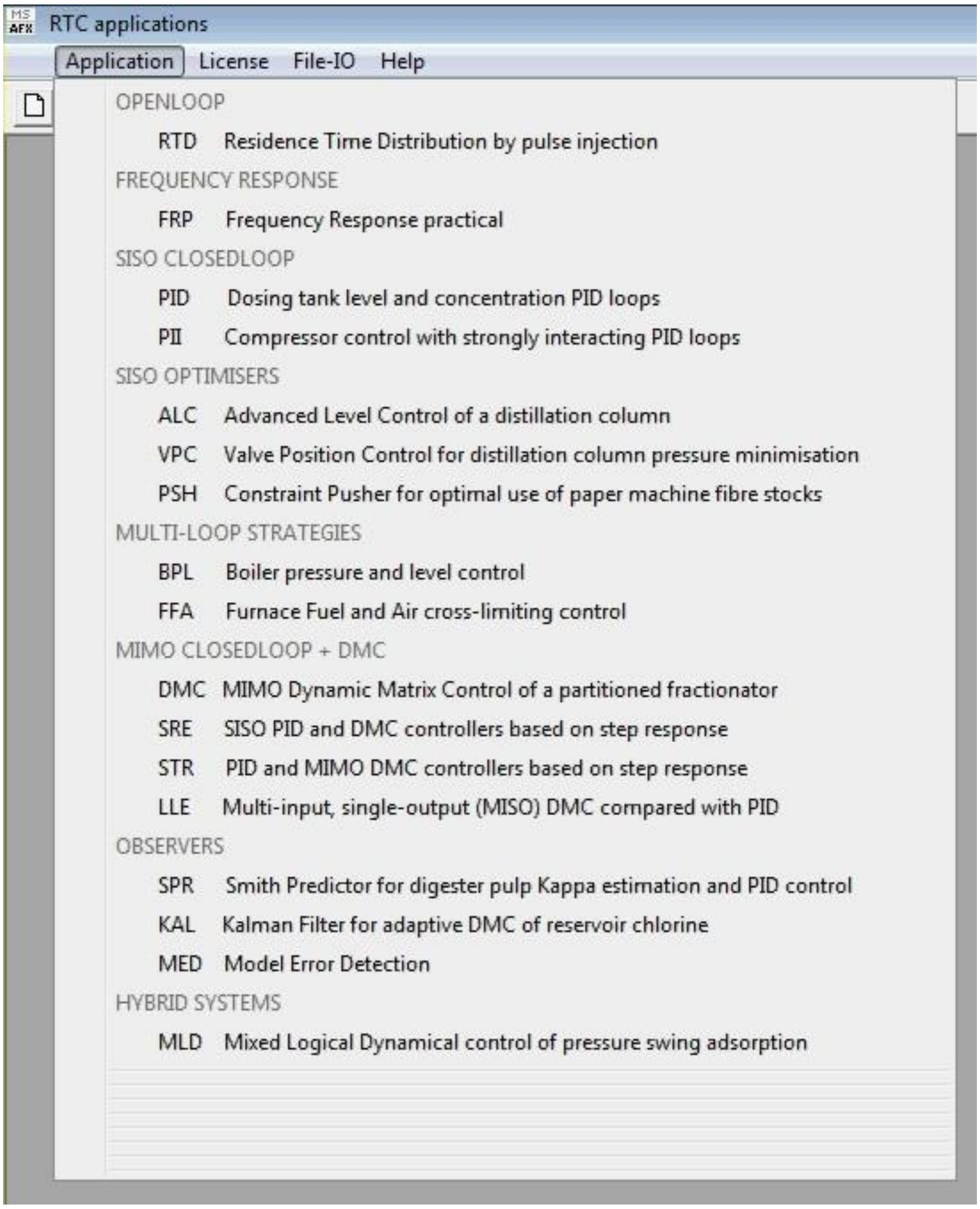
*f*_1.1


1.4 General features of the RTC interface

The RTC “Real Time Control” program is a general-purpose Modelling, Testing and Implementation platform, and it therefore has a hybrid of features. The layouts of the interface screens provided for the various applications (see “Contents”) have several common features (starting on *f*_1.2) which are explained below.

*f*_1.2


Application Choose from the listed applications (*f*_1.3).


RTC applications

Application License File-IO Help

OPENLOOP

RTD	Residence Time Distribution by pulse injection

FREQUENCY RESPONSE

FRP	Frequency Response practical

SISO CLOSEDLOOP

PID	Dosing tank level and concentration PID loops
PII	Compressor control with strongly interacting PID loops

SISO OPTIMISERS

ALC	Advanced Level Control of a distillation column
VPC	Valve Position Control for distillation column pressure minimisation
PSH	Constraint Pusher for optimal use of paper machine fibre stocks

MULTI-LOOP STRATEGIES

BPL	Boiler pressure and level control
FFA	Furnace Fuel and Air cross-limiting control

MIMO CLOSEDLOOP + DMC

DMC	MIMO Dynamic Matrix Control of a partitioned fractionator
SRE	SISO PID and DMC controllers based on step response
STR	PID and MIMO DMC controllers based on step response
LLE	Multi-input, single-output (MISO) DMC compared with PID

OBSERVERS

SPR	Smith Predictor for digester pulp Kappa estimation and PID control
KAL	Kalman Filter for adaptive DMC of reservoir chlorine
MED	Model Error Detection

HYBRID SYSTEMS

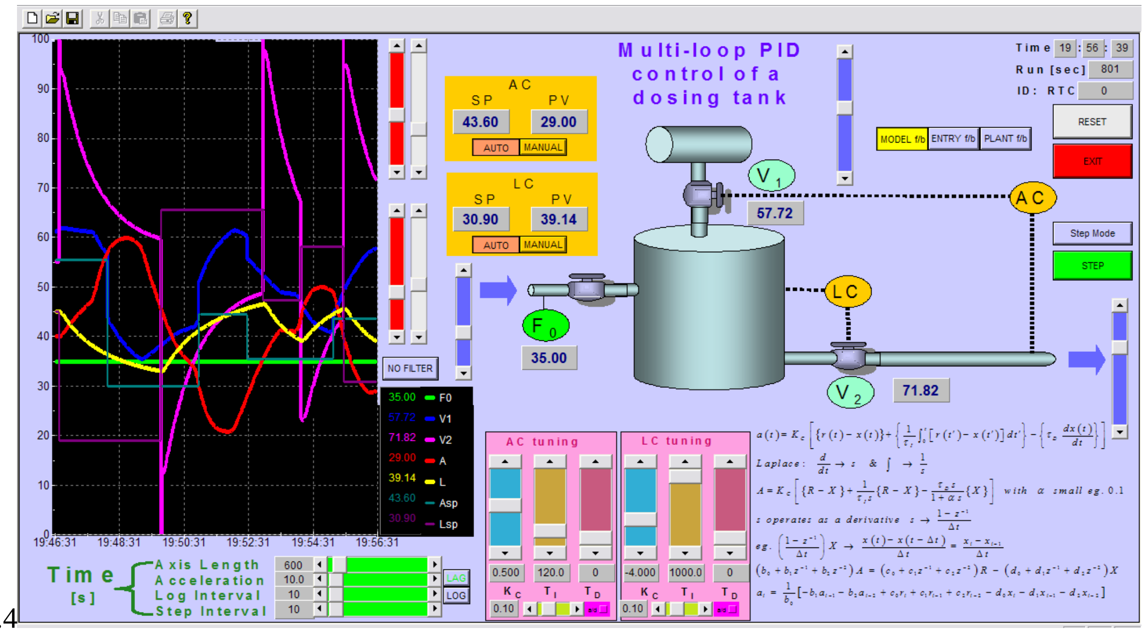
MLD	Mixed Logical Dynamical control of pressure swing adsorption

*f*_1.3

Help Information about origin of this program

License Information about origin of this program plus take careful note of the expiry date

Once an application in *f*_1.3 is selected (eg. PID – *f*_1.4), its interface screen has a number of features which are common for all applications.



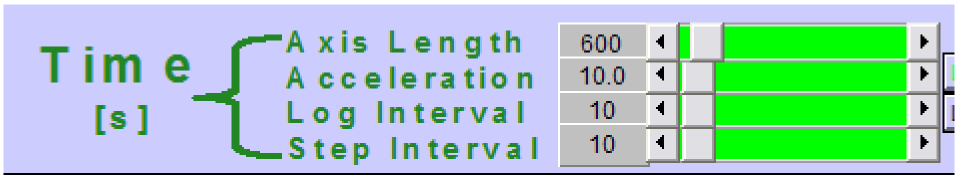
*f*_1.4

The scrollbar cursors can be moved in three ways:

* click, hold & move the cursor
* click the end-arrows
* click on the coloured region either side of the cursor.

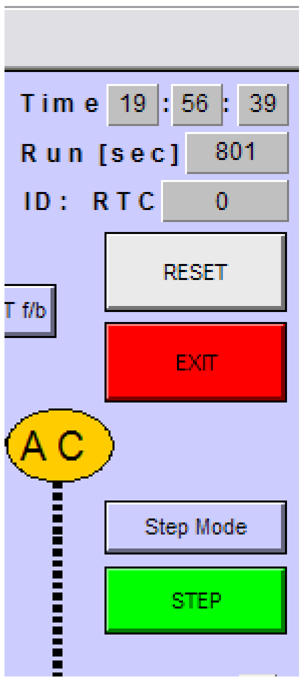
Each of these methods has a different sensitivity.

We select the more common buttons and controls shown on this application.



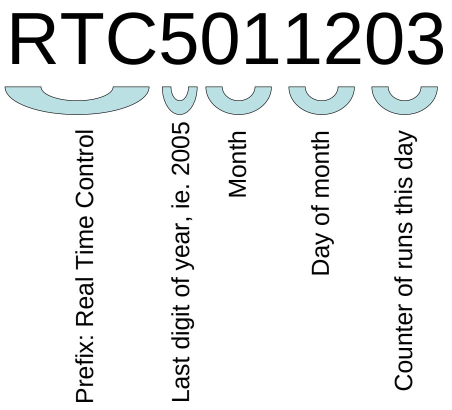
*f*_1.5

The time settings in *f*_1.5 are in seconds, except "Acceleration", which is the number of times faster than Real Time that the program will execute. If "Plant f/b" is selected (see *f*_1.8 below), ie. real-time plant feedback for online implementation, the "Acceleration" is automatically forced back to 1.0 . If the "Log" option is selected, records of all variables will be stored in the log file (see *f*_1.10 below) at the chosen log interval. If the "Step Mode" is selected (see *f*_1.6 below), the simulation will not run indefinitely, but only over the selected step interval. Once it stops, pressing the STEP button allows it to continue for another interval, etc. Depending on the chosen Acceleration, it may be desired to increase or reduce the time-axis length displayed. Note that the actual change only occurs four seconds after the <u>last</u> movement of the "Axis Length" cursor, to allow the user to make repeated finer adjustments of the cursor position.



*f*_1.6

Consider the run information at the top-right hand corner of the screen (*f*_1.6). The indicated time of day is the actual time of the solution, with all solutions starting at current real-time. Thus, if an acceleration is applied, this time will increasingly move ahead of real-time. The Run time elapsed is the time in seconds represented by the solution at any instant – again this will increase with an acceleration over unity. If the "Log" option is selected (see *f*_1.10 below), a Run Identity is allocated in the "ID" window. The ID numbers have the following structure (*f*_1.7):



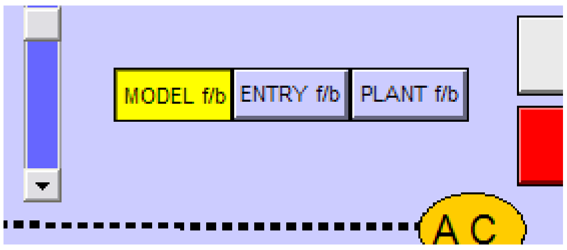
*f*_1.7

Each time "Log" is re-selected, or the program is "Reset" whilst logging, the previous log file is ended and closed, the counter of runs (last 2 digits) is incremented, and a new log file (eg. "RTC5011204.txt") is started. These are in the form of space-separated columns with the variable names as headings at the tops of the columns, and the model time and elapsed model Run seconds running down the leftmost columns.

The "Reset" button (*f*_1.6) effectively restarts the solution at the default values of the process variables. Certain settings such as tuning values or controller modes are not reset.

The "Exit" button (*f*_1.6) is used to stop the current application. This returns the user to the initial screen (*f*_1.2), where another application can be selected, or the program can be shut down.

The "Step Mode" (*f*_1.6) toggles this mode on or off. Once it is on, the "Step" button below it becomes active, allowing piece-wise solutions over intervals of length determined by the "Step Interval" setting (see above). This is useful for "freezing" the output, or for manual adjustment of manipulated or disturbance variable settings at a fixed interval. Another use is for manual entry of process variable feedback (see "Entry f/b" *f*_1.8 below), if the RTC program is being used to control a real plant or external model, which might not have an actual feedback signal connection.



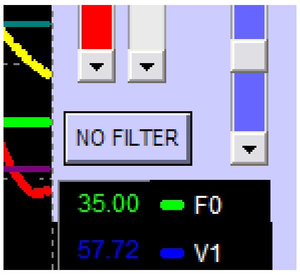
*f*_1.8

The set of three selectors "Model f/b", "Entry f/b" and "Plant f/b" (*f*_1.8) are mutually exclusive. For normal off-line testing of algorithms using model feedback, clearly the first of these, "Model f/b", is selected. As discussed above, it may be necessary to test a control algorithm on an actual plant or model (eg. "Aspen Dynamics") which is external to the program, but which does not offer a direct connection. In this case the "Entry f/b" feedback option is used, and the appropriate feedback variable scrollbars may be directly manipulated whilst the control algorithm runs. Beware that the "Acceleration" setting matches the external process in this case. Sometimes, as in DMC, controlled variable feedback is only reviewed by the algorithm on a fixed time interval. Then you will wish to select the "Step Mode" and set the appropriate "Step Interval". The RTC program will re-compute the manipulated variables for each interval, and these can be manually transferred to the actual plant or external model.

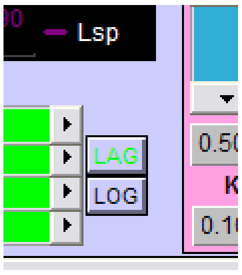
The "Plant f/b" option is for actual on-line data acquisition and control. When it is selected, the "Acceleration" is forced to unity for real-time work. In principle, driver software can be added to service any external A/D, D/A devices for plant input and output. So far, only the "Enhanced EDR" (EDRE) drivers have been implemented for the electronic interfaces supplied by Eagle Technology, Cape Town. See section 1.3, where it is explained that the default assumption for "PLANT f/b" is a hardware interface, yet a crude interchange of data by files can be automated. In each application, the manipulated variables (MV: output to plant) and the process variables (PV: input from plant) are expected to occur in a set order, counting from zero, eg. in the *f*_1.4 case above:

Input Channels:		Output Channels:	
Ch 0	A	Ch 0	V1
Ch 1	L	Ch 1	V2

In the case of the Eagle I/O boards, if more than one board is present, the channel numbers just continue the count. For more than one Eagle board to be recognised, the program must start in Execute mode ("!"), not Go (debug) mode ("↓") – ie. if the RTC program is being run under the Microsoft® Visual C++ environment.

*f*_1.9 

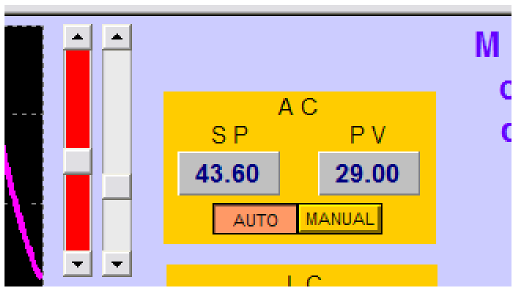
In connection with the "Plant f/b" external data acquisition feature, measurements are subjected to filter smoothing by default. However, if the raw inputs are desired, eg. for trouble-shooting, the "No Filter" button should be toggled to the on position (*f*_1.9).

*f*_1.10 

At the right of the time-scale adjustments are the "Lag" indicator and the "Log" toggle button (*f*_1.10). If the "Acceleration" is set too high, the solution may begin to lag behind its target time. The "Lag" indicator then turns red to warn you to reduce the acceleration.

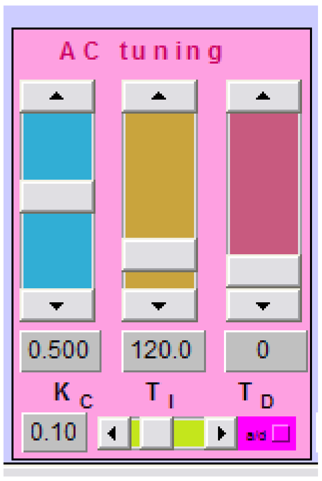
If the "Log" button is toggled on, it turns yellow to warn you that all process variables are being written to a log file at the selected log interval. As mentioned above, toggling the button off (or "Exit"-ing) will close and store the current file, and toggling it on again (or "Reset"-ing whilst logging) will start a new file. The files are named sequentially according to the Run "ID" described above (*f*_1.7), so it is advisable to note this ID number when a file is started. They are stored in the main directory from which the RTC program is executed.

Care must be taken not to generate uselessly large log files, which have the potential to crash the system by filling the hard drive. Note that the "Log Interval" refers to actual model time, so that an interval of 10 seconds would result in 10 records being stored every real-time second at an acceleration of 100. A small level of protection is afforded by automatically closing a log file once the 8640th record is written (ie. one day at 10 second intervals). This is implemented in case the program is left running and it is forgotten to toggle the "Log" button off. Care should be taken to purge all "RTC*.txt" files from the main program directory periodically.



*f*_1.11

A typical controller manipulation layout is shown in *f*_1.11 above, whether it is PID, DMC, ALC, etc. The "Auto" and "Manual" buttons are mutually exclusive. In "Manual" mode, the scrollbar of the relevant manipulated variable can be manually adjusted. In "Auto" mode, the set-point (SP), which is manually adjusted using the left-hand scroll-bar shown, is targeted by the algorithm, the relevant feedback PV being shown both numerically and by the right-hand scrollbar.



*f*_1.12

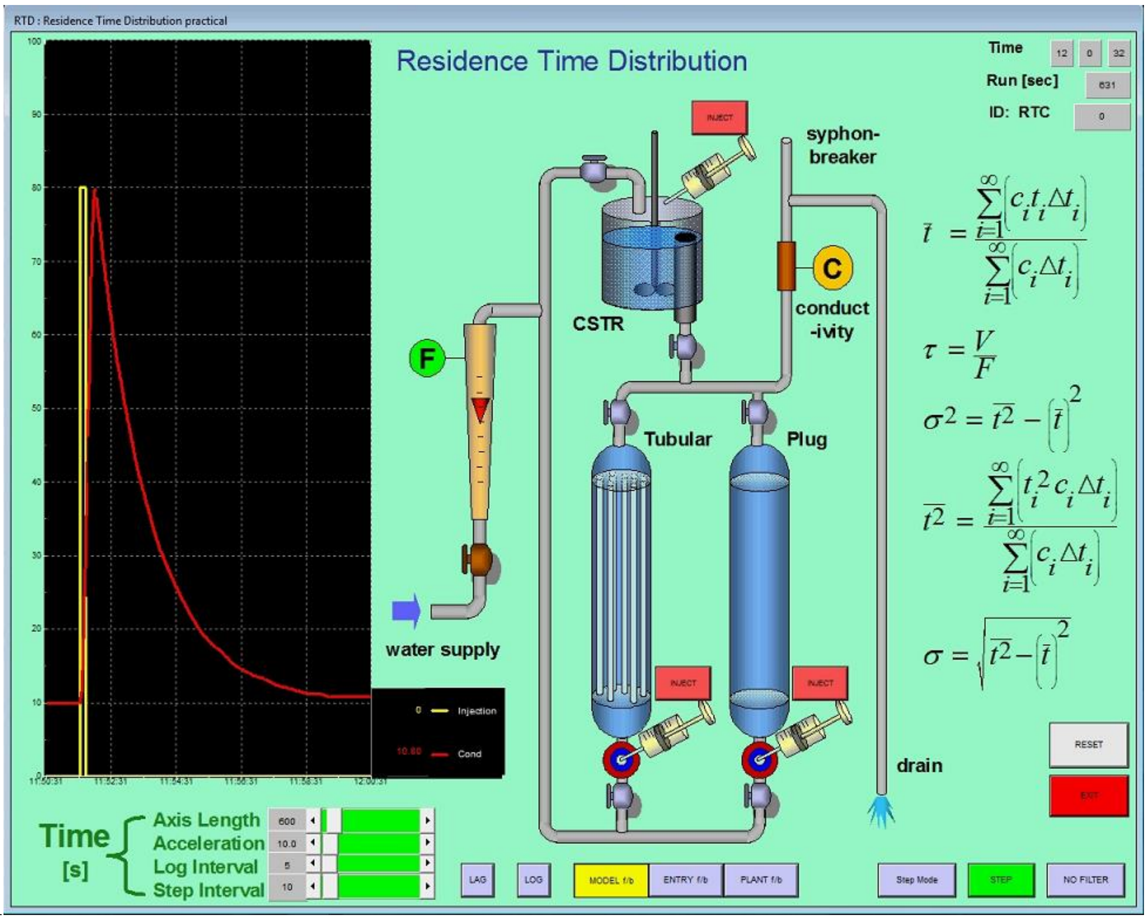
A tuning block for a PID controller is shown in *f*_1.12 above. The various parameters will be discussed in more detail in the next section. Beware that some K_C ranges have positive and/or negative parts. The recursive PID algorithm used has a parameter α for the approximation of the derivative term. It should normally be small (eg. 0.1) and may be set by the small horizontal scrollbar ("a") at the bottom. The "d" checkbox at the bottom right allows one to select that the derivative term uses just PV for checked "X" (as in common industrial controllers), or uses (PV-SP) for *not* checked, in which case the area is highlighted in pink.

Chapter 2 Simulations - Openloop

.

2.1 RTD – Residence Time Distribution by pulse injection

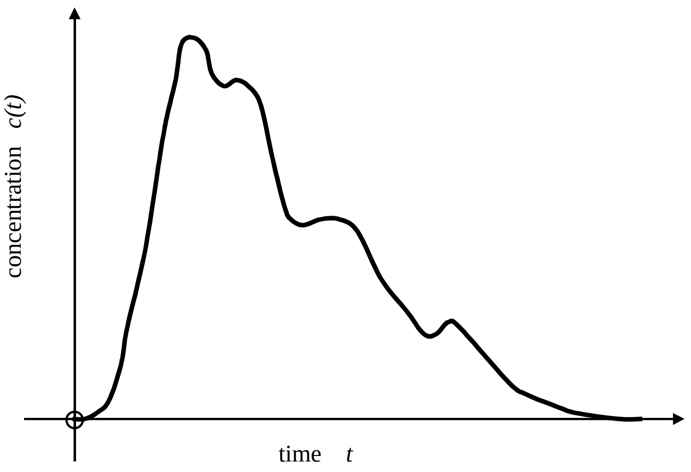
Examination of the RTD's of three different vessels by measurement of the exit concentration response to a pulse injection of tracer at the vessel entrance:


RTD : Residence Time Distribution practical
Residence Time Distribution
Time 12 0 32
Run [sec] 631
ID: RTC 0
INJECT
syphon-breaker
conduct-ivity
CSTR
Tubular
Plug
water supply
drain
F
C
0 Injection
10.80 Cond
$\bar{t} = \frac{\sum_{i=1}^{\infty}(c_i t_i \Delta t_i)}{\sum_{i=1}^{\infty}(c_i \Delta t_i)}$
$\tau = \frac{V}{F}$
$\sigma^2 = \overline{t^2} - (\bar{t})^2$
$\overline{t^2} = \frac{\sum_{i=1}^{\infty}(t_i^2 c_i \Delta t_i)}{\sum_{i=1}^{\infty}(c_i \Delta t_i)}$
$\sigma = \sqrt{\overline{t^2} - (\bar{t})^2}$
RESET
EXIT
Time [s]
Axis Length 600
Acceleration 10.0
Log Interval 5
Step Interval 10
LAG
LOG
MODEL f/b
ENTRY f/b
PLANT f/b
Step Mode
STEP
NO FILTER

*f*_2.1

2.1.1 Theory

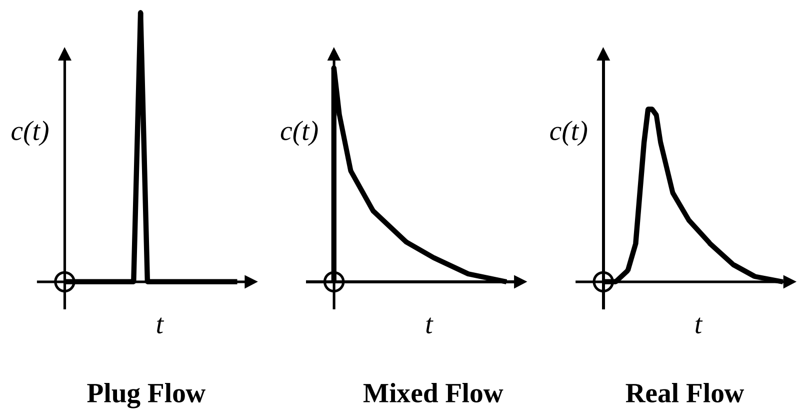
If we have a stream of fluid flowing into a vessel such as a reactor, and we wish to examine the degree of internal mixing in the vessel, we can inject a pulse of dye or some similar marking substance into the stream as it enters. Its subsequent behaviour will depend on the vessel design and the fluid flow properties. Sometimes the dye will move through the vessel quickly, passing directly from the inlet to the outlet (bypassing); sometimes the cloud of dye will move through the vessel without back- or forward-mixing with other fluid (plug flow); sometimes it would be well mixed in the vessel (mixed flow) and some could get into the corners where the flow velocities are low (dead zones). If the concentration of dye leaving the vessel is then measured as a function of time, a plot of concentration *vs* time could be constructed as shown.



*f*_2.2

Assuming that the dye is a good indicator of the behaviour of the rest of the fluid in the vessel, the shape of this curve will indicate the degree of mixing and the flow pattern. Levenspiel (1999) shows that it can be interpreted as the distribution of residence times in the vessel.

As an example, if the flow is through a tubular reactor where there is little mixing in the tube, we essentially have plug flow and the dye cloud would move through the reactor with much the same shape and dimensions as it had at the inlet: the concentration measured at the outlet would be zero up to a certain time, would then jump suddenly to a maximum value and then fall immediately back to zero: this is ideal plug flow. If the contents of the reactor are well mixed, the dye cloud would mix immediately with the rest of the fluid in the reactor and the residence time distribution (RTD) would have an exponential decay with time: this would be ideal mixed flow. These two ideal conditions are seldom met in practice and most real reactor systems have RTD curves that lie somewhere between these two extremes.



*f*_2.3

If the tracer injection point is close to the reactor inlet, “zero” time can be taken as the time of injection of the tracer pulse. The flow rate through the system is also considered to be constant.

The actual RTD curve, ie. the probability distribution *p*(*t*) of residence time *t*, is given my

$$p(t_i) = \frac{c_i}{\sum_{i=1}^{\infty}(c_i \cdot \Delta t_i)} \tag{2.1}$$

The mean residence time (MRT) in the reactor is then given by:

$$\bar{t} = \frac{\sum_{i=1}^{\infty}(c_i \cdot t_i \cdot \Delta t_i)}{\sum_{i=1}^{\infty}(c_i \cdot \Delta t_i)} \tag{2.2}$$

The denominator in this expression is evaluated from the area under the concentration response curve. The numerator is evaluated by constructing a table for each time sample t_i, and with a chosen time interval Δt_i, calculating $c_i \cdot t_i \cdot \Delta t_i$: the sum of these terms then gives the numerator.

The space time is the theoretical MRT given by:

$$\tau = \frac{V}{v} \tag{2.3}$$

where *V* is the volume of the reactor and *v* is the flowrate. The variance of residence time is given by:

$$\sigma^2 = \overline{t^2} - (\bar{t})^2 \tag{2.4}$$

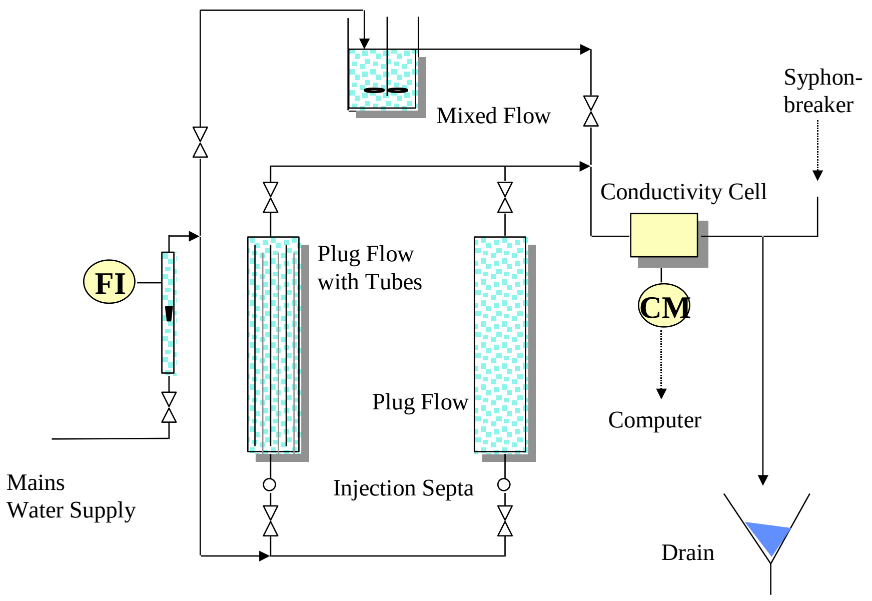
where $$\overline{t^2} = \frac{\sum_{i=1}^{\infty}(t_i^2 \cdot c_i \cdot \Delta t_i)}{\sum_{i=1}^{\infty}(c_i \cdot \Delta t_i)} \tag{2.5}$$

The standard deviation (SD), a measure of the spread of residence time, is thus clearly given by

$$\sigma = \sqrt{\overline{t^2} - (\bar{t})^2} \qquad (2.6)$$

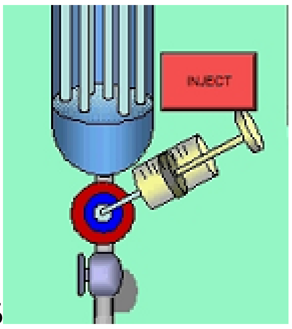
2.1.2 Tasks

The piping and instrumentation layout of the equipment is shown in *f*_2.4 below.

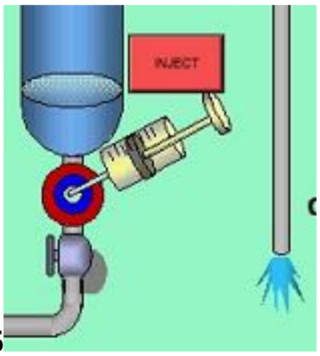


*f*_2.4

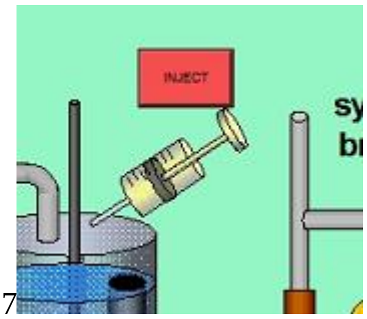
It is aimed to examine:
- the RTD curves for 3 different vessels
- the MRT of the fluid particles in the vessels compared to the Space Time of the vessels
- the SD of residence time *vs* Space Time for the vessels
- what differences there are between the vessels in terms of these parameters



*f*_2.5



*f*_2.6



*f*_2.7

In *f*_2.1 the respective vessels are: [a] PFR with axial mixing in tubes (*f*_2.5), [b] PFR (*f*_2.6), and [c] MFR (*f*_2.7).

(1) Press the LOG button and take note of the name of the RTC log file which begins

(2) Press the INJECT button for each vessel in turn, noting which vessel was injected at which time. Obviously wait until the exit concentration of the tracer returns to its datum value before proceding with the next injection.

(3) Press the LOG button again to terminate the file.

(4) Process the relevant sequences of the RTC log file in a spreadsheet program, in order to obtain the actual RTD curve, and the required statistics. Beware of concentration datum offsets.

(5) Only in the event of actual plant data (PLANT instead of MODEL setting), will an independent space time be available from a flow measurement.

References

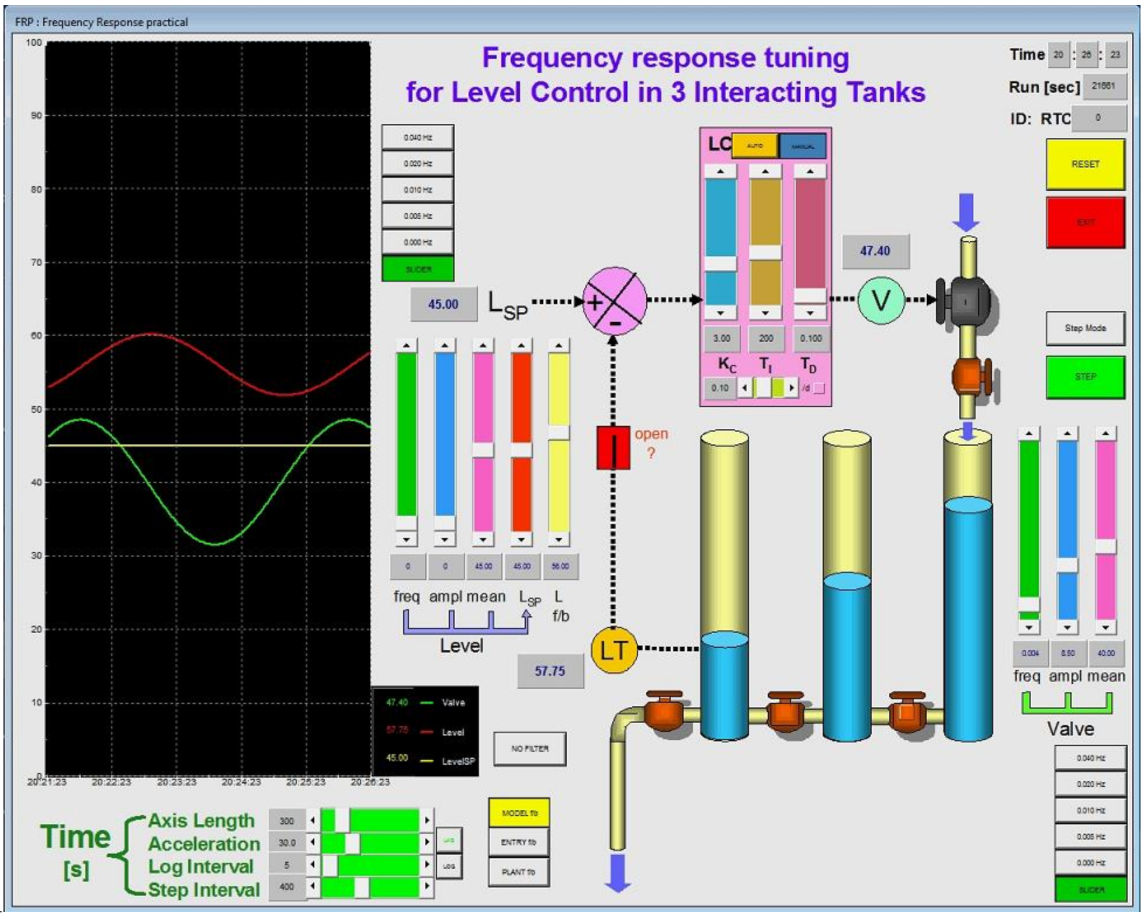
Levenspiel, O., “Chemical Reaction Engineering”, *Third Edition*, John Wiley & Sons, New York, 255 (1999)

Chapter 3 Simulations - Frequency response

.

3.1 FRP – Frequency response practical

The frequency response of an interacting tank system is measured to obtain Ziegler-Nichols PID parameters, which are tested using a setpoint step response:



*f*_3.1

Water is supplied under pressure from the water main. The 4-20mA output (Manipulated Variable: MV) of the control system passes through an I/P converter, whence the 3-15 psi signal moves a control valve which feeds water to the first of 3 interacting tanks. Each of the 3 tanks has a level-glass, and the tanks are separated by restriction valves. There is a final restriction valve at the exit of the third tank. The level in the third tank is the Process Variable (PV) of interest, and is measured using a bubble-tube sensor. This low pressure (cm H_20) is converted by a differential pressure (DP) cell to the 4-20mA range, which is the feedback signal to the controller.

3.1.1 Typical settings

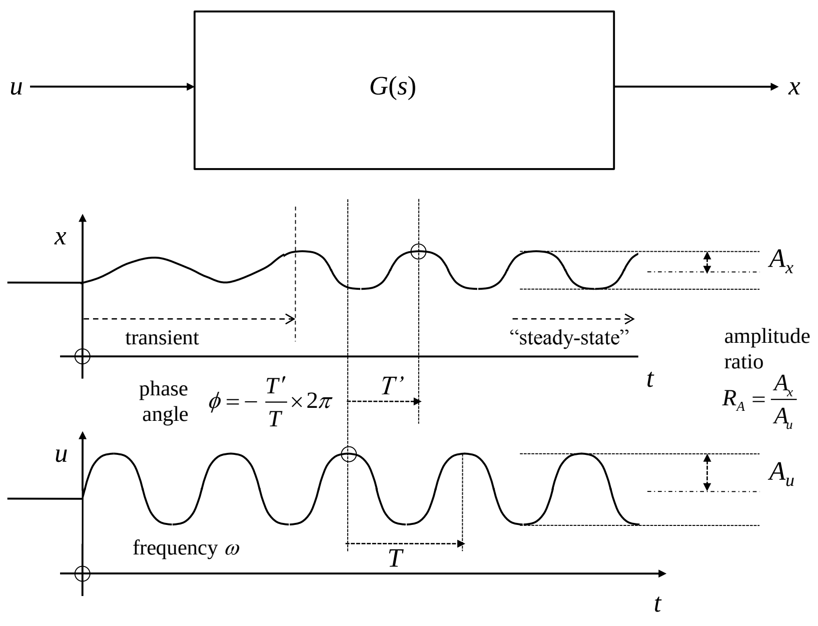
Valve Oscillation:	**Setpoint Oscillation:**	**PID Tuning:**
Mean=40	Mean=45	K_C = 3.0
Amplitude=6	Amplitude=6	T_I = 200
Frequency= 0.004	Frequency= 0.005	T_D = 0.1
		α = 0.1
		d : □
Time Axis:		
Length = 300		
Acceleration = 30		
Log Interval = 5		
Step Interval = 4000		

3.1.2 Theory

Although the 3 interacting tanks suggest a third-order system, the various transmitters and converters in the circuit may have sufficiently long time-constants to affect the system dynamics. A "black-box" approach will be adopted, and frequency testing will allow us to estimate suitable parameters $\mathbf{K_c}$, $\boldsymbol{\tau_I}$ and $\boldsymbol{\tau_D}$ for the PID controller without any knowledge of the intemals.

The Ziegler-Nichols frequency response method uses the response of the “open-loop” to sinusoidal excitation, to obtain parameters for the controller. All that is required in this case is to manipulate the feed valve sinusoidally, and determine the amplitude ratio and phase shift of the level in the third tank at a series of frequencies.

In general, the input of a sinusoidal excitation to a stable system leads to a "steady-state" output (frequency response), in which the amplitude and phase differ from those of the input (*f*_3.2).

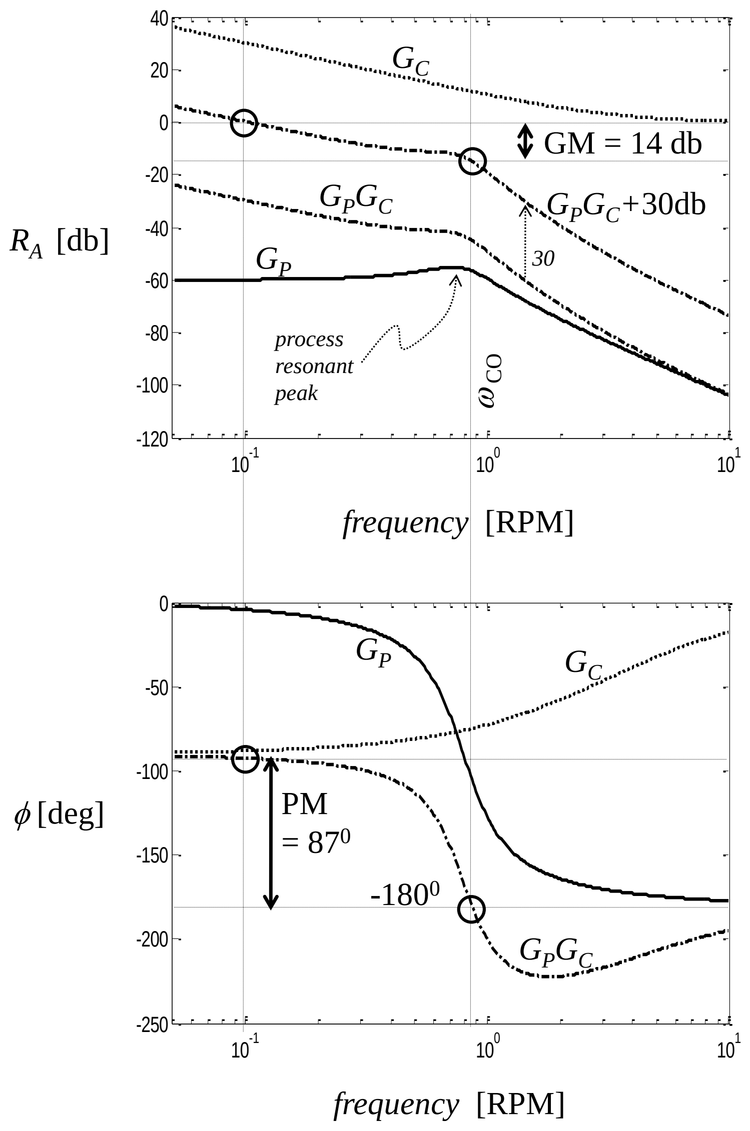


*f*_3.2

The amplitude ratio R_A and phase angle ϕ are found to vary according to the frequency ω of the excitation, and a typical "Bode Plot" of this relationship is shown in *f*_3.3. Here the original openloop process is represented by G_P. By way of example, this plot further shows curves for a proposed controller G_C, and the combined system in series, G_PG_C. Finally, a +30db adjustment is made to the gain of the controller G_C to achieve a gain margin (GM) of 14 db and a phase margin (PM) of 87^0. Recall that

$$\text{Gain Margin} = \frac{1}{R_A(\omega_{CO})} \quad \text{where } \omega_{CO} \text{ is the cross-over frequency where } \phi = -180^0 \tag{3.1}$$

$$\text{Phase Margin} = 180^0 + \phi\big|_{R_A=1} \quad \textit{ie.}\ 180^0 \text{ plus } \phi \text{ where } R_A = 1 \tag{3.2}$$



*f*_3.3

3.1.3 Tasks

(1) Obtain the frequency response of the uncontrolled open-loop process.

(2) Use a Bode Plot of this response to calculate the recommended Ziegler-Nichols settings for a three-term controller.

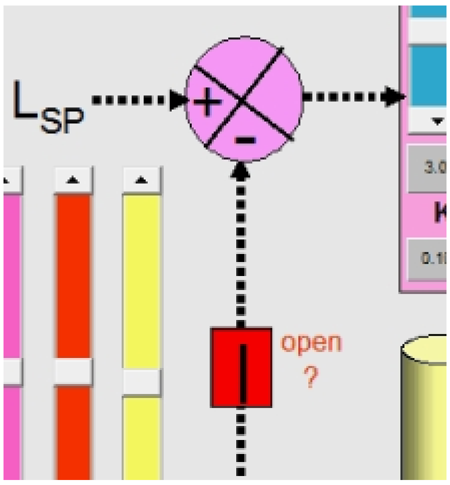
(3) Implement these settings on a PID controller. Assess the quality of control using a quadratic performance index calculated from the response to a set-point step-change.

(4) Attempt to improve the performance using an on-line trial-and-error tuning technique that involves adjustment of settings and observation of the closed-loop response to a set-point step-change.

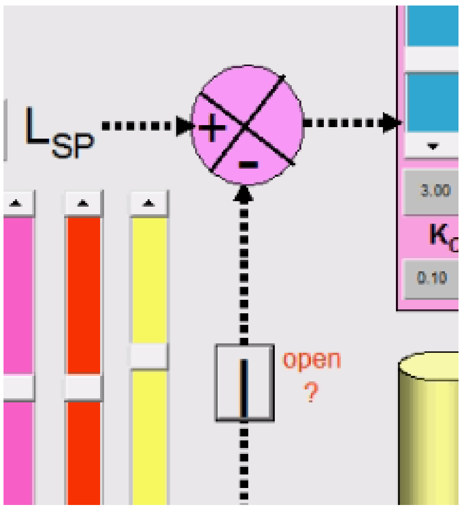
(5) For the better controller of (3) and (4), obtain the open-loop frequency response of controller and process in series. Use this to calculate the Gain Margin and Phase Margin of the controlled system.

3.1.4 Method

Note that the loop is *open* when the rectangle next to "open ?" shows in red (*f*_3.4), and is *closed* when it is clear (*f*_3.5). It is toggled between *open* and *closed* by pressing on the the rectangle.



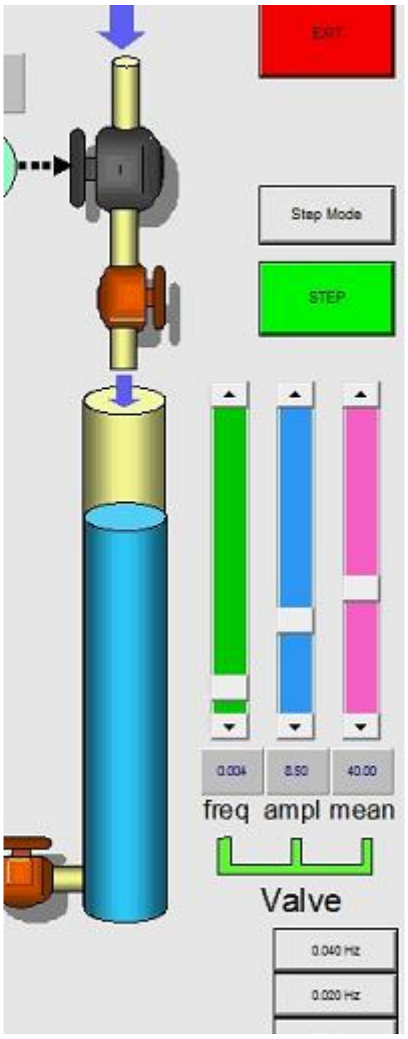
*f*_3.4



*f*_3.5

(a) Set the logging task on, with a time-interval of 2 seconds for data storage. You may stop this logging task at any time. Restarting it will generate a new file with the next name in the sequence.

(b) Open the loop. Obtain the open-loop frequency response (*without the controller: ie.* G_c=1) by manipulating the valve directly as follows (*f*_3.6): Choose frequencies of 0.04, 0.02, 0.01 and 0.005 Hz in that order, giving periods of 25 - 200 seconds. Be careful to note the *computer time* at every measurement sequence, so that the corresponding sequences can later be traced in the logged data. Obtain the amplitude ratio R_A and phase angle ϕ at each frequency, by plotting out the collected data, and comparing the valve trace as input to the tank level trace as output. Plot R_A and ϕ against frequency on log-log and log-linear axes respectively.



*f*_3.6

(c) Find the cross-over frequency ω_{co} and the amplitude ratio $R_A(\omega_{co})$ at this frequency. Then

ultimate gain	K_u	=	$1/\ R_A(\omega_{co})$		
ultimate period	P_u	=	$2\pi/\ \omega_{co}$	[with ω_{co} in radians/time]	(3.4)
		=	$1/\ \omega_{co}$	[with ω_{co} in cycles(repeats)/time]	(3.5)

The Ziegler-Nichols settings for a PID controller are as follows (Luyben, 1990):

$$K_c = K_u / 1.7 \tag{3.6}$$

$$\tau_I = P_u / 2 \tag{3.7}$$

$$\tau_D = P_u / 8 \tag{3.8}$$

These settings will serve as the "**calculated**" controller settings for the controlled response to a step test.

(d) From section 1.5.2, an **on-line trial and error** tuning technique (Luyben, 1990) will now be attempted:

(i) With the controller on manual, let the level in the third tank reach a steady value.

(ii) Take all of the integral action and derivative action out of the controller by setting maximum τ_I (eg. 99999), and minimum τ_D (0).
(iii) Position the set-point near the current tank level.
(iv) Set gain K_c, at a low value, say 0.5.
(v) Put the controller on automatic with the loop closed.
(vi) Make a small set-point change up or down, whichever is convenient, and observe the response of the controlled variable.
(vii) Keep doubling the gain K_c until the loop becomes very underdamped (persistent variation) and oscillatory.
(viii) Back-off on the gain to half this "ultimate" value.
(ix) Now start bringing in integral action by reducing τ_I by factors of 2 (doubling $1/\tau_I$), making small set-point steps at each value of τ_I to see the effect (Start with τ_I =120s).
(x) Find the value that makes it very underdamped and set τ_I to twice this value.
(xi) Now start bringing in derivative action by increasing τ_D. Make small set-point steps at each value of τ_D to see the effect. Find the value of τ_D which gives the tightest control without amplifying noise in the process measurement signal.
(xii) Increase K_c in steps until desirable damping and overshoot are obtained in reponse to a set-point step.

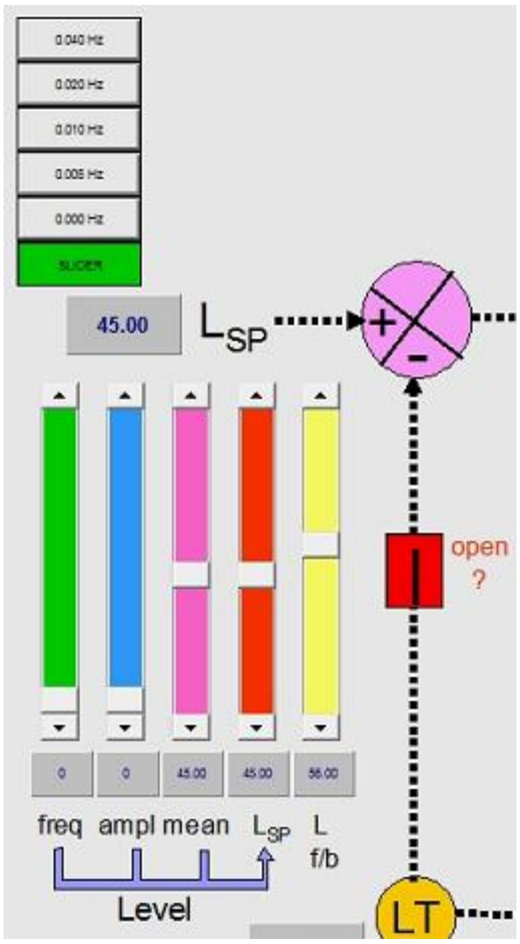
The controller parameters obtained in the above trial-and-error procedure will be referred to as the "**manual**" settings.

(e) For the purpose of comparison, now obtain the controlled response **to a particular step in set-point** for the **calculated** controller settings and the **manual** controller settings. Compare your two step-responses using a quadratic performance index:
$$I = \int_0^\infty (PV - SP)^2\,dt \qquad (3.9)$$
If the two steps were of different sizes, this must be accounted for, so that the index values may be compared (eg., assuming linearity, a step of twice the size will produce a QPI four times bigger, for control of the same quality).

(f) Using settings as in the better of **manual** and **calculated** above, obtain the openloop frequency response of the system, with

$$G_c = K_c\,[1 + 1/(\tau_I s) + \tau_D s] \qquad (3.10)$$

in series (controller on automatic, but loop opened). To achieve this, use the setpoint oscillation controls on the left (*f*_3.7) with the same frequencies of 0.04, 0.02, 0.01 and 0.005 Hz. You will have to manipulate the *mean* position of this signal to prevent the output level oscillation from drifting. This is because the integral action present requires that the mean position of the setpoint lies at the mean position of the actual level oscillation, otherwise there would be a net accumulation ("wind-up") on each cycle.



*f*_3.7

This second openloop Bode plot, with G_P and G_C in series, allows you to evaluate the **Gain Margin** and **Phase Margin** of this closed-loop control system.

3.1.5 Code extracts

- Controller and Model time-step....

```
void cFRP::Step()
{

        // Do the oscillations
        double daysec = (double)(theApp.m_hr*3600 + theApp.m_min*60 + theApp.m_sec);
        m_Level_sp  = __min(__max(m_LevelSPMean + m_LevelSPAmpl*sin(2*PI*m_LevelSPFreq*daysec) , 0), max_Level);
        if (!b_Auto[0]) m_Valve  = __min(__max(m_ValveMean + m_ValveAmpl*sin(2*PI*m_ValveFreq*daysec) , 0),
        max_Valve);

        bool b_changed_tuning = FALSE;
        for (i=0;i<nFRP;i++)
        {
                if (m_Kc[i] != Kc_last[i])
                {
                        b_changed_tuning = TRUE;
                        Kc[i] = m_Kc[i];
                        Kc_last[i] = Kc[i];
                }
                if (m_Ti[i] != Ti_last[i])
                {
                        b_changed_tuning = TRUE;
                        Ti[i] = m_Ti[i];
                        Ti_last[i] = Ti[i];
                }
                if (m_Td[i] != Td_last[i])
                {
                        b_changed_tuning = TRUE;
                        Td[i] = m_Td[i];
                        Td_last[i] = Td[i];
                }
                if (m_alpha[i] != alpha_last[i])
                {
                        b_changed_tuning = TRUE;
                        alpha[i] = m_alpha[i];
                        alpha_last[i] = alpha[i];
                }
                if (b_diff_err[i] != diff_err_last[i])
                {
                        b_changed_tuning = TRUE;
                        diff_err[i] = b_diff_err[i];
                        diff_err_last[i] = diff_err[i];
                }
        }
        if (b_changed_tuning) Initialise(TRUE,FALSE);   //PARTIAL

        if (theApp.FIRSTLOOP)
        {

                // initialise stacks for FRP loops
                r2[0] = r1[0] = r0[0] = m_Level;
                x2[0] = x1[0] = x0[0] = m_Level;
                m2[0] = m1[0] = m_Valve;

                // initialise Tank Levesls
                m_TankLevel[1] = max_Level;
                m_TankLevel[2] = (max_Level+m_Level)/2;
                m_TankLevel[3] = m_Level;
                theApp.FIRSTLOOP=0;
        }

        while (((theApp.t-tlast_FRP)>=dtFRP) || (theApp.b_FORCE_STEP))
        {
                if (!theApp.b_FORCE_STEP)
                {
                        tlast_FRP += dtFRP;     // must catch up by looping more than once if necessary
                }
                else
                {
                        // to SYNCHRONISE manual moves : b_FORCE_STEP was set TRUE on StepMode button
                        tlast_FRP = theApp.t;
                        theApp.b_FORCE_STEP = FALSE;
                }

                // Interpret present values
                if (!b_LoopBroken)
                {
                        x[0] = m_Level;
                        m_LevelScrollBar = m_Level;
                }
                else
                {
                        x[0] = m_LevelScrollBar;
                }
                r[0] = m_Level_sp;
                m[0] = m_Valve;

                for (i=0;i<nFRP;i++)
                {
                        if (!b_Auto[i])
                        {
```

```
                // Not on - reset to starting values
                r2[i] = r1[i] = r0[i] = x[i];
                x2[i] = x1[i] = x0[i] = x[i];
                m2[i] = m1[i] = m[i];
            }
            else
            {
                // On AUTO : calculate FRP action!
                // first cascade stack
                r2[i] = r1[i];
                r1[i] = r0[i];
                r0[i] = r[i];
                x2[i] = x1[i];
                x1[i] = x0[i];
                x0[i] = x[i];
                m2[i] = m1[i];
                m1[i] = m[i];

                // Action:
                m[i] = ( - a1[i] * m1[i]
                         - a2[i] * m2[i]
                              + b0[i] * r0[i]
                              + b1[i] * r1[i]
                              + b2[i] * r2[i]
                              - c0[i] * x0[i]
                              - c1[i] * x1[i]
                              - c2[i] * x2[i] ) / a0[i];
            }
        }

        // back to actual variables
        m_Valve = __max(__min(m[0],max_Valve),0);
    }

    // Model............................

    while ((theApp.t-theApp.tlast_ModelStep) >= dtFRP)
    {
        theApp.tlast_ModelStep += dtFRP;
        if (theApp.b_Model)
        {
            // Simple Euler model
            double F12 = 0.05*(m_TankLevel[1] - m_TankLevel[2]);
            double F23 = 0.05*(m_TankLevel[2] - m_TankLevel[3]);
            double F3 = 0.05*m_TankLevel[3];
            double dTL1 = 1.0*(0.07*m_Valve - F12)*dtFRP;
            double dTL2 = 1.0*(F12 - F23)*dtFRP;
            double dTL3 = 1.0*(F23-F3)*dtFRP;
            m_TankLevel[1] = m_TankLevel[1]+dTL1;
            m_TankLevel[2] = m_TankLevel[2]+dTL2;
            m_TankLevel[3] = m_TankLevel[3]+dTL3;
            m_Level = __max(__min(m_TankLevel[3],max_Level),0);
        }
    }
}
```

References

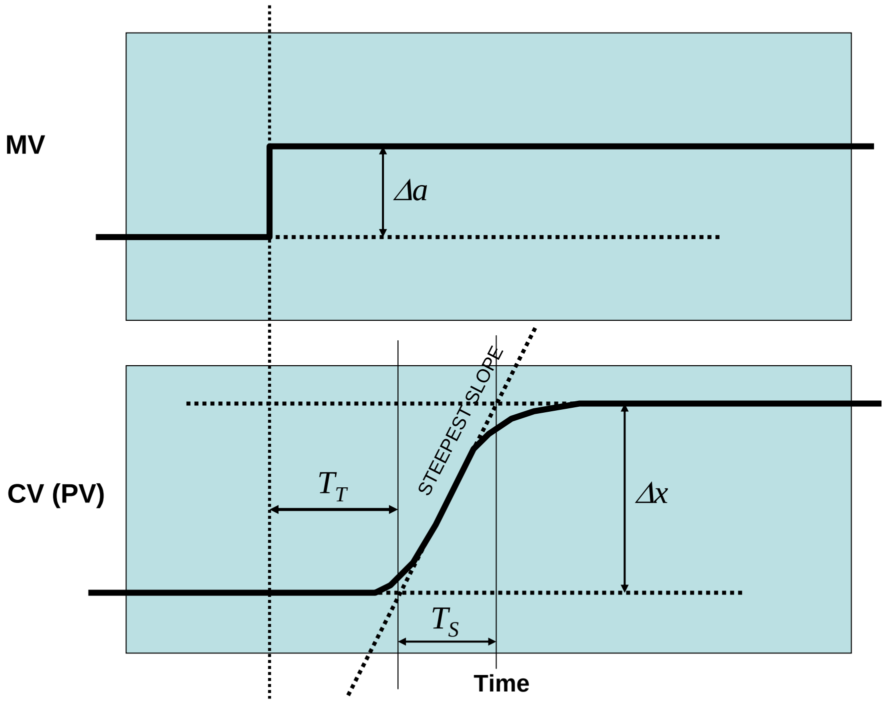
Luyben, W.L., “Process Modeling, Simulation and Control for Chemical Engineers”, Second Edition, McGraw-Hill, (1990).

Chapter 4 Simulations - SISO closedloop

4.1 Notes on tuning a PID loop

A number of tuning techniques are reported in the literature, such as Ziegler and Nichols (1942), Cohen and Coon (1953), and an on-line trial and error method. Some are based on the open-loop frequency response – ie. the steady oscillating output of the uncontrolled process whilst its input MV is being sinusoidally manipulated. Needless to say, not many operators of chemical plants subscribe to frequency response methods. Probably the most useful methods to find a tuning *starting point* are based on the step response – the so-called "reaction curve" methods. With a little experience, the persons responsible for setting up PID loops on plants will quickly bypass such methods, and choose K_C, T_I , (and possibly T_D) directly from their knowledge of the process, its speed of response, dead-time present, and the types of disturbances present. For example, we intuitively expect that T_I should be several times longer than the process response time, to prevent the controller from inducing its own oscillations. Moreover, we can get a good idea of the process response time just by viewing plant data records. Such skills should clearly be aimed for, but in the meantime, consider a more formal technique based on the reaction curve.

4.1.1 Reaction Curve tuning



*f*_4.1

An openloop step response is shown for a PV (process variable) output as a result of an MV (manipulated variable) step (*f*_4.1). A curve like this will be obtained provided the process does not have a natural pure integrator (eg. the level in a drum responds as a pure integral of the inflow). Note that the steepest slope of the response is $\Delta x / T_S$. The time T_T represents a transport lag (ie. dead-time lag). Certainly, if the process did consist solely of a dead-time lag and a first order response, T_T would be the actual dead-time, because the steepest part of a first-order response is at its start. In fact, many higher order systems are adequately represented by a combination of dead-time and first-order response, so it is understandable that the three parameters

Steady-state gain	K	=	$\Delta x / \Delta a$	(4.1)
Steepest slope	N	=	$\Delta x / T_S$	(4.2)
Initial delay	T_T			(4.3)

can form the basis of a "rule-of-thumb" tuning procedure such as that of Ziegler and Nichols:

Proportional:	K_c	=	$\Delta a / (N\, T_T)$	(4.4)
Proportional – Integral:	K_c	=	$0.9\ \Delta a / (N\, T_T)$	(4.5)
	T_I	=	$T_T\ / 0.3$	(4.6)
Proportional – Integral – Derivative:	K_c	=	$1.2\ \Delta a / (N\, T_T)$	(4.7)
	T_I	=	$T_T\ / 0.5$	(4.8)
	T_D	=	$0.5\ T_T$	(4.9)

When the delay T_T is small, it becomes difficult to estimate it accurately, possibly making K_C unduly large. This method obviously cannot be applied when there is no delay T_T, or when there is a pure integrator in the process. The literature abounds with guides to tuning in such circumstances, but for the meantime just note the on-line trial-and-error method below, which is performed in closed loop (ie. on "Auto").

4.1.2 On-line trial & error tuning

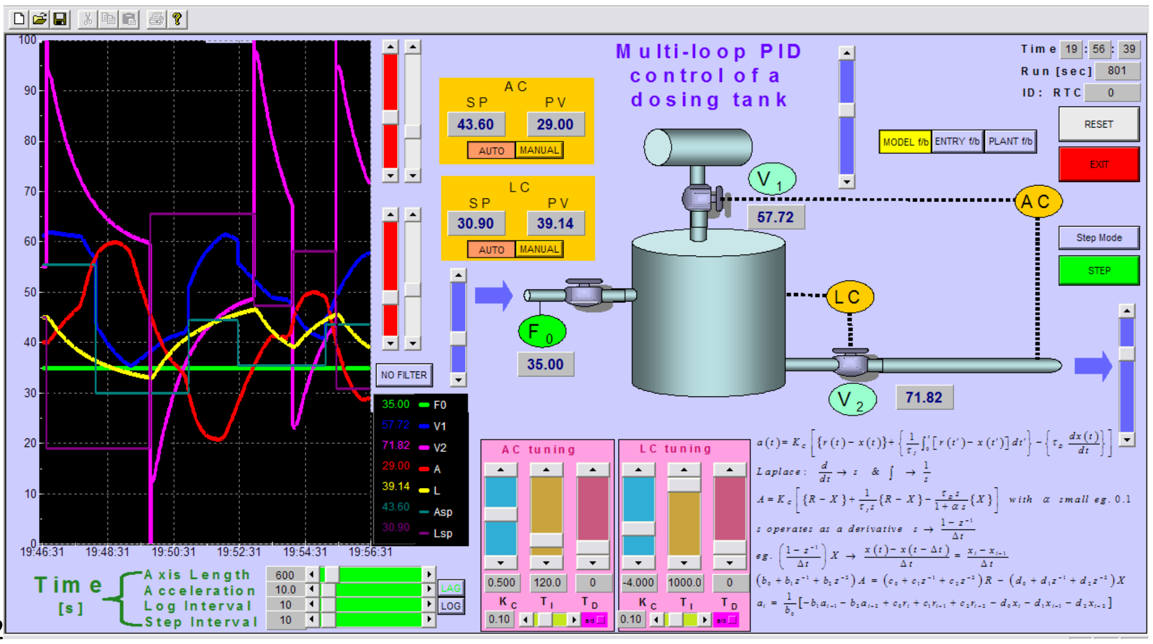
(a) Switch to manual and attain a steady operating point.
(b) Take all of the integral action and derivative action out of the controller by setting a "large" T_I , and minimum T_D (0).
(c) Position the set-point near the current value of the PV to be controlled.
(d) Set gain K_C to a "low" value
(e) Put the controller on automatic.
(f) Make a small set-point change up or down, whichever is convenient, and observe the response of the controlled variable.
(g) Keep doubling the gain K_C until the loop becomes very underdamped (persistent variation) and oscillatory.
(h) Back-off on the gain to half this "ultimate" value.
(i) Now start bringing in integral action by reducing T_I by factors of 2 (doubling $1/T_I$), making small set-point steps at each value of T_I to see the effect.
(j) Find the value that makes it very underdamped (bu not quite unstable) and set T_I to twice this value.

(k) Now start bringing in derivative action by increasing T_D. Make small set-point steps at each value of T_D to see the effect. Find the value of T_D which gives the tightest control without amplifying noise in the process measurement signal.

(l) Increase K_C in steps until desirable damping and overshoot are obtained in reponse to a set-point step.

4.2 PID - Dosing tank level and concentration PID loops

Understanding the PID algorithm and dealing with an integrating process with some interaction:



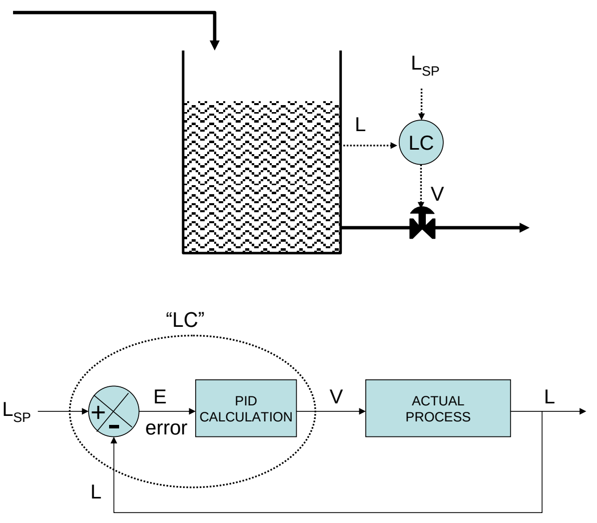
Multi-loop PID control of a dosing tank
AC SP 43.60 PV 29.00 AUTO MANUAL
LC SP 30.90 PV 39.14 AUTO MANUAL
Time 19 : 56 : 39
Run [sec] 801
ID: RTC 0
RESET
EXIT
MODEL f/b ENTRY f/b PLANT f/b
Step Mode
STEP
NO FILTER
V1 57.72
V2 71.82
F0 35.00
AC
LC
35.00 F0
57.72 V1
71.82 V2
29.00 A
39.14 L
43.60 Asp
30.90 Lsp
AC tuning 0.500 120.0 0 KC TI TD 0.10
LC tuning -4.000 1000.0 0 KC TI TD 0.10
Time [s] Axis Length 600 Acceleration 10.0 Log Interval 10 Step Interval 10
LAG LOG
19:46:31 19:48:31 19:50:31 19:52:31 19:54:31 19:56:31

f_4.2

4.2.1 Typical settings

$V_1 = 50$	**AC:**	$K_C = 0.8$	**LC:**	$K_C = -40$	Axis Length = 600
$V_2 = 55$		$T_I = 100$		$T_I = 40$	Acceleration = 10
$F_0 = 35$		$T_D = 2$		$T_D = 2$	Log Interval = 10
$A_{SP} = 40$		$\alpha = 0.1$		$\alpha = 0.1$	Step Interval = 10
$L_{SP} = 45$		d : ☐		d : ☐	

4.2.2 Theory



*f*_4.3

The three terms in the common PID controller serve the following purposes:

P **Proportional action:** *Moves V proportional to E and provides damping*

I **Integral action:** *Most processes do not have natural integration, so a permanent error E is required to keep V in a new position. Integral action provides a means to eliminate this offset, because V will continue to move with the integral of E.*

D **Derivative action:** *Makes a contribution to the movement of V in proportion to the derivative of E (actually, the derivative of –L in practice, to avoid large responses to steps in L_{SP}). The benefit of this is that large movements in V are possible before E has grown large, giving "lead" to the action (whereas integral action is retrospective and is said to give "lag"). The size of the gradient of E is clearly assumed to be related to the size of the incoming disturbance. Thus this action is not suitable when the L signal is "noisy" (carrying sharp, small disturbances), as this would cause V to oscillate wildly. Thus D action is used infrequently.*

The appropriate equation for the control action V then uses a term proportional to the error *(L_{SP} - L),* one proportional to the integral of this error, and, in the case shown, one proportional to the derivative of *(-L)* alone:

$$V(t)=K_C\left[\left\{L_{SP}(t)-L(t)\right\}+\left\{\frac{1}{\tau_I}\int_0^t\left[L_{SP}(t')-L(t')\right]dt'\right\}-\left\{\tau_D\frac{dL(t)}{dt}\right\}\right] \qquad (4.10)$$

Laplace: $\dfrac{d}{dt}\rightarrow s \quad \& \int \rightarrow \dfrac{1}{s}$:

$$V=K_C\left[\left\{L_{SP}-L\right\}+\frac{1}{\tau_I s}\left\{L_{SP}-L\right\}-\frac{\tau_D s}{1+\alpha s}\left\{L\right\}\right] \text{ with } \alpha \text{ small eg. 0.1.} \qquad (4.11)$$

The increase of the order of the denominator renders this term physically realisable. 's' acts as a derivative

$$s \rightarrow \frac{1-z^{-1}}{\Delta t} \tag{4.12}$$

where z^{-1} is the backwards shift operator for time-step Δt.

$$\text{eg.}\ \left(\frac{1-z^{-1}}{\Delta t}\right)X \rightarrow \frac{x(t)-x(t-\Delta t)}{\Delta t} = \frac{x_i - x_{i-1}}{\Delta t} \tag{4.13}$$

(The Tustin approximation $s \rightarrow \frac{2}{\Delta t}\left(\frac{z-1}{z+1}\right)$ could also be used) (4.14)

By substitution of $\frac{1-z^{-1}}{\Delta t}$ for s obtain

$$\left(a_0 + a_1 z^{-1} + a_2 z^{-2}\right)V = \left(b_0 + b_1 z^{-1} + b_2 z^{-2}\right)L_{SP} - \left(c_0 + c_1 z^{-1} + c_2 z^{-2}\right)L \tag{4.15}$$

ie.

$$V_i = \frac{1}{a_0}\left[-a_1 V_{i-1} - a_2 V_{i-2} + b_0 L_{SPi} + b_1 L_{SPi-1} + b_2 L_{SPi-2} - c_0 L_i - c_1 L_{i-1} - c_2 L_{i-2}\right] \tag{4.16}$$

This is the necessary recursive relationship for PID controller action *V*. Note that this relationship has natural anti-windup properties if the manipulated variable *V* is simply "clipped" at its upper and lower limits.

4.2.3 Tasks

(1) With both controllers on "Auto", let the process stabilise at an operating point, then switch both controllers to "Manual".

(2) Step V_1 up by about 25. Sketch the response. What type of response does this appear to be? Is the level affected ? Why?

(3) Freezing the response with the "Step Mode" button, estimate the parameters Δa, Δx, T_T and T_S required for the calculation of controller parameters according to the "Reaction Curve" method in section 4.1.1 (These will be used later).

(4) Step V_2 up by about 25. Sketch the response. What type of response does this appear to be? Why? Is the composition affected? Is there cross-coupling in this process, which could lead to control loop interaction ("riding") ?

(5) Switch both controllers back onto "Auto". Wait until the system re-attains a steady operating point.

(6) Step the composition setpoint up by about 25. Sketch the composition response. Look at the valve motion. Can you explain it?

(7) Repeat (6) with only the PV in the derivative term (Check the box at bottom right of the tuning controls, and the pink highlight should disappear. This is dicussed at the end of

section 1.4). How does the setpoint step response differ from (6)? Why? Reduce T_D to zero. Do (6) and (7) still differ?

(8) See if you can detect any effect of the α parameter using the composition controller (bottom scrollbar in the controller tuning block).

(9) Step the level setpoint up by about 25. Sketch what you see. Why does valve V2 return to where it started? What happens to the composition?

(10) Decrease F_0 by about 25. Why does V_2 change position permanently now?

(11) Try to improve the tuning of the composition control loop so that the composition is not so badly disturbed by the level controller, eg. during level setpoint steps. First try a controller based on the results obtained in the "Reaction Curve" test in (3) above.

(12) With both loops closed (ie. on "Auto"), see if the "trial and error" closed loop tuning method (section 4.1.2) gives any improvement on the performance of the level control loop.

4.2.4 Code extracts

• At start-up or when controller parameters are changed, the following assignments are executed....

```
// New parameters for PID Controllers
double dtPID2 = pow(dtPID,2);
for (i=0;i<nPID;i++)
{
        a0[i] = Ti[i]*( dtPID  +  alpha[i])/dtPID2;
        a1[i] = Ti[i]*(-dtPID  -2*alpha[i])/dtPID2;
        a2[i] = Ti[i]*(           alpha[i])/dtPID2;
        b0[i] = Kc[i]*( dtPID2 + alpha[i]*dtPID + Ti[i]*dtPID +    alpha[i]*Ti[i] +   Ti[i]*Td[i]*diff_err[i])/dtPID2;
        b1[i] = Kc[i]*(        - alpha[i]*dtPID - Ti[i]*dtPID - 2* alpha[i]*Ti[i] -  2*Ti[i]*Td[i]*diff_err[i])/dtPID2;
        b2[i] = Kc[i]*(                                            alpha[i]*Ti[i] +   Ti[i]*Td[i]*diff_err[i])/dtPID2;
        c0[i] = Kc[i]*( dtPID2 + alpha[i]*dtPID + Ti[i]*dtPID +    alpha[i]*Ti[i] +   Ti[i]*Td[i]             )/dtPID2;
        c1[i] = Kc[i]*(        - alpha[i]*dtPID - Ti[i]*dtPID - 2* alpha[i]*Ti[i] - 2*Ti[i]*Td[i]             )/dtPID2;
        c2[i] = Kc[i]*(                                            alpha[i]*Ti[i] +   Ti[i]*Td[i]             )/dtPID2;
}
```

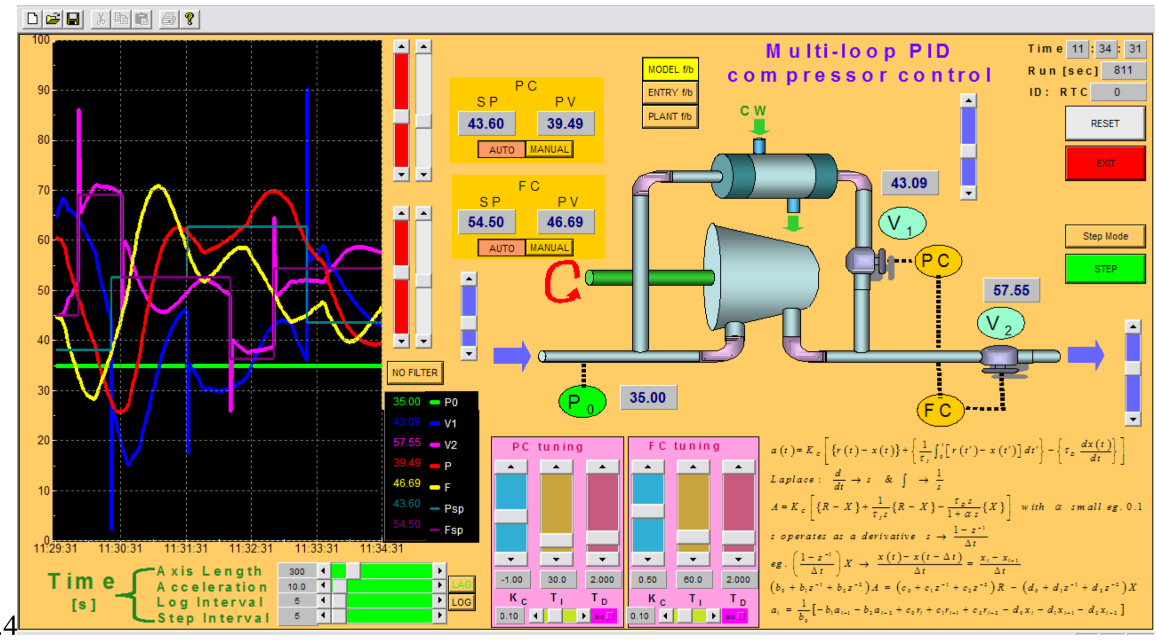
• On each time-step, different calculations are executed depending on whether it is the first call, or whether the controllers are on "Auto" or "Manual".....

```
if (theApp.FIRSTLOOP)
{
        // initialise stacks for PID loops
        r2[0] = r1[0] = r0[0] = m_A;
        r2[1] = r1[1] = r0[1] = m_L;
        x2[0] = x1[0] = x0[0] = m_A;
        x2[1] = x1[1] = x0[1] = m_L;
        m2[0] = m1[0] = m_V1;
        m2[1] = m1[1] = m_V2;

        theApp.FIRSTLOOP=0;
}
```

```
while (((theApp.t-tlast_PID)>=dtPID) | (theApp.b_FORCE_STEP))
{
        if (!theApp.b_FORCE_STEP)
        {
                tlast_PID += dtPID;    // must catch up by looping more than once if necessary
        }
        else
        {
                tlast_PID = theApp.t;        // to SYNCHRONISE manual moves :
                                            // b_FORCE_STEP was set TRUE on StepMode button
                theApp.b_FORCE_STEP = FALSE;
        }

        // Interpret present values
        x[0] = m_A;
        x[1] = m_L;
        r[0] = m_A_sp;
        r[1] = m_L_sp;
        m[0] = m_V1;
        m[1] = m_V2;


        for (i=0;i<nPID;i++)
        {
                if (!b_Auto[i])
                {
                        // Not on - reset to starting values
                        r2[i] = r1[i] = r0[i] = x[i];
                        x2[i] = x1[i] = x0[i] = x[i];
                        m2[i] = m1[i] = m[i];
                }
                else
                {
                        // On AUTO : calculate PID action!
                        // first cascade stack
                        r2[i] = r1[i];
                        r1[i] = r0[i];
                        r0[i] = r[i];
                        x2[i] = x1[i];
                        x1[i] = x0[i];
                        x0[i] = x[i];
                        m2[i] = m1[i];
                        m1[i] = m[i];

                        // Action:
                        m[i] =          ( - a1[i] * m1[i]
                                         - a2[i] * m2[i]
                                         + b0[i] * r0[i]
                                         + b1[i] * r1[i]
                                         + b2[i] * r2[i]
                                         - c0[i] * x0[i]
                                         - c1[i] * x1[i]
                                         - c2[i] * x2[i] ) / a0[i];
                }
        }

        // back to actual variables & clip
        m_V1 = __max(__min(m[0],max_V1),0);
        m_V2 = __max(__min(m[1],max_V2),0);
}
```

4.3 PII - Compressor control with strongly interacting PID loops

Tuning for satisfactory operation under conditions of strong interaction:

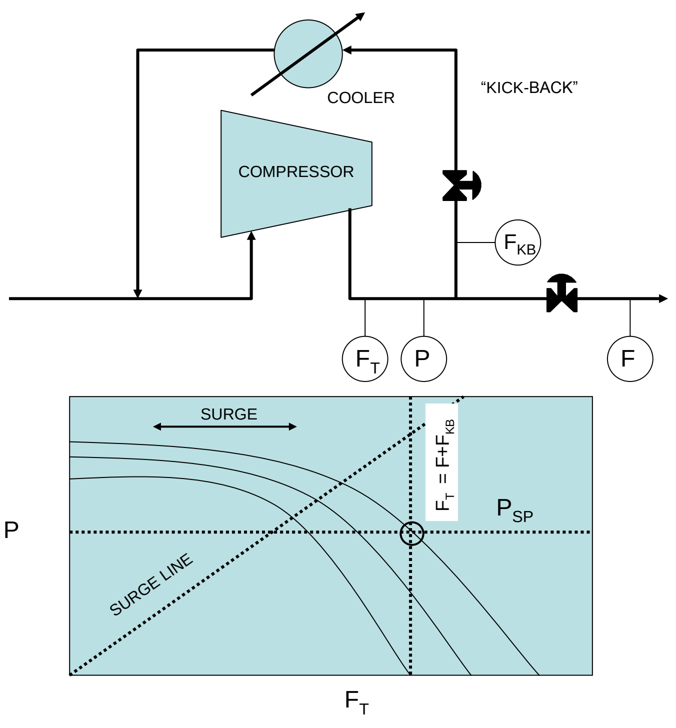


f_4.4

4.3.1 Typical settings

$V_1 = 50$	**PC:**	$K_C = -1$	**FC:**	$K_C = 0.5$	Axis Length = 300
$V_2 = 55$		$T_I = 30$		$T_I = 60$	Acceleration = 10
$P_0 = 35$		$T_D = 2$		$T_D = 2$	Log Interval = 5
$P_{SP} = 40$		$\alpha = 0.1$		$\alpha = 0.1$	Step Interval = 5
$F_{SP} = 45$		d : □		d : □	

4.3.2 Theory



*f*_4.5

This is a basic surge control arrangement on a compressor. If there were no pressure control, the compressor would go into surge if the user flow demand (*F*) cut back (*f*_4.5). The PC and FC arrangement shown in *f*_4.4 effectively forces the compressor to operate at the indicated point, assuming that the speed (curved lines) is governed at a fixed value.

One expects the arrangement shown to be highly interactive, because the valves draw from a common point. If the FC valve opens, pressure will drop, requiring a correction by PC, and vice-versa. In practice, to prevent these two loops from "riding" each other, one loop has to be made much slower than the other. In this exercise the PC loop will be made fast and the FC loop slow, as the PC loop offers the primary surge protection.

4.3.3 Tasks

(1) With both controllers on "Auto", let the process stabilise at an operating point, then switch both controllers to "Manual".

(2) Step V_1 up by about 25. What type of response does this appear to be? Is the flow affected? Why?

(3) Step V_2 up by about 25. Is the pressure affected? Why?

(4) Switch the pressure controller to "Auto". Wait for the system to steady, then step the pressure setpoint down by about 25. Note the response of both flow and pressure.

(5) Switch the flow controller to "Auto". Wait for the system to steady, then step the flow setpoint down by about 25. Note the response of both flow and pressure.

(6) Try to speed up the flow response as much as possible. Primarily you will be attempting to find a suitable combination of K_C and T_I of FC, with T_D possibly set to zero. Note the point at which the system goes unstable. Back off to a point which is a compromise between speed and dampening of the oscillations.

(7) Disturb the system with a step in the suction pressure of the compressor and check the controlled system'sperformance.

4.3.4 Code extracts

- Model time-step....

```
// Convolution model based on step responses

        while ((theApp.t-theApp.tlast_ModelStep) >= dtPII_StepResp/(double)Ninterp)
        {
                theApp.tlast_ModelStep += dtPII_StepResp/(double)Ninterp;

                dmlinterp(1) = m_V1-mlastinterp(1);
                dmlinterp(2) = m_V2-mlastinterp(2);
                dmlinterp(3) = m_P0-mlastinterp(3);

                mlastinterp(1) = m_V1;
                mlastinterp(2) = m_V2;
                mlastinterp(3) = m_P0;


                //update interpolated past vector
                // ACCUMULATE the moves falling off the past moves stack
                for (j=1; j<=(Ppii+Qpii); j++)
                {
                        dminterpACC(j) += dmpinterp(j);
                }
                // past moves shift one down dmpinterp stack   (to incorporate latest move)
                for (i=1; i<=(Npii*Ninterp-1); i++)
                {
                        for (j=1; j<=(Ppii+Qpii); j++)
                        {
                                dmpinterp((i-1)*(Ppii+Qpii)+j) = dmpinterp(i*(Ppii+Qpii)+j);
                        }
                }
                for (j=1; j<=(Ppii+Qpii); j++)
                {
                        dmpinterp((Npii*Ninterp-1)*(Ppii+Qpii)+j) = dmlinterp(j);
                }

                if (theApp.b_Model)
                {
                        cMatrix dx = dB0interp*dmpinterp;
                        // Add in any integral effects
                        for (i=1;i<=Rpii;i++)
                        {
                                for (j=1;j<=(Ppii+Qpii);j++)
                                {
                                        dx(i) += dB0interp(i,j)*dminterpACC(j);
                                }
                        }

                        P = P + dx(1);
                        F = F + dx(2);
                        m_P = __max(__min(P,max_P),0);
```

```
                    m_F = __max(__min(F,max_F),0);
            }
    }
```

References

Cohen, G. and G. Coon (1953) “Theoretical consideration of retarded control”, *Transactions of the ASME,* **75**, 827-834.

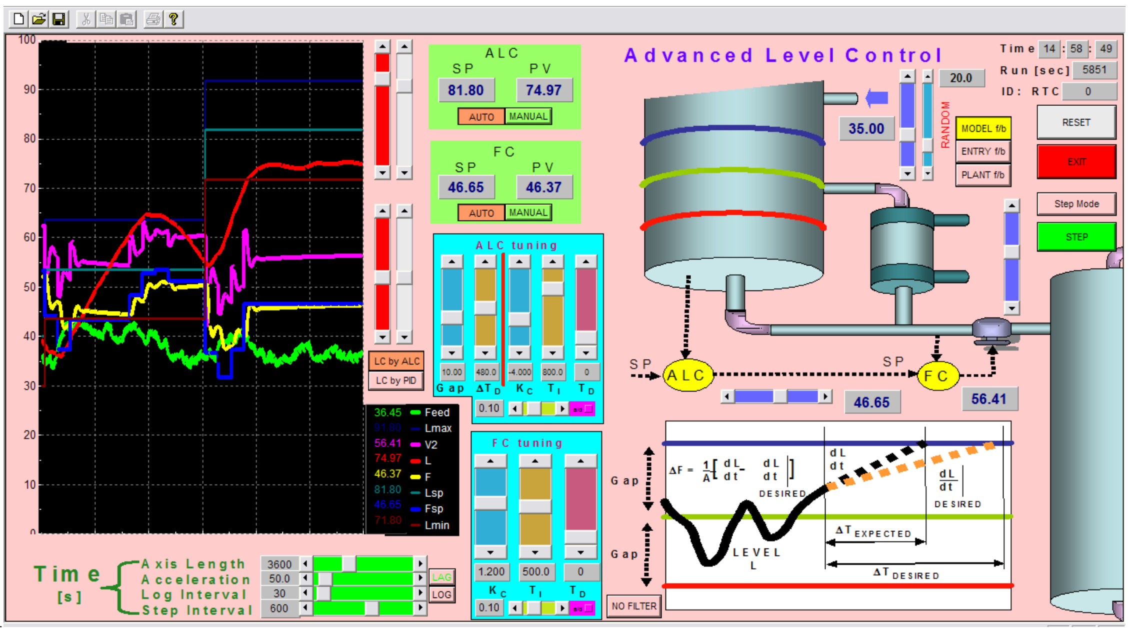
Ziegler, J.G. and N.B. Nichols (1942) “Optimum settings for automatic controllers”, *Transactions of the ASME,* **64**, 759–768.

Chapter 5 Simulations - SISO optimisers

.

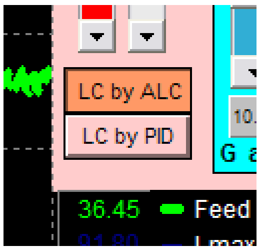
5.1 ALC - Advanced level control of a distillation column

Understanding and tuning an algorithm for minimisation of flow fluctuations between plant items:



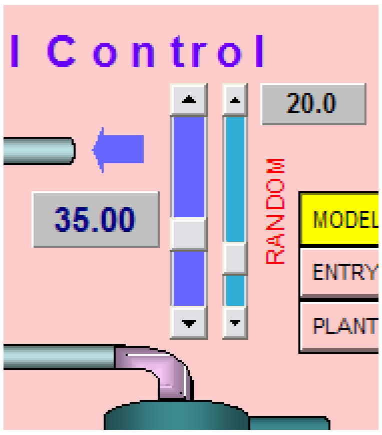
*f*_5.1

In this application, a comparison will be made between the performance of an Advanced Level Controller, and an ordinary PID controller maintaining the level. In both instances the controller cascades to the same slave flow controller. A new control switch is provided to switch between the two supervising controllers (*f*_5.2).



*f*_5.2

To provide a realistically variable environment for the level control to contend with, a feature is provided for addition of a random component in the feed flow rate into the column. This has a zero mean, and is added to the normal flow setting to obtain the actual flow into the column. The size of the random component is determined by the RANDOM scrollbar setting (*f*_5.3).

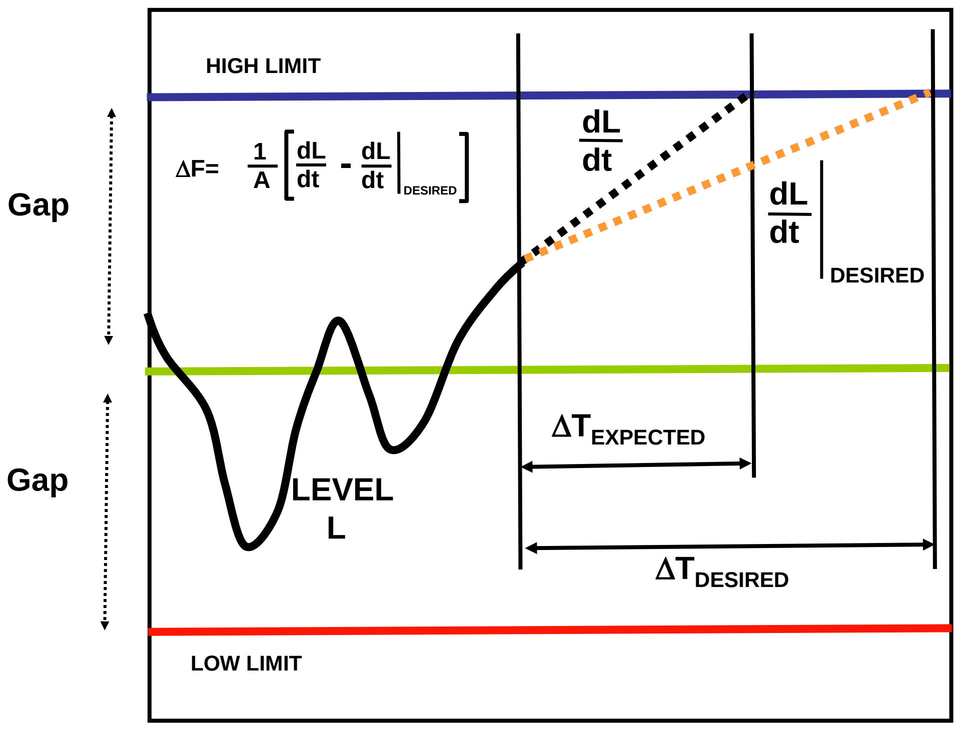


*f*_5.3

5.1.1 Typical settings

L= 40	**ALC:**	*Gap* = 10	**LC:**	K_C = -4	**FC:**	K_C = 1.2	Axis Length = 3600
L_{SP} = 45		ΔT_D = 480 (PID)		T_I = 800		T_I = 500	Acceleration = 50
F_{SP}= 35		(desired)		T_D = 2		T_D = 2	Log Interval = 30
F = 45				α = 0.1		α = 0.1	Step Interval = 600
F valve = 55				d : □		d : □	

5.1.2 Theory



*f*_5.4

The idea is to make as few adjustments to the outflow as possible, so as to introduce minimal variations into the downstream units. In this way, those units downstream can be controlled more tightly, closer to specification. This idea is made possible by maximum utilisation of the “buffering capacity” of the vessel in which level is to be controlled – indeed, modern plants are often designed with additional capacity here so as to facilitate ALC. There is an assumption that one is dealing with shorter positive and negative flow variations about some mean. If the outflow could be set up at the estimated mean, it might be possible to “ride out” the incoming flow variations. In the old days, operators/instrument technicians might have been quite proud of how tightly they were controlling level in a vessel, but if one looked at the corresponding flow or valve chart, it could be virtually “painted” with huge variations of the control action involved. This is all very well for the vessel considered, but what about the operations downstream?

There would be many variations of algorithms that could minimise the outflow variations – with different merits in different situations. In fact, this problem is not so straightforward, with algorithms running into several pages of code. The one proposed here is based on the idea of a desired time margin (ΔT_D , ie. $\Delta T_{DESIRED}$) representing a minimum acceptable time within which the upper or lower limit of level could be reached. If the present rate of level rise or fall indicates that the limit will be reached sooner, some minimal evasive action must be taken, in one step, with the aim of not having to make another adjustment soon. The diagram shows a way to adjust the outflow setpoint, but as can be seen in the code in section 5.1.4, there is more to it than just this.

5.1.3 Tasks

(1) Switch to the PID controller and set it on Auto. Press “Reset” and set the random component on the inflow down to zero. Let the system steady out.

(2) Step up the inflow by about 10, and note the level and outflow response.

(3) Repeat (1) and (2), but this time with the ALC controller. Note that the "*Gap*" setting is relative to the present controller setpoint. How do the level and outflow responses differ?

(4) Increase the RANDOM setting on the inflow from 0 to about 20. Compare the operation of the PID and ALC controller again. Comment on the swings in level, violation of the limits, and the frequency of outflow adjustments.

(5) Change the ΔT_D setting, and investigate what effect it has on the ALC controller with the same random variation of the inflow

(6) Change the Gap setting, and investigate what effect it has on the ALC controller with the same random variation of the inflow

5.1.4 Code Extracts

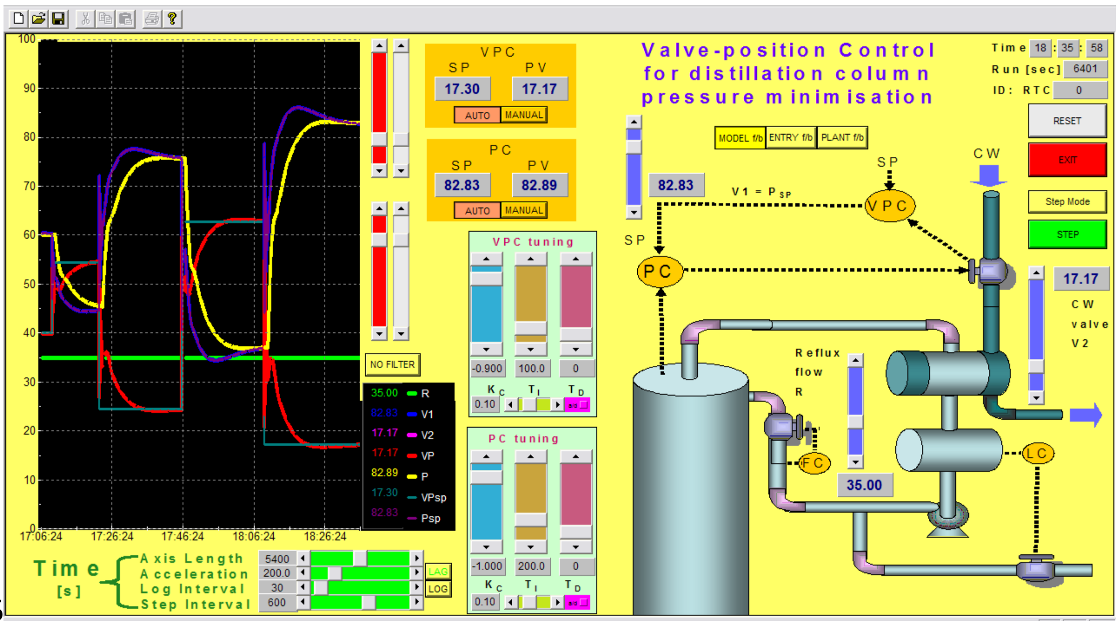
• On each time-step, different calculations are executed depending on whether it is the first call, or whether the controllers are on "Auto" or "Manual".....

```
if (theApp.FIRSTLOOP)
{
        // initialise stacks for ALC loops
        r2[0] = r1[0] = r0[0] = m_AL;
        r2[1] = r1[1] = r0[1] = m_F;
        x2[0] = x1[0] = x0[0] = m_AL;
        x2[1] = x1[1] = x0[1] = m_F;
        m2[0] = m1[0] = m_V1 = m_F;  // Special for ALC
        m2[1] = m1[1] = m_V2;

        theApp.FIRSTLOOP=0;
}

while (((theApp.t-tlast_ALC)>=dtALC) | (theApp.b_FORCE_STEP))
{
        if (!theApp.b_FORCE_STEP)
        {
                tlast_ALC += dtALC;    // must catch up by looping more than once if necessary
        }
        else
        {
                tlast_ALC = theApp.t;       // to SYNCHRONISE manual moves : b_FORCE_STEP
                                           //was set TRUE on StepMode button
                theApp.b_FORCE_STEP = FALSE;
        }

        // Interpret present values
        if (b_Auto[0])
        {
                b_Auto[1] = TRUE;  // Special for ALC
                m_F_sp = m_V1;   // Special for ALC
        }

        x[0] = m_AL;
        x[1] = m_F;
        r[0] = m_AL_sp;
```

```
r[1] = m_F_sp;
m[0] = m_V1;
m[1] = m_V2;


for (i=0;i<nALC;i++)
{
        if (!b_Auto[i])
        {
                // Not on - reset to starting values
                r2[i] = r1[i] = r0[i] = x[i];
                x2[i] = x1[i] = x0[i] = x[i];
                m2[i] = m1[i] = m[i];
                if (i==2) n_ALC_counter = 0;
        }
        else
        {
                // On AUTO : calculate control action!
                // first cascade stack
                r2[i] = r1[i];
                r1[i] = r0[i];
                r0[i] = r[i];
                x2[i] = x1[i];
                x1[i] = x0[i];
                x0[i] = x[i];
                m2[i] = m1[i];
                m1[i] = m[i];

                // Action:
                if ((i==1) | (!b_LCbyALC))
                {
                        // NORMAL PID CONTROL
                        m[i] = (  - a1[i] * m1[i]
                                - a2[i] * m2[i]
                                + b0[i] * r0[i]
                                + b1[i] * r1[i]
                                + b2[i] * r2[i]
                                 - c0[i] * x0[i]
                                 - c1[i] * x1[i]
                                 - c2[i] * x2[i] ) / a0[i];
                }
                else
                {
                        // ADVANCED LEVEL CONTROL
                        //present gradient
                        double dF=0;
                        n_ALC_counter -=1;
                        if (n_ALC_counter <= 0)
                        {
                                double dLdt = (x1[0]-x2[0]) / dtALC;
                                // delay - bit shorter than full flow response
                                int n_counter_reset = int(0.6*NALC*dtALC_StepResp/dtALC);
                                // dT_expected
                                double dT_expected, level_margin;
                                double dT_emergency = 0.7*m_ALC_dTdesired;
                                //UPPER SIDE
                                level_margin = (r0[0] + m_ALC_Gap) - x0[0];
                                if (level_margin < 0)
                                {
                                        dF=((-level_margin+dLdt*dT_emergency) * Area) / dT_emergency;
                                        n_ALC_counter = n_counter_reset;
                                }
                                else if (dLdt >= 0)
                                {
                                        dT_expected = level_margin / (dLdt + 1e-10);
                                        if((dT_expected < m_ALC_dTdesired) & (level_margin < 0.3*
                                                m_ALC_Gap))  // let it jump around in middle bit
                                        {
                                                dF = (level_margin * Area) * (1/dT_expected –
                                                        1/m_ALC_dTdesired);
                                                n_ALC_counter = n_counter_reset; //have to give it time
                                        }
                                }
                                //LOWER SIDE
                                level_margin = x0[0] - ( r0[0] - m_ALC_Gap) ;
                                if (level_margin < 0)
                                {
                                        dF=-((-level_margin-dLdt*dT_emergency) * Area) / dT_emergency;
                                        n_ALC_counter = n_counter_reset;
                                }
                                else if (dLdt <= 0)
                                {
                                        dT_expected = level_margin / (-dLdt+1e-10);
                                        if((dT_expected < m_ALC_dTdesired) & (level_margin < 0.3*
                                                m_ALC_Gap))  // let it jump around in middle bit
                                        {
                                                dF = -(level_margin * Area) * (1/dT_expected –
                                                        1/m_ALC_dTdesired);
                                                n_ALC_counter = n_counter_reset; //have to give it time
                                        }
                                }
                        }
                        m[0] += dF;
                }
        }
}

// back to actual variables
m_V1 = __max(__min(m[0],max_V1),0);
m_V2 = __max(__min(m[1],max_V2),0);
}
```

5.2 VPC - Valve position control for distillation column pressure minimisation

Use of a combination of PID loops for simple process operating point optimisation:



f_5.5

One of the early applications of Valve Position Control was distillation column pressure minimisation, as proposed by Shinskey, who worked for Foxboro Corporation, and who had a knack for arranging PID controllers in ingenious ways, even one controller manipulating the tuning parameters of another! Generally speaking, relative volatilities increase as pressure drops, so the operation of a distillation column at as low pressure as possible will enhance separation. Thus one needs to maximise the condenser duty. If it is water-cooled as in this example, one wants to open the cooling water supply valve as widely as possible. Why not just open it to 100% and leave it? This is not advisable, as in the operation there are likely to be other disturbances causing pressure fluctuations, and one would need some adjustment of the cooling water, both positive and negative, to deal with them rapidly to avoid complete upset of the column. Thus one needs to reserve some positive movement of the valve, and would like to maintain it at a setpoint of, say, 90% open. The means of doing this are by manipulating the setpoint of the pressure controller itself, relatively slowly so as not to upset the column. Another reason why the valve position controller should be relatively slower than the pressure controller is because the implicit dependence of these two loops has the potential to become completely unstable!

Note that this procedure is a kind of optimisation of a process. Indeed, there are many other applications of this simple optimisation technique, eg. maximising the use of a cheap coolant (CW), with only the residual duty being taken up by a very effective, but expensive, refrigerant. These principles could be applied to any material flows in a process, not just coolants – eg. maximum re-use of treated effluent water prior to fresh water, etc.

5.2.1 Typical settings

$VP = V_2 = 40$	**VPC:**	$K_C = -0.9$	**PC:**	$K_C = -1$	Axis Length = 5400
$VP_{SP} = 40$		$T_I = 100$		$T_I = 200$	Acceleration = 200
$P = 60$		$T_D = 0$		$T_D = 0$	Log Interval = 30
$P_{SP} = V_1 = 60$		$\alpha = 0.1$		$\alpha = 0.1$	Step Interval = 600
R (reflux) = 35		d : □		d : □	

5.2.2 Theory

The VPC clearly cannot function unless the PC is on “Auto”, accepting the remote setpoint. This occurs automatically one the VPC is set to “Auto”. If the system pressure is disturbed, the PC will try to correct it, but as it begins to move the valve V_2, it will find its own setpoint changing., and this can easily lead to unstable oscillations. Thus the VPC should be tuned to react somewhat more slowly than the PC.

5.2.3 Tasks

(1) With both controllers on “Auto”, let the system reach steady-state. Then switch the VPC to “manual”, and do a setpoint step-test on the PC. See if you can improve the performance of the PC by manipulating its K_C, T_I and T_D.

(2) Switch the VPC to “Auto”. Step its setpoint by about +25. See if you can improve the tuning of the VPC using its K_C, T_I and T_D. At what values does the system go unstable?

(3) Once you finalise tuning for PC and VPC, let the system reach steady state with both loops closed. Now step the reflux (R) as a disturbance. Can you account for all of the graphical responses?

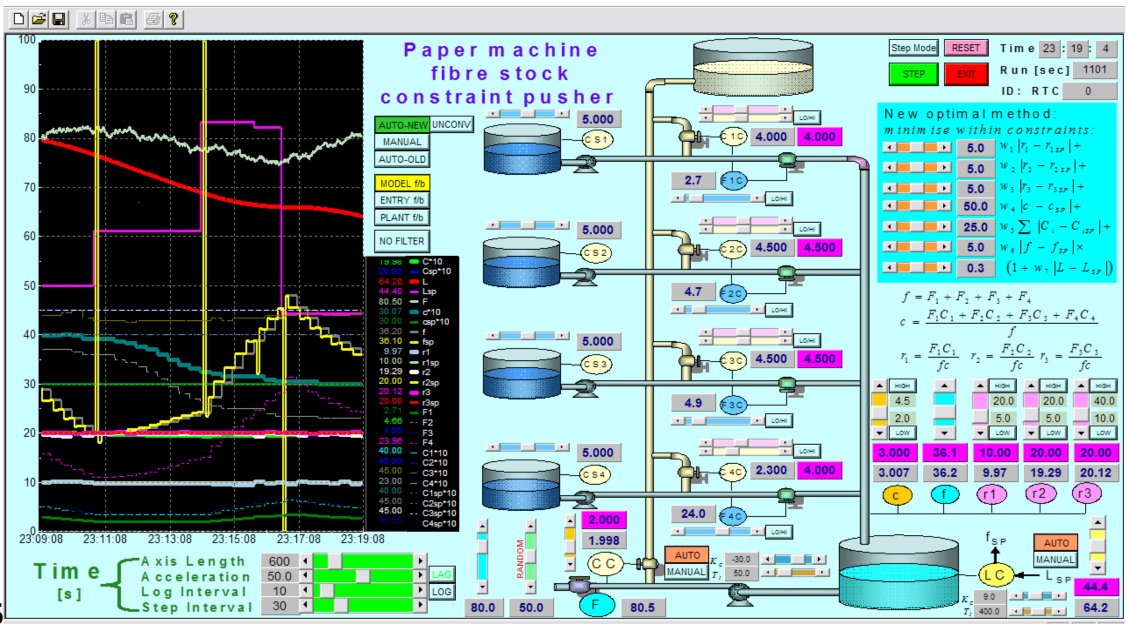
5.2.4 Code Extracts

• The only difference to two *independent* PID loops (as in section 4.2) occurs ahead of the two loop solutions on each time step, where the following code is inserted:

```
// Interpret present values
if (b_Auto[0])              // VPC has been switched to "Auto"
{
        b_Auto[1] = TRUE;   // force the PC onto “Auto”
        m_P_sp = m_V1;          // pass down remote setpoint
}
else
{
        m_V1 = m_P;         // for bumpless startup, keep VPC output variable close to present value
}
```

5.3 PSH - Constraint pusher for optimal use of paper machine fibre stocks

Understanding and tuning a constrained optimisation with a performance objective function:



$$w_1|r_1 - r_{1SP}| + w_2|r_2 - r_{2SP}| + w_3|r_3 - r_{3SP}| + w_4|c - c_{SP}| + w_5\sum|C_i - C_{iSP}| + w_6|f - f_{SP}| \times (1 + w_7|L - L_{SP}|)$$

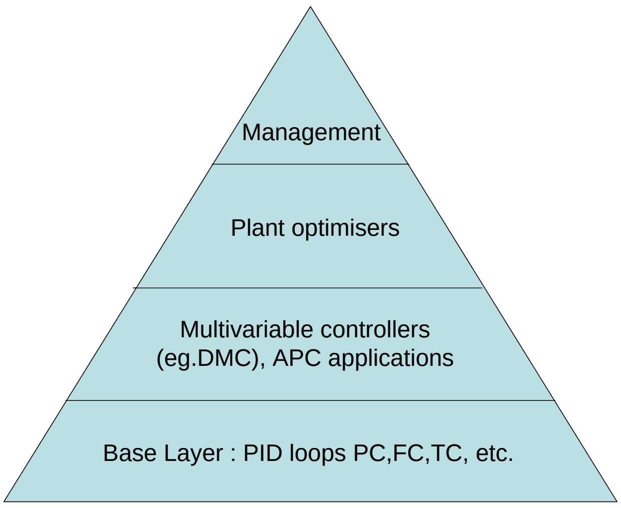
$$f = F_1 + F_2 + F_3 + F_4$$

$$c = \frac{F_1C_1 + F_2C_2 + F_3C_3 + F_4C_4}{f}$$

$$r_1 = \frac{F_1C_1}{fc} \quad r_2 = \frac{F_2C_2}{fc} \quad r_3 = \frac{F_3C_3}{fc}$$

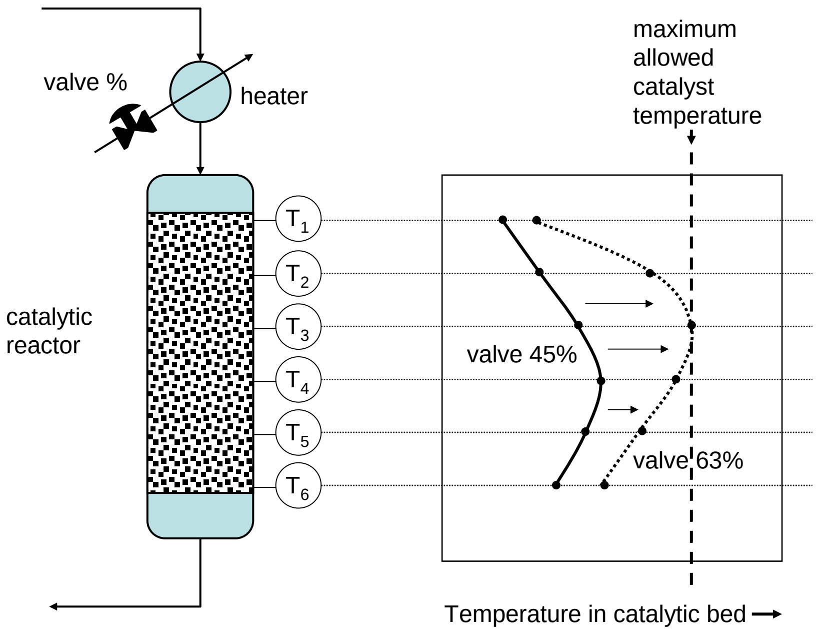
*f*_5.6

With the access to to all variables in modern DCS or SCADA systems, optimisers (and *constrained* optimisers as in this case), are becoming more common. Plant optimisers are viewed as being near the top of an hierarchical process control pyramid of the form *f*_5.7:


Management
Plant optimisers
Multivariable controllers (eg.DMC), APC applications
Base Layer : PID loops PC,FC,TC, etc.

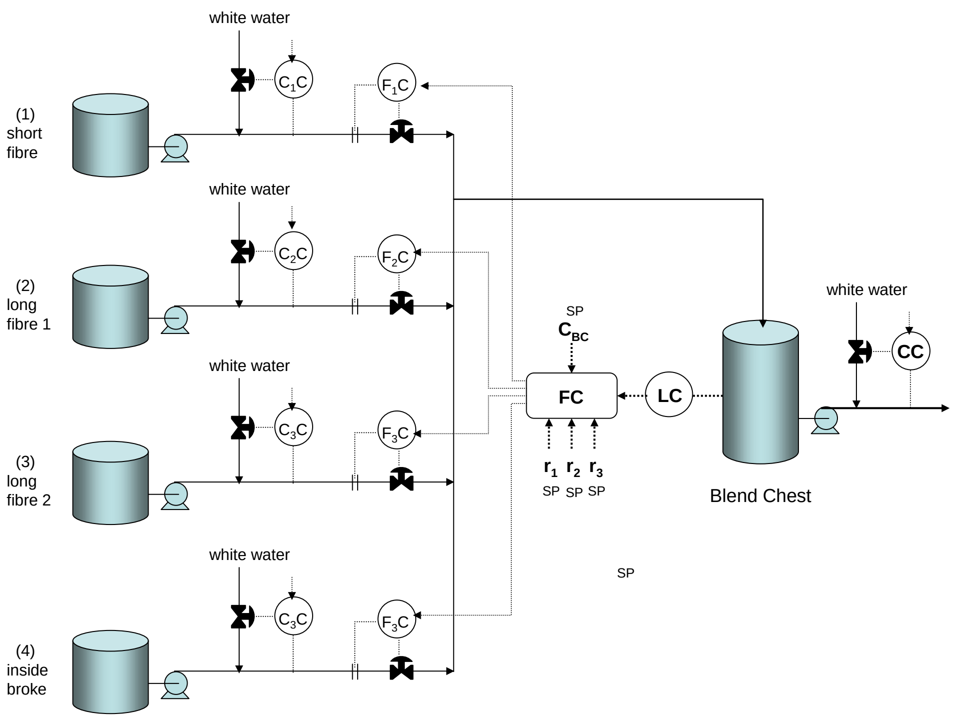
*f*_5.7

Generally speaking, the target information for each strategy moves downward in this pyramid. Actually, the type of optimiser envisaged in the pyramid tends to embrace the entire plant with an economic objective. It usually does not have any dynamic compensation – ie. it assumes a steady-state balance (and often has onboard a data-reconciliation package to create a proper mass and energy balance as a starting point). Such optimisers can run into thousands of variables and equations. However, the type of constrained optimiser we shall consider here fits more logically into the APC layer. Here is a simple example of a constraint-pushing optimiser (*f*_5.8):


valve %
heater
catalytic reactor
T1
T2
T3
T4
T5
T6
maximum allowed catalyst temperature
valve 45%
valve 63%
Temperature in catalytic bed

*f*_5.8

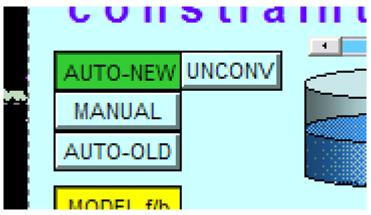
Usually one attempts to operate a catalytic reactor at as high a temperature as possible, in order to maximise reaction rate and conversion. There will be a limit associated with the equipment – typically there is a maximum allowed operating temperature for the catalyst to avoid sintering. So a simple constraint pusher here will find the maximum temperature in the catalytic bed at any time, and keep on opening the heater valve until this maximum reaches the limit.



*f*_5.9

Let us now consider the application at hand (*f*_5.9): the optimal use of a paper machine's fibre stocks. These are suspensions of tiny wood fibres in water, at varying "consistencies", ie. mass fibre per mass water. The basic stock supplies are held in storage tanks, and each stock supply is separately adjusted, as it is drawn, to a setpoint consistency by the addition of white water. It should be noted that the stocks are suspensions of wood-pulp fibres in the range of 1-5% m/m. The white-water is merely wash-water, etc, which has a very low (negligible) fibre content, so it can only be used to adjust the stock consistencies *downwards*.

In this application, the constraining issues, and the scope for optimisation, may be less obvious than the reactor above, so some explanation is required. The constraint-pusher optimiser was developed when deficiencies in an existing fibre stock preparation section were noted, and it is easier to understand the optimiser if one first reviews the old system (which can also be run in this simulator). The NEW or OLD method (or MANUAL) can be selected with the controls below (*f*_5.10). The "UNCONV" flashes yellow if the new solution cannot converge, in which case the solution just reverts to the OLD algorithm. Positioning the system within constraints before switching to the NEW algorithm should avoid this.



*f*_5.10

In the old method, the overall FC algorithm would calculate setpoints for F_1, F_2, F_3 and F_4 (knowing C_1, C_2, C_3, and C_4), in order to achieve the following conditions in the total flow into the Blend Chest:

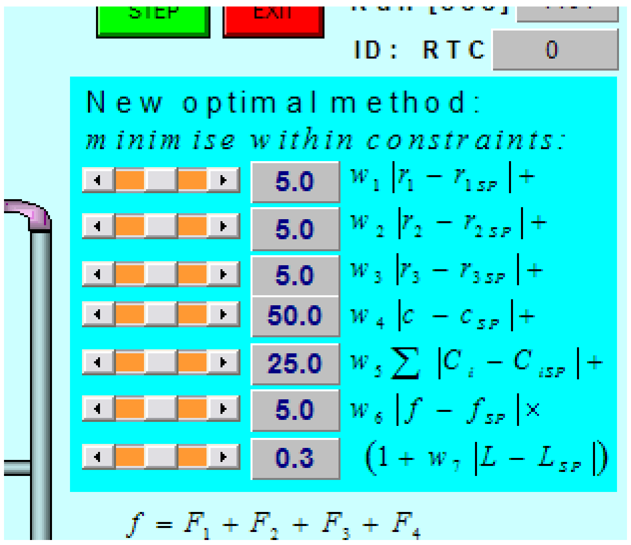
f_{SP}	: total flow demanded by LC
c_{SP}	: consistency of total flow (ie. mass percent of fibre solids in water)

$r_{1\,SP}$	: mass percent of Short Fibre in total fibre
$r_{2\,SP}$	: mass percent of Long Fibre 1 in total fibre
$r_{3\,SP}$	: mass percent of Long Fibre 2 in total fibre
$(100 - r_{1\,SP} - r_{2\,SP} - r_{3SP})$	: mass percent of Inside Broke in total fibre

Plant data records showed frequent upsets in the stock consistency and fibre ratios. These were discovered to arise because the ratio-ing algorithm (FC) was issuing setpoints to the various stock FC's, but had no strategy to deal with the possibility that desired flows and consistencies in the source stocks might not be achievable. For example, a basic stock supply consistency could drop well below the consistency setpoint of the associated consistency controller.

The origin of the new "constraint pusher" was in recognising that eight setpoints could be manipulated (for C_1, C_2, C_3, C_4, F_1, F_2, F_3, F_4), not just four (*for* F_1, F_2, F_3, F_4). If the objectives regarding f, c, r_1, r_2, and r_3 could be properly formulated in an objective function, then C_1, C_2, C_3, C_4, F_1, F_2, F_3, and F_4 could be manipulated until restricted by constraints, yet those remaining unconstrained could still be used for cross-compensation, according to the priorities. On a more basic level, one recognises that eight liquid streams are merely being combined, four from the basic stock vessels, plus one white water flow each. (It is noted that there is some freedom regarding the choice of a source of white water – so the maintenance of typical setpoints for C_1, C_2, C_3, C_4 is included as a weak secondary objective.)

The weightings w_i applied to the various setpoint deviations in the objective function are set in a dedicated block (*f*_5.11). Note the 7th weight, designed to enhance the weight on flow deviations when there is a large deviation from the level setpoint.



ID: RTC 0

New optimal method:

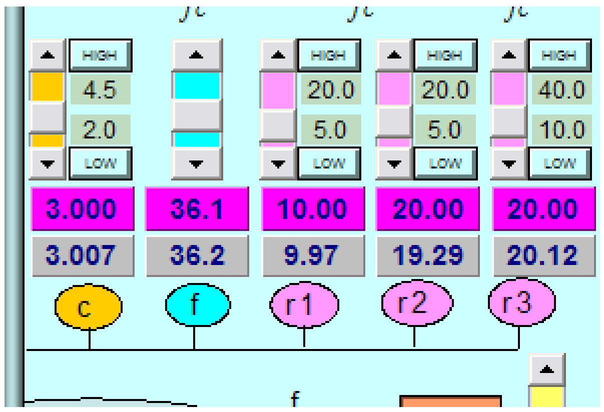
minimise within constraints:

5.0	$w_1 \lvert r_1 - r_{1SP} \rvert +$
5.0	$w_2 \lvert r_2 - r_{2SP} \rvert +$
5.0	$w_3 \lvert r_3 - r_{3SP} \rvert +$
50.0	$w_4 \lvert c - c_{SP} \rvert +$
25.0	$w_5 \sum \lvert C_i - C_{iSP} \rvert +$
5.0	$w_6 \lvert f - f_{SP} \rvert \times$
0.3	$(1 + w_7 \lvert L - L_{SP} \rvert)$

$$f = F_1 + F_2 + F_3 + F_4$$

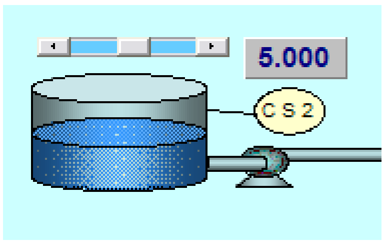
*f*_5.11

The various setpoints above are set in the following block (*f*_5.12), representing conditions of the combined stream entering the Blend Chest:



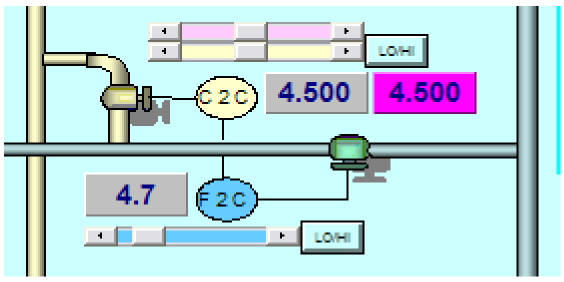
*f*_5.12

The setpoints are highlighted in pink, and are set by the corresponding scrollbars. The actual value of the controlled variable is shown below the setpoint value. The (fixed) set upper and lower constraints for *c*, r_1, r_2 and r_3 are shown on a light grey background next to their scrollbars. Adjacent to the top and bottom of each scrollbar is an indicator which shows the status of each of these 4 variables with respect to its constraints. If it is within its constraints, both indicators are off. If it is *at* either constraint, the associated indicator flashes green. If it violates either constraint, the associated indicator flashes red. The NEW algorithm recognises these constraints, so the indicators will be useful to check its performance.



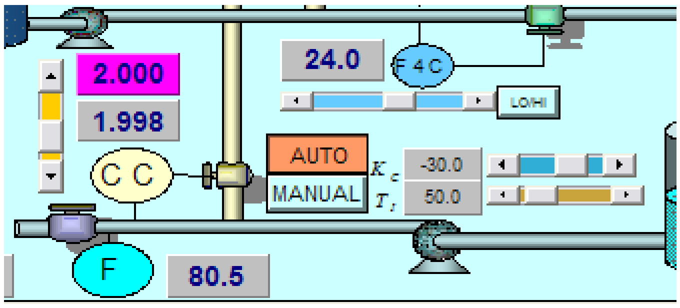
*f*_5.13

The consistency of each of the supply stocks can be set using the scrollbar shown above (*f*_5.13). Take note of the arrangement around the individual flow and consistency controllers for each stock, shown below (*f*_5.14). The desired setpoint for this stock consistency is set on the upper pink scrollbar, and this consistency is shown numerically highlighted in pink. The actual consistency is shown on the white scrollbar (and numerically to the left of the setpoint), and will usually differ when the NEW algorithm is running, because it pays little attention to these desired setpoints. Should the actual consistency find itself running up against a constraint, the adjacent "LO/HI" indicator will flash red. This will usually happen when it cannot be adjusted any higher than the supply consistency (or lower than zero). Similarly, the flow setting below the pipe has a "LO/HI" "constrained" indicator, which will flash red when the controlling setpoint has adjusted the scrollbar setpoint to either end of its range.

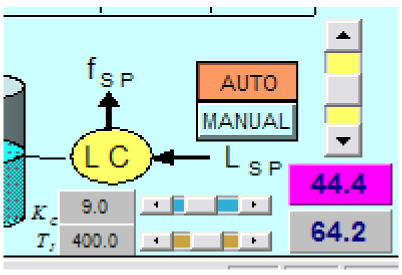


*f*_5.14

PI controllers (T_D =0) are provided for the final consistency trim control (*f*_5.15), and the Blend Chest level control (*f*_5.16). Again, the setpoint is highlighted in pink above the value of the actual feedback variable. The two vertical scrollbars apply to the setpoints. Clearly, the setpoint for *c* in the combined flow into the Blend Chest would have to be set *higher* than the desired final consistency trim setpoint, since the only means of control is by addition of white water.

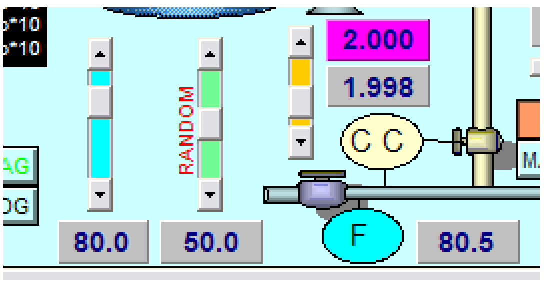


*f*_5.15



*f*_5.16

After the final consistency controller, a flow setting can be manipulated to simulate demand conditions on the flow of prepared stock. There is a mean setting (left hand scroll bar) and a zero-mean RANDOM component of size determined by the RANDOM scrollbar (*f*_5.17). The resultant flow is indicated to the right of the "F" symbol below the pipe. Note that this final flow includes the adjacent white-water addition, so if the white water flow fluctuates, the flow drawn from the Blend Chest will fluctuate inversely.



*f*_5.17

5.3.1 Typical settings

	LC (PID):			
Stock:	L_{SP} = 70	**Setpoints**	**Weights**	Time Axis = 1800
C_{1SP} = 4.0	K_C = 25	c_{SP} = 3	w_1 = 5	Time accel. = 50
C_{2SP} = 4.5	T_I = 250	r_{1SP} = 10	w_2 = 5	Log Interval = 30

C_{3SP} = 4.5	(T_D = 0)	r_{2SP} = 20	w_3 = 5	Step Interval = 60
C_{4SP} = 4.0		r_{3SP} = 20	w_4 = 50	
Supply:	**CC (PID):**		w_5 = 25	
C_{S1} = 5.0	C_{SP} = 2.0		w_6 = 5	
C_{S2} = 5.0	K_C = -30		w_7 = 0.3	
C_{S3} = 5.0	T_I = 50			
C_{S4} = 5.0	(T_D = 0)			
	Flow drawn Mean *F*= 80			
	Flow drawn RANDOM= 50			

5.3.2 Theory

The combined stream property arriving at the Blend Chest can be calculated as follows:

$$f = F_1 + F_2 + F_3 + F_4 \tag{5.1}$$

$$c = \frac{F_1C_1 + F_2C_2 + F_3C_3 + F_4C_4}{f} \tag{5.2}$$

$$r_1 = \frac{F_1C_1}{fc} \tag{5.3}$$

$$r_2 = \frac{F_2C_2}{fc} \tag{5.4}$$

$$r_3 = \frac{F_3C_3}{fc} \tag{5.5}$$

It is clear that the variables to be manipulated are non-linearly related to the setpoint objectives. Thus a constrained optimisation based on Linear Programming is not possible. Define an objective function as follows

$$\begin{aligned} J\left(C_1,C_2,C_3,C_4,F_1,F_2,F_3,F_4\right) \; = \; & w_1\left|r_1 - r_{1SP}\right| + \\ & w_2\left|r_2 - r_{2SP}\right| + \\ & w_3\left|r_3 - r_{3SP}\right| + \\ & w_4\left|c - c_{SP}\right| + \\ & w_5\sum_{i=1}^{4}\left|C_i - C_{iSP}\right| + \\ & w_6\left|f - f_{SP}\right| \times \left(1 + w_7\left|L - L_{SP}\right|\right) \end{aligned} \tag{5.6}$$

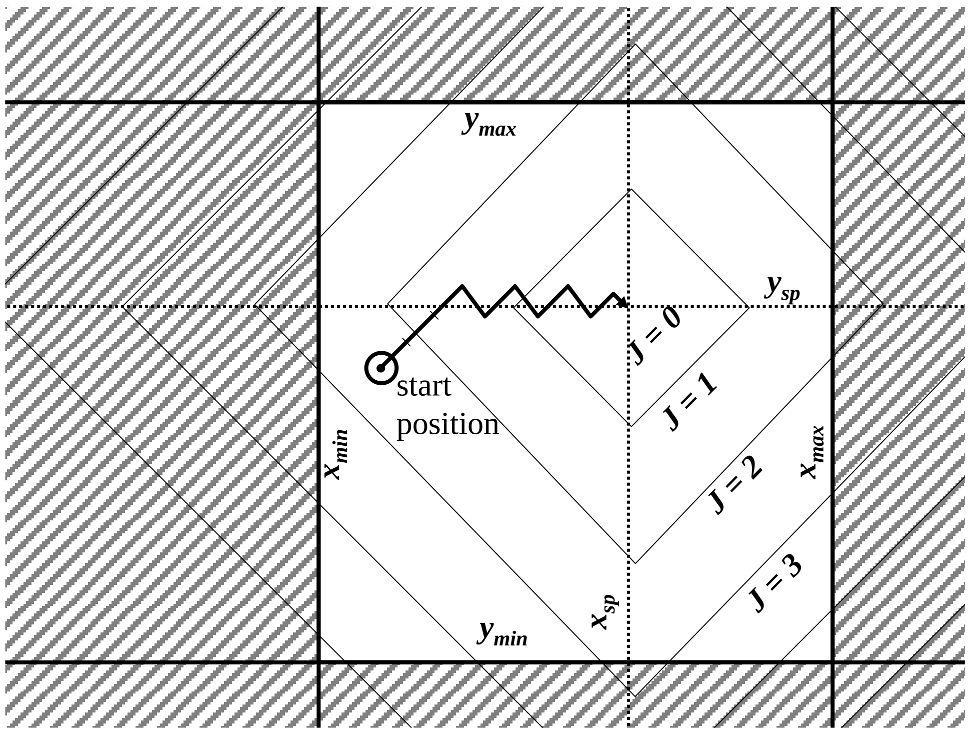
where we aim to minimise *J* by suitable choice of C_1, C_2, C_3, C_4, F_1, F_2, F_3, and F_4 within the constraints on all of these variables. It has been chosen to use the *magnitude* of the deviations rather than the *square* (not for any particular reason). The intention with the 6th term is merely to increase the weighting on compliance with the total flow setpoint (from the LC) in the event that deviations of the level *L* from L_{SP} happen to be large. The variable *L* is just a measurement, and not influenced by the free variables in the optimisation. Notice the w_5

term which aims to have a weak attraction of the individual controlled stock consistencies towards their setpoints. In this way the white water will not be drawn all with one of the stocks.

A somewhat crude but reasonably effective constrained optimisation search has been devised for this problem. The search proceeds outwards from the last C_1, C_2, C_3, C_4, F_1, F_2, F_3, and F_4 position, adjusting each of these variables in turn (positive and negative fractions of its range, before returning it to its original value), to find which of the 16 possible results (excluding any that fall outside constraints) gives the biggest reduction in *J*. Then the centre position moves on to this best point, and the search is repeated. When there is no smaller *J*, or its value oscillates within a small tolerance, the search stops.

As an example, consider the following 2-dimensional optimisation (*f*_5.18):

$$\begin{aligned} &\textit{Minimise} \quad J(x,y) = |x - x_{SP}| + |y - y_{SP}| \\ &\textit{for:} \quad x_{\min} < x < x_{\max} \\ &\qquad\quad y_{\min} < y < y_{\max} \end{aligned} \tag{5.7}$$



*f*_5.18

5.3.3 Tasks

(1) Switch the system to "AUTO-OLD" (*f*_5.10), reduce the RANDOM component on the final flow to zero, and wait for the system to become steady. Notice that once the old algorithm has got the fibre ratios right, it has no further freedom to adjust the combined flow *c* into the Blend Chest. All of the control is done by the final *C* trim control. Reduce some of the individual consistency setpoints slightly, and see the *c* value drop.

(2) The Inside Broke contributes a lot of the fibre. See what happens when you reduce its supply consistency slowly down towards 1.0%. Eventually it is impossible to meet both the fibre ratio requirement, and the total flow requirement, so the solution fails. (This version does not have enough protection against this at present, so the result is numerically catastrophic!).

(3) Repeat the test in (3) for a higher final consistency setpoint (say 3%), and you will see that final consistency cannot be controlled, because the Blend Chest consistency drops below 3% (before the OLD algorithm actually fails). [If you restart the solution with "Reset", you will have to reposition the cursors moved so far]. If you cannot get steady operation like this, just raise the final consistency setpoint and you will see that the maximum is determined by c.

(4) With the offset final consistency in (3), set the c setpoint higher than the final consistency setpoint, and switch from the "AUTO-OLD" to the "AUTO-NEW" algorithm. If the "UNCONV" indicator flashes yellow, it is probably because you have started outside of the constraints, and the solution will be reverting to the OLD method by default. Check which constraint is flashing and temporarily move the associated setpoint until the system is within constraints, in which case the NEW solution will take over. Then you can return setpoints to where they were.

(5) With the NEW algorithm still operating, with zero RANDOM component on the final flow, check the responses to steps in the setpoints of c, r_1, r_2, r_3, L, and C.

(6) Now raise the final flow RANDOM component to about 50, and observe the general performance of the NEW algorithm in the face of these disturbances. Check how it handles drops in the stock supply consistencies.

(7) Adjust the objective function weights w_1, w_2, w_3, w_4, w_5, w_6, and w_7 , and see if you get predictable effects. For example, see how the level-control performance improves with w_6, and w_7.

5.3.4 Code Extracts

- The following code is executed on every time-step:

```
//renormalise fibre ratios if necessary
double tot1to3 = m_r1+m_r2+m_r3;
if (tot1to3>100)
{
        m_r1 = 100*m_r1/tot1to3;
        m_r2 = 100*m_r2/tot1to3;
        m_r3 = 100*m_r3/tot1to3;
}

if (theApp.FIRSTLOOP)
{
        // initialise stacks for PSH loops
        r2[0] = r1[0] = r0[0] = m_Lbc;
        x2[0] = x1[0] = x0[0] = m_Lbc;
        m2[0] = m1[0] = m_fopt;
        r2[1] = r1[1] = r0[1] = m_Cfinal;
        x2[1] = x1[1] = x0[1] = m_Cfinal;
        m2[1] = m1[1] = m_fww;
        theApp.FIRSTLOOP=0;
}

while (((theApp.t-tlast_PSH)>=dtPSH) | (theApp.b_FORCE_STEP))
{
        if (!theApp.b_FORCE_STEP)
        {
                tlast_PSH += dtPSH;    // must catch up by looping more than once if necessary
        }
        else
        {
                tlast_PSH = theApp.t;        // to SYNCHRONISE manual moves : b_FORCE_STEP
                                             // was set TRUE on StepMode button
                theApp.b_FORCE_STEP = FALSE;
        }

        // Interpret present values
        x[0] = m_Lbc;
        r[0] = m_Lbc_sp;
        m[0] = m_fopt;
        x[1] = m_Cfinal;
        r[1] = m_Cfinal_sp;
        m[1] = m_fww;


        for (i=0;i<nPSH;i++)
        {
                if (!b_Auto[i])
                {
                        // Not on - reset to starting values
                        r2[i] = r1[i] = r0[i] = x[i];
                        x2[i] = x1[i] = x0[i] = x[i];
                        m2[i] = m1[i] = m[i];
                }
                else
                {
                        // On AUTO : calculate PSH action!
                        // first cascade stack
                        r2[i] = r1[i];
                        r1[i] = r0[i];
                        r0[i] = r[i];
                        x2[i] = x1[i];
                        x1[i] = x0[i];
                        x0[i] = x[i];
                        m2[i] = m1[i];
                        m1[i] = m[i];

                        // Action:
                        m[i] = ( - a1[i] * m1[i]
                                - a2[i] * m2[i]
                                + b0[i] * r0[i]
                                + b1[i] * r1[i]
                                + b2[i] * r2[i]
                                - c0[i] * x0[i]
                                - c1[i] * x1[i]
                                - c2[i] * x2[i] ) / a0[i];
                        // back to actual variables as appropriate
                        if(i==0) m_fopt_sp = __max(__min(m[0],max_fopt),0);
                        if(i==1) m_fww = __max(__min(m[1],m_Ffinal),0);   // cannot be bigger than
                                                                           // final combined flow!
                }
        }
}

// Constraint Pusher Optimiser.......
while ((theApp.t-tlast_Pusher) >= dtPSH_Pusher)
{
        tlast_Pusher = theApp.t;
        // Now do the Flow Apportioning
        if (b_ManualOldNew)
        {
                // No control
                b_unconverged = FALSE;
                m_C1 = __min(m_C1sp,m_Cs1);
                m_C2 = __min(m_C2sp,m_Cs2);
                m_C3 = __min(m_C3sp,m_Cs3);
                m_C4 = __min(m_C4sp,m_Cs4);
        }
        else if(b_AutoOld)
        {
                // Old method
                b_unconverged = FALSE;
Default_Strategy:
                m_C1 = __min(m_C1sp,m_Cs1);
                m_C2 = __min(m_C2sp,m_Cs2);
                m_C3 = __min(m_C3sp,m_Cs3);
                m_C4 = __min(m_C4sp,m_Cs4);

                cMatrix A, x, y, temp;
                A.Init(4,4);
                x.Init(4,1);
                y.Init(4,1);
                double m_r4_sp = 100 - m_r1_sp - m_r2_sp - m_r3_sp;
                A(1,1) = m_r1_sp-100         ; A(1,2) = m_r1_sp*m_C2/m_C1    ;
                A(1,3) = m_r1_sp*m_C3/m_C1   ; A(1,4) = m_r1_sp*m_C4/m_C1;
                A(2,1) = m_r2_sp*m_C1/m_C2   ; A(2,2) = m_r2_sp-100          ;
                A(2,3) = m_r2_sp*m_C3/m_C2   ; A(2,4) = m_r2_sp*m_C4/m_C2;
                A(3,1) = m_r3_sp*m_C1/m_C3   ; A(3,2) = m_r3_sp*m_C2/m_C3    ;
                A(3,3) = m_r3_sp-100         ; A(3,4) = m_r3_sp*m_C4/m_C3;
                A(4,1) = m_r4_sp*m_C1/m_C4+100; A(4,2) = m_r4_sp*m_C2/m_C4+100;
                A(4,3) = m_r4_sp*m_C3/m_C4+100; A(4,4) = m_r4_sp;
                y(4,1) = 100*m_fopt_sp;
                temp = A.Inv();
                x = temp * y;
                m_F1 = __min(__max(x(1), 0), max_F1);
                m_F2 = __min(__max(x(2), 0), max_F2);
                m_F3 = __min(__max(x(3), 0), max_F3);
                m_F4 = __min(__max(x(4), 0), max_F4);
        }
        else
        {
                // New Optimal Constraint-pusher method
                //====================================
                //Inputs
                cMatrix x, xmin, xmax, dx;
                x.Init(8,1); xmin.Init(8,1); xmax.Init(8,1); dx.Init(8,1);
                x(1) = m_F1; xmin(1) = 0; xmax(1) = max_F1       ; dx(1) = max_F1/200;
                x(2) = m_F2; xmin(2) = 0; xmax(2) = max_F2       ; dx(2) = max_F2/200;
                x(3) = m_F3; xmin(3) = 0; xmax(3) = max_F3       ; dx(3) = max_F3/200;
                x(4) = m_F4; xmin(4) = 0; xmax(4) = max_F4       ; dx(4) = max_F4/200;
                x(5) = m_C1; xmin(5) = 0; xmax(5) = __min(max_C1,m_Cs1)      ; dx(5) = max_C1/200;
                x(6) = m_C2; xmin(6) = 0; xmax(6) = __min(max_C2,m_Cs2)      ; dx(6) = max_C2/200;
                x(7) = m_C3; xmin(7) = 0; xmax(7) = __min(max_C3,m_Cs3)      ; dx(7) = max_C3/200;
                x(8) = m_C4; xmin(8) = 0; xmax(8) = __min(max_C4,m_Cs4)      ; dx(8) = max_C4/200;
                //Outputs
                cMatrix y, ysp, ymin, ymax;
                y.Init(5,1); ysp.Init(5,1); ymin.Init(5,1); ymax.Init(5,1);
                y(1) = m_r1;      ysp(1) = m_r1_sp;      ymin(1) = m_Lr1c;      ymax(1) = m_Hr1c;
                y(2) = m_r2;      ysp(2) = m_r2_sp;      ymin(2) = m_Lr2c;      ymax(2) = m_Hr2c;
                y(3) = m_r3;      ysp(3) = m_r3_sp;      ymin(3) = m_Lr3c;      ymax(3) = m_Hr3c;
                y(4) = m_copt;    ysp(4) = m_copt_sp;    ymin(4) = m_Lcc;       ymax(4) = m_Hcc;
                y(5) = m_fopt;    ysp(5) = m_fopt_sp;    ymin(5) = 0;           ymax(5) = max_fopt;
                double tolerance = 1e-8;
                double J = m_w1*fabs(y(1)-m_r1_sp) + m_w2*fabs(y(2)-m_r2_sp) + m_w3*fabs(y(3)-m_r3_sp) +
                        m_w4*fabs(y(4)-m_copt_sp)+
                        m_w5*( fabs(x(5)-m_C1sp) + fabs(x(6)-m_C2sp) + fabs(x(7)-m_C3sp) + fabs(x(8)-m_C4sp) ) +
                        m_w6*fabs(y(5)-m_fopt_sp)*(1+ m_w7*fabs(m_Lbc-m_Lbc_sp));
                b_unconverged = TRUE;
                double b_nosolution = TRUE;
                double Jminlast = J;
                double Jmin = J;
                int jmin;
                double sensemin = 0;
                for (int iter=1; iter<=1000; iter++)
                {
                        jmin=0;
                        for (j=1; j<=8; j++) for (int k=0; k<=1; k++)
                        {
                                if (x(j)<xmin(j)) x(j) = xmin(j);  // rescue it back
                                if (x(j)>xmax(j)) x(j) = xmax(j);  // rescue it back
                                double sense = k*2-1;
                                x(j) += sense*dx(j);
                                if((x(j)>=xmin(j)) & (x(j)<=xmax(j)))
                                {
                                        y(5) = x(1)+x(2)+x(3)+x(4);  // f
                                        y(4) = ( x(1)*x(5)+x(2)*x(6)+x(3)*x(7)+x(4)*x(8) ) / (y(5) + 1e-5); //c
                                        y(3) = 100*x(3)*x(7)/( y(4)*y(5) + 1e-5); //r1
                                        y(2) = 100*x(2)*x(6)/( y(4)*y(5) + 1e-5); //r2
                                        y(1) = 100*x(1)*x(5)/( y(4)*y(5) + 1e-5); //r3
                                        if ((y(1)>=ymin(1)) & (y(2)>=ymin(2)) &(y(3)>=ymin(3)) &
                                                (y(4)>=ymin(4)) & (y(5)>=ymin(5)) &(y(1)<=ymax(1)) &
                                                (y(2)<=ymax(2)) &(y(3)<=ymax(3)) &(y(4)<=ymax(4)) &
                                                 (y(5)<=ymax(5)) )
                                        {
                                          b_nosolution = FALSE;
                                          J =     m_w1*fabs(y(1)-m_r1_sp) + m_w2*fabs(y(2)-m_r2_sp) +
                                                  m_w3*fabs(y(3)-m_r3_sp) +
                                                  m_w4*fabs(y(4)-m_copt_sp)+
                                                  m_w5*( fabs(x(5)-m_C1sp) +
                                                  fabs(x(6)-m_C2sp) + fabs(x(7)-m_C3sp) + fabs(x(8)-m_C4sp) ) +
                                                   m_w6*fabs(y(5)-m_fopt_sp)*(1+ m_w7*fabs(m_Lbc-m_Lbc_sp));
                                          if (J<Jmin)
                                          {
                                                  Jmin = J;
                                                  jmin = j;
                                                  sensemin = sense;
                                          }
                                        }
                                }
                                x(j) -= sense*dx(j);  // return to datum;
                        }
                        if (b_nosolution) goto Default_Strategy;
                        // converged?
                        if ((jmin == 0) | (fabs(Jmin-Jminlast)<tolerance))
                        {
                                b_unconverged = FALSE;
                                break;
                        }
                        else
                        {
                                Jminlast = Jmin;
                                x(jmin) += sensemin*dx(jmin);  // move in this direction
                        }
                }
                if(b_unconverged) goto Default_Strategy;
                m_F1 = x(1);
                m_F2 = x(2);
                m_F3 = x(3);
                m_F4 = x(4);
                m_C1 = x(5);
                m_C2 = x(6);
                m_C3 = x(7);
                m_C4 = x(8);
        }
}

m_fopt = m_F1+m_F2+m_F3+m_F4;
```

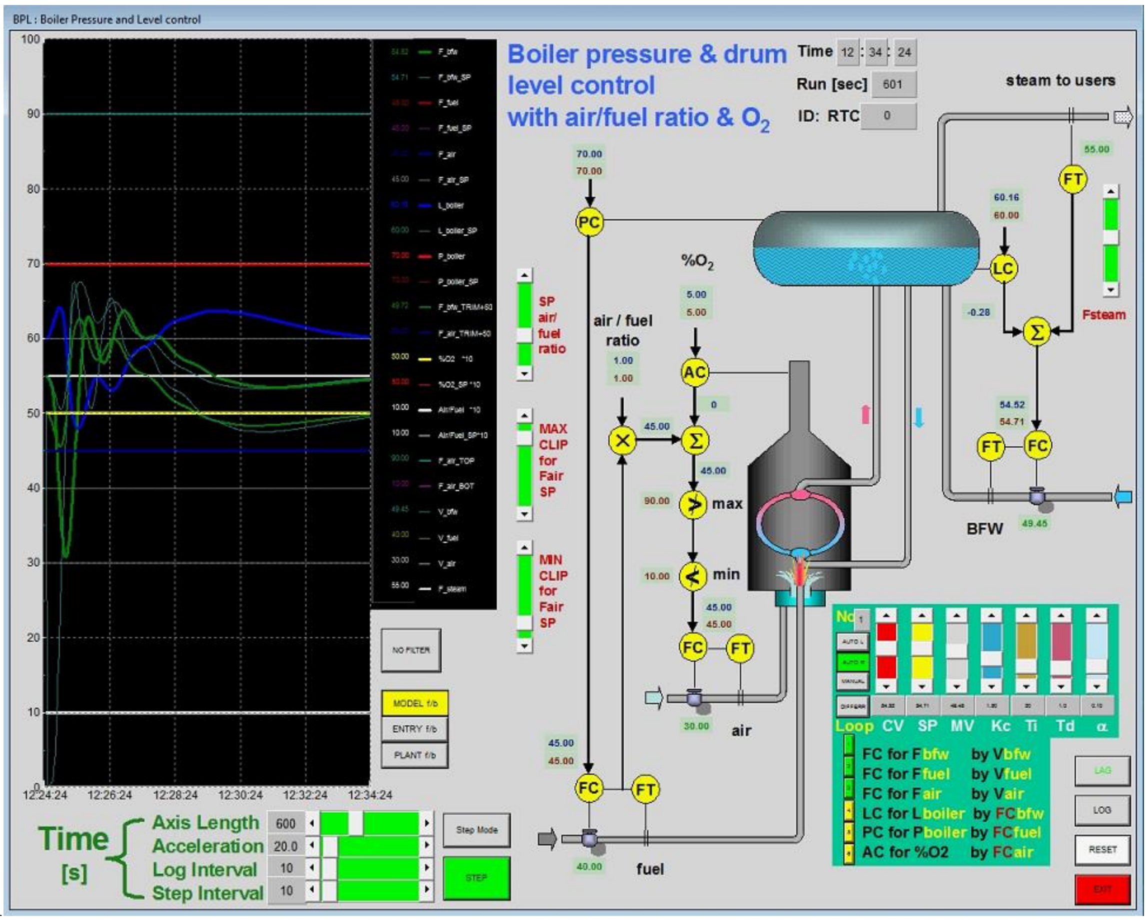
```
m_copt = ( m_F1*m_C1+m_F2*m_C2+m_F3*m_C3+m_F4*m_C4 ) / (m_fopt+1e-5);
m_r1 = 100 * m_F1*m_C1 / (m_fopt*m_copt+1e-5);
m_r2 = 100 * m_F2*m_C2 / (m_fopt*m_copt+1e-5);
m_r3 = 100 * m_F3*m_C3 / (m_fopt*m_copt+1e-5);

// Model step only

while ((theApp.t-theApp.tlast_ModelStep) >= dtPSH_StepResp/(double)Ninterp)
{
        if (theApp.b_Model)
        {
                //Do a proper calculation because there are non-linear effects
                double fbc = __max((m_Ffinal-m_fww),0);
                m_Lbc = m_Lbc + (m_fopt - fbc) * (dtPSH_StepResp/(double)Ninterp/60) / 2 ;
                m_Cbc = m_Cbc + ((m_fopt*m_copt - fbc*m_Cbc) *
                                (dtPSH_StepResp/(double)Ninterp/60))/(m_Lbc*10+1e-5);
                double m_Cfinal_inst = fbc*m_Cbc / (m_Ffinal+1e-5);
                double Smooth_Cfinal = 1-10*dtPSH_StepResp/(double)Ninterp/(Npsh*dtPSH_StepResp);
                m_Cfinal = (1-Smooth_Cfinal)*m_Cfinal_inst + Smooth_Cfinal*m_Cfinal;
                m_Lbc    = __max(__min(m_Lbc,max_Lbc),0);
                m_Cfinal = __max(__min(m_Cfinal,max_Cfinal),0);
        }
}
```

Chapter 6 Simulations - Multi-loop strategies

.

6.1 BPL – Boiler pressure and level control

Three-element boiler drum level control with pressure control, air-fuel ratio control and oxygen trim control



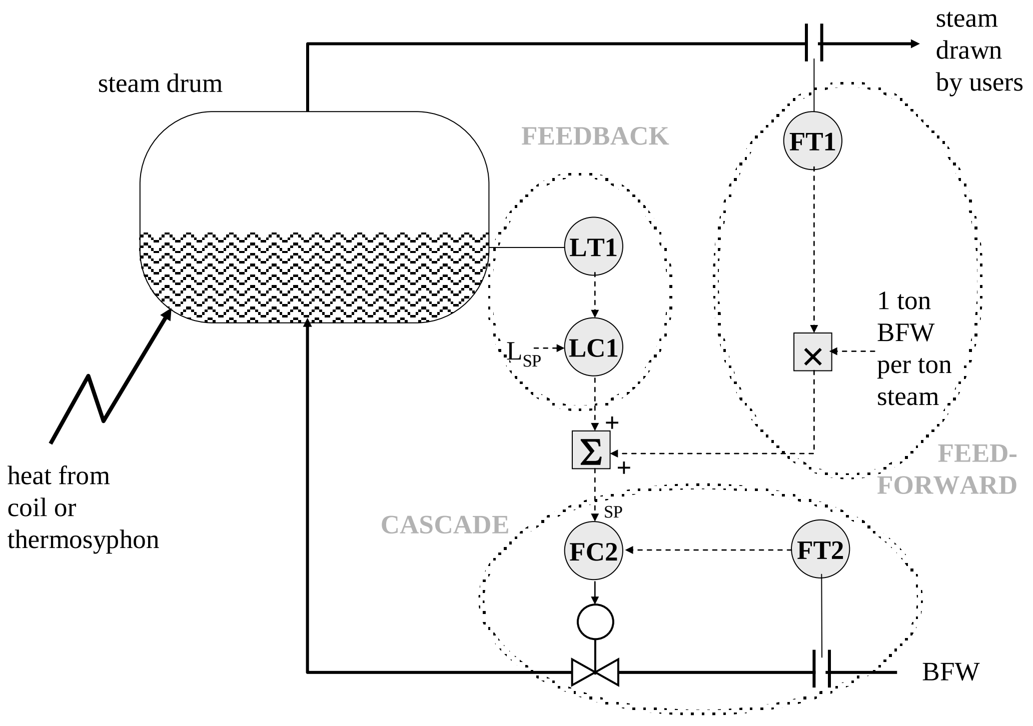
*f*_6.1

6.1.1 Typical settings

FC for	**FC for**	**FC for**	**LC for**	**PC for**	**AC for**
$\boldsymbol{F_{BFW}}$	$\boldsymbol{F_{FUEL}}$	$\boldsymbol{F_{AIR}}$	$\boldsymbol{L_{BOILER}}$	$\boldsymbol{P_{BOILER}}$	**%$\boldsymbol{O_2}$**
by $\boldsymbol{V_{BFW}}$	**by** $\boldsymbol{V_{FUEL}}$	**by** $\boldsymbol{V_{AIR}}$	**by** $\boldsymbol{F_{BFWSP}}$	**by** $\boldsymbol{F_{FUELSP}}$	**by** $\boldsymbol{F_{AIRSP}}$
K_C=1.5	K_C=1.0	K_C=1.1	K_C=1.0	K_C=1.0	K_C=1.5
τ_I=30	τ_I=30	τ_I=30	τ_I=200	τ_I=100	τ_I=50
τ_D=1.0	τ_D=1.0	τ_D=1.0	τ_D=0	τ_D=0	τ_D=0
α=0.1	α=0.1	α=0.1	α=0.1	α=0.1	α=0.1
F_{BFWSP}=55	F_{FUELSP}=45	F_{AIRSP}=45	L_{SP}=60	$P_{BOILERSP}$=70	%O_{2SP}=5
REMOTE	*REMOTE*	*REMOTE*	*LOCAL*	*LOCAL*	*LOCAL*

6.1.2 Theory

The figure below shows the well-known "3-element" control scheme for regulation of the water level in a boiler drum (*f*_6.2). The three elements are clearly feedback, feedforward and a supervised flow control loop.



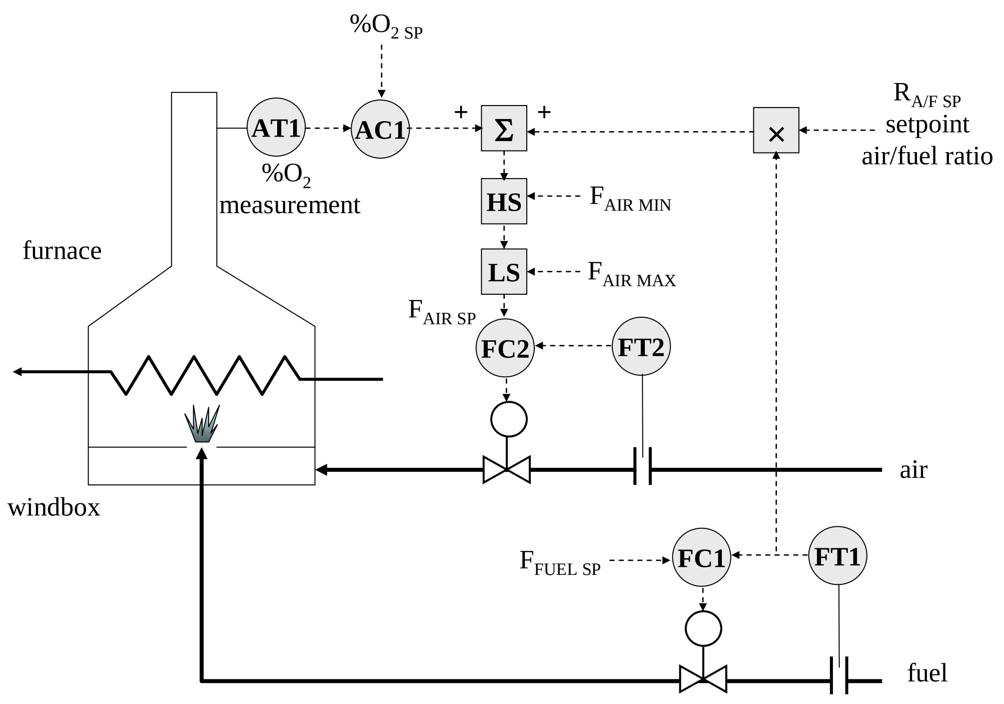
*f*_6.2

All feedforward controllers require a "model", and in this case it is seen to be a very simple one, with one ton of *steam drawn* translating exactly to one ton of *BFW to be supplied.* Since the system integrates, and the flow measurements cannot be perfect, a feedback "trim" is essential, if for nothing else, just to get the level to its initial setpoint!

The delegation of the task of maintaining a desired BFW flow-rate to a slave flow control loop isolates the rest of the algorithm from such factors affecting the BFW flow as BFW and drum pressure fluctuations, and indeed non-linearity of the valve itself. The direct summation of feedforward and feedback BFW demands may appear to require twice as much BFW as necessary, until one recalls that the entire algorithm is working on the basis of *deviations* from the initial “switch on” condition.

Regarding the firing of the furnace, the amount of air supplied for combustion must always be in excess – ie. there should be a non-zero residual O_2 concentration in the flue gas. This is to ensure that ignition does not occur somewhere in the exit ductwork, or the top of a stack, where the flue gases first encounter oxygen. On the other hand, one does not want to supply too much air for the combustion, because recovery of the heat from that extra air is not easy, so that the furnace efficiency decreases. Maintenance of about 3% residual O_2 on a molar basis (compared to a maximum of 21% in air) seems typical industrially. If there is no actual measurement of oxygen, then a larger margin may be necessary to ensure safety. This is certainly the case when only feedforward control is used, ie. air/fuel ratio control.

The figure *f*_6.3 below shows the full metering control with oxygen trim control presented by Smith and Corripio (1985). This is an air/fuel ratio control with an additive feedback trim from a flue gas oxygen controller. The implication at the summer is that the feedback signal from the AC will request a zero adjustment if the ratio controller, which is working with absolute flow rates, is already achieving the setpoint $\%O_2$. This feedforward-feedback arrangement will minimise $\%O_2$ deviations from setpoint. The air flow rate controller is a slave in the cascade, whilst the fuel flow setpoint will arrive from the operator, or another controller such as the process stream TC or a boiler PC as in the simulation under consideration. The scheme shown includes a high and low “clip” on the setpoint air flow-rate. The low clip is a good safety measure, but the high clip could lead to incomplete combustion at high fuel demands. A slight variation of this scheme is sometimes encountered where instead of supplying an additive trim to the air flow rate setpoint, the oxygen controller manipulates the setpoint air/fuel ratio $R_{AF\,SP}$ directly.

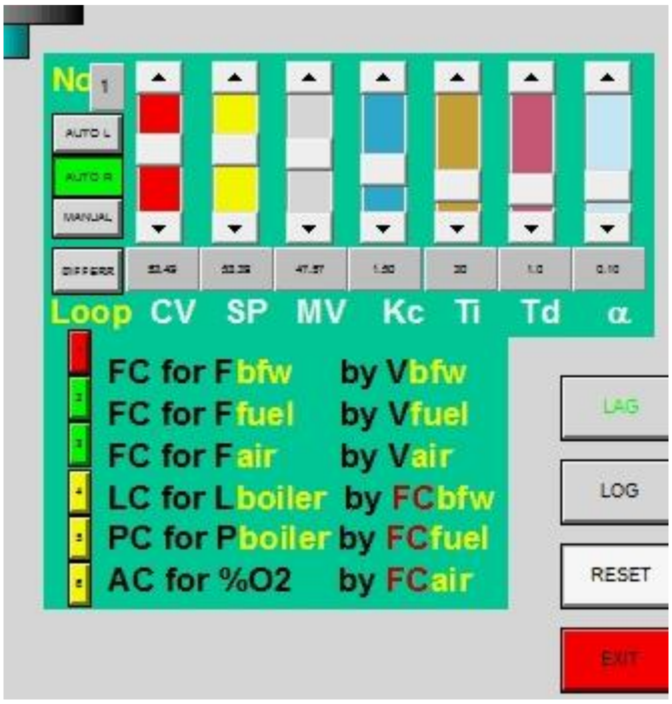


*f*_6.3

The scheme has a drawback to do with the system dynamics. One notes that the fuel flow rate will always change in advance of the air adjustments, simply because there will be dynamic lag of the actual air flow as the ratio controller moves the air flow setpoint in response to the measured fuel flow variations. This situation is described as "*fuel leads air in*" and "*fuel leads air out*". The latter situation is safe, because air will temporarily be in excess. However, the "*fuel leads air in*" is dangerous, because the implication is that there will temporarily be a deficit of oxygen.

6.1.3 Tasks

(1) Start the simulation with the original settings as in section 6.1.1. Let the system reach steady-state. The individual loops are selected using the column of numbered buttons on the left of the diagram below (*f*_6.4). Then the settings pertaining to that loop are shown in the top part of the same diagram. Starting by selecting the supervising loops (6,5,4), switch each to MANUAL. Loops 1,2 and 3 will automatically switch to MANUAL also. Step the boiler feed water valve, by pressing on the coloured part of the MV scrollbar of loop 1, above or below the current cursor position. Explain what happens. Before it goes too far, give the valve a reverse step. Explain what happens.



*f*_6.4

(2) Rescue the situation in (1) by switchting loop 4, the boiler drum level LC, to AUTO L. This will switch the BFW FC to AUTO R, as a slave in the cascade, and bring the level to setpoint. Examine the quality of this control.

(3) Select the air flow FC, loop 3. Switch it to auto with a local setpoint (AUTO L). Step the air flow setpoint a small amount by pressing rapidly on end-arrow of the yellow SP scrollbar, about 3 or 4 times. Explain what you see.

(4) Go to loop 6, the AC for $\%O_2$ control. Switch it to AUTO L. It will take over loop 3 as a slave. Check that $\%O_2$ is brought to its setpoint. Now step the $\%O_2$ setpoint. Is the tuning of this cascade adequate?

(5) Select loop 2, the FC for fuel. Switch it to AUTO L. Step the fuel flow setpoint a small amount by pressing a few times on one of the arrows on the SP scrollbar. Explain what happens. You can recue the situation by switching loop 5, the PC for poiler drum pressure, to AUTO L. The AC for $\%O_2$ control is already on AUTO L, so the $\%O_2$ is taken care of.

(6) Select loop 5, which is the steam drum pressure PC. Step the pressure setpoint a small amount by pressing a few times on one of the end-arrows of the pressure setpoint (SP). Observe the response. Note that the $\%O_2$ in the flue gas becomes quite upset. There are large dips below setpoint.

(7) Unsettled behaviour in (6) is caused by the sequence of reactions from the pressure PC to the fuel FC to theAir/Fuel Ratio PLUS to the resultant $\%O_2$, which is again reacted to by the $\%O_2$ AC via the air FC. Probably the responses of the sequence of controllers need to be slowed down from one to the next. Attempt to improve the overall tuning of this system.

6.1.4 Code extracts

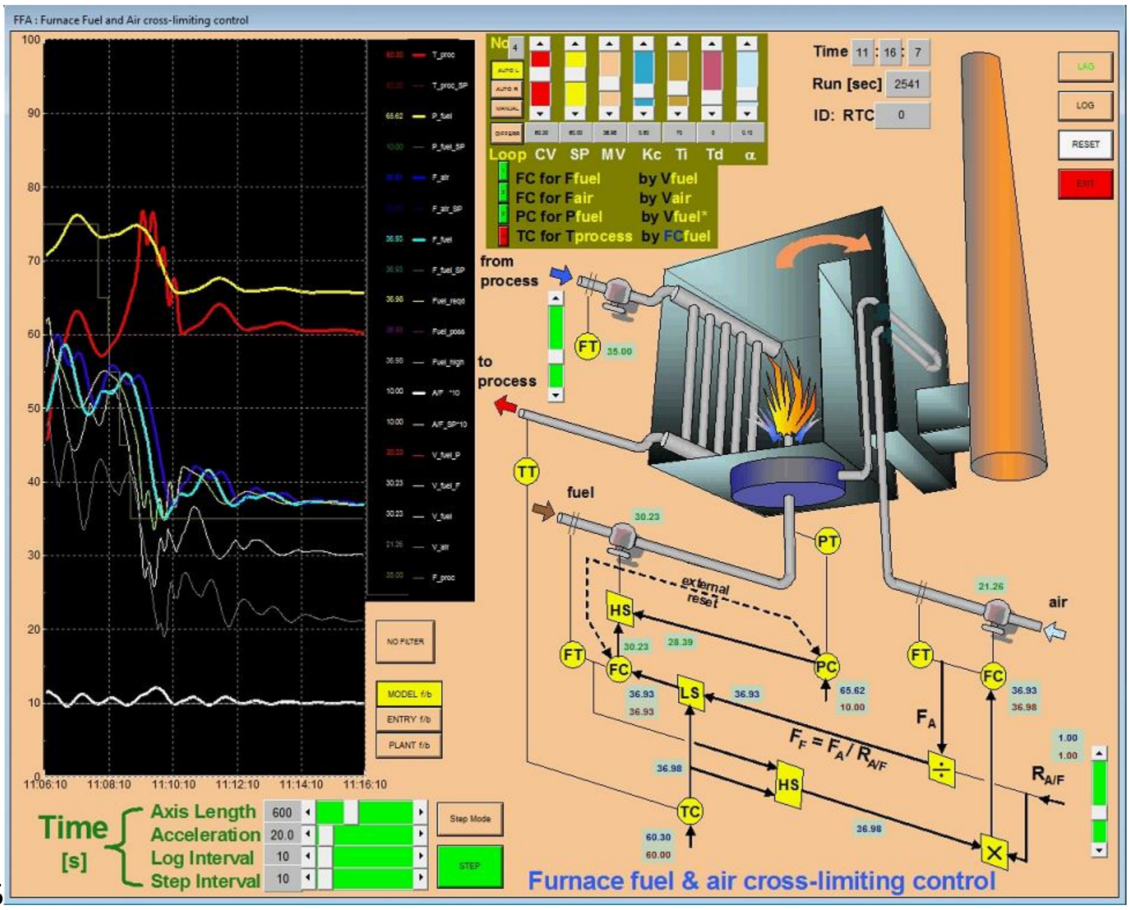
- Cascaded control loop interlocks and calculation of control actions:

```
if (theApp.FIRSTLOOP)
{
        //   .  (omitted code)
        //   .
        // Set out first AutoLocal / AutoRemote selections
        for (j=0; j<nBPL; j++) b_AutoRemote[j] = FALSE;   // Reset all remotes
        for (i=0; i<nBPL; i++)
        {
                if ( b_AutoLocal[i] )
                {
                        for (j=0; j<nBPL; j++)
                        {
                                if (LoopTable[i][j] < 0) b_AutoLocal[j] = FALSE;
                                if (LoopTable[i][j] > 0) b_AutoRemote[j] = TRUE;
                        }
                }
        }

        theApp.FIRSTLOOP=0;
}

while (((theApp.t-tlast_BPL)>=dtBPL) || (theApp.b_FORCE_STEP))
{
        if (!theApp.b_FORCE_STEP)
        {
                tlast_BPL += dtBPL;     // must catch up by looping more than once if necessary
        }
        else
        {
                tlast_BPL = theApp.t;  // to SYNCHRONISE manual moves : b_FORCE_STEP set TRUE on StepMode button
                theApp.b_FORCE_STEP = FALSE;
        }

        // Interpret present values

        // Loop03: Lboiler by Fbfw_sp1
        x[ 3] = m_Lboiler;
        r[ 3] = m_Lboiler_sp;
        m[ 3] = m_Fbfw_sp1;  // was centred to hold the nominal variable in mid-range for plotting
        if (b_AutoLocal[3]) m_Fbfw_sp = __max(__min(m_Fbfw_sp1 + m_Fsteam, max_Fbfw),0);

        // Loop05: Ao2 by Fair_sp1
        x[ 5] = m_Ao2;
        r[ 5] = m_Ao2_sp;
        m[ 5] = m_Fair_sp1;  // was centred to hold the nominal variable in mid-range for plotting
        if (b_AutoLocal[5]) m_Fair_sp = __max(__min(m_Fair_sp1 + m_Rairfuel_sp*m_Ffuel, m_Fair_topclip), m_Fair_botclip);


        // Loop00: Fbfw by Vbfw
        x[ 0] = m_Fbfw;
        r[ 0] = m_Fbfw_sp;
        m[ 0] = m_Vbfw;
        // Loop01: Ffuel by Vfuel
        x[ 1] = m_Ffuel;
        r[ 1] = m_Ffuel_sp;
        m[ 1] = m_Vfuel;
        // Loop02: Fair by Vair
        x[ 2] = m_Fair;
        r[ 2] = m_Fair_sp;
        m[ 2] = m_Vair;
        // Loop04: Pboiler by Ffuel_sp
        x[ 4] = m_Pboiler;
        r[ 4] = m_Pboiler_sp;
        m[ 4] = m_Ffuel_sp;


        for (i=0;i<nBPL;i++)
        {
```

```
                        if ((!b_AutoLocal[i]) && (!b_AutoRemote[i]))
                        {
                                // Not on - reset to starting values
                                if (i==3) {m[i] = m_Ffuel-m_Fsteam;}    // for bumpless start
                                if (i==5) {m[i] = 0;}                                           // for bumpless start
                                r2[i] = r1[i] = r0[i] = x[i];
                                x2[i] = x1[i] = x0[i] = x[i];
                                m2[i] = m1[i] = m[i];
                        }
                        else
                        {
                                // On AUTO (Local or remote) : calculate BPL action!
                                // first cascade stack
                                r2[i] = r1[i];
                                r1[i] = r0[i];
                                r0[i] = r[i];
                                x2[i] = x1[i];
                                x1[i] = x0[i];
                                x0[i] = x[i];
                                m2[i] = m1[i];
                                m1[i] = m[i];

                                // Action:
                                m[i] = ( - a1[i] * m1[i]
                                              - a2[i] * m2[i]
                                                   + b0[i] * r0[i]
                                                   + b1[i] * r1[i]
                                                   + b2[i] * r2[i]
                                                   - c0[i] * x0[i]
                                                   - c1[i] * x1[i]
                                                   - c2[i] * x2[i] ) / a0[i];

                                // back to actual variables
                                switch (i)
                                {

                                case  0: // Loop00: Fbfw by Vbfw
                                        m_Vbfw = __max(__min(m[0],max_Vbfw),0);
                                        break;
                                case  1: // Loop01: Ffuel by Vfuel
                                        m_Vfuel = __max(__min(m[1],max_Vfuel),0);
                                        break;
                                case  2: // Loop02: Fair by Vair
                                        m_Vair = __max(__min(m[2],max_Vair),0);
                                        break;
                                case  3: // Loop03: Lboiler by Fbfw_sp1
                                        m_Fbfw_sp1 = __max(__min(m[3],max_Fbfw_sp1),min_Fbfw_sp1);
                                        break;
                                case  4: // Loop04: Pboiler by Ffuel_sp
                                        m_Ffuel_sp = __max(__min(m[4],max_Ffuel_sp),0);
                                        break;
                                case  5: // Loop05: Ao2 by Fair_sp1
                                        m_Fair_sp1 = __max(__min(m[5],max_Fair_sp1),min_Fair_sp1);  // special clipping;
                                        break;
                                default:
                                        break;
                                }
                        }
                }
        }
```

6.2 FFA – Furnace fuel and air cross-limiting control

An interlocked air/fuel ratio control system which ensures that air leads fuel in and fuel leads air out



f_6.5

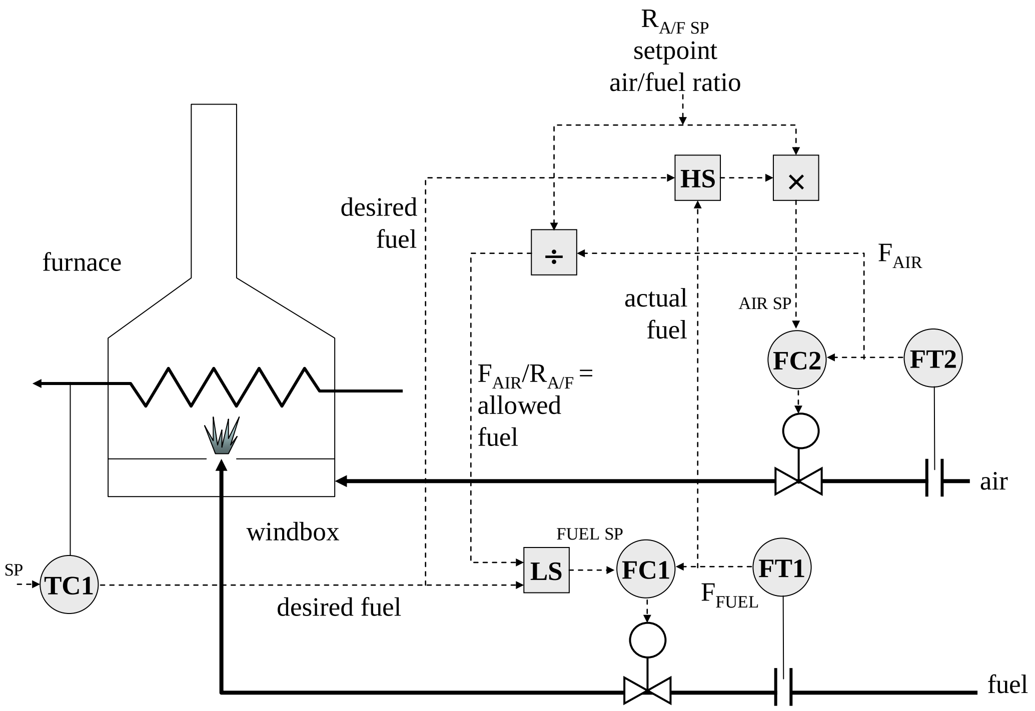
6.2.1 Typical settings

TC for ***T***				

τ_D=0	τ_D=1.0	τ_D=1.0	τ_D=1.0
α=0.1	α=0.1	α=0.1	α=0.1
$T_{FURNACE}$=60	F_{FUELSP}=45	F_{AIRSP}=45	P_{FUELSP}=10
LOCAL	*REMOTE*	*REMOTE*	*LOCAL*

6.2.2 Theory

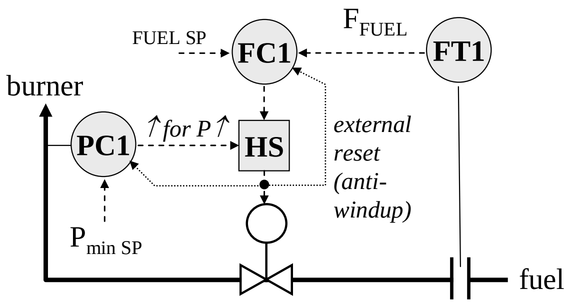
The dynamic lag of the air flow controller in the direct ratioing of air to fuel in section 6.1 was seen to cause a temporary drop in the air/fuel ratio when the fuel flow increased. The cross-limiting scheme shown below (*f*_6.6) overcomes this problem by using both measured flows to ensure a minimum air/fuel ratio at all times.



*f*_6.6

The cross-limiting control scheme is best explained using an example. Consider the situation where the system is initially steady maintaining the correct air/fuel ratio. Now TC1 demands an increase in fuel in order to maintain its setpoint temperature. This demand will be ignored at the LS, because the *desired fuel* will be higher than the *allowed fuel* (which initially will be the *actual fuel*). However, the HS will pass this higher demand rather than the *actual fuel*, so after multiplication by the air/fuel ratio, the air flow will start to rise to match the *desired fuel*. As the actual air flow rises, following its controller setpoint, the cut-off limit (*allowed fuel*) arriving at the LS rises in proportion, gradually allowing the fuel setpoint to increase, and ultimately the *actual fuel* will rise to the *desired fuel*. One notes that "*air leads fuel in*", which is a safe action. Conversely if one follows the sequence of events when *desired fuel* decreases, it will be found that "*air follows fuel out*", which is again a safe action. So in any transient, there will temporarily be a safe excess of air, which will return to the correct ratio when the process settles down again.

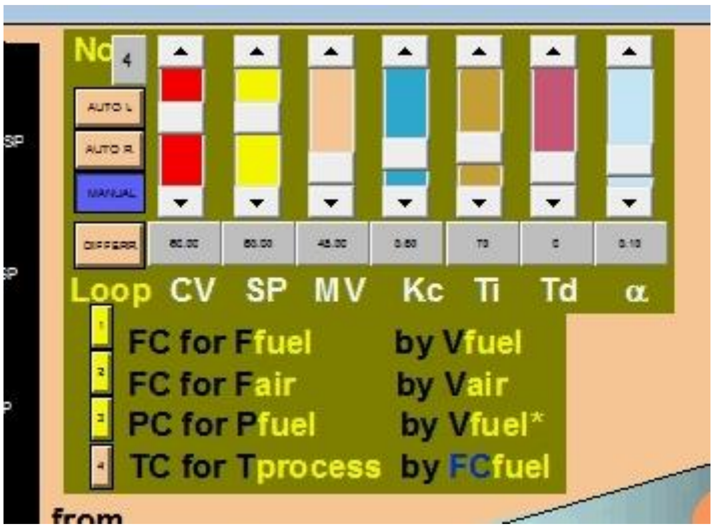
A furnace control scheme as above may also have an additional safety feature to prevent "*flame-out*". It is understandable that the fuel valve could swing to the shut position temporarily, depending on the controller gain, as it seeks to hold the flow setpoint. This would cause an irreversible situation where the flame is lost. As the valve moves open again, uncombusted fuel will accumulate in the furnace box and ducting, and may explode all at one should a source of ignition be found eg. at a neighbouring furnace sharing the same ducting. Thus it is necessary to ensure a minimum fuel flow. This can be done using an override controller which senses the fuel pressure just before the burner nozzle. The figure *f*_6.7 below shows an arrangement where an override pressure controller maintains a minimum pressure, and implicitly a flow.



*f*_6.7

6.2.3 Tasks

(1) Start the simulation with the original settings as in section 6.2.1. Let the system reach steady-state. The individual loops are selected using the column of numbered buttons on the left of the diagram below (*f*_6.8). Then the settings pertaining to that loop are shown in the top part of the same diagram. Start by selecting the supervising loop TC (4), and the over-riding loop PC (3), and switching them both to MANUAL. In this case the FC loops 1 and 2 will automatically switch from remote setpoint AUTO R to local setpoint AUTO L. Step the fuel flow setpoint by pressing on the coloured part of the SP scrollbar of loop 1, above or below the current cursor position. Explain what happens.



*f*_6.8

(2) With the air flow control loop 2 set to local setpoint, AUTO L, adjust the air flow setpoint by pressing on the arrowheads of the SP scrollbar of loop 2, in order to bring the air/fuel ratio back to 1.0.

(3) Select the TC control loop (4), and switch this controller to AUTO L. The associated cross-limiting algorithm takes over the two flow controllers on remote (AUTO R). Wait for the system to settle down. Observe the trace of the actual air/fuel ratio, which is shown at 10 times its magnitude as “A/F *10”. What is its value? Make some big steps in the the process flow-rate through the furnace, by pressing on the coloured parts of the associated scrollbar. Can you get deviations of the actual air/fuel ratio *below* the setpoint? How well is the furnace exit temperature maintained?

(4) Perform setpoint steps in the furnace exit temperature and the air/fuel ratio. Comment on the responses, noting also where the air/fuel ratio falls below setpoint.

(5) With both the override PC and TC still on AUTO L, reduce the minimum pressure setpoint as far as possible. Explain the response of lack thereof. Now start increasing the minimum pressure setpoint. Can you explain the effect? Repeat this test after reducing the derivative time τ_D to zero. Next, increase the minimum pressure setpoint so that it is well above the current measured pressure. Explain what happens. Is the furnace exit temperature maintained? What about the air/fuel ratio?

6.2.4 Code extracts

- On each time-step, calculation of the control action for each of the *nFFA* = 4 control loops:

```
// Interpret present values

m_Ffuel_highest = __max(m_Ffuel_desired,m_Ffuel);
m_Ffuel_allowed = m_Fair/(m_Rairfuel_sp+SMALL);


// Loop03: Tprocess by Ffuel_desired
x[ 3] = m_Tprocess;
r[ 3] = m_Tprocess_sp;
m[ 3] = m_Ffuel_desired;
```

```
if (b_AutoLocal[3])
{
        m_Ffuel_sp = __max(__min(__min(m_Ffuel_desired,m_Ffuel_allowed), max_Ffuel),0);
        m_Fair_sp  = __max(__min(m_Ffuel_highest*m_Rairfuel_sp, max_Fair),0);
}

// Loop00: Ffuel by Vfuel1
x[ 0] = m_Ffuel;
r[ 0] = m_Ffuel_sp;
m[ 0] = m_Vfuel1;
// Loop01: Fair by Vair
x[ 1] = m_Fair;
r[ 1] = m_Fair_sp;
m[ 1] = m_Vair;
// Loop02: Pfuel by Vfuel2
x[ 2] = m_Pfuel;
r[ 2] = m_Pfuel_sp;
m[ 2] = m_Vfuel2;


for (i=0;i<nFFA;i++)
{
        if ((!b_AutoLocal[i]) && (!b_AutoRemote[i]))
        {
                r2[i] = r1[i] = r0[i] = x[i];
                x2[i] = x1[i] = x0[i] = x[i];
                m2[i] = m1[i] = m[i];
        }
        else
        {
                // On AUTO (Local or remote) : calculate FFA action!
                // first cascade stack
                r2[i] = r1[i];
                r1[i] = r0[i];
                r0[i] = r[i];
                x2[i] = x1[i];
                x1[i] = x0[i];
                x0[i] = x[i];
                m2[i] = m1[i];
                m1[i] = m[i];

                // Action:
                m[i] = ( - a1[i] * m1[i]
                        - a2[i] * m2[i]
                            + b0[i] * r0[i]
                            + b1[i] * r1[i]
                            + b2[i] * r2[i]
                            - c0[i] * x0[i]
                            - c1[i] * x1[i]
                            - c2[i] * x2[i] ) / a0[i];

                // back to actual variables
                switch (i)
                {

                case  0: // Loop00: Ffuel by Vfuel1
                        m_Vfuel1 = __max(__min(m[0],max_Vfuel),0);
                        break;
                case  1: // Loop01: Fair by Vair
                        m_Vair = __max(__min(m[1],max_Vair),0);
                        break;
                case  2: // Loop02: Pfuel by Vfuel2
                        m_Vfuel2 = __max(__min(m[2],max_Vfuel),0);
                        break;
                case  3: // Loop03: Tprocess by Fbfw_desired
                        m_Ffuel_desired = __max(__min(m[3],max_Ffuel),0);
                        break;
                default:
                        break;
                }
        }
}

m_Vfuel1_DISPLAY = m_Vfuel1;
m_Vfuel2_DISPLAY = m_Vfuel2;
```

```
		if (b_AutoLocal[3])
		{
			m_Vfuel =__max(m_Vfuel1,m_Vfuel2);
			m_Vfuel1 = m_Vfuel; // anti-windup : external reset
			m_Vfuel2 = m_Vfuel; // anti-windup : external reset
		}
		else
		{
			{
				if ((b_AutoLocal[0]) && (!b_AutoLocal[2]))
				{
					m_Vfuel = m_Vfuel1;
					m_Vfuel2 = m_Vfuel;  // for bumpless start
				}
				else
				{
					if ((!b_AutoLocal[0]) && (b_AutoLocal[2]))
					{
						m_Vfuel = m_Vfuel2;
						m_Vfuel1 = m_Vfuel;  // for bumpless start
					}
				}
			}
		}
	}
```

References

Smith, C.A. and A.B. Corripio (1997) *Principles and Practice of Automatic Process Control*, 2nd Edition, John Wiley & Sons, 204.

Chapter 7 Simulations - MIMO closedloop and DMC

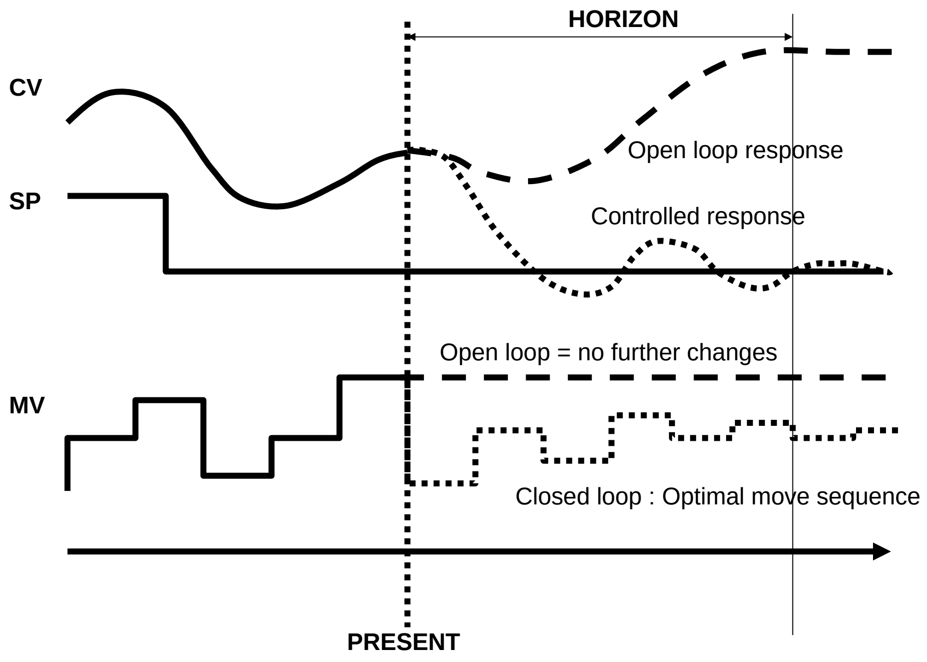
.

7.1 A note on Dynamic Matrix Control

Dynamic Matrix Control is a form of Model Predictive Control (MPC) which uses a step-response convolution model for prediction of the effect of possible control actions. Since the early work of Cutler and Ramaker, (1979) and Garcia and Morshedi (1984), these controllers, particularly DMC, have proved their worth in many industrial applications.

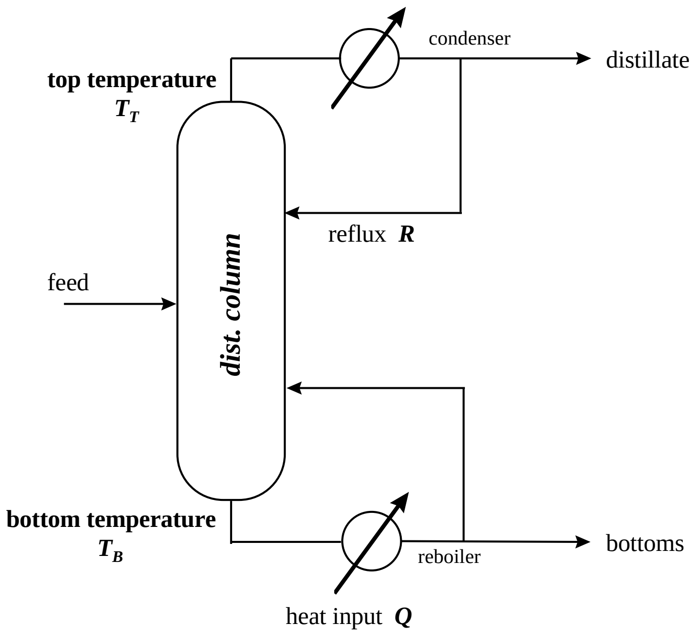
Dynamic Matrix Controllers have become popular in industry because they are easily understood and set up, can handle dead-time, inverse response and constraints optimally, and are particularly useful for multivariable systems in which there is cross-interaction between inputs and outputs. Cross-interaction becomes problematic where the response times are similar, making it difficult to decouple the system with individual loops. Distillation columns are typical examples of systems where such interaction exists, but there are many other process operations which present similar problems for the alternative multiple PID loop approach. The DMC algorithm is able to time and coordinate control "moves" on the manipulated variables (MV's) in such systems to simultaneouly keep all CV's (controlled variables) on setpoint, accounting too, in a feedforward sense, for measured disturbance variables (DV's). Should constraints on the MV's, CV's, or any combinations thereof prevent the setpoints from being reached, the closest possible approach to the setpoints will be made according to the relevant weightings applied to the setpoint deviations.

The DMC algorithm is a particular form of MPC (Model Predictive Control), which belongs to the family of IMC (Internal Model Control). In order to anticipate what combination of moves will minimise the performance objective up to a moving time horizon (*f*_7.1), a model relating outputs to inputs is required. The particular model for DMC is based on measured process step-responses, and it is the "Dynamic Matrix" which scales and shifts these to build the complete response for a series of "moves". The same method is used to account for the contribution of past moves to the future output. The ease of building this model is one feature that has made DMC popular.

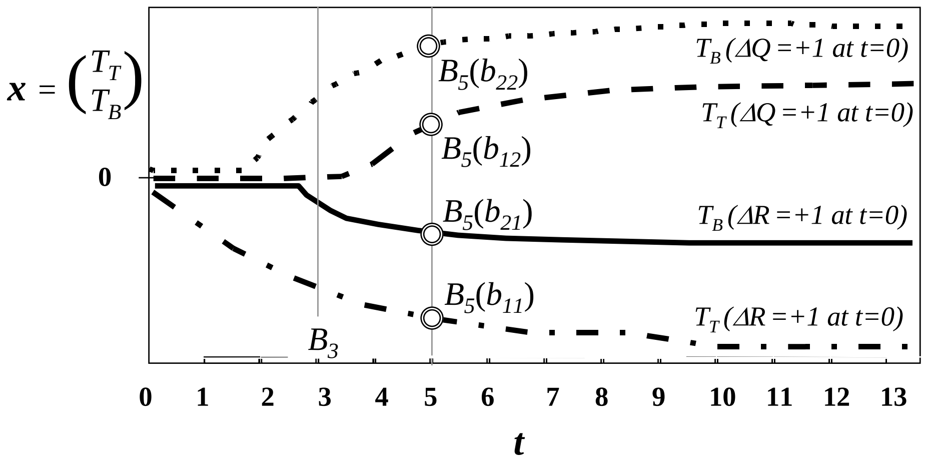


*f*_7.1

Consider a 2-input, 2-output system, for example a distillation column (*f*_7.2) in which reflux flow (R) and reboiler duty (Q) cause variations in the Top Temperature (T_T) and Bottom Temperature (T_B). If the system is steady and we make a step in R, we shall get two separate responses for T_T and T_B. Likewise, we shall get distinct responses for T_T and T_B for a step in Q. This is shown graphically in *f*_7.3, for unit positive steps in R and Q. Note that only changes in T_T and T_B from their original steady values are considered.



*f*_7.2



*f*_7.3

Here the sub-matrices B_i have the form, eg. $B_5 = \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix}_5$, where the b_{ij} are the corresponding deviations of the outputs from their initial values, as a result of the unit input steps.

For the input vector ***m*** (R,Q), now consider not just one step but a series of control vector moves $\boldsymbol{\Delta m_1}$, $\boldsymbol{\Delta m_2}$, ... ,$\boldsymbol{\Delta m_M}$, over a sequence of M time steps. If the system is linear, we can construct the resultant sequence in ***x*** (T_T,T_B) over P intervals by shifting, scaling and superposing the above step responses in the convolution model:

$$\begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ : \\ x_M \\ x_{M+1} \\ : \\ x_P \end{pmatrix} = \begin{bmatrix} B_1 & 0 & 0 & 0 & 0 & \dots & 0 \\ B_2 & B_1 & 0 & 0 & 0 & \dots & 0 \\ B_3 & B_2 & B_1 & 0 & 0 & \dots & 0 \\ : & : & : & : & : & & : \\ B_M & B_{M-1} & \dots & B_1 & 0 & \dots & 0 \\ B_M & B_M & B_{M-1} & \dots & B_1 & \dots & 0 \\ : & : & : & : & : & & : \\ B_M & B_M & B_M & B_M & B_{M-1} & \dots & B_1 \end{bmatrix} \begin{pmatrix} \Delta m_1 \\ \Delta m_2 \\ \Delta m_3 \\ : \\ \Delta m_M \\ \Delta m_{M+1} \\ : \\ \Delta m_P \end{pmatrix} \qquad (7.1)$$

(Steady-state response achieved M time intervals ahead with $M < P$)

We represent this convolution model for future outputs as $\boldsymbol{x} = \boldsymbol{B\Delta m}$, where the "matrix of matrices" ***B*** is generally known as the *Dynamic Matrix*. Now defining the P x M matrices:

$$
\boldsymbol{B}_{OL} = \begin{bmatrix}
\boldsymbol{B}_M & \boldsymbol{B}_M & \boldsymbol{B}_{M-1} & \boldsymbol{B}_{M-2} & \boldsymbol{B}_{M-3} & \cdots & \boldsymbol{B}_3 & \boldsymbol{B}_2 \\
\boldsymbol{B}_M & \boldsymbol{B}_M & \boldsymbol{B}_M & \boldsymbol{B}_{M-1} & \boldsymbol{B}_{M-2} & \cdots & \cdots & \boldsymbol{B}_3 \\
\boldsymbol{B}_M & \boldsymbol{B}_M & \boldsymbol{B}_M & \boldsymbol{B}_M & \boldsymbol{B}_{M-1} & \cdots & \cdots & \vdots \\
\boldsymbol{B}_M & \boldsymbol{B}_M & \boldsymbol{B}_M & \boldsymbol{B}_M & \boldsymbol{B}_M & \cdots & \cdots & \vdots \\
\vdots & \vdots & \vdots & \vdots & \vdots & & \vdots & \boldsymbol{B}_{M-1} \\
\boldsymbol{B}_M & \boldsymbol{B}_M & \boldsymbol{B}_M & \boldsymbol{B}_M & \boldsymbol{B}_M & \cdots & \boldsymbol{B}_M & \boldsymbol{B}_M \\
\vdots & \vdots & \vdots & \vdots & \vdots & & \vdots & \vdots \\
\boldsymbol{B}_M & \boldsymbol{B}_M & \boldsymbol{B}_M & \boldsymbol{B}_M & \boldsymbol{B}_M & \cdots & \boldsymbol{B}_M & \boldsymbol{B}_M
\end{bmatrix} \tag{7.2}
$$

$$
\boldsymbol{B}_0 = \begin{bmatrix}
\boldsymbol{B}_M & \boldsymbol{B}_{M-1} & \boldsymbol{B}_{M-2} & \boldsymbol{B}_{M-3} \cdots & \boldsymbol{B}_2 & \boldsymbol{B}_1 \\
\boldsymbol{B}_M & \boldsymbol{B}_{M-1} & \boldsymbol{B}_{M-2} & \boldsymbol{B}_{M-3} \cdots & \boldsymbol{B}_2 & \boldsymbol{B}_1 \\
\boldsymbol{B}_M & \boldsymbol{B}_{M-1} & \boldsymbol{B}_{M-2} & \boldsymbol{B}_{M-3} \cdots & \boldsymbol{B}_2 & \boldsymbol{B}_1 \\
\boldsymbol{B}_M & \boldsymbol{B}_{M-1} & \boldsymbol{B}_{M-2} & \boldsymbol{B}_{M-3} \cdots & \boldsymbol{B}_2 & \boldsymbol{B}_1 \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
\boldsymbol{B}_M & \boldsymbol{B}_{M-1} & \boldsymbol{B}_{M-2} & \boldsymbol{B}_{M-3} \cdots & \boldsymbol{B}_2 & \boldsymbol{B}_1
\end{bmatrix} \tag{7.3}
$$

and the present measurements (*P*) and past inputs (*M*):

$$
\boldsymbol{x}_{0MEAS} = \begin{pmatrix} \boldsymbol{x}_{0MEAS} \\ \boldsymbol{x}_{0MEAS} \\ \boldsymbol{x}_{0MEAS} \\ \boldsymbol{x}_{0MEAS} \\ \vdots \\ \boldsymbol{x}_{0MEAS} \end{pmatrix} \quad \text{and} \quad \Delta\boldsymbol{m}_{PAST} = \begin{pmatrix} \Delta\boldsymbol{m}_{-M+1} \\ \Delta\boldsymbol{m}_{-M+2} \\ \Delta\boldsymbol{m}_{-M+3} \\ \Delta\boldsymbol{m}_{-M+4} \\ \vdots \\ \Delta\boldsymbol{m}_0 \end{pmatrix} \tag{7.4-7.5}
$$

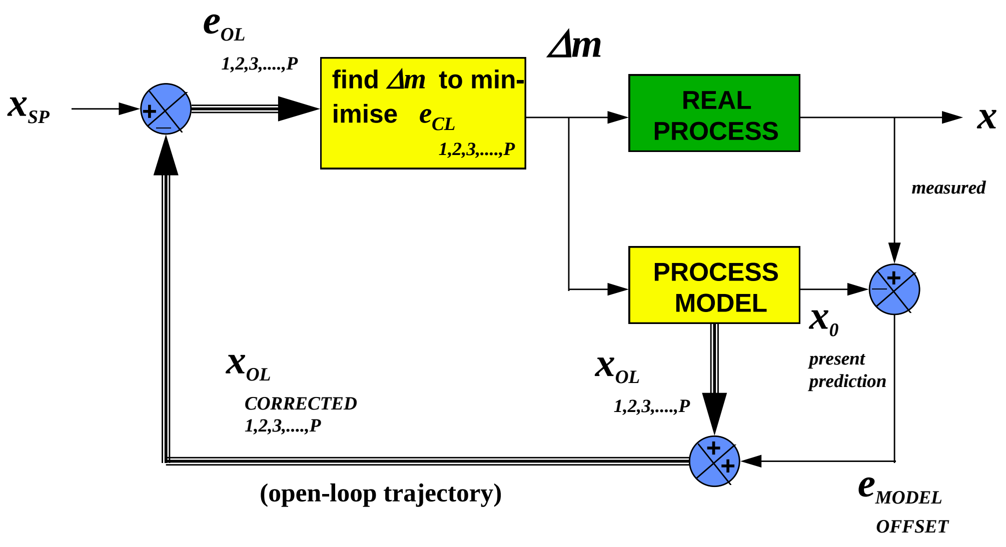
then the “open-loop” response, corrected for present model offset, is

$$\boldsymbol{x}_{OL} = \boldsymbol{x}_{0MEAS} + [\boldsymbol{B}_{OL} - \boldsymbol{B}_0]\, \Delta\boldsymbol{m}_{PAST} \tag{7.6}$$

and the “closed loop” response up to the *P*-step horizon is obtained by including the contribution of the future control input steps $\Delta\boldsymbol{m}$:

$$\boldsymbol{x}_{CL} = \boldsymbol{x}_{OL} + \boldsymbol{B}\, \Delta\boldsymbol{m} \tag{7.7}$$

On each time-step it is possible to compute the future open-loop response $\boldsymbol{x}_{OL}$ based on past inputs and the present output. Thus the control problem to achieve the desired trajectory $\boldsymbol{x}_{CL}$ amounts to finding suitable $\Delta\boldsymbol{m}$ as in figure *f*_7.4 below.



*f*_7.4

A constrained multivariable Linear Dynamic Matrix Controller (LDMC), based on the linear programming solution of Chang and Seborg (1983), and the formulation of Morshedi *et al* (1985), has been developed as follows:

Define $\boldsymbol{x_{SP}}$ to contain a sequence of set-points for the outputs up to the time horizon P steps ahead, so that the open loop error may be calculated in advance as $\boldsymbol{x_{OL}} - \boldsymbol{x_{SP}}$. Then the closed-loop error for a control move sequence $\boldsymbol{\Delta m}$ will be

$$\boldsymbol{e_{CL}} = \boldsymbol{x_{CL}} - \boldsymbol{x_{SP}} \tag{7.8}$$

$$= \boldsymbol{x_{OL}} - \boldsymbol{x_{SP}} + \boldsymbol{B}\ \boldsymbol{\Delta m} \tag{7.9}$$

$$= \boldsymbol{e_{OL}} + \boldsymbol{B}\ \boldsymbol{\Delta m} \tag{7.10}$$

Generally only a limited sequence of N moves ($\boldsymbol{\Delta m}^*$) are optimised ($N << P$). This is equivalent to setting $\boldsymbol{\Delta m_k = 0}$ for $k > N$, or alternately replacing $\boldsymbol{B}$ with the non-square P x N matrix:

(7.11)

Then

$$\boldsymbol{e_{CL}} = \boldsymbol{e_{OL}} + \boldsymbol{A}\,\boldsymbol{\Delta m^*}$$

Now define a quadratic objective function, dependent only on the strategy $\boldsymbol{\Delta m^*}$.

$$\begin{aligned} J(\boldsymbol{\Delta m^*}) &= (\boldsymbol{e_{CL}})^{T}\,\boldsymbol{W}\,(\boldsymbol{e_{CL}}) + (\boldsymbol{\Delta m^*})^{T}\boldsymbol{\Lambda}\,(\boldsymbol{\Delta m^*}) \\ &= (\boldsymbol{e_{OL}} + \boldsymbol{A}\,\boldsymbol{\Delta m^*})^{T}\,\boldsymbol{W}\,(\boldsymbol{e_{OL}} + \boldsymbol{A}\,\boldsymbol{\Delta m^*}) + (\boldsymbol{\Delta m^*})^{T}\boldsymbol{\Lambda}\,(\boldsymbol{\Delta m^*}) \end{aligned} \tag{7.12}$$

By minimising J with respect to $\boldsymbol{\Delta m^*}$, we are able to find an optimal sequence of control moves, $\boldsymbol{\Delta m^*}$, which achieve "minimum" deviation from the set-point trajectory up to the time horizon P, for "minimum" control move effort. It is the weights in the matrices $\boldsymbol{W}$ and $\boldsymbol{\Lambda}$**,** generally diagonal, which determine the extents to which deviations of either parameter are discouraged. Higher "gains" will generally be associated with higher values in $\boldsymbol{W}$ than $\boldsymbol{\Lambda}$. The values in $\boldsymbol{\Lambda}$ cause *"move suppression".* Note that the diagonal weights act on the squared deviations and moves. Thus, in general, a weight would be increased *four times* to reduce the associated deviation or move to *half* of its previous value.

It is easily shown that differentiation of J with respect to the elements of $\boldsymbol{\Delta m^*}$, and setting the result to the zero vector, yields the *unbounded quadratic optimum* control move strategy

$$\boldsymbol{\Delta m_{UQO}} = -\,[\boldsymbol{A^T}\,\boldsymbol{W}\boldsymbol{A} + \boldsymbol{\Lambda}\,]^{-1}\,\boldsymbol{A^T}\,\boldsymbol{W}\;\boldsymbol{e_{OL}} \tag{7.13}$$

In our solution we now wish to account for the constraints of the system. These will include the ranges over which the the manipulated variables $\boldsymbol{m}$ can be moved, where

$$\boldsymbol{m} = \boldsymbol{L}\,\boldsymbol{\Delta m^*} + \boldsymbol{m_{INIT}} \tag{7.14}$$

$$\boldsymbol{L} = \begin{bmatrix} I & 0 & 0 & 0 & 0 & \dots & 0 \\ I & I & 0 & 0 & 0 & \dots & 0 \\ I & I & I & 0 & 0 & \dots & 0 \\ I & I & I & I & 0 & \dots & 0 \\ : & : & : & : & : & & : \\ I & I & I & I & I & \dots & I \end{bmatrix} \tag{7.15}$$

and allowed ranges for the controlled variables $\boldsymbol{x}$. A global method which will seek the minimum of J within these constraints requires *Quadratic Programming*, and is quite computation-intensive. Here we rather follow the method of Morshedi *et al* (1985), and seek that combination of control moves which will get us as close as possible to $\boldsymbol{\Delta m_{UQO}}$, yet keep us within the constraints. This re-definition of the problem then allows us to use *Linear Programming* to handle the constraints. Although it does not guarantee us the *quadratic optimum*, we expect to be close (and identical within the constraints), and thus we shall call this result $\boldsymbol{\Delta m_{BQO}}$ (bounded quadratic optimum). Define the residuals

$$\boldsymbol{r} = \boldsymbol{\Delta m_{BQO}} - \boldsymbol{\Delta m_{UQO}} \tag{7.16}$$

In order to allow minimisation of the *absolute* differences, we represent $\boldsymbol{r}$ using two non-negative quantitities (one of which will be forced to zero in the LP solution):

$$\boldsymbol{r} = [\, \boldsymbol{r}^{+} - \boldsymbol{r}^{-} \,] \tag{7.17}$$

Then the *Linear Programming Problem* is formulated as follows:

Objective function: $\boldsymbol{w}^{T} [\, \boldsymbol{r}^{+} + \boldsymbol{r}^{-} \,]$ (to be minimised) (7.18)

where $\boldsymbol{w}$ is a weighting vector, possibly chosen as below to improve the approach to $\boldsymbol{\Delta m_{UQO}}$, by following the steepest descent of *J* (Mulholland and Prosser, 1997) subject to the constraints:

Input limits: $\boldsymbol{L}\,[\boldsymbol{r}^{+} - \boldsymbol{r}^{-}] \geq \boldsymbol{m}_{MIN} - \boldsymbol{m}_{INIT} - \boldsymbol{L}\,\Delta\boldsymbol{m}_{UQO}$ (7.19)

$\boldsymbol{L}\,[\boldsymbol{r}^{+} - \boldsymbol{r}^{-}] \leq \boldsymbol{m}_{MAX} - \boldsymbol{m}_{INIT} - \boldsymbol{L}\,\Delta\boldsymbol{m}_{UQO}$ (7.20)

Input combination $\boldsymbol{HL}\,[\boldsymbol{r}^{+} - \boldsymbol{r}^{-}] \geq \boldsymbol{\delta}_{MIN} - \boldsymbol{Hm}_{INIT} - \boldsymbol{HL}\,\Delta\boldsymbol{m}_{UQO}$ (7.21)

limits: $\boldsymbol{HL}\,[\boldsymbol{r}^{+} - \boldsymbol{r}^{-}] \leq \boldsymbol{\delta}_{MAX} - \boldsymbol{Hm}_{INIT} - \boldsymbol{HL}\,\Delta\boldsymbol{m}_{UQO}$ (7.22)

Input ramp limits: $[\boldsymbol{r}^{+} - \boldsymbol{r}^{-}] \geq -\Delta\boldsymbol{m}_{MAX} - \Delta\boldsymbol{m}_{UQO}$ (7.23)

$[\boldsymbol{r}^{+} - \boldsymbol{r}^{-}] \leq +\Delta\boldsymbol{m}_{MAX} - \Delta\boldsymbol{m}_{UQO}$ (7.24)

Output limits: $\boldsymbol{A}\,[\boldsymbol{r}^{+} - \boldsymbol{r}^{-}] \geq \boldsymbol{x}_{MIN} - \boldsymbol{x}_{OL} - \boldsymbol{A}\,\Delta\boldsymbol{m}_{UQO}$ (7.25)

$\boldsymbol{A}\,[\boldsymbol{r}^{+} - \boldsymbol{r}^{-}] \leq \boldsymbol{x}_{MAX} - \boldsymbol{x}_{OL} - \boldsymbol{A}\,\Delta\boldsymbol{m}_{UQO}$ (7.26)

After solving for $\boldsymbol{r}^{+}$ and $\boldsymbol{r}^{-}$, one of which will be zero in each pair**,** the necessary input control actions are obtained from

$$\boldsymbol{\Delta m_{BQO}} = [\, \boldsymbol{r}^{+} - \boldsymbol{r}^{-} \,] + \boldsymbol{\Delta m_{UQO}} \tag{7.27}$$

$\boldsymbol{\Delta m_{BQO}}$ contains optimal values for the limited sequence of steps $\boldsymbol{\Delta m_1}, \boldsymbol{\Delta m_2}, \ldots, \boldsymbol{\Delta m_N}$, but it is only the first step $\boldsymbol{\Delta m_1}$ which is actually implemented, before the entire optimisation process is repeated on the next time-step. The effect of optimising more than one step is that the first step can be severe (overshooting), with subsequent steps correcting the steady-state response.

In equation 7.27 it is noted that if $\boldsymbol{\Delta m_{BQO}}$ cannot reach $\boldsymbol{\Delta m_{UQO}}$ (owing to constraints), then the LP will minimise the weighted sum of the deviation of each manipulated variable move from its unconstrained optimal value. This is not the same as minimising the quadratic objective function given in equation 7.12:

$$J(\boldsymbol{\Delta m^*}) = (\boldsymbol{e_{OL}} + \boldsymbol{A}\,\boldsymbol{\Delta m^*})^{\mathrm{T}}\ \boldsymbol{W}\ (\boldsymbol{e_{OL}} + \boldsymbol{A}\,\boldsymbol{\Delta m^*}) + (\boldsymbol{\Delta m^*})^{\mathrm{T}}\boldsymbol{\Lambda}\,(\boldsymbol{\Delta m^*}) \qquad (7.28)$$

The weights in ***W***, specified to minimise the closed-loop error $\boldsymbol{e_{CL}} = \boldsymbol{e_{OL}} + \boldsymbol{A}\ \boldsymbol{\Delta m^*}$, will be found to have no effect in the constrained situation, with manipulated variables apparently making no attempt to find a better position on the constraints to minimise $(\boldsymbol{e_{CL}})^2$ .The following procedure is used to deal with this situation, bringing the solution closer to the full *QDMC* treatment (Mulholland and Prosser, 1997):

In a constrained situation we would see a steady-state error

$$\boldsymbol{e_{SS}} = -\boldsymbol{A_{SS}}\ \boldsymbol{\Delta m_{SS}} \qquad (7.29)$$

where $\boldsymbol{\Delta m_{SS}}$ is nominally a single unconstrained move that could be made, ultimately giving a steady-state response which would eliminate this error. The small $\boldsymbol{A_{SS}}$ matrix clearly has the last point of each response – ie. it is B_M. To be consistent with the unconstrained criterion, *viz.* minimisation of *J*, we now really want to minimise

$$J_{CON} = \boldsymbol{e_{SS}}^T\,\boldsymbol{W}\,\boldsymbol{e_{SS}} = \boldsymbol{\Delta m_{SS}}^T\ \boldsymbol{A_{SS}}^T\,\boldsymbol{W}\,\boldsymbol{A_{SS}}\ \boldsymbol{\Delta m_{SS}} \qquad (7.30)$$

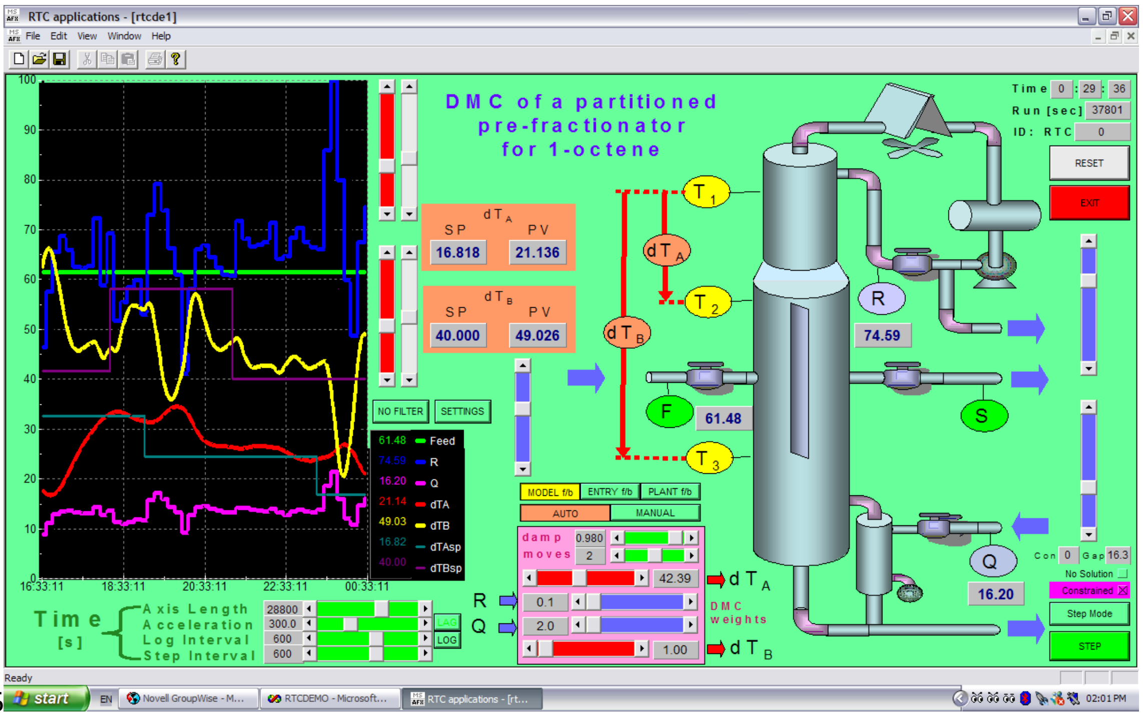
(To use ***W*** we have expanded $\boldsymbol{e_{SS}}$, $\boldsymbol{A_{SS}}$ and $\boldsymbol{\Delta m_{SS}}$ by repetition for all points up to the horizon). This has a gradient with respect to $\boldsymbol{\Delta m_{SS}}$ which is $2\ \boldsymbol{A_{SS}}^T\ \boldsymbol{W}\ \boldsymbol{A_{SS}}\ \ \boldsymbol{\Delta m_{SS}}$. Recognising that $\boldsymbol{\Delta m_{UQO}} - \boldsymbol{\Delta m_{BQO}}$ will give a fair estimate of $\boldsymbol{\Delta m_{SS}}$, we propose therefore to weight the constrained manipulated variables approach to their optimum values according to the *strength* (positive or negative) they have in changing the value of J_{CON}. Thus we take ***w*** in equation 7.6 as

$$\boldsymbol{w} = |\,\boldsymbol{A_{SS}}^T\,\boldsymbol{W}\,\boldsymbol{A_{SS}}\,(\boldsymbol{\Delta m_{UQO}} - \boldsymbol{\Delta m_{BQO}})\,| \qquad (7.31)$$

With ***w*** adjusted on each-time-step like this, it is understandable that the solution could start to to oscillate between two constrained points. Thus the computed values of ***w*** are filtered on each-time-step to slow down the transients.

7.2 DMC – MIMO Dynamic Matrix Control of a partitioned fractionator

Understanding and tuning this popular predictive controller based on measured process step responses:



*f*_7.5

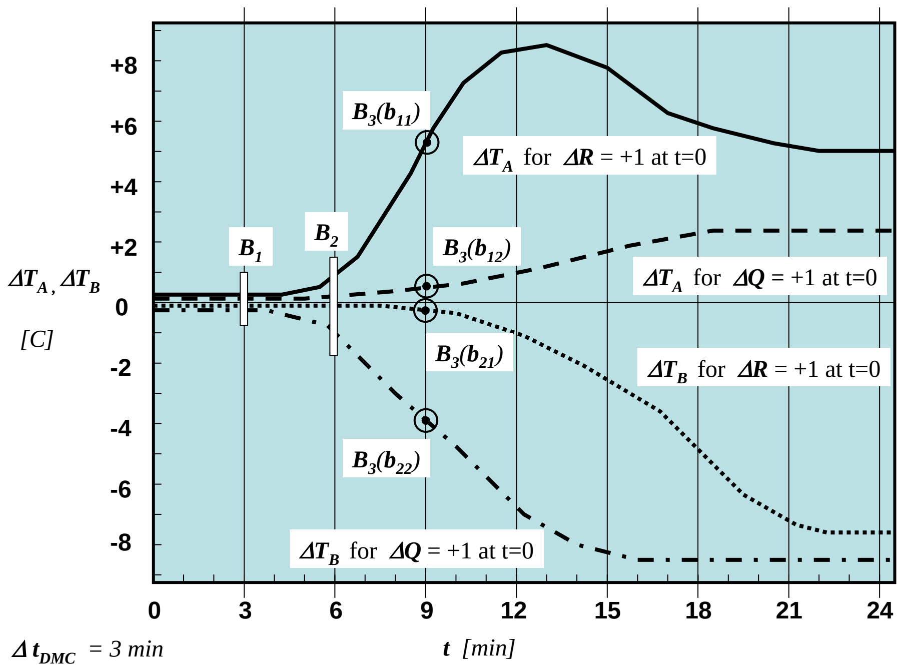
7.2.1 Typical settings

	Weights:	**Constraints:**	
$\Delta T_{A\ SP}$= 19.795	dev ΔT_A = 1	max R = 100	Axis Length = 28800
$\Delta T_{B\ SP}$= 41.646	dev ΔT_B = 1	max Q= 50	Acceleration = 300
R = 68.37	move R = 0.1	max ΔT_A = 50	Log Interval = 600
Q = 14.20	move Q = 2.0	max ΔT_B = 100	Step Interval = 600
F = 61.48		%dev from R/Q = 5	

7.2.2 Theory

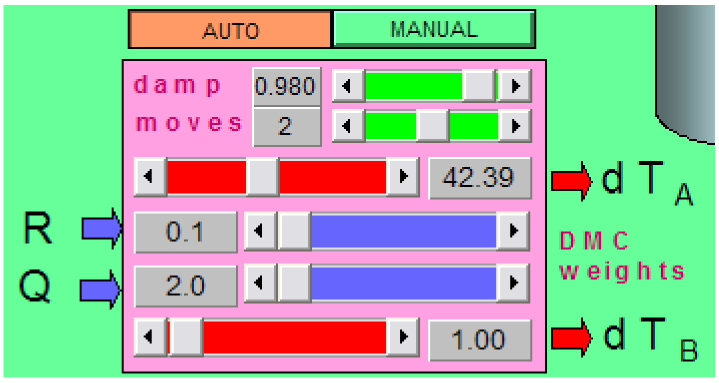
The particular example considered in this application is the control of temperatures at two points in a partitioned distillation column. This is akin to "dual-composition control", and is achieved by manipulating the reflux flow *R*, and the reboiler duty *Q*. To handle pressure fluctuations, it is actually the *difference* between the temperature at the considered point, and that at the top of the column that are considered, viz. $\Delta T_A = T_A - T_{TOP}$ and $\Delta T_B = T_B - T_{TOP}$. Here "A" is just above a dividing wall (separating feed side from sidestream-offtake side), and "B" is just below this dividing wall. This type of partitioned column is used to provide a middle-cut without having to use two columns, and the partition reduces the short-circuiting of feed to the offtake.

If the system is steady and we make a step in *R*, we shall get two separate responses ΔT_A and ΔT_B. Likewise, we shall get distinct responses for ΔT_A and ΔT_B for a step in *Q*. This is shown graphically in the figure below (*f*_7.6), for unit positive steps in *R* and *Q*. Note that only *changes* ΔT_A and ΔT_B from the original steady temperature gaps are considered. These step response measurements are then used to construct the DMC controller, as discussed in sdection 7.1.



*f*_7.6

Below is seen the tuning panel for the DMC controller (*f*_7.7). Against "dT$_A$" and "dT$_B$" are the weights on the deviations of the variables ΔT_A and ΔT_B from their setpoints. Actually, the objective function requires that these deviations *squared*, then summed with their appropriate weights (and the move suppression terms), be minimised. The weights against "R" and "Q" are the "move suppression" weights on these MV's. As these weights are increased, the *size* of each "move" (step) in the relevant MV, squared and then multiplied by the relevant weight, increasingly adds a penalty to the objective function. Note that the move suppression weights do not penalise the overall movement of the MV's, rather just the *rate* at which they move.



damp 0.980
moves 2
0.1
2.0
42.39
1.00

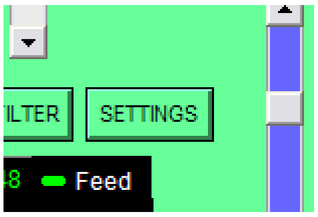
f_7.7

Also seen on the tuning panel is a setting "moves" for the number of moves into the future to be optimised. Increasing this is one way of making a DMC more vigorous. With one move it must minimise setpoint deviations over the whole horizon, and it is therefore obliged to be conservative. With two moves, the first can be much stronger, causing overshoot, because the second move is being relied on to correct the future response back towards the setpoint. In any case, it is only the first move that is actually implemented, before the system moves forward one time-step, and the entire optimisation computation is repeated.

The DMC tuning panel also allows the setting of a variable "damp" between 0 and 1. This is a recent enhancement that handles a limitation of the constrained optimisation technique used in LDMC, the particular form of DMC used here. The theoretical background to this enhancement is discussed in section 7.1, so will simply be outlined here:

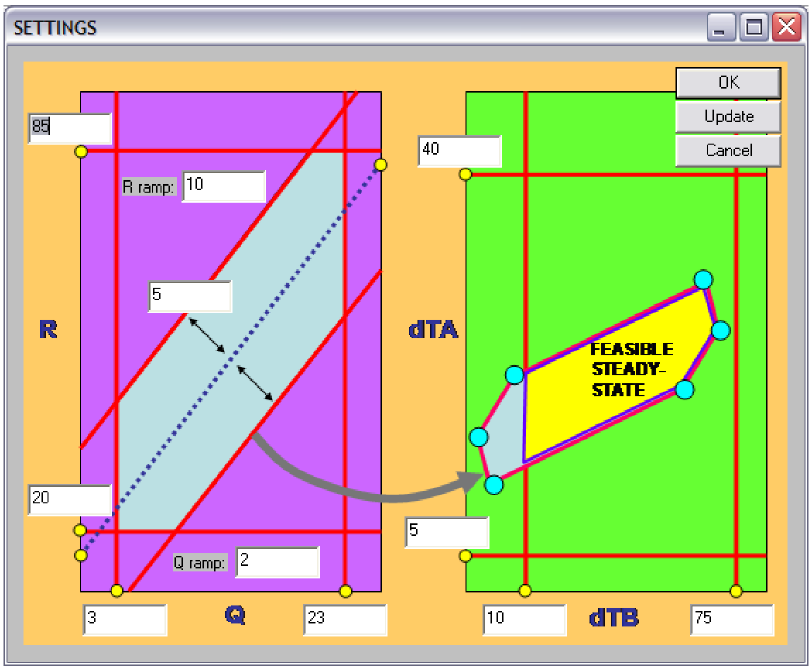
LDMC has the advantage that it is robust and efficient, making use of a *Linear Programming* solution rather than the *Quadratic Programming* solution which is often used (QDMC). A difference arises when the solution is constrained. The *QP* solution would still attempt to minimise the original *quadratic objective function*, whereas the *LP* solution switches to minimising the (weighted) "distance" of the MV's from those values which would actually achieve the minimum of this function. The difference is quite subtle, and only noticed in particular circumstances. However, to make the present LDMC behave like QDMC, a recent enhancement provides automatic adjustment of the *LP weights* on these MV distances, to recognise the contribution of each MV in minimising the quadratic objective function. These weights cannot be changed rapidly, as this could induce oscillation in the constrained solution. Thus "damp" is used to slow down the rate of change of the *LP weights*. Note that *LDMC* and *QDMC* of course produce identical solutions when the quadratic objective function has its minimum within all of the constraints.

In the case of the separation of 1-octene as the middle-cut from a mixture of olefins, studies have shown that *R* and *Q* cannot be more than about 5% off a line representing proportionality between *R* and *Q*, else conditions in the column cannot be maintained – ie. plates boil empty, or no vapour reaches the top to provide reflux and downflow. This is because the reflux flow *R* largely determines the cooling duty at the top of the column. Thus in addition to the usual upper and lower limits specified for *R, Q,* ΔT_A and ΔT_B, this application also has angled constraints 5% above and 5% below the appropriate line of proportionality.



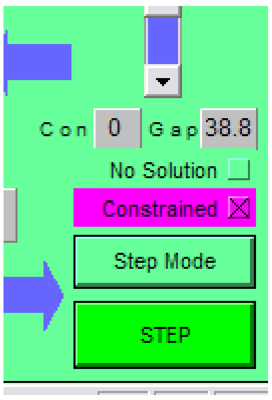
*f*_7.8

The application has a "Settings" button which is seen above (*f*_7.8). This gives access to a window for setting the various constraints as below (*f*_7.9). Press "Update" to implement the new constraint arrangement without leaving the window, or "OK" to implement *and* leave the window.



*f*_7.9

It is also possible to apply a "ramp" constraint to each MV. The effect is to limit the *size* of positive or negative changes in the MV's which are made on each time-step of the DMC algorithm. If at any point in time the defined constraints (MV, CV, combinations thereof, or ramp) are found to be restraining the controller, the "Constrained □" block shown below becomes highlighted in pink. The "norm" of the gap, between allowed MV values and those actually used, is indicated as the "*Gap*" parameter above. It is still planned that "Con" will show which (one) of the constraints is responsible when constrained. The "No Solution" block becomes highlighted if there is no feasible solution in the DMC algorithm – ie. no combination of MV values can be found which will satisfy all constraints. This typically occurs when the DMC is started outside of the constraints, and it cannot get within the constraints within one step. Then MV values are just left at their last settings. You can experiment to establish the effect of the constraints by un-checking the box in "Constrained □", which will cause all constraints to be ignored (*f*_7.10).



*f*_7.10

7.2.3 Tasks

(1) With the DMC on "Auto", let the system reach steady-state. Then switch it to "Manual", and examine the open-loop responses as follows. Step the reflux *R* up by about 10. Observe the responses until steady-state, and note that ΔT_A has an inverse response, something that would cause serious difficulty in PID control. Now step the reboiler duty *Q* by about -2. Note the different speeds of the ΔT_A and ΔT_B responses.

(2) Switch the DMC back on "Auto". Press "Reset" to start at the normal operating point. Step up the ΔT_A setpoint by about 10. Why does dT_B deviate from setpoint? In fact, it is likely that neither ΔT_A nor ΔT_B finally go to setpoint. See if a manipulation of the weights on ΔT_A and ΔT_B setpoint deviations improves the situation, or allows you to manipulate the relative sizes of the constrained deviations. Here we are making use of the enhancement mentioned at the end of section 7.1. The *LP* weights will alter to favour that MV which can make the biggest contribution in minimising the objective function. If they alter too rapidly the solution could become oscillatory. In this case, see the effect of increasing the "damp" parameter. When this reaches 1.0, the *LP* weights remain at their last value – ie. further changes by the algorithm are completely damped out.

(3) Uncheck the box in "Constrained □", to remove the constraints, and see how setpoints are exactly attained. Check the box again to see the solution go back to the constrained version.

(4) Change the setpoint of ΔT_B close to its constrained value. See the solution become unconstrained as *R/Q* ends within the 5% margin constraint on this ratio.

(5) Do some further manipulations of the move suppression weights on *Q* and *R* (and possibly the setpoint deviation weights on ΔT_A and ΔT_B) to determine whether you can get a variation in the *speed* at which the system moves towards the new setpoint after a step. It will be easier to detect changes if you turn the constraints off. Recall that the weights in $\boldsymbol{W}$ and $\boldsymbol{\Lambda}$, which are diagonal, act on the squared deviations and moves. Thus, in general, a weight would be increased *four times* to reduce the associated deviation or move to *half* of its previous value.

(6) Repeat the setpoint step test in (5) to determine the effect of increasing the the number of optimised moves (“moves”) from 1 to 2 and to 3.

(7) The column feed *F* is included as a DV (disturbance variable) in the DMC algorithm – ie. there will be feedforward compensation for feed flow changes. Test this out.

7.2.4 Code extracts

- The following recalculation of the DMC gain matrix is performed at the “Initialise” stage, and each time that any one of the weights *W1, W2, L1,* or *L2* is changed:

```
// Load Wdmc  (weights on deviation-squared from setpoint)
for (i=1;i<=Ndmc;i++) for (ii=1;ii<=Rdmc;ii++)
{
        j = (i-1)*Rdmc+ii;
        if (ii==1) Wdmc(j,j)= W1;  // dTA  deviation from SP // so can weight each controlled variable differently
        if (ii==2) Wdmc(j,j)= W2;  // dTB  deviation from SP
}
// Load Ldmc  (weights on control-move-squared)
for (i=1;i<=Mdmc;i++) for (ii=1;ii<=Pdmc;ii++)
{
        j = (i-1)*Pdmc+ii;
        if (ii==1) Ldmc(j,j)= L1;  // REFLUX FLOW MOVE // so can weight each controlled variable differently
        if (ii==2) Ldmc(j,j)= L2;  // REBOILER DUTY MOVE
}
// Load Bss (ignores possibility of integration)
cMatrix Bss;
Bss.Init(Rdmc,Pdmc);
for (i=1;i<=Rdmc;i++)
{
        for (j=1;j<=Pdmc;j++)
        {
                rcol=(j-1)*Rdmc+i;
                Bss(i,j)=resp(Ndmc,rcol);
        }
}
cMatrix temp1,temp2;
temp1.Init(Rdmc,Rdmc);
for (j=1;j<=Rdmc;j++) temp1(j,j) = Wdmc(j,j); // only the first time step for the output states
temp2 = temp1 * Bss;
temp1 = ~Bss;   // transpose
phi = temp1*temp2; // will be used for sum of square setpoint deviation error

// Set up the DMC gain matrix...............
// Kdmc = inv[B'WB +L] B'W
BT= ~B; // transpose
BTWBPL = BT*Wdmc*B + Ldmc;
BTWBPLi=BTWBPL.Inv();  // inversion
BTW = BT*Wdmc;
Kdmc = BTWBPLi*BTW;
```

- The following code is executed on every time-step of the DMC algorithm:

```
if ((theApp.FIRSTLOOP) | (m_nopt != nopt_last))   // have to sart 'w' at default in this case
{
        w.Init(m_nopt*Pdmc,1);
        for (i=1;i<=m_nopt;i++) for (j=1;j<=Pdmc;j++)
```

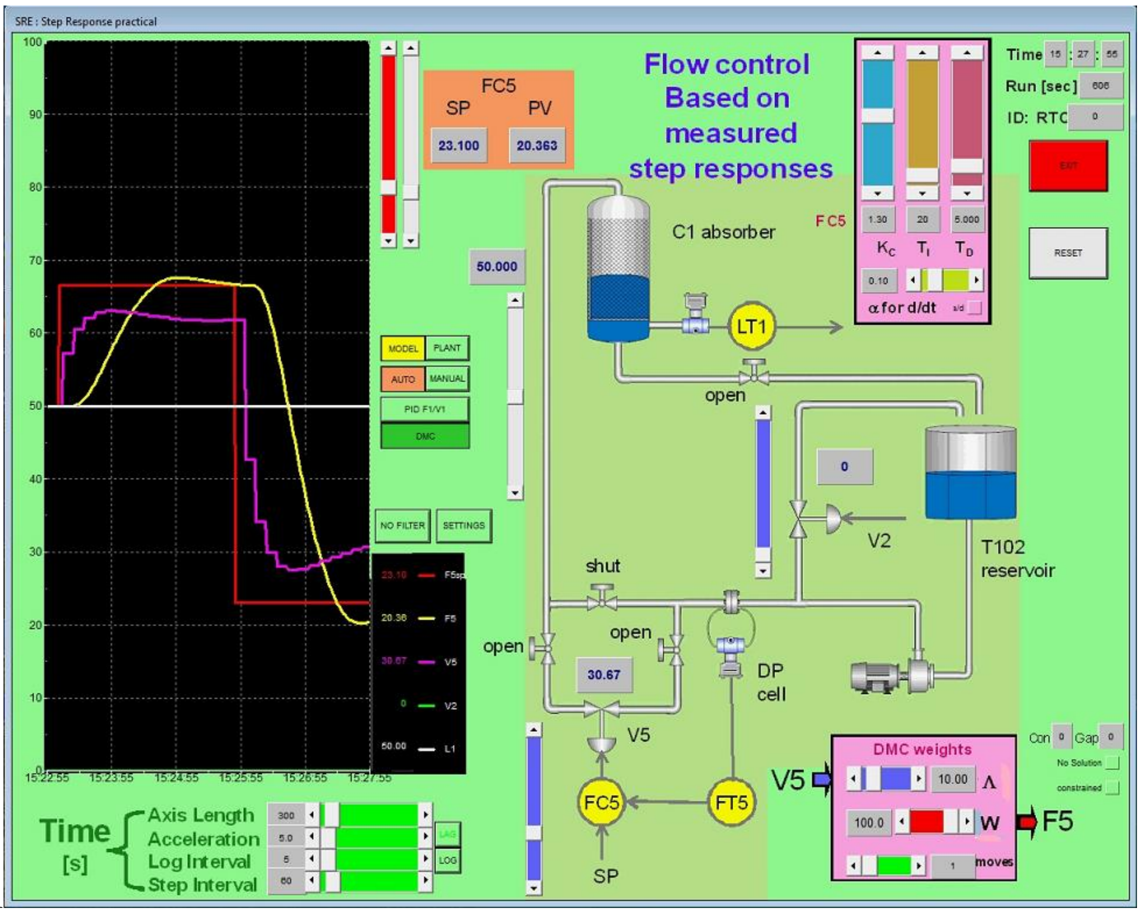
```
		{
			// fractions of eng value
			if (j=1) w((i-1)*Pdmc+j) = 1.0/Rdatum;
			if (j=2) w((i-1)*Pdmc+j) = 1.0/Qdatum;
		}
	}

	if ((m_W1 != W1_last) || (m_W2 != W2_last) || (m_L1 != L1_last) || (m_L2 != L2_last)
				|| (m_nopt != nopt_last)  || (m_cdamp != cdamp_last))
	{
		W1 = m_W1;
		W1_last = W1;
		W2 = m_W2;
		W2_last = W2;
		L1 = m_L1;
		L1_last = L1;
		L2 = m_L2;
		L2_last = L2;
		nopt = m_nopt;
		nopt_last = nopt;
		cdamp = m_cdamp;
		cdamp_last = cdamp;
		Initialise(TRUE,FALSE);   //PARTIAL
	}

	if (theApp.FIRSTLOOP)
	{
		mlast(1) = RefluxMassFlow;                       //[t/h]
		mlast(2) = ReboilerDuty;                         //[MW]
		mlast(3) = FeedMassFlow;                         //[t/h]
		lasterr = -9999.0;  // signal
		mlastinterp(1) = RefluxMassFlow;                 //[t/h]
		mlastinterp(2) = ReboilerDuty;                   //[MW]
		mlastinterp(3) = FeedMassFlow;                   //[t/h]
		theApp.FIRSTLOOP=0;
	}

	while (((theApp.t-tlast_DMC)>=dtDMC) | (theApp.b_FORCE_STEP))
	{
		if (!theApp.b_FORCE_STEP)
		{
			tlast_DMC += dtDMC;     // must catch up by looping more than once if necessary
		}
		else
		{
			tlast_DMC = theApp.t;        // to SYNCHRONISE manual moves : b_FORCE_STEP was set TRUE
							//on StepMode button
			theApp.b_FORCE_STEP = FALSE;
		}

		dml(1) = RefluxMassFlow-mlast(1);                //[t/h]
		dml(2) = ReboilerDuty-mlast(2);                  //[MW]
		dml(3) = FeedMassFlow-mlast(3);                  //[t/h]

		mlast(1) = RefluxMassFlow;
		mlast(2) = ReboilerDuty;
		mlast(3) = FeedMassFlow;

		// past moves shift one down dmp stack
		for (i=1; i<=(Ndmc-1); i++)
		{
			for (j=1; j<=(Pdmc+Qdmc); j++)
			{
				dmp((i-1)*(Pdmc+Qdmc)+j) = dmp(i*(Pdmc+Qdmc)+j);
			}
		}
		for (j=1; j<=(Pdmc+Qdmc); j++)
		{
			dmp((Ndmc-1)*(Pdmc+Qdmc)+j) = dml(j);  // previous move
		}
```

```
if (b_AUTO)
{
        // on AUTO ! ---- do DMC Control !
        for (i=1;i<=Ndmc;i++)
        {
                xsp((i-1)*Rdmc+1) = dTAsp;
                xsp((i-1)*Rdmc+2) = dTBsp;
                x0((i-1)*Rdmc+1) = dTA;
                x0((i-1)*Rdmc+2) = dTB;
        }

        if (lasterr(1) == -9999.0)
        {
                // Initialise
                lasterr = x0 - B0*dmp;
                graderr = 0;
        }
        else
        {
                // smooth estimate of integral error
                double alpha = 0.05;   // filter constant
                graderr = alpha*(x0 - B0*dmp - lasterr)  +  (1.0-alpha)*graderr;
                for (i=1;i<=Ndmc;i++)for (j=1;j<=Rdmc;j++) trajerr((i-1)*Rdmc+j) = i*graderr(j); // ramp
                lasterr = x0 - B0*dmp;
        }

        // set constraints
        m0.Init((Mdmc*Pdmc),1);
        ml=m0;
        mh=m0;
        dmmax = m0;
        for (i=1;i<=Mdmc;i++) for (j=1;j<=Pdmc;j++)
        {
                m0((i-1)*Pdmc+j) = mlast(j);  // repeat it through the vector
                if (j==1) ml((i-1)*Pdmc+j) = LoLim_R;
                if (j==2) ml((i-1)*Pdmc+j) = LoLim_Q;
                if (j==1) mh((i-1)*Pdmc+j) = HiLim_R;
                if (j==2) mh((i-1)*Pdmc+j) = HiLim_Q;
                if (j==1) dmmax((i-1)*Pdmc+j) = RampLim_R;   // ramp limits
                if (j==2) dmmax((i-1)*Pdmc+j) = RampLim_Q;
        }

        xl = x0;   // just to initialise
        xh = x0;
        for (i=1;i<=Ndmc;i++)
        {
                xl((i-1)*Rdmc+1) = LoLim_dTA;
                xl((i-1)*Rdmc+2) = LoLim_dTB;
                xh((i-1)*Rdmc+1) = HiLim_dTA;
                xh((i-1)*Rdmc+2) = HiLim_dTB;
        }

        Hdmc.Init(m0.nRow,m0.nRow);     // for combination of elements of m
        zh = m0; // to get size right
        zl = m0;
        double fraction_allowed = Gap_Constraint_Percent/100;  //allowed fractional deviation fr. equilib line
        double slope  = Qdatum / Rdatum;
        for (int k=1;k<=Mdmc;k++)
        {
                int axoff = (k-1)*Pdmc;
                Hdmc(axoff+1,axoff+1) = 1;
                Hdmc(axoff+1,axoff+2) = -1/slope;
                Hdmc(axoff+2,axoff+1) = -slope;
                Hdmc(axoff+2,axoff+2) = 1;
                zh(axoff+1) =   fraction_allowed*Rdatum + Rdatum  - Qdatum/slope;
                zh(axoff+2) =   fraction_allowed*Qdatum + Qdatum  - slope*Rdatum;
                zl(axoff+1) =  -fraction_allowed*Rdatum + Rdatum  - Qdatum/slope;
                zl(axoff+2) =  -fraction_allowed*Qdatum + Qdatum  - slope*Rdatum;
        }
```

```
                    if (DMCstep(Ndmc,Mdmc,Pdmc,Qdmc,Rdmc,
                            &x0,&xsp,&dmp,
                            &B,&B0,&Bol,&Wdmc,&Ldmc,&Kdmc,&Hdmc,&w,&phi,
                            &m0,&mh,&ml,&dmmax,&xh,&xl,&zh,&zl,&trajerr,&dmuqo,&dm))
                    {
                            RefluxMassFlow = mlast(1) + dm(1);
                            ReboilerDuty = mlast(2) + dm(2);
                            m_dmc_gap = 100*sqrt(pow((dm(1)-dmuqo(1))/Rdatum,2)/2.0 +
                                            pow((dm(2)-dmuqo(2))/Qdatum,2)/2.0); // normalised % gap
                            if (m_dmc_gap > 0)
                            {
                                    b_constrained = TRUE;  // shortcut - see slack variable method with ry below !!!####
                            }
                            else
                            {
                                    b_constrained = FALSE;
                            }
                    }
                    m_R = RefluxMassFlow;
                    m_Q = ReboilerDuty;
            }
    }
```

7.3 SRE – SISO PID and DMC controllers based on step response

Use of a measured openloop step response to design SISO PID and DMC controllers, and comparison of their performance in response to a setpoint step



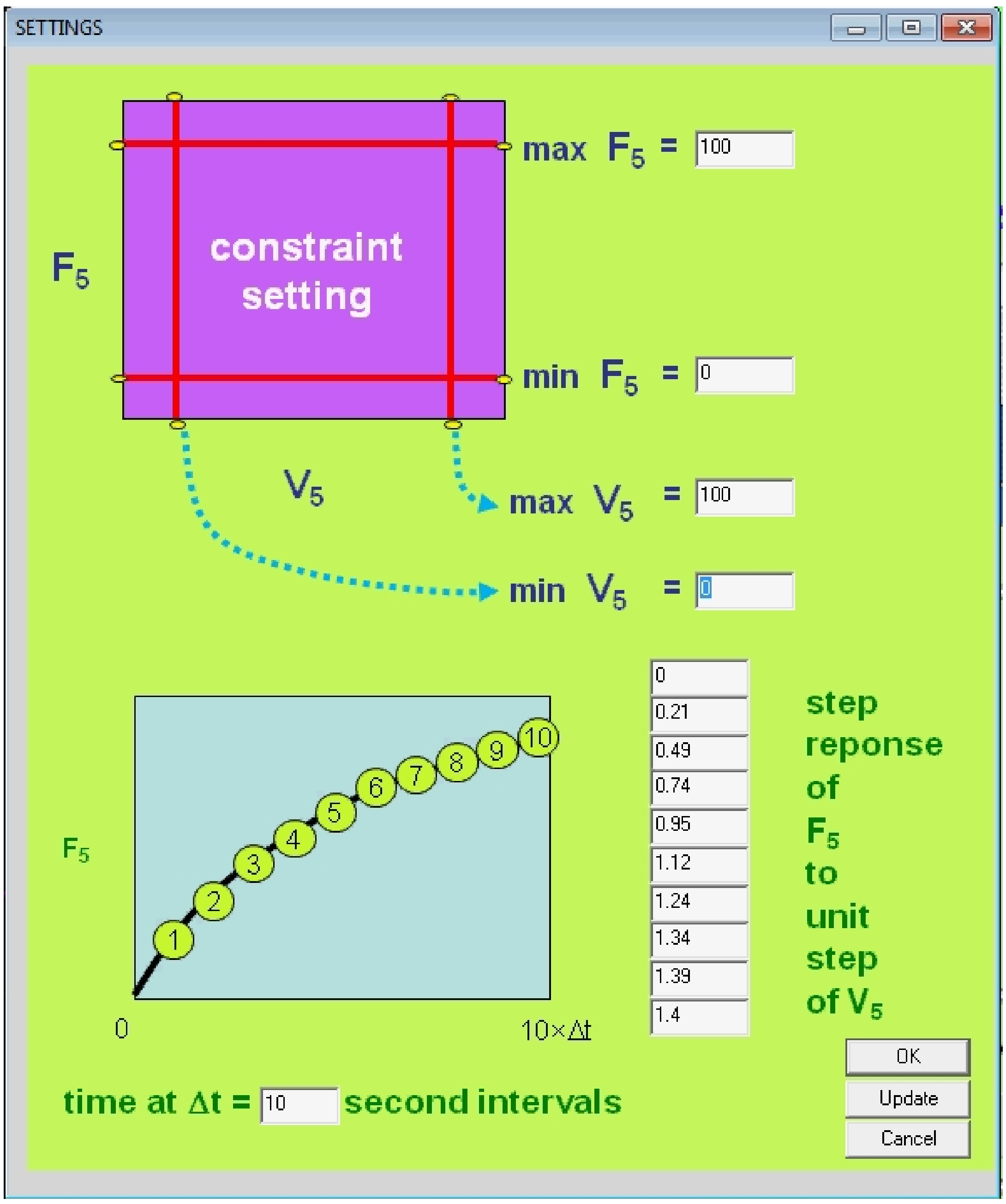
f_7.11

7.3.1 Typical settings

FC for *F5* by *V5* K_C= 1.3	**DMC for *F5* by *V5*** *Move suppression weight:* *V5* : 10

τ_I=20			
τ_D=5	*Setpoint deviation weight:*	*F5*	: 100
α=0.1			
F5$_{SP}$=50	*Number of optimised future moves*		: 1
V5=50			

Measured step reponse & constraint settings:



max F_5 = 100

min F_5 = 0

max V_5 = 100

min V_5 = 0

1	0.21
2	0.49
3	0.74
4	0.95
5	1.12
6	1.24
7	1.34
8	1.39
9	1.4

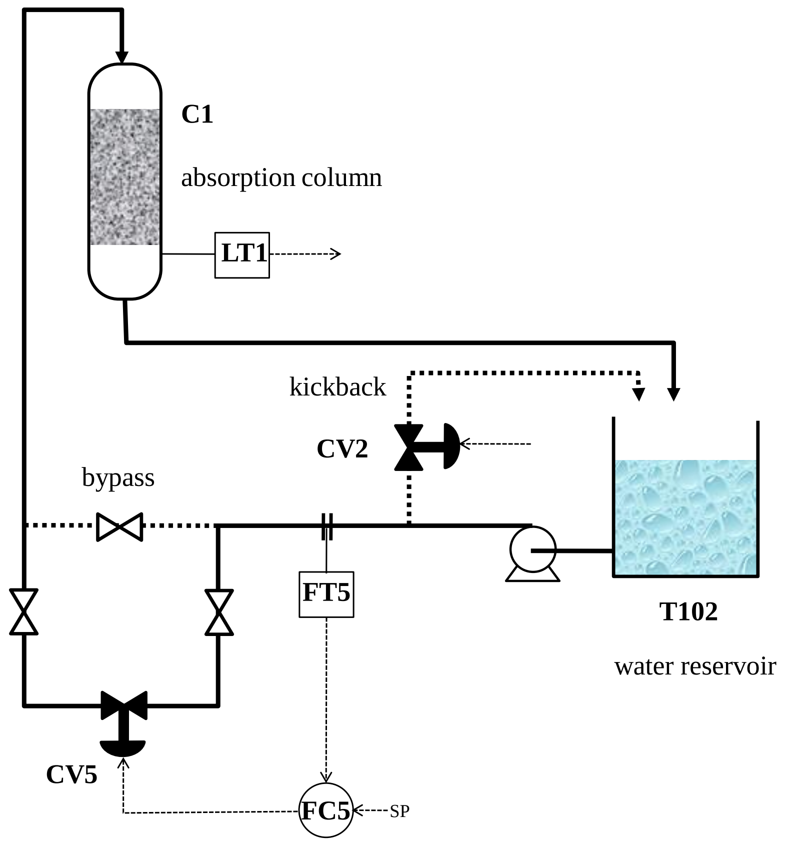
0

step reponse of F_5 to unit step of V_5

time at Δt = 10 second intervals

*f*_7.12

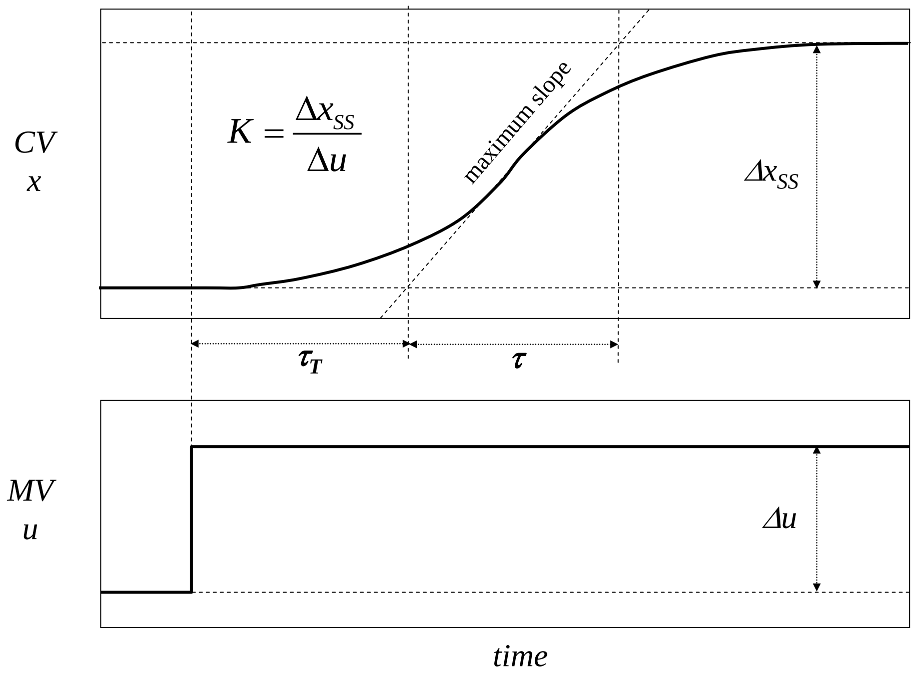
7.3.2 Theory



*f*_7.13

A pump supplies water from tank T102 to an elevated packed column as shown in the figure. The kickback valve CV2 can be used to vary the resistance line of the network, changing the head that can be supported on the pump characteristic. The orifice plate and DP cell FT5 are used to measure the flow to the distributor at the top of the column, for different valve positions of CV5. Thus the focus will be on the relationship between FT5 and CV5. This exercise will normally be conducted with CV2 at 0% open. CV5 is initially varied manually in openloop to obtain the desired openloop responses. The controller settings based on these responses are then implemented in the closedloop FC5, and tested.

The Reaction Curve tuning method (Richards, 1979) is detailed in section 4.1.1. It is based on the “open-loop” response to a step input. More specifically, the controller is left out of the circuit altogether, and the *manipulated variable* (valve position) is stepped. This will result in a response of the proposed *controlled variable*, which for most systems has the form shown in figure *f*_7.14. In the flow control system considered here, F5 will respond to a step in V5.



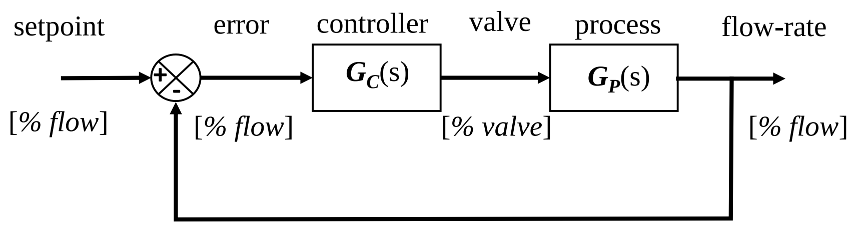
*f*_7.14

Table 7.1 Ziegler-Nichols *reaction curve* controller settings

	P	PI	PID
K_C	$\frac{\tau}{K\tau_T}$	$\frac{0.9\tau}{K\tau_T}$	$\frac{1.2\tau}{K\tau_T}$
τ_I	-	$\frac{\tau_T}{0.3}$	$\frac{\tau_T}{0.5}$
τ_D	-	-	$\frac{\tau_T}{2}$

In the "reaction curve" tuning method, Ziegler and Nichols (1942) proposed "good" settings for **P**, **PI** & **PID** controllers based on these measurements as in Table 7.1. Presently, both a **P** and a **PID** controller will be designed on this basis, and tested in closed loop. Furthermore, as described in section 7.1, the reaction curve will also be used to create the matrices $\boldsymbol{B}$, $\boldsymbol{B_{OL}}$ and $\boldsymbol{B_0}$ required for a SISO dynamic matrix controller.

The *gain margin* of the **PID** control loop will be calculated using the assumed *first-order-plus-dead-time* model. The process reaction curve represents a **% flow** output in response to **% valve-open** input. Clearly, process models derived on this basis will lump the dynamics of the control element, the pump and the measuring element, and other converters in the circuit. The attitude taken is that anything lying between the **% valve-open** input and **% flow** output is "process" (*f*_7.15):



*f*_7.15

Observation of the response of the process to a step input suggests that it might be adequately described by a 1st order lag with dead-time in series:

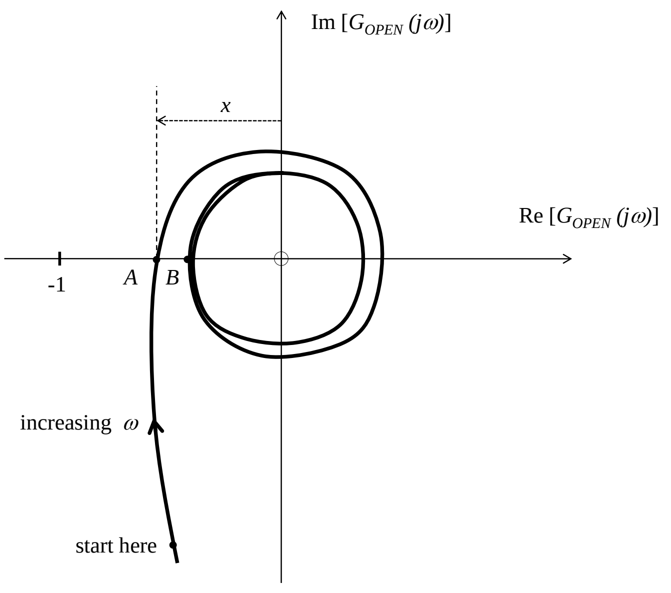
$$G_P(s) = \frac{Ke^{-\tau_T s}}{\tau s + 1} \tag{7.32}$$

This may be fitted to the process step responses using average values of the parameters τ_T, the dynamic lag τ, and the open-loop gain $K = \Delta x / \Delta u \,\big|_{ss}$

The assumed form of $G_p(s)$ may then be used in conjunction with the chosen **PID** controller to obtain an open-loop transfer function:

$$G_O(s) = G_P(s) G_C(s) = \left[\frac{Ke^{-\tau_T s}}{\tau s + 1}\right]\left[K_C\left(1 + \frac{1}{\tau_I s} + \tau_D s\right)\right] \tag{7.33}$$

The corresponding Nyquist plot has the form shown in figure *f*_7.16 below



*f*_7.16

In order to establish the Gain Margin of this assumed system, we must find the point A at which the Nyquist Plot crosses the – ve Real axis with the biggest negative real value. Notice that the transport lag term $e^{-\tau_T s}$ would normally cause a continuous spiral inwards to the origin as $s = j\omega$ increases in the frequency response. (and $\left|G_O\left(j\omega\right)\right|$ decreases). For this particular idealised open-loop, the phase angle does become increasingly negative as expected, but the magnitude becomes constant as frequency increases. Why? What will be the asymptotic magnitude?

We are naturally interested in the crossing 'A' which is most negative (largest distance *x* from the origin), since this contour will be the first to enclose (–1+*j*0) with increasing K_c and thus cause closed-loop instability. Thus care must be taken in plotting $G_O\left(j\omega\right)$ to start with a sufficiently low frequency (e.g. 0.01 rad s^{-1}) to avoid proceeding inwards on one of the inner contours, and perhaps only finding point 'B'.

Starting at a low frequency, plot points at successively higher frequencies until the – ve real axis is crossed. Join these points with a smooth curve to find A, then

$$\text{Gain Margin}= G_O\left(j\omega\right) = \frac{1}{x} = -20\log_{10}\left(x\right) \text{ db} \tag{7.34}$$

(Note that we <u>could</u> simply have solved for ω which gives $angle\left\{G_O\left(j\omega\right)\right\} = 180^0$, then substituted this ω into $\left|G_O\left(j\omega\right)\right|$ to obtain x.)

7.3.3 Tasks

(1) Obtain the dynamic response of the flow-rate F5 to a step in valve position V5.

(3) Derive recommended Ziegler-Nichols parameters based on this "Reaction Curve" for **P** and **PID** controllers for the SISO loop F5 by V5.

(4) Use your measured step-response matrix to set up a Dynamic Matrix Controller (**DMC**)

(5) Assess the performance of each of these controllers in closed-loop control by stepping the set-points and evaluating a quadratic performance index.

(6) Use an assumed system model to calculate the Gain Margin of the system under control by the **PID** controller.

7.3.4 Method

(1) Start the RTC program and select the "SRE" application.

(2) The graphic user interface (GUI) now allows you to set the positions of control valves V2 and V5. Set V2 to 0% open and V5 to 50% open, and allow the system to achieve steady state.

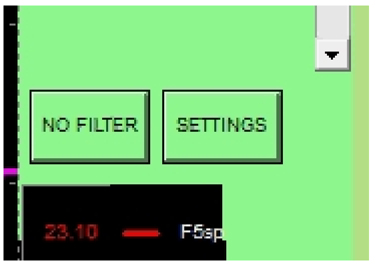
(3) Start logging to a data file with a time-interval of 5 seconds.

(4) Note the computer time for future reference to your logging file.

(5) Now input a step of +10% in valve V5 position. (In the case of control of a real plant, an initial test for valve hysteresis should be done by comparing a response occurring following continuous movement of the valve in one direction, with that occurring following a reversing step. Furthermore, to minimise the impact of non-linearity, all steps should be made from a fixed initial condition, close to the normal anticipated operating point).

(6) Calculate the recommended Ziegler-Nichols setting K_c for a **P** controller, and K_C, τ_I and τ_D for each **PID** controller. Set these values using the scroll-bars for each case.

(7) Choose 10 equally-spaced points in the step response (The suggested spacing is 10s, so beware that the log-file interval is 5s. Also, the default interval on the SETTINGS screen may need to be changed to 10s). The portion of curve should represent a smooth variation of F5 up to steady-state. Divide the measurements by the size of the input step (+10%) to get a "unit step response". Enter these values in the response sequence under the settings section, reached by pressing the SETTINGS button (*f*_7.17).

*f*_7.17 

The Settings screen also allows you to change the sample interval, plus constraints that may be applied to V5 and F5 (for DMC control only). Additional constraints include "ramp" limits, ie. the maximum change in V5.

(8) The associated Dynamic Matrix Controller is tuned using the DMC weight settings on the GUI. These set the terms in the diagonal weighting matrices $\boldsymbol{W}$ (for F5 setpoint deviations) and $\boldsymbol{\Lambda}$ (for V5 move suppression). The other tuning parameter is the "number of optimised moves", also set on the GUI.

(9) Start a new data log file if necessary. For each closed-loop set-point step response, start with the system on "manual" at steady-state, with V5 at the same initial settings. Match the set-point to the feedback F5 before switching to "auto". Once you are on "auto", let the system settle for a short while (eg. 30s). Now obtain the level responses to a step of + 20% in flow F5 set-point with the system under closed-loop **P**, **PID** and **DMC** control. In the case of **DMC** control, investigate the effect of changing from 1 to 2 or 3 "optimised moves", as well as the effect of varying the V5 and F5 weights.

(10) Let the system reach steady-state under DMC control. Now create a load disturbance by stepping V2 open by 20%. Note the response.

(11) Process your log-file as follows: Plot the individual openloop and closedloop responses, and comment on them. For each closedloop response, calculate a quadratic performance index for each flow controller, starting at the time when the setpoint was stepped, and continuing for a fixed period thereafter.

$$QPI = \sum_{i=1}^{N} \left[F_5\left(i\Delta t\right) - F_{5SP}\left(i\Delta t\right) \right]^2 \qquad (7.35)$$

Comment on the differences caused by the various configurations and settings.

(12) Using the assumed form of the closedloop transfer function under **PID** control, construct a section of the Nyquist plot for the **PID** controller, and use this to estimate the Gain Margin.

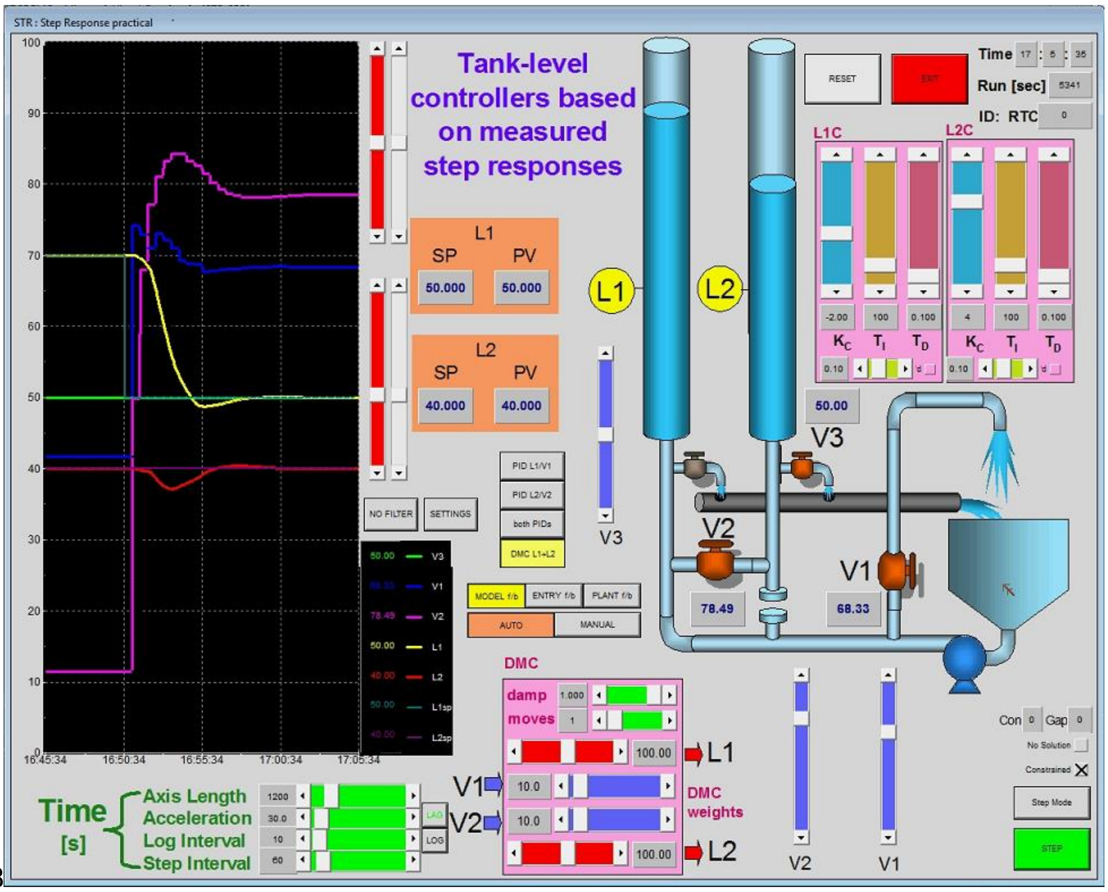
(13) Comment on all of your observations and calculated results.

7.3.5 MATLAB® program for the Nyquist plot

```
% Magnitude and Phase Angle of a Process with PID controller
clc
close all
clear

% define the variables:
%ZN
Kc = -0.8; %
tauI = 30;
tauD = 4;
%Open Loop
K = -0.678;
tau = 23.33;
tauT = 18.43; % (L) lag time
w=[0.01:0.001:10];% Freq array

for j = 1:length(w)
   s = w(j)*2*pi*i; % vector of different 's' values of frequency
   % calculating the transfer function G(s)
   G(j) = Kc*(1 + 1/(tauI*s) + tauD*s)*(K * exp(-tauT*s))/(1+ tau*s);
end

% Plotting the Nyquist plot (Re vs. Im)
x = real(G);
y = imag(G);
plot(x,y);
xlabel('Re(G(s))');
ylabel('Im(G(s))');
title('Nyquist Plot');
line([-0.8 0.8],[0 0],'color','k','linestyle','--');
axis manual;
line([0 0],[-0.8 0.8],'color','k','linestyle','--');
grid on;
```

7.4 STR – PID and MIMO DMC controllers based on step response

Use of a measured openloop step response to design PID and DMC controllers, and comparison of their performance in response to a setpoint step



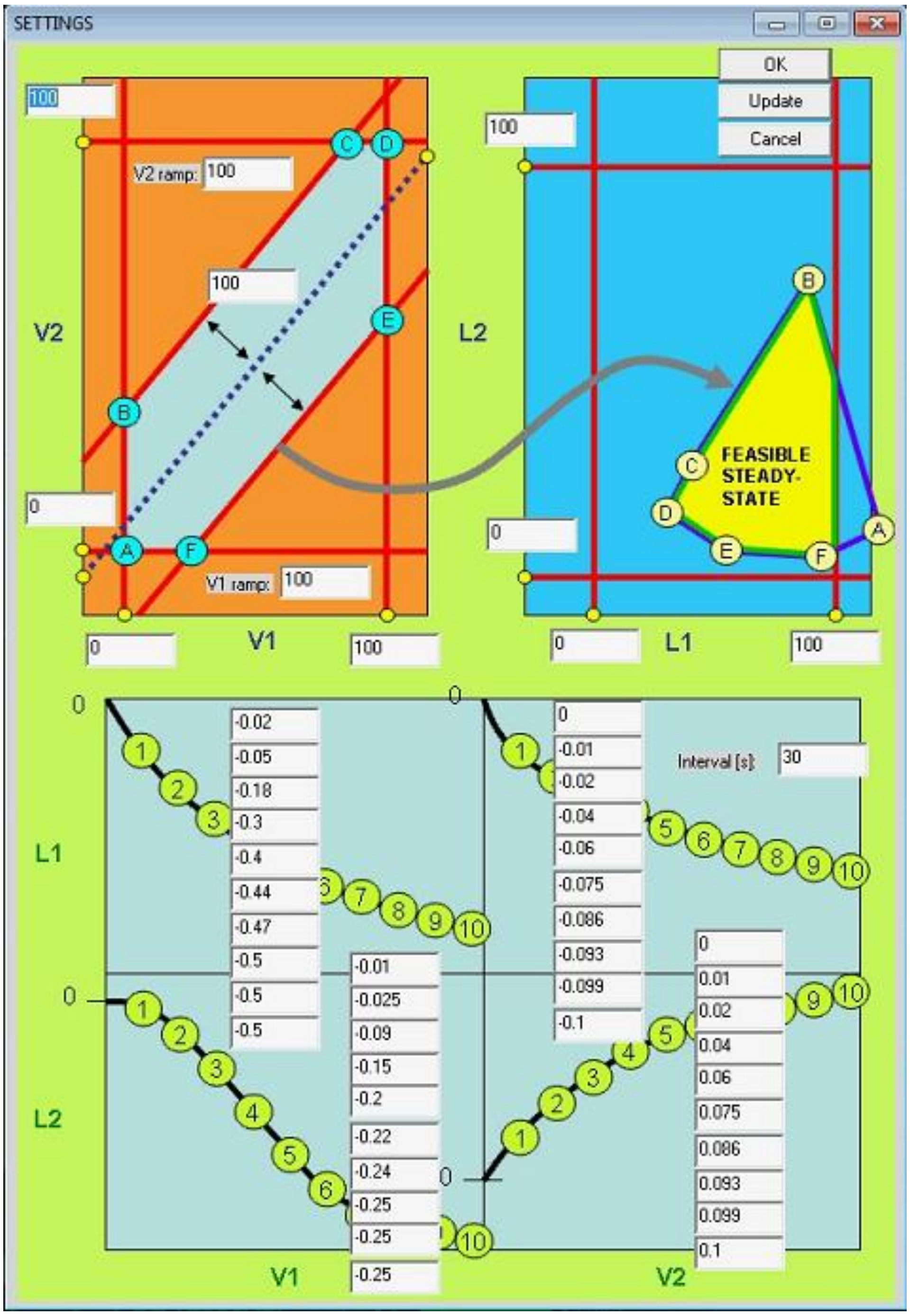
f_7.18

7.4.1 Typical settings

LC for *L*₁ **by** *V*₁	**LC for** *L*₂ **by** *V*₂	**DMC for** *L*₁ **&** *L*₂ **by** *V*₁ **&** *V*₂		
K_C= -2.0	K_C=4.0	*Move suppression weights:*	V_1	: 10

τ_I=100	τ_I=100		V_2	: 10
τ_D=0.1	τ_D=0.1			
α=0.1	α=0.1	*Setpoint deviation weights:*	L_1	: 100
L_{1SP}=60	L_{2SP}=40		L_2	: 100
V_1=55	V_2=45	*Number of optimised future moves*		: 1

Measured step reponses & constraint settings:


SETTINGS

OK
Update
Cancel

100
V2 ramp: 100
100
0
V1 ramp: 100
0
100
100
0
0
100
Interval [s]: 30

-0.02
-0.05
-0.18
-0.3
-0.4
-0.44
-0.47
-0.5
-0.5
-0.5

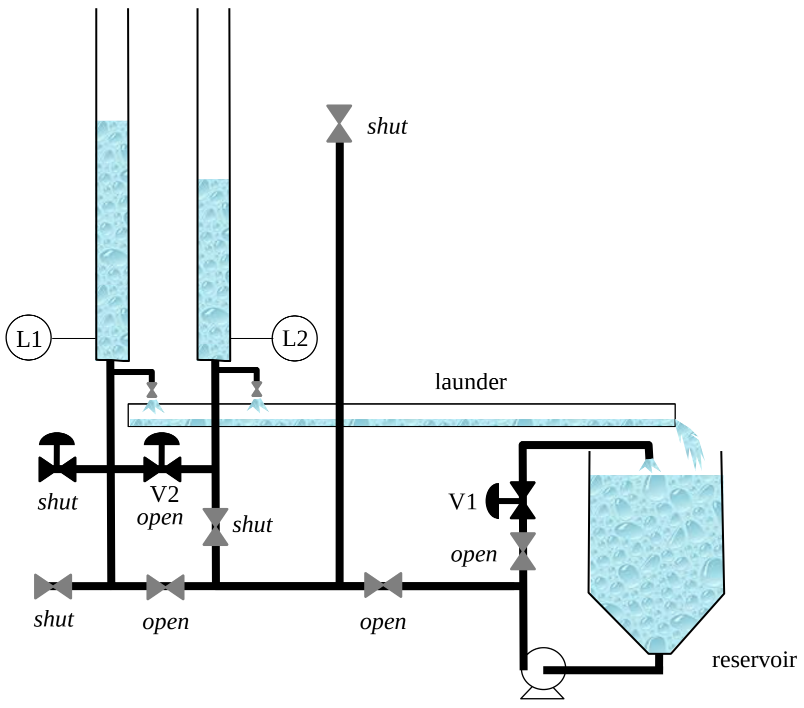
-0.01
-0.025
-0.09
-0.15
-0.2
-0.22
-0.24
-0.25
-0.25
-0.25

0
-0.01
-0.02
-0.04
-0.06
-0.075
-0.086
-0.093
-0.099
-0.1

0
0.01
0.02
0.04
0.06
0.075
0.086
0.093
0.099
0.1

f_7.19

7.4.2 Theory

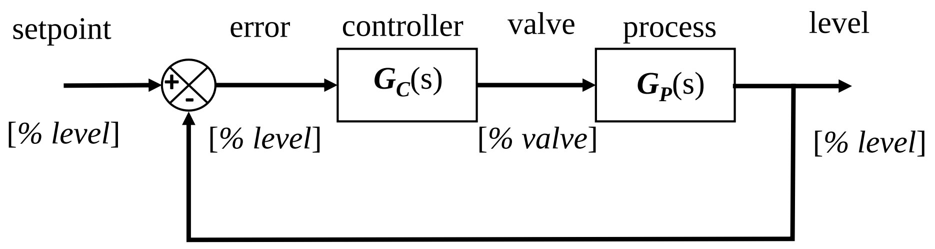


*f*_7.20

A pump supplies water from a reservoir to two interconnected 8m tall tanks as shown in *f*_7.20. The return valve V1 near the pump effectively varies the resistance line of the network, changing the head that can be supported on the pump characteristic. A head increase builds up tank level, whilst a head decrease is accommodated by dumping tank water through bleed valves to a return launder (or, indeed, by reverse-flow through the water supply line). The focus will be on the relationships V1$\rightarrow$L1 and V2$\rightarrow$L2 for the two PID loops, and [V1,V2] $\rightarrow$[L1,L2] for the DMC controller. Note the settings of the remaining valves in the figure *f*_7.20.

The Reaction Curve tuning method (Richards, 1979) is detailed in section 4.1.1. It is based on the "open-loop" response to a step input. More specifically, the controller is left out of the circuit altogether, and the *manipulated variable* (valve position) is stepped. This will result in a response of the proposed *controlled variable,* which for most systems has the form shown in *f*_4.1. In the pump-tank system considered here, both L1 and L2 will respond to separate steps in V1 and V2, creating a *matrix* of step responses. As described in section 7.1, these may be used to create the *matrices of matrices,* $\boldsymbol{B}$, $\boldsymbol{B_{OL}}$ and $\boldsymbol{B_0}$ required to construct a dynamic matrix controller.

In this study, the *gain margin* of each of the separate PID control loops will be calculated using the assumed *first-order-plus-dead-time* model. The available process reaction curves represent a **% level** output in response to **% valve-open** input. Clearly, process models derived on this basis will lump the dynamics of the control element, the pump/tank process, the measuring element, and other converters in the circuit. The attitude taken is that anything lying between the **% valve-open** input and **% level** output is "process" (*f*_7.21):



*f*_7.21

Observation of the response of the process to a step input suggests that it might be adequately described by a 1st order lag with dead-time in series:

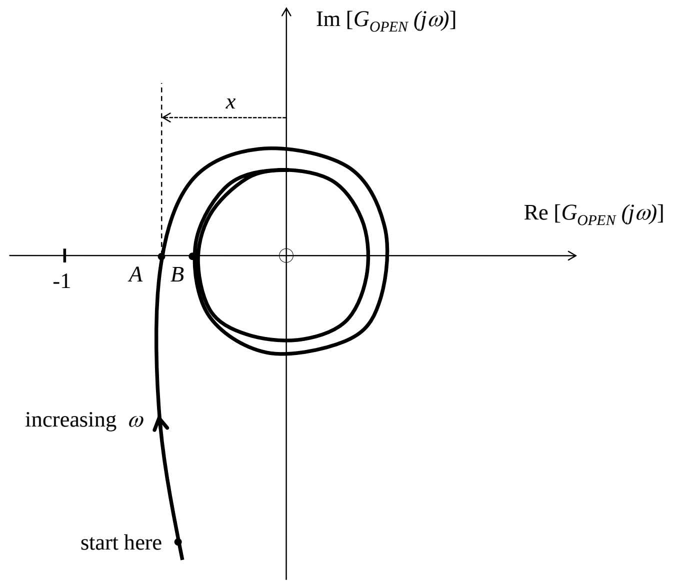
$$G_P(s) = \frac{Ke^{-\tau_T s}}{\tau s + 1} \tag{7.36}$$

This may be fitted to the process step responses using average values of the parameters T_T (τ_T) , the dynamic lag T_S (τ), and the open-loop gain $K = \Delta x / \Delta a \big|_{SS}$ (*f*_4.1).

The assumed form of $G_P(s)$ may then be used in conjunction with the chosen PID controller to obtain an open-loop transfer function:

$$G_O(s) = G_P(s) G_C(s) = \left[\frac{Ke^{-\tau_T s}}{\tau s + 1}\right]\left[K_C\left(1 + \frac{1}{\tau_I s} + \tau_D s\right)\right] \tag{7.37}$$

The corresponding Nyquist plot has the form shown in the figure below



*f*_7.22

In order to establish the Gain Margin of this assumed system, we must find the point A at which the Nyquist Plot crosses the – ve Real axis with the biggest negative real value. Notice that the transport lag term $e^{-\tau_T s}$ would normally cause a continuous spiral inwards to the origin as $s = j\omega$ increases in the frequency response. (and $\left|G_O\left(j\omega\right)\right|$ decreases). For this particular idealised open-loop, the phase angle does become increasingly negative as expected, but the magnitude becomes constant as frequency increases. Why?

We are naturally interested in the crossing 'A' which is most negative, since this contour will be the first to enclose (–1+*j*0) with increasing K_c and thus cause closed-loop instability. Thus care must be taken in plotting $G_O\left(j\omega\right)$ to start with a sufficiently low frequency (e.g. 0.01 rad s^{-1}) to avoid proceeding inwards on one of the inner contours, and perhaps only finding point 'B'.

Starting at a low frequency, plot points at successively higher frequencies until the – ve real axis is crossed. Join these points with a smooth curve to find A, distance *x* from the origin, then

$$\text{Gain Margin} = G_O\left(j\omega\right) = \frac{1}{x} = -20\log_{10}\left(x\right) \text{ db} \tag{7.38}$$

(Note that we <u>could</u> simply have solved for ω which gives $angle\left\{G_O\left(j\omega\right)\right\} = 180^0$, then substituted this ω into $\left|G_O\left(j\omega\right)\right|$ to obtain x.)

7.4.3 Tasks

(1) Obtain the dynamic responses of the levels L1 and L2 to separate steps in valve positions V1 and V2.

(3) Derive recommended Ziegler-Nichols parameters based on these “Reaction Curves” for **P** and **PID** controllers for separate SISO loops L1 by V1, and L2 by V2.

(4) Use your measured step-response matrix to set up a Dynamic Matrix Controller (**DMC**)

(5) Assess the performance of each of these controllers in closed-loop control by stepping the set-points and evaluating a quadratic performance index.

(6) Use an assumed system model to calculate the Gain Margins of the the two SISO systems under control by their **PID** controllers.

7.4.4 Method

(1) Start the RTC program and select the “STR” application.

(2) The graphic user interface (GUI) now allows you to set the positions of control valves V1 and V2. Set to mid-range and allow the system to achieve steady state.

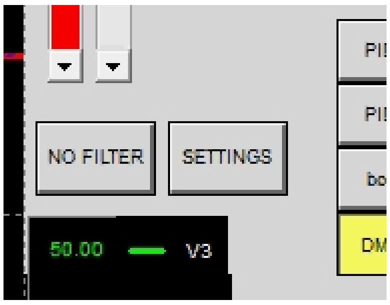
(3) Start logging to a data file with a time-interval of 5 seconds.

(4) Note the computer time for future reference to your logging file.

(5) Now input a step of +20% in valve V1 position. Repeat to get the responses to a +20% step in V2. (In the case of control of a real plant, an initial test for valve hysteresis should be done by comparing a response occurring following continuous movement of the valve in one direction, with that occurring following a reversing step. Furthermore, to minimise the impact of non-linearity, all steps should be made from a fixed initial condition, close to the normal anticipated operating point).

(6) Calculate the recommended Ziegler-Nichols settings K_c, τ_I and τ_D for each **PID** controller. Set these values using the scroll-bars for each case.

(7) Choose 10 equally-spaced points (default **30s** intervals – see instruction below for changing this) on a smooth “average” step-response to define the variation up to steady-state for the two L1 responses, and the two L2 responses. Divide the measurements by the size of the input step (20%) to get a “unit step response”. Enter these values in the Dynamic Matrix under the settings section, reached by pressing the SETTINGS button (*f*_7.23).



f_7.23

This screen also allows you to change the sample interval, plus constraints that may be applied to V1, V2, L1 and L2 (for DMC control only). Additional constraints include "ramp" limits, ie. the maximum change in V1 or V2 per interval, as well as a facility to to maintain the ratio of V1 to V2 within defined bounds.

(8) The associated Dynamic Matrix Controller is tuned using the DMC weight settings on the GUI. These set the terms in the diagonal weighting matrices ***W*** (for L1 and L2 setpoint deviations) and ***Λ*** (for V1 and V2 move suppression). The other tuning parameter is the "number of optimised moves", also set on the GUI.

(9) Start a new data log file if necessary. For each closed-loop set-point step response, start with the system on "manual" at steady-state, with V1 and V2 at the same initial settings. Match the set-points to the feedback L1, L2 before switching to "auto". Once you are on "auto", let the system settle for a short while (eg. 30s). Now obtain the level responses to a step of + 20% in level L1 set-point, and separately to a +20% step in L2 setpoint, with the system under closed-loop **PID** and **DMC** control. In the case of **PID** control, compare the situation when the other **PID** controller is left on MANUAL, with that when both **PID** controllers are on AUTO. In the case of **DMC** control, investigate the effect of changing from 1 to 2 or 3 "optimised moves", as well as the effect of varying the L1, L2, V1 and V2 weights.

(10) Process your log-file as follows: Plot the individual openloop and closedloop responses, and comment on them. For each closedloop response, calculate a quadratic performance index for each level control, starting at the time when the setpoint was stepped, and continuing for a fixed period thereafter.

$$QPI_1 = \sum_{i=1}^{N}\left[L_1(i\Delta t) - L_{1SP}(i\Delta t)\right]^2 \tag{7.39}$$

$$QPI_2 = \sum_{i=1}^{N}\left[L_2(i\Delta t) - L_{2SP}(i\Delta t)\right]^2 \tag{7.40}$$

Comment on the differences caused by the various configurations and settings.

(11) Using the assumed form of the process transfer function (first-order plus dead-time), construct a section of the Nyquist plot for each **PID** controller, and use this to estimate the Gain Margin in each case.

7.4.5 MATLAB® procedure for calculation of frequency response

```
% Magnitude and Phase Angle of a Process with PID controller
K = 1;
Tt = 3;    % [s]
T = 20;    % [s]
Kc = 1;
Ti = 10;   % [s]
Td = 2;    % [s]
% Input frequencies (Hz):
freq = [  0.0001;
          0.001 ;
          0.01  ;
          0.1   ;
          1.0   ;
         10.0   ;
        100.0]  ;
for k=1:size(freq,1)
   s = (2*pi*freq(k))*i;
   Gc = Kc*(1 + 1/(Ti*s) + Td*s);
   mag_Gc = 20*log10(abs(Gc));  % [db]
   ang_Gc = 180*angle(Gc)/pi;   % [deg]
   Gp = K*exp(-Tt*s)/(T*s+1);
   mag_Gp = 20*log10(abs(Gp));  % [db]
   ang_Gp = 180*angle(Gp)/pi;   % [deg]

   mag_GcGp = mag_Gc+mag_Gp;
   ang_GcGp = ang_Gc+ang_Gp;
   [freq(k) mag_GcGp ang_GcGp]
end
```

7.4.6 Code extracts

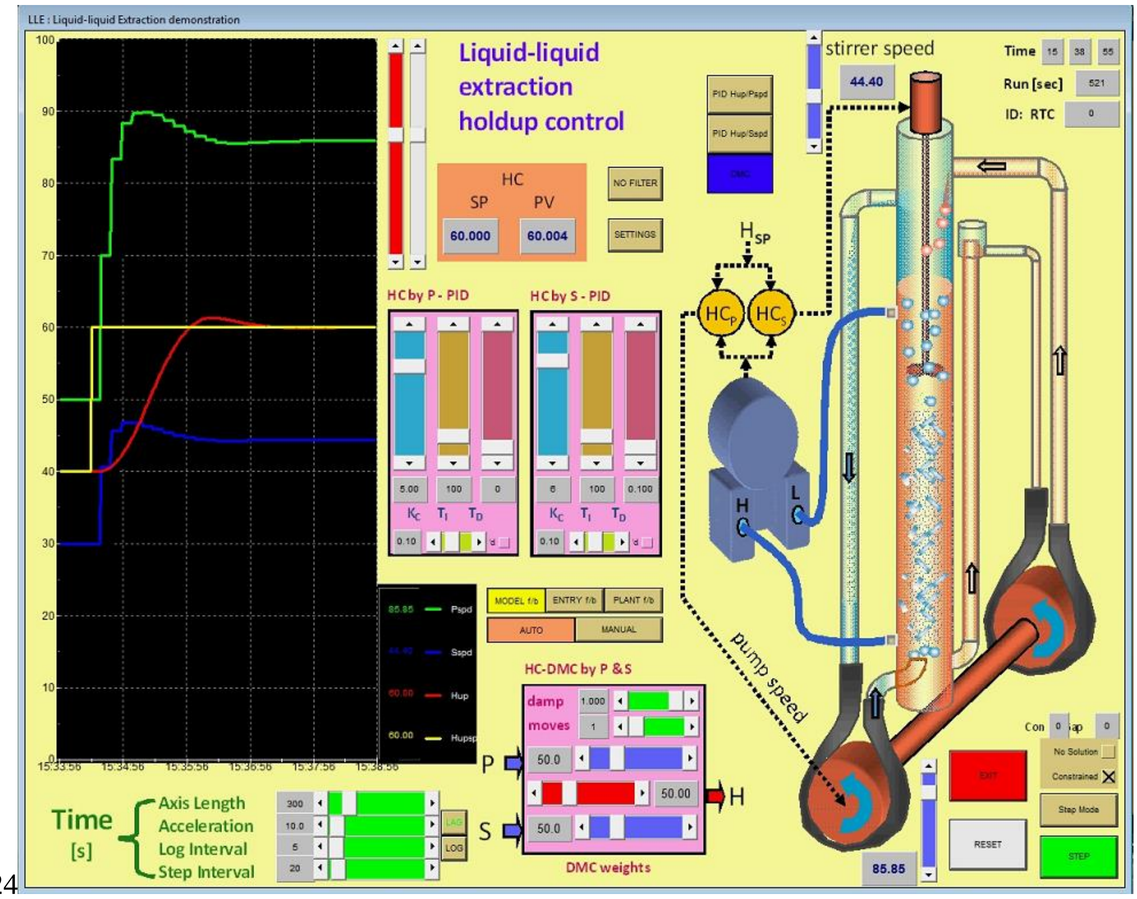
- Final loading of arrays for the LPSOLVE linear programming solution, plus unpacking of the results returned by LPSOLVE (Michel Berkelaar - http://sourceforge.net/projects/lpsolve/):

```
// Finally have to order first the "objective", then the "less thans" then the "greater thans"
CLPproblem LP;
LP.n   = NTc-1;
LP.m1  = NTr-1;  //temporary
LP.m2  = 0;
LP.m3  = 0;
LP.eps = 1e-12;  // improve the way this is set !

LP.SetUp();
LP.m1=0;
LP.m=0;
int lessthans[MAX_VECTOR_SIZE];
int greaterthans[MAX_VECTOR_SIZE];
for (i=1;i<=NTr;i++)
{
        if(I[i]==-1)
        {
                LP.m1 += 1;
                lessthans[LP.m1]=i;
        }
        if(I[i]==+1)
        {
                LP.m2 += 1;
                greaterthans[LP.m2]=i;
```

```
		}
	}
	LP.m = LP.m1 + LP.m2;

	for (i=1;i<=NTr;i++)
	{
		if(i==1)   //objective
		{
			for (j=1;j<=NTc;j++)
			{
				LP.a(i,j) = G(1,j);
			}
		}
		if((i>1) && (i<=(1+LP.m1)))   //a lessthan
		{
			for (j=1;j<=NTc;j++)
			{
				LP.a(i,j) = G(lessthans[i-1],j);
			}
		}
		if(i>(1+LP.m1))   //a greaterthan
		{
			for (j=1;j<=NTc;j++)
			{
				LP.a(i,j) = G(greaterthans[i-(1+LP.m1)],j);
			}
		}
	}


	// SOLVE THE LP
	int icase = LP.Solve();
	m_lastconstraint = 0;
	if (icase!=0)
	{
		*dm=0;		// don't move
		b_no_solution = TRUE;
		return(FALSE);  // problem!
	}
	else
	{
		// find the result values  in the tableau
		double r[MAX_VECTOR_SIZE],ry[MAX_VECTOR_SIZE];
		for (j=1; j<=MAX_VECTOR_SIZE; j++)
		{
			ry[j]=99;
		}
		for (j=1; j<=LP.n; j++)
		{
			 int jj=LP.izrov[j];
			 if (jj<=LP.n)
			 {
					r[jj]=0.0;
			 }
			 else
			 {
					ry[jj-LP.n]=0.0;
			 }
		}
		for (i=1; i<=LP.m; i++)
		{
			int ii = LP.iposv[i];
			if (ii!=0)
			{
				if (ii<=LP.n)
				{
					r[ii] = LP.a(i+1,1);
				}
				else
				{
					ry[ii-LP.n] = LP.a(i+1,1);
				}
			}
		}
		// check if any variable is constrained, if BOTH bits of the slack variable are zero
		for (j=1; j<int(double(MAX_VECTOR_SIZE)/2.0); j++)
		{
			if((ry[2*(j-1)+1]==0.0) && (ry[2*(j-1)+2]==0.0))
			{
				// ####  b_constrained = TRUE;  FAULTY AT THE MOMENT!  ####MM041021
				m_lastconstraint = j;
			}
		}


		// now recombine the two bits of r
		for (i=1;i<=P;i++)
		{
			(*dm)(i) = (*dmuqo)(i) + r[2*(i-1)+1]-r[2*(i-1)+2];
		}

		return(TRUE); // no problems
	}
```

7.5 LLE – Multi-Input, Single-Output (MISO) DMC compared with PID

Examination of how the DMC move suppression weights affect the contributions of two inputs to a single-output system.



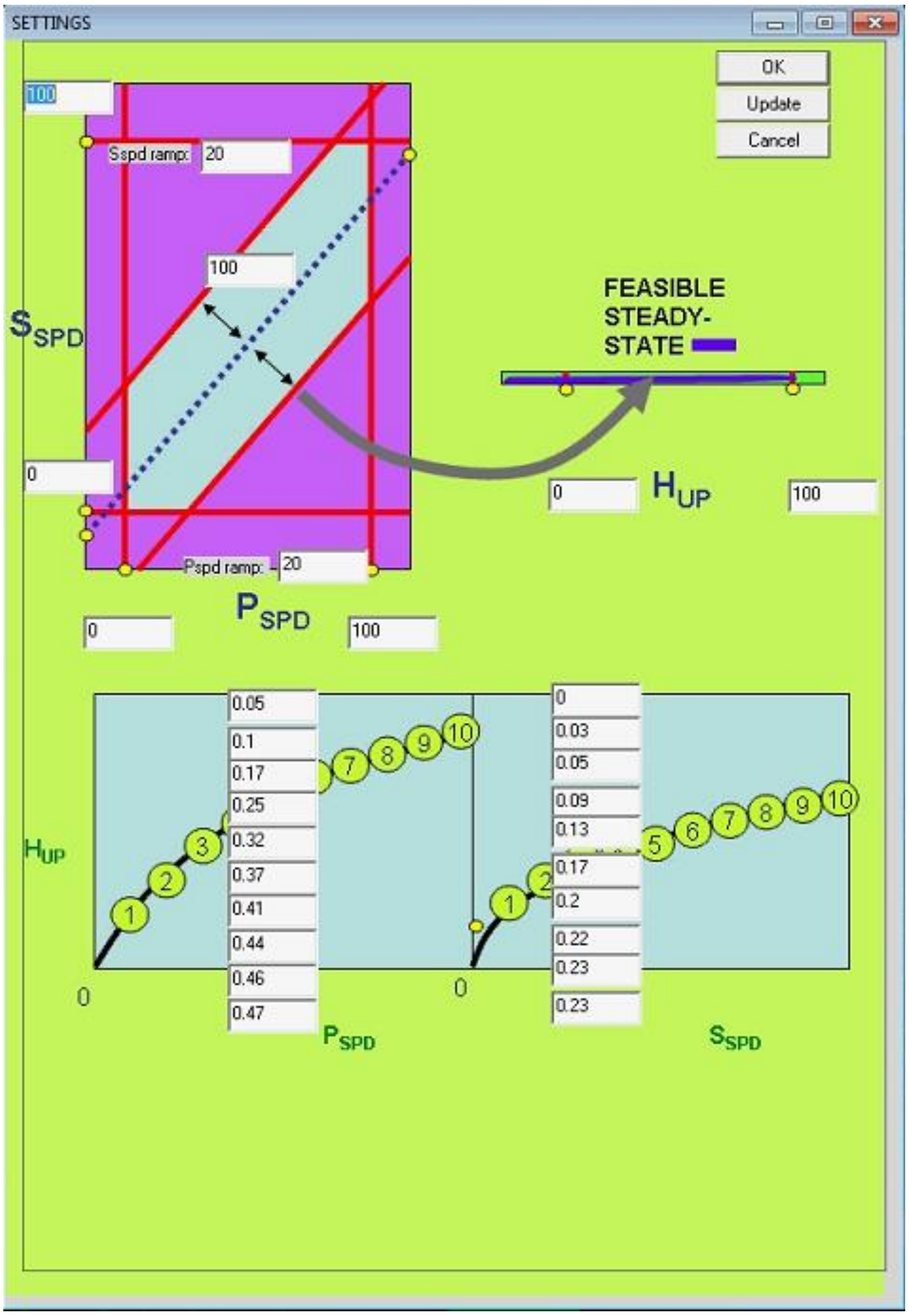
f_7.24

7.5.1 Typical settings

HC for *H* by *P*$_S$	**HC for *H* by *S*$_S$**	**DMC for *H* by *P*$_S$ & *S*$_S$**		
K_C= 5	K_C=6	*Move suppression weights:*	P_S	: 50
τ_I=100	τ_I=100		S_S	: 50

τ_D=0	τ_D=0.1			
α=0.1	α=0.1	*Setpoint deviation weight:*	*H*	: 50
H_{SP}=40	H_{SP}=40			
P_S=50	S_S=30	*Number of optimised future moves*		: 1

Measured step reponses & constraint settings:



S_{SPD}: 100, 0

Sspd ramp: 20

100

Pspd ramp: 20

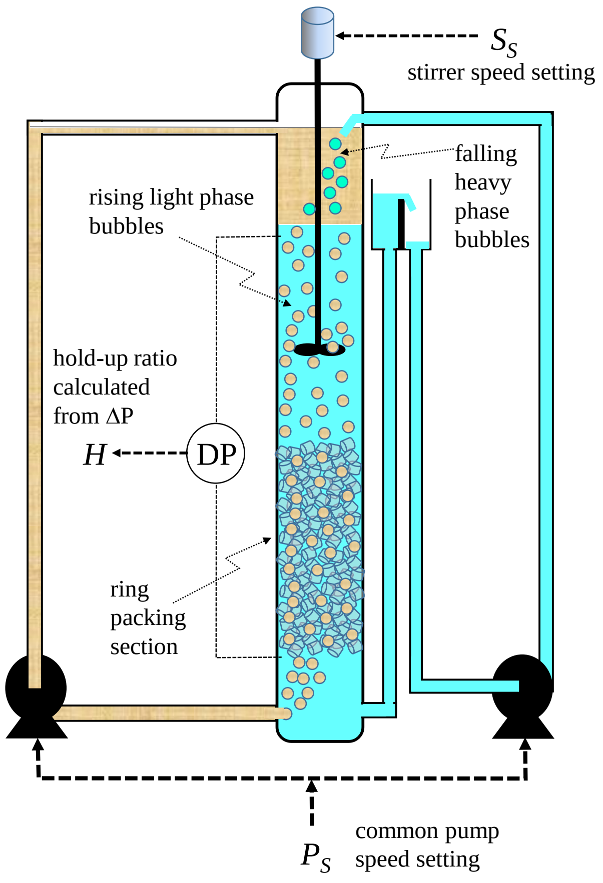
P_{SPD}: 0, 100

H_{UP}: 0, 100

P_{SPD}	S_{SPD}
0.05	0
0.1	0.03
0.17	0.05
0.25	0.09
0.32	0.13
0.37	0.17
0.41	0.2
0.44	0.22
0.46	0.23
0.47	0.23

f_7.25

7.5.2 Theory


S_S
stirrer speed setting
falling heavy phase bubbles
rising light phase bubbles
hold-up ratio calculated from ΔP
H
DP
ring packing section
common pump speed setting
P_S

*f*_7.26

A double-headed peristaltic pump supplies both heavy and light liquid phases to a liquid-liquid extraction contacting column. The rising light phase bubbles first pass through a packed section to increase interphase contact area. Above the packing the light phase continues to rise through the continuous heavy phase, under the influence of a stirrer which reduces bubble size and enhances mass transfer. Above this section the light phase coalesces in a layer which returns to the bottom of the column via an overflow and its pump head. The heavy phase leaves at the bottom of the column as shown, with its own overflow near the top, and is returned to the top of the column via its head of the pump.

Since the peristaltic heads are driven by the same variable-speed motor, their pump rates are identical and determined by the pump speed setting P_S. One expects that as this speed is increased the fraction of light phase in the measured section (*viz. H* the "hold-up ratio") will increase owing to the increased superficial velocities of the two phases. If the stirrer speed S_S is increased, the fraction of small rising bubbles will increase. The increasing surface area will increase the drag per unit mass, slowing down the bubbles and again increasing the holdup-ratio. In practice one likes to keep the hold-up ratio in such a column high, to maximise mass transfer. But if it gets too high one reaches a condition called "flooding". Here the feed of either phase cannot get through the column quickly enough, causing bodily displacement of phases. Thus a control system is required to maintain the hold-up ratio, normally just below the flooding point. The aim of this exercise is to examine the control of such a system, where two inputs (P_S,S_S) can be manipulated optimally, according to a defined criterion, in order to regulate a single output, H.

7.5.3 Tasks

(1) Obtain the dynamic responses of the hold-up ratio H to separate steps in pump speed P_S and stirrer speed S_S.

(2) Use your measured step-responses to set up a Dynamic Matrix Controller (**DMC**)

(3) Investigate the performance of the **DMC** controller for different settings of the move suppression weights on P_S and S_S, and the setpoint deviation weight on H.

7.5.4 Method

(1) Start the RTC program and select the “LLE” application.

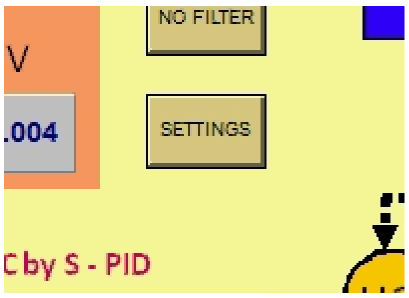
(2) The graphic user interface (GUI - *f*_7.24) now allows you to set the *pump speed* and the *stirrer speed*. Set both to mid-range and allow the system to achieve steady state.

(3) Start logging to a data file with a time-interval of 5 seconds.

(4) Note the computer time for future reference to your logging file.

(5) Now input a step of +20% in *pump speed* by pressing the cursor twice in the coloured part of the scroll-bar. Repeat to get the responses to a +20% step in *stirrer speed*.

(6) The **DMC** controller is this application will step at 10 s intervals. Thus choose 10 equally-spaced points at **10 s** intervals on a smooth “average” step-response to define the variation up to steady-state for the response to *pump speed* and the response to *stirrer speed*. Divide the measurements by the size of the input step (10%) to get a “unit step response” in each case. Enter these values into the Dynamic Matrix under the settings section, reached by pressing the SETTINGS button (*f*_7.27).



*f*_7.27

This screen also allows you to change constraints that may be applied to pump speed, P_S, stirrer speed, S_S, and the hold-up ratio H, for **DMC** control only. Additional constraints include "ramp" limits, ie. the maximum change in P_S or S_S per interval, as well as a facility to maintain the ratio of S_S to P_S within defined bounds.

(7) The associated Dynamic Matrix Controller is tuned using the **DMC** weight settings on the GUI. These set the terms in the diagonal weighting matrices $\boldsymbol{W}$ (for H setpoint deviations) and $\boldsymbol{\Lambda}$ (for P_S and S_S move suppression). The other tuning parameter is the "number of optimised moves", also set on the GUI.

(8) Start a new data log file if necessary. Select the **DMC** controller, and switch the system to AUTO. Let the system settle to a steady-state. Now obtain the *hold-up H* response to a + 20% step in setpoint. Choose various combinations of high and low move suppression weights on P_S and S_S and monitor how the work done to achieve set-point steps is shared between these two MV's. What happens when the setpoint deviation weight on H is reduced, or the number of optimised *moves* is changed ?

(9) Compare the **DMC** step responses with **PID** control using P_S as MV, and **PID** control using S_S as MV.

(10) Process your log-file as follows: Plot the individual openloop and closedloop step responses, and comment on them. For each closedloop *hold-up* control response, calculate a quadratic performance index, starting at the time when the setpoint was stepped, and continuing for a fixed period thereafter.

$$QPI = \sum_{i=1}^{N}\left[H\left(i\Delta t\right)-H_{SP}\left(i\Delta t\right)\right]^2 \tag{7.41}$$

Comment on the differences caused by the various configurations and settings.

References

Chang T.S. and D.E. Seborg, "A linear programming approach for multivariable feedback control with inequality constraints", *Int. J. Control,* **37**, 583-597, (1983).

Cutler CR and BL Ramaker, "Dynamic Matrix Control - A computer control algorithm", *AIChE National Meeting,* Houston, Texas, (1979)

Garcia CE and AM Morshedi, "Quadratic Programming Solution of Dynamic Matrix Control (QDMC)", *Proc. Am. Control Conf.,* San Diego, California (1984)

Morshedi AM, CR Cutler and TA Skrovanek, "Optimal Solution of Dynamic Matrix Control with Linear Programming Techniques (LDMC)", *Proc. Am. Control Conf.,* Boston , Massachusetts, 199-208, (1985) Mulholland M and JA Prosser, "Linear Dynamic Matrix Control of a Distillation Column", *SA Inst. Chem. Eng 8th Nat. Meeting,* Cape Town, April 16-18 (1997).

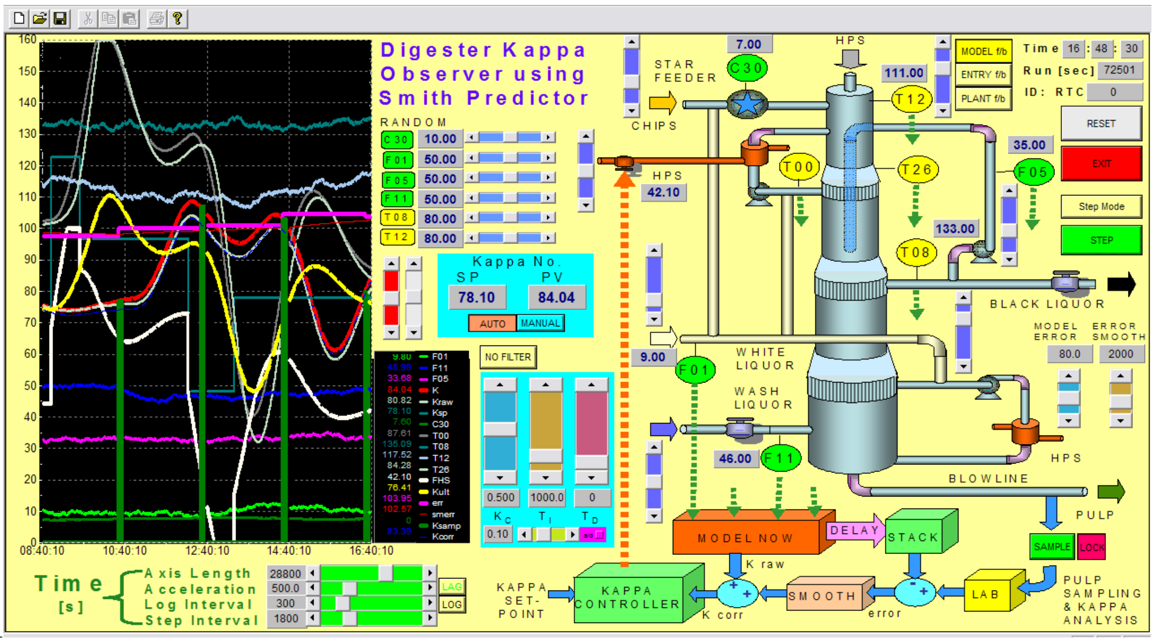
Richards, RJ, "An Introduction to Dynamics and Control", Longman, London, pp. 280-281, (1979).

Chapter 8 Simulations - Observers

.

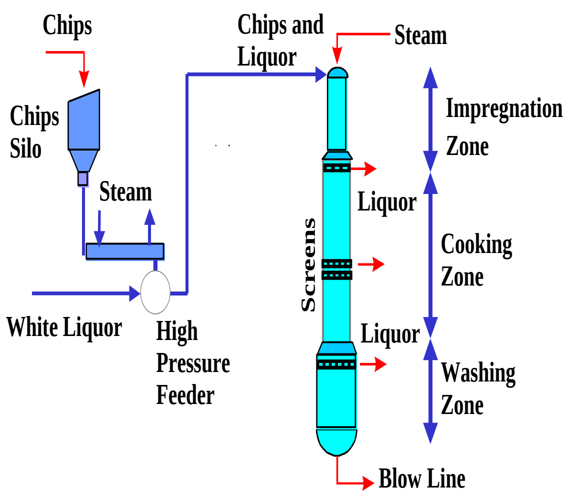
8.1 SPR - Smith Predictor for digester pulp Kappa estimation and control

Use of a continuous quality estimator based on intermittent lab samples, & compensating for dead time:



f_8.1

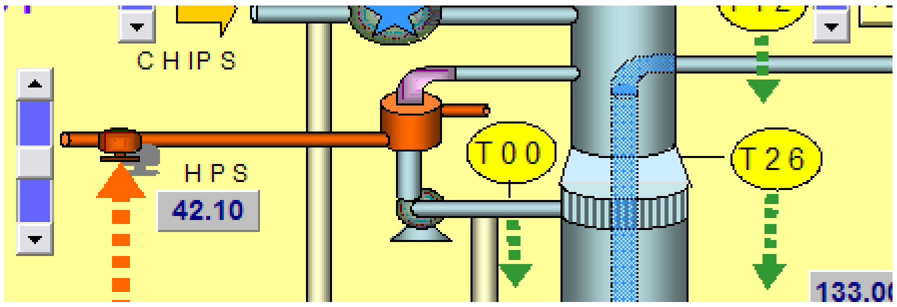
Smith Predictors as part of “Quality Estimators” are becoming very popular in industry. A basic robust model of the dependence of some stream property is constructed, often by regressing laboratory analyses against corresponding plant conditions (*P*,*T*,*F* measurements). This model is then run on-line, predicting the product property. Periodically, laboratory analyses are done for the property. The analysis is compared with the prediction that was made at the time the sample was drawn, and the “offset” error used to correct future on-line predictions. There is often a significant dynamic part and dead-time in the response of the actual process for the desired property. These can be included in an IMC (Internal Model Control) format as a virtual measurement for control feedback.



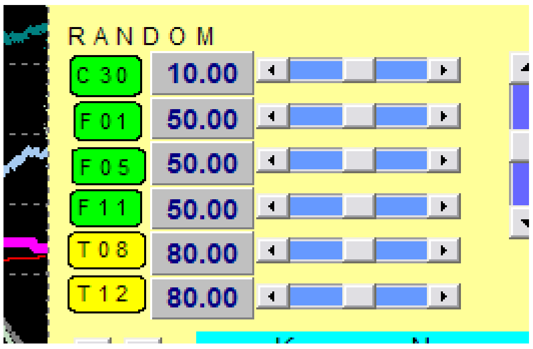
*f*_8.2

The application considered here is the prediction of wood-pulp Kappa number (extent of delignification) at the exit of a Kraft wood digester (*f*_8.2). Interesting features of this problem include the long dead-time delay (8-12 hours) as the wood chips move in plug flow down the reactor, a further delay of about 2 hours whilst the exit "blow-line" pulp undergoes laboratory analysis, and the dependence of the Kappa number on eight process variables distributed "in time" along the reactor length.

In the Kraft digester shown in *f*_8.1, wood chips enter at the top at rate *C30* together with "white liquor", the fresh chemical cooking solution containing caustic soda and sulphite. There are zones of counter-current and co-current solution flow as the chips progress downwards. At the top, most of the reaction takes place, whilst further down there are cooling and washing stages, with the circulating flows being withdrawn through peripheral screens, and returned to the desired level down concentric axial tubes from the top of the reactor. Analysis of Variance (ANOVAR) studies have shown that the flows *C30,F01,F05,F11* and temperatures *T00,T08,T12,T26* have the main influence on the final Kappa number. In addition, the effects of the temperatures *T00* and T26 were best correlated 2 hours and 4 hours respectively *prior* to the time at which pulp leaves the digester via the blow-line. A simple linear regression was used to obtain a model for the Kappa number on this basis, yielding eight coefficients and one bias. In the simulation, *T00* and *T26* (*f*_8.3) have been made completely dependent on the supply of *HPS* (high pressure steam) to the circulation heater at the top of the reactor (this will also be the MV for closed loop control).

*f*_8.3 

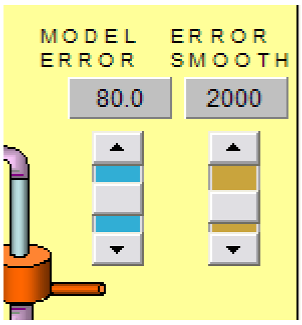
For the remaining variables affecting Kappa number (*C30,F01,F05,F11,T08,T12*), a facility is provided both to set a mean value, and a chosen degree of random variation about this mean using the following panel (*f*_8.4).


RANDOM

C30	10.00
F01	50.00
F05	50.00
F11	50.00
T08	80.00
T12	80.00

*f*_8.4

Samples of pulp can be drawn from the blow-line in either of two ways. If the "Lock" button is toggled off, then the "Sample" button can be pressed each time a sample is required. This event will be marked by a 5-minute sample pulse (green line) from the "true" (but unobservable) blow-line Kappa trace (red). If the "Lock" button is toggled on, then samples are drawn automatically every two hours. In both cases, it will be seen that the error trace ("*err*") steps *2 hours after the sample was drawn*, allowing for the delay in the laboratory, with the error being established by looking up the historical raw prediction of Kappa appropriate to the original sampling time. A smoothed version of this error is also generated ("*smerr*": red trace) to be used in correcting current raw predictions. (NOTE that all error traces are offset by +100 so that positive and negative variations are visible).


MODEL ERROR: 80.0

ERROR SMOOTH: 2000

*f*_8.5

The degree of smoothing applied to the error correction is determined by the "ERROR SMOOTH" scrollbar on the right (*f*_8.5). The adjoining "MODEL ERROR" scrollbar is used (for simulation purposes) to set the extent of a slowly-varying difference between the expected behaviour of the system (based on the regressed correlation used in the Smith Predictor), and the actual Kappa value emerging from the blow-line ("*K*" : red trace). The challenge for a Smith Predictor is to correctly identify this varying offset, and apply it in correcting the current raw prediction to provide the final corrected prediction of Kappa presently emerging from the reactor ("*Kcorr*": thin blue trace).

In practice, even this corrected Kappa value is not ideal to use in a feedback loop, because there is still a long dead-time (4 hours from *T26* and 2 hours from *T00*) and dynamic (1 hour) response between the manipulation of the *HPS*, and the emergence at the blow-line. Rather, one wants to control a virtual Kappa value representing where it will finally converge should the present settings persist. This is called the "ultimate" Kappa ("*Kult*" : yellow trace). Thus, when the PID controller is switched to “Auto”, it is this feedback loop that is closed: from *Kult* back to the *HPS*.

8.1.1 Typical settings

	mean	**random**			
C30:	7	10	*Kappa SP*= 80	PID: $Kc = 0.5$	Axis Length = 28800
F01:	9	50	*HPS* = 50	$T_I = 1000$	Acceleration = 500
F05:	35	50		$T_D = 0$	Log Interval = 300
F11:	46	50	MODEL ERROR = 80	$\alpha = 0.1$	Step Interval = 1800
T08:	133	80	ERROR SMOOTH =2000	d : □	
T12:	111	80			

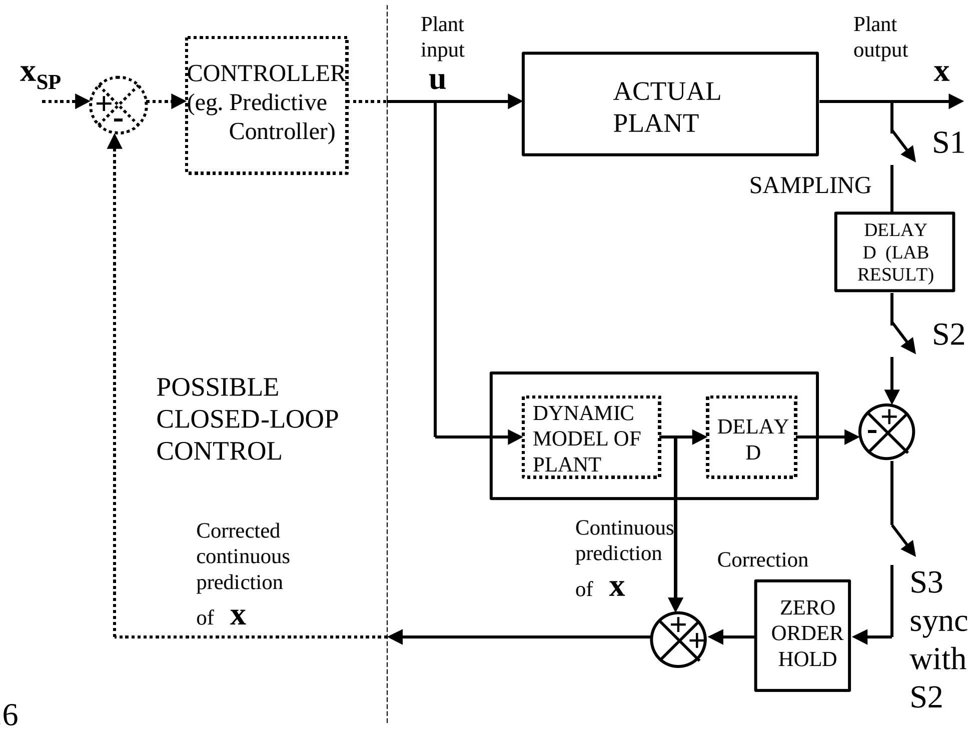
8.1.2 Theory

Smith Predictors are typically used in dead-time compensation, though sometimes it is more a case of measurement delay, for example in the case of an-online gas chromatograph. The algorithm certainly does need to have the delay time appropriate to each sample, so that the raw prediction at that time can be looked up in an historical buffer file. For a fixed instrument delay this might not be problematic, but it can become a problem where the measurement delay is caused by process plug flow. Here the delay will become dependent on the flow-rate through the process, and a continuous estimate of this delay would help to locate the appropriate data record.

Because of the possibility of timing errors in locating the appropriate historical record, it is advisable to “smooth” the error prediction, say in a single-exponential filter as is done here.

$$smerr(t) = \beta\, smerr(t-\Delta t) + (1-\beta)\, err(t) \qquad \text{with} \qquad 0<\beta<1 \tag{8.1}$$

Furthermore, such smoothing will avoid “bumping” the final corrected value in steps. Note in particular that Smith Prediction uses an *old* error to correct *new* predictions. The following diagram (*f*_8.6) shows a version of the Smith Predictor for intermittent sampling, without error smoothing. Note how the loop can optionally be closed through a controller, in which case an Internal Model Control (IMC) format has been achieved.



*f*_8.6

8.1.3 Tasks

(1) Switch the PID controller to "Manual", all random errors to zero (ie. on *C30,F01,F05,F11,T08,T12*), and the MODEL ERROR and ERROR SMOOTH parameters to zero. Wait for the system to achieve steady-state. Now step the *HPS* up by about 20. Note the responses – *Kult* predicts the full dynamic response without dead-time delays, whilst *Kraw* gives it with the appropriate dead-times: here we see that the *T00* contribution is delayed 2 hours and the *T26* contribution is delayed 4 hours. In this response *Kraw* is coincident with *K*, the true Kappa value, because we have not introduced a MODEL ERROR for simulation purposes.

(2) Now increase the MODEL ERROR to 160, and note how *true K* now randomly drifts around *Kraw*, which is steady on account of the fixed inputs. Note that the *err* trace steps 2 hours after each sample is taken, with the error for that sample. Now increase ERROR SMOOTH to 2000 and see how the *smerr* curve separates from *err* and smoothly varies.

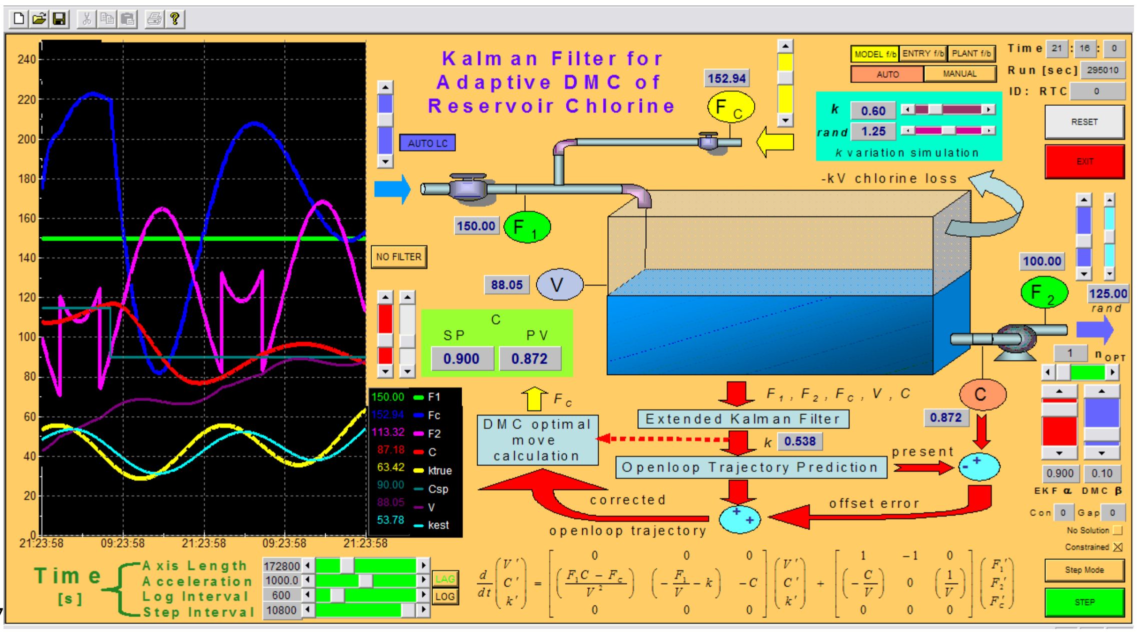
(3) Set the Kappa setpoint to about 60 and switch the PID controller to "Auto". Now the *HPS* (and *T00, T26*) are varied to cause *Kraw* to vary in such a way that *Kult* and thus *K* (true) track the Kappa setpoint as well as possible in the face of the varying model error, which is being detected by the sampling. Note that *K* (true) is not controlled as well as Kult because of the heavy smoothing applied to the detected error. Reduce ERROR SMOOTH to zero and check if *K* control improves. Switch off the sample "Lock" and take more frequent samples yourself to see if this improves it. The only remaining problem is the 2-hour delay for each sample!

(4) Put MODEL ERROR back to a reasonable value – say 80, and ERROR SMOOTH back to about 2000. Now consider some typical operation, where the Smith Predictor will

also provide compensation for variations in the other process variables. Set the RANDOM components for *C30*,*F01*,*F05*,*F11*,*T08*,*T12* all to about mid-range, and note the performance of the Predictor and Controller over a period.

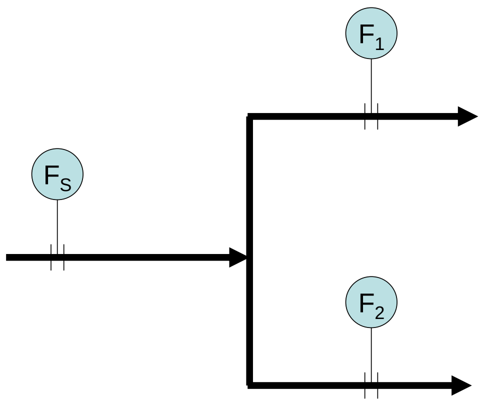
8.2 KAL - Kalman filter for adaptive DMC of reservoir chlorine

Setting up a Kalman filter to estimate an unmeasurable variable for continuous adaptation of DMC:



*f*_8.7

The Kalman filter is an “optimal observer” which one uses to find a set of process variable values which are a compromise between the available measurements, and the modelled relationship which should exist between these measurements. For example, one might be measuring three flows – a main supply and two flows that split from it (*f*_8.8).

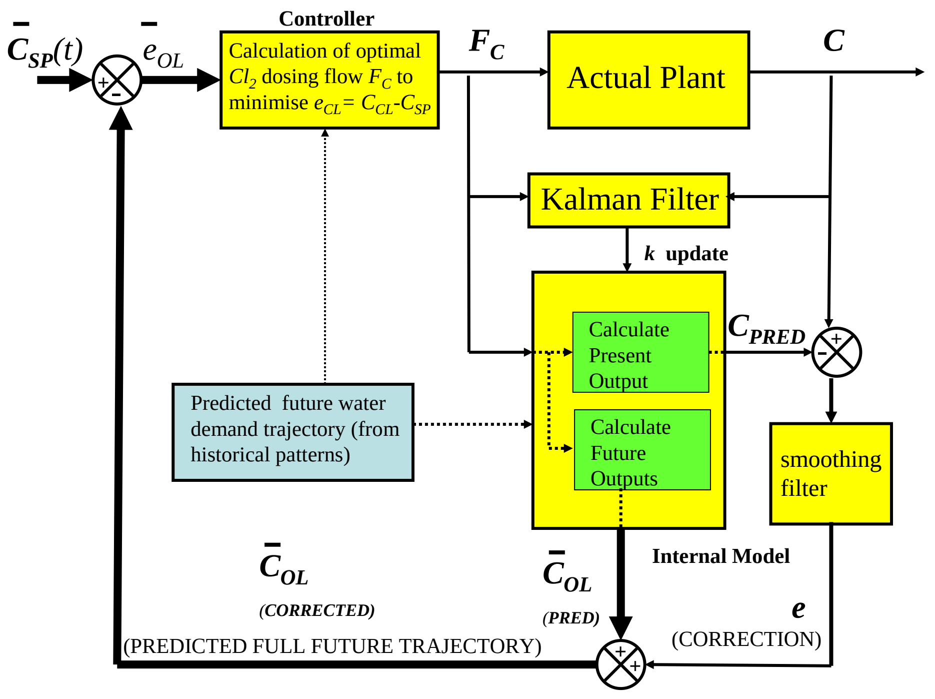


Model: $F_1 + F_2 = F_S$

*f*_8.8

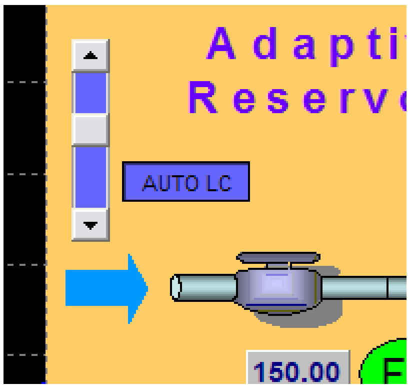
The model suggests that the two split flows should add up to the supply flow. But the measurements probably do not exactly confirm this. So where does the error lie? In the Kalman filter, one assigns expected errors to the model and each flow measurement. The filter runs as a continuous "reconciliation" based on these expected errors. Perhaps you are confident about the model (low error expected), and about the two split flow measurements (low errors expected), but not the supply measurement. Then the filter will provide continuous estimates of all three flows in which the estimated supply flow will be close to the sum of the measured split flows. Note that this "compromise" is not quite the same as process data reconciliation, where a constraint is imposed to achieve proper mass and/or energy balances.

In fact, the Kalman filter generally handles dynamic processes, where such measurements are dynamically inter-related – eg. a measured tank level rising in response to a measured inflow. Moreover, we do not need to restrict ourselves to process states and inputs. We could consider the size of a tank, a heat transfer coefficient, or, as in the present application, a chemical reaction rate constant, all to be variables which can or cannot be observed, but for which reconciled values are required which are a compromise between available observations and the expected model relationship. These variables may well be nonlinearly arranged in the model equations (eg. flow × composition in a mass balance). In this case we can locally-linearise the expression on each step, and still use the Kalman filter. Such a procedure is termed "Extended Kalman Filter" (EKF) and this is the form used in this application. In *f*_8.9 it is seen that the Kalman Filter acts to update the model within an Internal Model Control (IMC) structure.



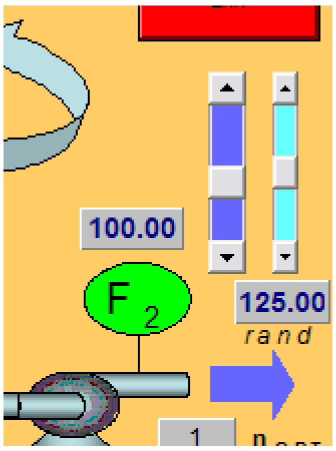
*f*_8.9

The problem considered in this application revolves around *adaptive control*, in particular, an adaptive algorithm for DMC, Dynamic Matrix Control. At a waterworks there is a large "conditioning" reservoir at the exit, to give the water sufficient contact time with a chlorine dose to sterilise it before it enters the consumer distribution network. The mean contact time depends on the RTD (Residence Time Distribution) in the conditioning reservoir - as a simplification, think of this as the volume / through-flow. But the volume varies, as a result of separate variations of inflow and outflow. A set of differential equations is required to describe the system properly, but note for the meantime that the response of the outflow chlorine composition C will be related to the input chlorine dosing F_C in a way which depends on current water flows (F_1 & F_2) , level (V), *and* the natural rate of loss of chlorine, determined by a first-order rate constant k (ie. $-kCV$). Even this rate constant has been observed to vary significantly with time (eg. 0.5 day^{-1} – 2.0 day^{-1}). So if we are to use a DMC algorithm to control the exit chlorine concentration, by manipulating the chorine dosing rate (sodium hypochlorite solution to the reservoir inflow), we would want to change the step-response curves on which the DMC is based in real-time, based on the identified k, and flows and level – ie. *adaptive* control. So the algorithm used has two components: a Kalman filter for continuous estimation of k, using measurements of dosing, flows, level and the exit chlorine composition, and a continuously-updated DMC calculation.



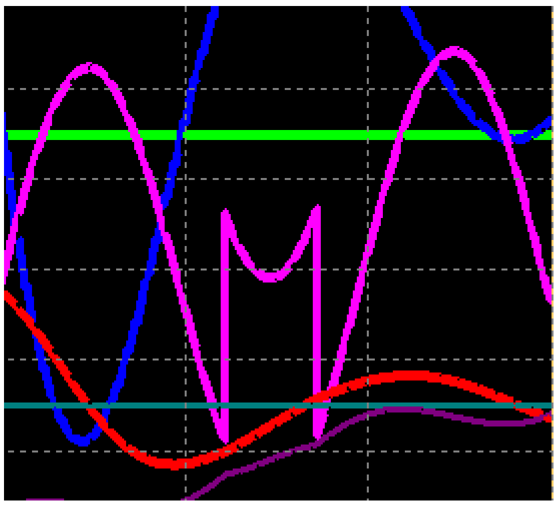
*f*_8.10

In this application the supply of water to the conditioning reservoir F_1 can be controlled manually, or alternatively it can be supervised by a reservoir level controller by toggling on "AUTO LC" (*f*_8.10). This level controller is of the "on-off" deadband type, giving a high inflow (150) when level reaches a defined low limit, and a low inflow (50) when it reaches a defined high limit. Note that "Level" is represented by the volume "*V*" in the reservoir.



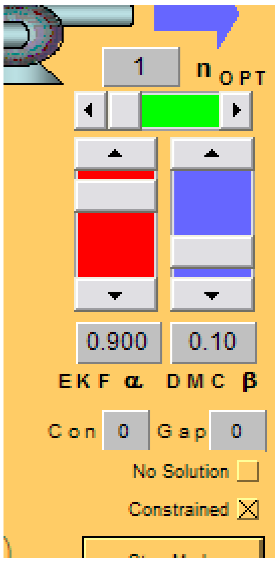
*f*_8.11

The outflow F_2 represents the flow drawn by consumers (*f*_8.11). If the random component "rand" is set down to zero, this flow can be set directly using the left scrollbar. If the "rand" component is used, then the left scrollbar represents the mean about which the random component makes positive or negative contributions, of size proportional to "rand". The actual random variation has a daily oscillation, with a smaller-amplitude weekly oscillation superimposed. Each night, when consumers are drawing least (and electricity is cheapest), there is a surge of flow superimposed when pumps are switched on to transfer water to a higher supply reservoir some distance away (pink trace in diagram *f*_8.12).



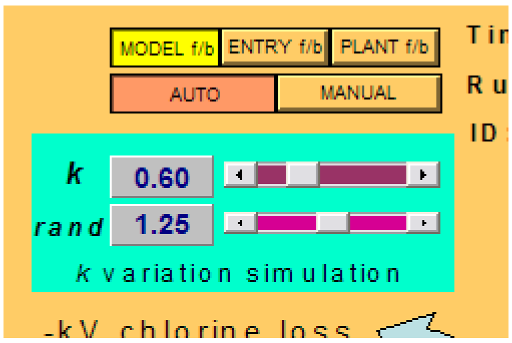
*f*_8.12

The EKF and DMC operation block *f*_8.13 has scrollbar settings for α (EKF) and β (DMC). The EKF gain parameter α increases the effort made by the extended Kalman filter in tracking the observations, at the expense of ignoring the model more. Thus the estimated decay constant *k* will vary more rapidly, and internal estimates of the exit chlorine concentration *C* (and reservoir volume *V*) will track their corresponding measurements more closely. Because the DMC has only one input and one output, only one gain parameter β is required, representing the ratio of ***W/Λ*** ie. the penalty weight on the *C* setpoint deviation over the move suppression weight on the hypochlorite dosage flow F_C. As β goes up, more vigorous control action can be expected.



*f*_8.13

As in the DMC applications given in sections 7.2-7.5, the "*Gap*" parameter (*f*_8.13) indicates how far the solution has been separated from the optimal MV settings, on account of constraints. In this problem, the only constraints operating are the maxima and minima of F_C and *C* (as evidenced by their scrollbar ranges). Should a constraint be met, the "Constrained □" block is highlighted in pink, and constraints can be ignored, if so desired, if the "□" is unchecked. If no feasible solution exists, the "No Solution" box is highlighted, and the MV value remains unchanged. The number of future control moves considered in the optimisation on each DMC step is set by "n_{OPT}", and it can be expected, as discussed in section 7.1, that control actions will become more vigorous, as n_{OPT} is increased to 2.



*f*_8.14

Note in this application that the "Auto"/"Manual" switch for the DMC is near the top right of the screen (*f*_8.14). Just below this is a special block for simulation purposes only. In this block one can alter the mean true chlorine decay rate constant *k*, as well as set the size of a slow oscillatory variation about this mean. This feature is to allow testing of the EKF's ability to identify and track *k* variations from the measurement data available.

8.2.1 Typical settings

$F_1 = 100$	$F_2 = 100$	EKF & DMC:	$\alpha = 0.9$	Axis Length = 172800
$V = 124$	rand (for F_2) = 125		$\beta = 0.2$	Acceleration = 1000
$C_{SP} = 1.0$	$k = 0.6$		$n_{OPT} = 1$	Log Interval = 600
	rand (for k) = 1.25			Step Interval = 10800

8.2.2 Theory

Kalman filter:

Three equations describe expected time-variations in the system:

$$\frac{dV}{dt} = F_1 - F_2 \tag{8.2}$$
$$\frac{d[VC]}{dt} = F_C - F_2C - kVC \tag{8.3}$$
$$\frac{dk}{dt} = 0 \tag{8.4}$$

We obtain *dC/dt* from the second equation as follows

$$\frac{d[VC]}{dt} = V\frac{dC}{dt} + C\frac{dV}{dt}\ , \quad \text{so} \tag{8.5}$$
$$V\frac{dC}{dt} = F_C - F_2C - kVC - C[F_1 - F_2] \tag{8.6}$$
$$= F_C - F_1C - kVC \tag{8.7}$$
$$\frac{dC}{dt} = \frac{F_C}{V} - \frac{F_1C}{V} - kC \tag{8.8}$$

Linearising the three differential equations using deviations $(V', C', k', F_1', F_2', F_C')$ about the present operating point (V, C, k, F_1, F_2, F_C), obtain

$$\frac{d}{dt}\begin{pmatrix} V' \\ C' \\ k' \end{pmatrix} = \begin{bmatrix} 0 & 0 & 0 \\ \left(\frac{F_1C - F_C}{V^2}\right) & \left(-\frac{F_1}{V} - k\right) & -C \\ 0 & 0 & 0 \end{bmatrix} \begin{pmatrix} V' \\ C' \\ k' \end{pmatrix} + \begin{bmatrix} 1 & -1 & 0 \\ \left(-\frac{C}{V}\right) & 0 & \left(\frac{1}{V}\right) \\ 0 & 0 & 0 \end{bmatrix} \begin{pmatrix} F_1' \\ F_2' \\ F_C' \end{pmatrix} \tag{8.9}$$

This is a set of first order differential equations of the form

$$\frac{d\boldsymbol{x}}{dt} = \boldsymbol{Ax} + \boldsymbol{Bm} \tag{8.10}$$

where $\boldsymbol{x}$ is the state vector and $\boldsymbol{m}$ is the input vector. This could be accurately integrated for the time-step Δt using the matrix exponential, but we perform a simple Euler integration instead:

$$\boldsymbol{x}_{t+\Delta t} = [\boldsymbol{A}\Delta t + \boldsymbol{I}]\boldsymbol{x}_t + [\boldsymbol{B}\Delta t]\boldsymbol{m}_t \qquad (8.11)$$

We represent this with the discrete system notation

$$\boldsymbol{x}_{i+1} = \boldsymbol{A}_i\,\boldsymbol{x}_i + \boldsymbol{B}_i\,\boldsymbol{m}_i \qquad (8.12)$$

$$\text{where } \boldsymbol{A}_i = [\boldsymbol{A}\Delta t + \boldsymbol{I}]_i \text{ and } \boldsymbol{B}_i = [\boldsymbol{B}\Delta t]_i \qquad (8.13)$$

Consider a situation where we can observe the inputs ***m*** and a selection ***G*** (perhaps *all*) of the states ***x***. The matrix ***G*** will typically be rectangular, selecting some of the states to an observation vector ***y***.

$$\boldsymbol{y}_i = \boldsymbol{G}\,\boldsymbol{x}_i \text{ with eg. } \boldsymbol{G} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \qquad (8.14)$$

The job of a Kalman filter is to match **y**i to the corresponding set of measurements $\hat{\mathbf{y}}_i$. It does this by interfering with the normal integration step of the system, using an updated optimal gain matrix ***K***i.

$$\boldsymbol{x}_{i+1} = \boldsymbol{A}_i\,\boldsymbol{x}_i + \boldsymbol{B}_i\,\boldsymbol{m}_i + \boldsymbol{K}_i[\hat{\boldsymbol{y}}_i - \boldsymbol{G}\,\boldsymbol{x}_i] \qquad (8.15)$$

One way of thinking about this is: If $\hat{\mathbf{y}}_i$ is “above” $\mathbf{y}_i = \mathbf{G}\mathbf{x}_i$, then ***K***i can be thought of as “positive”, and adding positive increments onto $\boldsymbol{x}_i$ on each time step until **x**i rises sufficiently for $\mathbf{y}_i = \mathbf{G}\mathbf{x}_i$ to reach $\hat{\mathbf{y}}_i$. Kalman showed how to obtain an optimal gain matrix ***K*** when the statistics of error vectors $\boldsymbol{\delta}$ and $\boldsymbol{\mu}$ in the system are known as follows:

$$\textit{Model predictions}: \boldsymbol{x}_{i+1} = \boldsymbol{A}_i\,\boldsymbol{x}_i + \boldsymbol{B}_i\,\boldsymbol{m}_i + \boldsymbol{\delta}_i \qquad (8.16)$$

$$\textit{Measurements}: \hat{\boldsymbol{y}}_i = \boldsymbol{G}\,\boldsymbol{x}_i + \boldsymbol{\mu}_i$$

where the $\boldsymbol{\delta}_i$ and $\boldsymbol{\mu}_i$ contain uncorrelated, zero-mean random terms with normal distributions as follows (where $E\{...\}$ represents the "expectation" or average):

$$\boldsymbol{Q} = E\{\boldsymbol{\delta}\,\boldsymbol{\delta}^T\} \quad \textit{(model-error covariance matrix)} \qquad (8.17)$$

$$\boldsymbol{R} = E\{\boldsymbol{\mu}\,\boldsymbol{\mu}^T\} \quad \textit{(measurement-error covariance matrix)} \qquad (8.18)$$

$$E\{\boldsymbol{\delta}\,\boldsymbol{\mu}^T\} = [\boldsymbol{0}] \qquad (8.19)$$

$$E\{\boldsymbol{\delta}\} = \boldsymbol{0} \qquad (8.20)$$

$$E\{\boldsymbol{\mu}\} = \boldsymbol{0} \qquad (8.21)$$

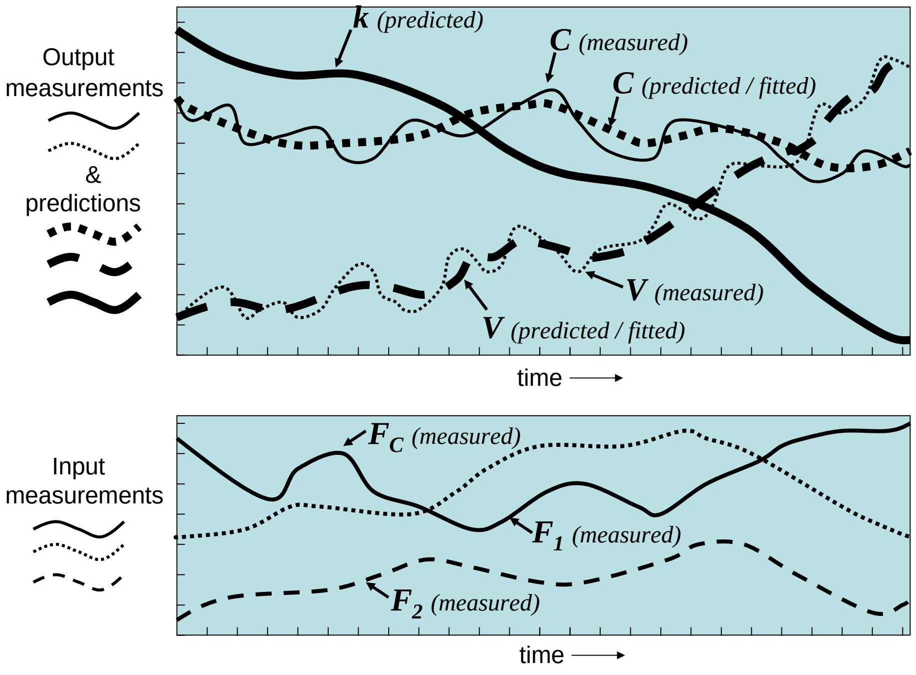
Under this circumstance, the optimal gain matrix ***K***i is yielded recursively on each time-step of the discrete filter as follows:

$$\boldsymbol{K}_i = \boldsymbol{M}_i\,\boldsymbol{G}^T[\boldsymbol{G}\,\boldsymbol{M}_i\,\boldsymbol{G}^T + \boldsymbol{R}]^{-1} \qquad (8.22)$$

$$\boldsymbol{x}_{i+1} = \boldsymbol{A}_i\,\boldsymbol{x}_i + \boldsymbol{B}_i\,\boldsymbol{m}_i + \boldsymbol{K}_i[\hat{\boldsymbol{y}}_i - \boldsymbol{G}_i\,\boldsymbol{x}_i] \qquad (8.23)$$

$$\boldsymbol{M}_{i+1} = \boldsymbol{A}_i\,[\boldsymbol{I} - \boldsymbol{K}_i\,\boldsymbol{G}_i]\boldsymbol{M}_i\,\boldsymbol{A}_i^T + \boldsymbol{Q} \qquad (8.24)$$

The ***Q*** matrix is set to represent the errors expected in the model predictions, whilst the ***R*** matrix is set to represent the errors expected in the measurements. Usually, no correlation is anticipated between the elements within each error vector, so the two error covariance matrices are diagonal, with the variance of each term (error-squared) on the diagonal. The "larger" the ***Q*** values, the more likely it is that the filter will follow the measurements, and conversely, the larger the ***R*** values, the more likely it is that it will follow the model. The "gain" is determined by the relative sizes between ***Q*** and ***R*** elements. Even if a measurement of a state is not available, as in this case with a non-square ***G*** matrix, a prediction will be produced for that state (*k*), as seen in the following diagram (*f*_8.15).



*f*_8.15

It is as well to emphasise at this point that the diagonal terms on the **Q** and **R** matrices relate to square errors. Thus, eg., if it is desired to halve the tracking deviations from a measurement, the corresponding **R** axis term must be divided by 4. The filter is started off with an assumed $\mathbf{M}_0$ diagonal with small values on the diagonal, to allow rapid variation of **M** at the start.

Updating the step-response for the DMC:

Recall that the system

$$\frac{d}{dt}\begin{pmatrix} V' \\ C' \\ k' \end{pmatrix} = \begin{bmatrix} 0 & 0 & 0 \\ \left(\frac{F_1 C - F_C}{V^2}\right) & \left(-\frac{F_1}{V} - k\right) & -C \\ 0 & 0 & 0 \end{bmatrix} \begin{pmatrix} V' \\ C' \\ k' \end{pmatrix} + \begin{bmatrix} 1 & -1 & 0 \\ \left(-\frac{C}{V}\right) & 0 & \left(\frac{1}{V}\right) \\ 0 & 0 & 0 \end{bmatrix} \begin{pmatrix} F_1' \\ F_2' \\ F_C' \end{pmatrix} \tag{8.25}$$

is integrated discretely as

$$\mathbf{x}_{i+1} = \mathbf{A}_i \mathbf{x}_i + \mathbf{B}_i \mathbf{m}_i \tag{8.26}$$

At any time, a local step response of C to F_C is obtained as follows: At t=0 set

$$\mathbf{x}_0 = \begin{pmatrix} V' \\ C' \\ k' \end{pmatrix}_0 = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} \text{ and } \mathbf{m}_0 = \begin{pmatrix} F_1' \\ F_2' \\ F_C' \end{pmatrix}_0 = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} \tag{8.27}$$

then integrate as in the discrete equation above, up to the horizon, having evaluated **A**, **B** with the present local values of F_1, F_C, C, V and k. It is the response of C' that we need as the unit step response for the DMC convolution model.

8.2.3 Tasks

(1) Set the "rand" contribution of the true chlorine decay rate constant k to zero. Wait for the Kalman filter estimate k_{est} to reach k_{true}. Now consider the effect of various EKF α values on the tracking of the (unknown) k_{true}, by stepping k_{true} with different α settings. Return k_{true} to about 0.6. (NOTE that the plotted "k" values are multiplied by 100).

(2) Set the reservoir "AUTO LC" off, remove the "rand" component from the outflow, and set inflow F_1 exactly equal to outflow F_2 (around 100) to keep the level steady at some reasonable V value near 100. Wait for the system to reach steady-state. Using setpoint step-response tests, assess the effects of different values of the DMC gain parameter β, and the number of optimised steps n_{OPT}.

(3) Set all parameters back to the typical values listed in section 8.2.1 (press "Reset"). Set the reservoir level control on "AUTO LC", to simulate normal operation of the conditioning reservoir, including a slowly-varying decay rate constant k (rand of k set at 1.25). Note the performance of the EKF and the DMC. Check if any improvements are possible through changes in α or β.

8.2.4 Code extracts

- The following code is executed on every time-step of the system:

```
m_V = __max(m_V,1e-7); // can't have an empty reservoir

// Re-evaluate Model & EKF matrices on every step
// Have to find the  nearest steady-state:
double F1ss = (m_F1+m_F2)/2;
double F2ss = F1ss;
//first do it for the estimate of k
double Vss_est = m_V_est;
double kss_est = m_k_est;
double Term1 = m_Fc/Vss_est;
double Term2 = m_C_est*(F1ss/Vss_est+kss_est);
double Term = (Term1+Term2)/2;
double Fcss_est = Vss_est*Term;
double Css_est = Term/(F1ss/Vss_est+kss_est);
cMatrix xr,mr;
xr.Init(3,1);
mr.Init(3,1);
```

```
Acont_est=0;
Acont_est(2,1) = (F1ss*Css_est - Fcss_est) / pow(Vss_est,2);
Acont_est(2,2) = -F1ss/Vss_est - kss_est;
Acont_est(2,3) = -Css_est;
Bcont_est=0;
Bcont_est(1,1)=1;
Bcont_est(1,2)=-1;
Bcont_est(2,1)=-Css_est/Vss_est;
Bcont_est(2,3)=1/Vss_est;
// discrete matrices for KALMAN FILTER step : step small enough for Euler
Adiscr_est = Acont_est*(dtKAL/(double)Ninterp)/(24*3600);
Adiscr_est(1,1) = Adiscr_est(1,1)+1;
Adiscr_est(2,2) = Adiscr_est(2,2)+1;
Adiscr_est(3,3) = Adiscr_est(3,3)+1;
Bdiscr_est = Bcont_est*(dtKAL/(double)Ninterp)/(24*3600);

//NOW do it for the TRUE value of k etc
double Vss = m_V;
double kss = m_k;
Term1 = m_Fc/Vss;
Term2 = m_C*(F1ss/Vss+kss);
Term = (Term1+Term2)/2;
double Fcss = Vss*Term;
double Css = Term/(F1ss/Vss+kss);
xr.Init(3,1);
mr.Init(3,1);
Acont_true=0;
Acont_true(2,1) = (F1ss*Css - Fcss) / pow(Vss,2);
Acont_true(2,2) = -F1ss/Vss - kss;
Acont_true(2,3) = -Css;
Bcont_true=0;
Bcont_true(1,1)=1;
Bcont_true(1,2)=-1;
Bcont_true(2,1)=-Css/Vss;
Bcont_true(2,3)=1/Vss;
// discrete matrices for model step : step small enough for Euler
Adiscr_true = Acont_true*(dtKAL/(double)Ninterp)/(24*3600);
Adiscr_true(1,1) = Adiscr_true(1,1)+1;
Adiscr_true(2,2) = Adiscr_true(2,2)+1;
Adiscr_true(3,3) = Adiscr_true(3,3)+1;
Bdiscr_true = Bcont_true*(dtKAL/(double)Ninterp)/(24*3600);

if (theApp.FIRSTLOOP)
{
        mlast(1) = m_Fc;
        mlast(2) = m_F1;
        mlast(3) = m_F2;
        theApp.FIRSTLOOP=0;
}

while (((theApp.t-tlast_KAL)>=dtKAL) | (theApp.b_FORCE_STEP))
{
        if (!theApp.b_FORCE_STEP)
        {
                tlast_KAL += dtKAL;     // must catch up by looping more than once if necessary
        }
        else
        {
                tlast_KAL = theApp.t;               // to SYNCHRONISE manual moves : b_FORCE_STEP
                                                // was set TRUE on StepMode button
                theApp.b_FORCE_STEP = FALSE;
        }

        // RECALCULATE DMC MATRICES EACH TIME!  (BASED ON ESTIMATED k: ADAPTIVE!)---------------------
-

        // openloop step response of C to a unit step in Fc,F1,F2
        xr=0;
        mr=0;
        mr(3)=1;  //Fc input
        for (i=1;i<=Nkal;i++)
```

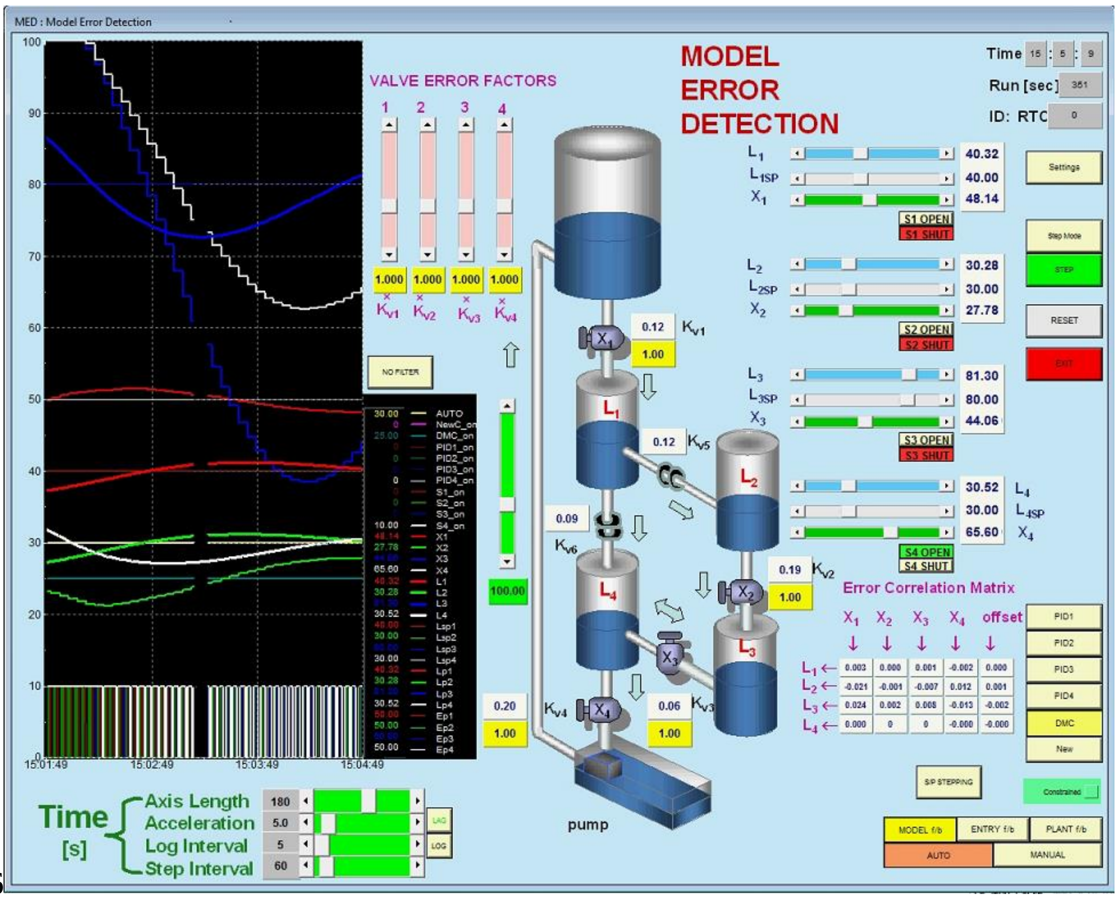
```
{
        for (j=1;j<=Ninterp;j++)
        {
                xr = Adiscr_est*xr + Bdiscr_est*mr;
        }
        resp(i,1)=xr(2);  //C output
}
resp(Nkal,1)=resp(Nkal-1,1);  // else integration is implied !
xr=0;
mr=0;
mr(1)=1;  //F1 input
for (i=1;i<=Nkal;i++)
{
        for (j=1;j<=Ninterp;j++)
        {
                xr = Adiscr_est*xr + Bdiscr_est*mr;
        }
        resp(i,2)=xr(2);  //C output
}
resp(Nkal,2)=resp(Nkal-1,2);  // else integration is implied !
xr=0;
mr=0;
mr(2)=1;  //F2 input
for (i=1;i<=Nkal;i++)
{
        for (j=1;j<=Ninterp;j++)
        {
                xr = Adiscr_est*xr + Bdiscr_est*mr;
        }
        resp(i,3)=xr(2);  //C output
}
resp(Nkal,3)=resp(Nkal-1,3);  // else integration is implied !

unsigned ii,jj,iB,jB,rcol,rrow;
// Load B
for (i=1;i<=Nkal;i++) for (ii=1;ii<=Rkal;ii++) for (j=1;j<=Mkal;j++) for (jj=1;jj<=Pkal;jj++)
{
        iB=(i+j-2)*Rkal+ii;
        if (iB<=Nkal*Rkal)
        {
                jB=(j-1)*Pkal+jj;
                rcol=(jj-1)*Rkal+ii;
                B(iB,jB)=resp(i,rcol);
        }
}
// Load Bol
for (i=1;i<=Nkal;i++) for (ii=1;ii<=Rkal;ii++) for (j=1;j<=Nkal;j++) for (jj=1;jj<=(Pkal+Qkal);jj++)
{
        iB=(i-1)*Rkal+ii;
        jB=(j-1)*(Pkal+Qkal)+jj;
        rrow = __min(Nkal-j+i+1, Nkal);
        rcol=(jj-1)*Rkal+ii;
        Bol(iB,jB)=resp(rrow,rcol);
}
// Load B0
for (i=1;i<=Nkal;i++) for (ii=1;ii<=Rkal;ii++) for (j=1;j<=Nkal;j++) for (jj=1;jj<=(Pkal+Qkal);jj++)
{
        iB=(i-1)*Rkal+ii;
        jB=(j-1)*(Pkal+Qkal)+jj;
        rrow = Nkal-j+1;
        rcol=(jj-1)*Rkal+ii;
        B0(iB,jB)=resp(rrow,rcol);
}
// Load W_kal  (weights on deviation-squared from setpoint)
for (i=1;i<=Nkal;i++) for (ii=1;ii<=Rkal;ii++)
{
        j = (i-1)*Rkal+ii;
        if (ii==1) Wkal(j,j)= beta*pow((1/max_C),2);    // C  deviation from SP
                                                // so can weight each controlled variable differently
}
```

```
// Load L_kal  (weights on control-move-squared)
for (i=1;i<=Mkal;i++) for (ii=1;ii<=Pkal;ii++)
{
        j = (i-1)*Pkal+ii;
        if (ii==1) Lkal(j,j)= 1*pow((1/max_Fc),2);      // Fc MOVE
                                                   // so can weight each controlled variable differently
}

// Set up the KAL gain matrix...............
// Kkal = inv[B'WB +L] B'W
BT= ~B;
BTWBPL = BT*Wkal*B + Lkal;
BTWBPLi=BTWBPL.Inv();  // inversion
BTW = BT*Wkal;
Kkal = BTWBPLi*BTW;
//-------------------------------------------------------------------------------

dml(1) = m_Fc-mlast(1);  //MM050103 (Fc-Fcss)-mlast(1);
dml(2) = m_F1-mlast(2);  //MM050103 (F1-F1ss)-mlast(2);
dml(3) = m_F2-mlast(3);  //MM050103 (F2-F2ss)-mlast(3);

mlast(1) = m_Fc;
mlast(2) = m_F1;
mlast(3) = m_F2;

// past moves shift one down dmp stack
for (i=1; i<=(Nkal-1); i++)
{
        for (j=1; j<=(Pkal+Qkal); j++)
        {
                dmp((i-1)*(Pkal+Qkal)+j) = dmp(i*(Pkal+Qkal)+j);
        }
}
for (j=1; j<=(Pkal+Qkal); j++)
{
        dmp((Nkal-1)*(Pkal+Qkal)+j) = dml(j);  // previous move
}

if (b_AUTO)
{
        // on AUTO ! ---- do KAL Control !
        for (i=1;i<=Nkal;i++)
        {
                xsp((i-1)*Rkal+1) = m_C_sp;
                x0((i-1)*Rkal+1) = m_C;
        }
        // set constraints
        m0.Init((Mkal*Pkal),1);
        ml=m0;
        mh=m0;
        for (i=1;i<=Mkal;i++) for (j=1;j<=Pkal;j++)
        {
                m0((i-1)*Pkal+j) = mlast(j);  // repeat it through the vector
                ml((i-1)*Pkal+j) = 0;
                mh((i-1)*Pkal+j) = max_Fc;
        }
        dmmax = m0; dmmax = 20;

        xl = x0;
        xh = x0;
        for (i=1;i<=Nkal;i++)
        {
                xl((i-1)*Rkal+1) = 0;
                xh((i-1)*Rkal+1) = max_C;
        }

        w.Init(Mkal*Pkal); //weights for deviations from unconstrained optimal move
        for (i=1;i<=Mkal;i++)
        {
                // fractions of eng value
```

```
                        w((i-1)*Pkal+1) = 1.0/Fcdatum;
                }

                // Solve the LP problem....
                if (KALstep(Nkal,Mkal,Pkal,Qkal,Rkal,
                        &x0,&xsp,&dmp,
                        &B,&B0,&Bol,&Wkal,&Lkal,&Kkal,&w,
                        &m0,&mh,&ml,&dmmax,&xh,&xl,&trajerr,&dmuqo,&dm))
                {
                        m_Fc = mlast(1) + dm(1);
                        m_kal_gap = fabs(dm(1)-dmuqo(1));
                        if (m_kal_gap > 0)
                        {
                                b_constrained = TRUE;  // shortcut - see slack variable method with ry below
                        }
                        else
                        {
                                b_constrained = FALSE;
                        }
                }
                m_Fc = m_Fc;
        }
}


// now move the model (BASED ON TRUE value of k) forward on a finer time-step (just use Euler integration)
while ((theApp.t-theApp.tlast_ModelStep) >= dtKAL/(double)Ninterp)
{
        theApp.tlast_ModelStep += dtKAL/(double)Ninterp;

        mr(1) = (m_F1-F1ss);
        mr(2) = (m_F2-F2ss);
        mr(3) = (m_Fc-Fcss);

        xr(1) = (m_V-Vss);
        xr(2) = (m_C-Css);
        xr(3) = (m_k-kss);

        xr = Adiscr_true*xr + Bdiscr_true*mr;

        if (theApp.b_Model)
        {
                m_V = __min(__max(xr(1)+Vss,0),max_V);
                m_C = __min(__max(xr(2)+Css,0),max_C);
        }

        //--------------------------------------------------------------------------------------
        // Now Kalman Filter step based on estimated k value, to find k!
        mr(1) = (m_F1-F1ss);
        mr(2) = (m_F2-F2ss);
        mr(3) = (m_Fc-Fcss_est);

        xr(1) = (m_V_est-Vss_est);   // these must be based on their estimates! ####MM050104
        xr(2) = (m_C_est-Css_est);
        xr(3) = (m_k_est-kss_est);

        yf(1) = m_V - Vss_est;   // observations
        yf(2) = m_C - Css_est;

        cMatrix GfT,temp;
        GfT = ~Gf;
        temp = (Gf* ( Mf * GfT) ) + Rf;
        Kf = Mf * ( GfT * temp.Inv() );
        xr = Adiscr_est*xr + Bdiscr_est*mr + Kf*(yf - Gf*xr);
        temp = ~Adiscr_est;
        Mf = Adiscr_est*(If - Kf*Gf)*Mf*temp +Qf;   //recursive 'M'

        //recover absolute values
        m_V_est = __min(__max((xr(1) + Vss_est), 0),max_V);
        m_C_est = __min(__max((xr(2) + Css_est), 0),max_C);
        m_k_est = __min(__max((xr(3) + kss_est), 0),max_k);
        //----------------------------------------------------------------------------------------

        // Level control if requested ... has to be done in phase with model step
        if(b_AutoLC)
        {
                if ((m_V<=m_Vlow) & (m_Vlast>m_Vlow))
                {
                        m_F1 = m_F1high;
                }
                if ((m_V>=m_Vhigh) & (m_Vlast<m_Vhigh))
                {
                        m_F1 = m_F1low;
                }
                m_F1 = __min(__max(m_F1,0),max_F1);
        }
        m_Vlast = m_V;
}
```

8.3 MED – Model error detection

Identification of MIMO model error dependence using a least squares fit to a batch of real-time data in a moving window:



Time 15 : 5 : 9
Run [sec] 351
ID: RTC 0

	1	2	3	4
Valve error factor	1.000	1.000	1.000	1.000
	K_{V1}	K_{V2}	K_{V3}	K_{V4}

L_1	40.32
L_{1SP}	40.00
X_1	48.14
L_2	30.28
L_{2SP}	30.00
X_2	27.78
L_3	81.30
L_{3SP}	80.00
X_3	44.06
L_4	30.52
L_{4SP}	30.00
X_4	65.60

S1 OPEN S1 SHUT
S2 OPEN S2 SHUT
S3 OPEN S3 SHUT
S4 OPEN S4 SHUT

K_{V1} 0.12 / 1.00; K_{V2} 0.19 / 1.00; K_{V3} 0.06 / 1.00; K_{V4} 0.20 / 1.00; K_{V5} 0.12; K_{V6} 0.09

Error Correlation Matrix

	X_1 ↓	X_2 ↓	X_3 ↓	X_4 ↓	offset ↓
$L_1 \leftarrow$	0.003	0.000	0.001	-0.002	0.000
$L_2 \leftarrow$	-0.021	-0.001	-0.007	0.012	0.001
$L_3 \leftarrow$	0.024	0.002	0.008	-0.013	-0.002
$L_4 \leftarrow$	0.000	0	0	-0.000	-0.000

Time [s]

Axis Length	180
Acceleration	5.0
Log Interval	5
Step Interval	60

*f*_8.16

8.3.1 Typical settings

Level setpoints:	Valves:
L_{SP1} = 40	X_1 = 50
L_{SP2} = 30	X_2 = 50
L_{SP3} = 80	X_3 = 50
L_{SP4} = 30	X_4 = 50

8.3.2 Theory

*f*_8.17

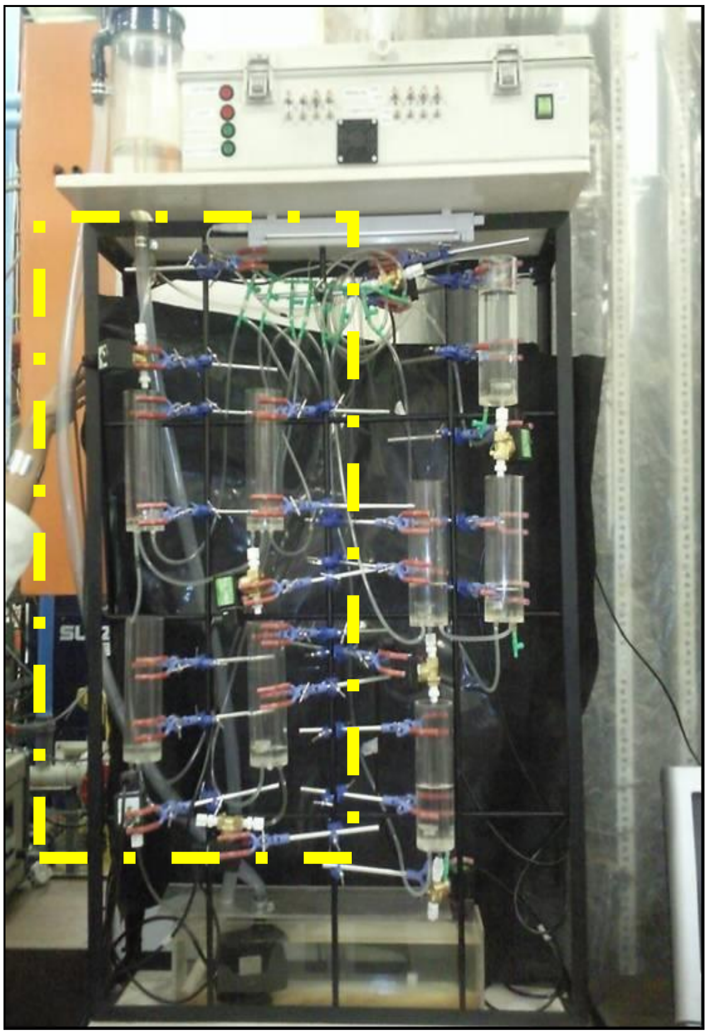
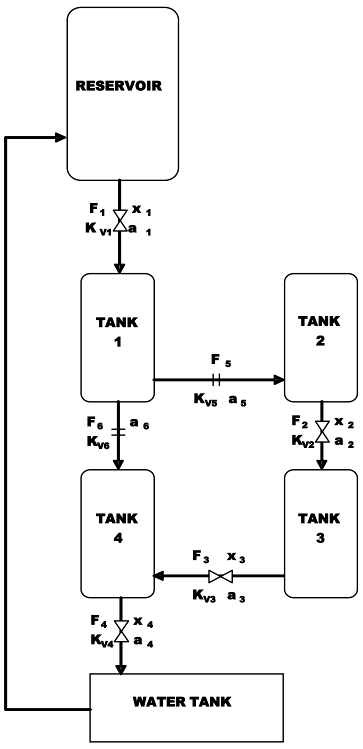


Figure *f*_8.17 shows the experimental rig with the considered four vessels on the left (Kannie and Managalparsad, 2010).



$$\frac{dL_1}{dt} = \frac{1}{A_1}\left(F_1 - F_5 - F_6\right)$$

$$\frac{dL_2}{dt} = \frac{1}{A_2}\left(F_5 - F_2\right)$$

$$\frac{dL_3}{dt} = \frac{1}{A_3}\left(F_2 - F_3\right)$$

$$\frac{dL_4}{dt} = \frac{1}{A_4}\left(F_3 + F_6 - F_4\right)$$

*f*_8.18

(8.28-8.31)

$$F_1 = x_1 K_{V1}\sqrt{15 + \mathrm{a}_1}$$

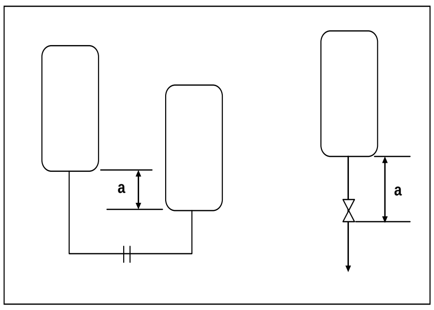
$$F_2 = x_2 K_{V2}\sqrt{L_2 + \mathrm{a}_2}$$

$$F_3 = x_3 K_{V3}\sqrt{L_3 - L_4 + \mathrm{a}_3}$$

$$F_4 = x_4 K_{V4}\sqrt{L_4 + \mathrm{a}_4}$$

$$F_5 = 1 \times K_{V5}\sqrt{L_1 - L_2 + \mathrm{a}_5}$$

$$F_6 = 1 \times K_{V6}\sqrt{L_4 + \mathrm{a}_6}$$



*f*_8.19

(8.32-8.37)

Kannie and Managalparsad (2010) modelled the system as above (*f*_8.18- *f*_8.19), obtaining the parameters K_{V1}-K_{V6} and a_1-a_6 by a series of experiments on the apparatus.

```
av[1]=200.00;    kv[1]=0.123;
av[2]=3.629;     kv[2]=0.187;
```

```
av[3]=4.25;     kv[3]=0.0632;
av[4]=2.268;    kv[4]=0.202;
av[5]=-4.6;     kv[5]=0.122;
av[6]=0.000;    kv[6]=0.090;
```

The valves on the apparatus were open or shut solenoid valves. Thus flow was proportioned on a 0-100% scale by time-slicing each valve on a fairly quick cycle so as not to add any extra measureable transients. The fractional opening of valves is obvious at the bottom of the screen (*f*_8.16). However, the model uses the fractional valve setting directly, so that MODEL operation of the software will not exhibit any dependence on the fractional switching cycles.

Observation of Errors:

As the system runs, there is a measurement of each tank level coming from the plant. In parallel, the assumed plant model is run, using the same valve settings as are being used on the plant. Since this will inevitably have large offsets, the assumed plant model is run in a very particular way. An horizon is established by going backwards in time for a fixed period (10s), and the measured levels at that time are used as the assumed model starting point. The assumed model is then run forward in time up to the present, and the error in each level is obtained as the difference between the present level and the assumed model prediction of it. This procedure is repeated on every-time step, so there is an updated set of level errors obtained for every time-step.

Error Correlation Matrix:

For the purpose of identifying the realtionship between each level error and each valve (as a possible source of the error), a *window* of data for the 4 level errors, and simultaneous 4 valve settings is maintained in a stack, going back *N*=200 points in time. The way that this stack is processed on each time-step, to update the error/valve correlation matrix, will now be described.

The method will be developed initially by considering *only one* of the 4 tank level errors, which will nominally be called *e*. The value of this error at position *i* in the stack 1≤*i*≤*N* will be represented e_i. It will be assumed that a linear relationship with fixed offset exists between this error and the valve settings at time *i*:

$$e_i \approx \begin{pmatrix} x_{1i} \\ x_{2i} \\ x_{3i} \\ x_{4i} \\ 1 \end{pmatrix}^T \begin{pmatrix} \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ \alpha_4 \\ \alpha_5 \end{pmatrix} = \boldsymbol{X}_i^T \boldsymbol{\alpha} \tag{8.38}$$

So the objective is to find the set of coefficients $\boldsymbol{\alpha}$ which cause the best least squares fit objective value $J(\boldsymbol{\alpha})$ over the whole stack (equally-weighted) as follows:

$$J(\boldsymbol{\alpha}) = \sum_{i=1}^{N} \left(e_i - \boldsymbol{X}_i^T \boldsymbol{\alpha}\right)^T \left(e_i - \boldsymbol{X}_i^T \boldsymbol{\alpha}\right) \tag{8.39}$$

$$= \sum_{i=1}^{N} \left(e_i^T - \boldsymbol{\alpha}^T \boldsymbol{X}_i \right)\left(e_i - \boldsymbol{X}_i^T \boldsymbol{\alpha} \right) \tag{8.40}$$

$$= \sum_{i=1}^{N} e_i^T e_i - e_i^T \boldsymbol{X}_i^T \boldsymbol{\alpha} - \boldsymbol{\alpha}^T \boldsymbol{X}_i e_i + \boldsymbol{\alpha}^T \boldsymbol{X}_i \boldsymbol{X}_i^T \boldsymbol{\alpha} \tag{8.41}$$

The middle two terms are transposes and scalar, so one can write

$$J(\boldsymbol{\alpha}) = \sum_{i=1}^{N} e_i^T e_i - 2\boldsymbol{\alpha}^T \boldsymbol{X}_i e_i + \boldsymbol{\alpha}^T \boldsymbol{X}_i \boldsymbol{X}_i^T \boldsymbol{\alpha} \tag{8.42}$$

Differentiating with respect to the vector $\boldsymbol{\alpha}$,

$$\frac{\partial J(\boldsymbol{\alpha})}{\partial \boldsymbol{\alpha}} = \sum_{i=1}^{N} -2\boldsymbol{X}_i e_i + 2\boldsymbol{X}_i \boldsymbol{X}_i^T \boldsymbol{\alpha} = \boldsymbol{0} \tag{8.43}$$

to minimise. Thus

$$\left[\sum_{i=1}^{N} \boldsymbol{X}_i \boldsymbol{X}_i^T \right] \boldsymbol{\alpha} = \left(\sum_{i=1}^{N} \boldsymbol{X}_i e_i \right) \tag{8.44}$$

$$\boldsymbol{\alpha} = \left[\sum_{i=1}^{N} \boldsymbol{X}_i \boldsymbol{X}_i^T \right]^{-1} \left(\sum_{i=1}^{N} \boldsymbol{X}_i e_i \right) \tag{8.45}$$

Bearing in mind that only an arbitrary level error *e* has been considered so far, one can consider all *j* levels simultaniously, $1 \le j \le 4$, using

$$\boldsymbol{A} = \begin{bmatrix} \boldsymbol{\alpha}_1 & \boldsymbol{\alpha}_2 & \boldsymbol{\alpha}_3 & \boldsymbol{\alpha}_4 \end{bmatrix} = \begin{bmatrix} \alpha_{11} & \alpha_{12} & \alpha_{13} & \alpha_{14} \\ \alpha_{21} & \alpha_{22} & \alpha_{23} & \alpha_{24} \\ \alpha_{31} & \alpha_{32} & \alpha_{33} & \alpha_{34} \\ \alpha_{41} & \alpha_{42} & \alpha_{43} & \alpha_{44} \\ \alpha_{51} & \alpha_{52} & \alpha_{53} & \alpha_{54} \end{bmatrix} \tag{8.46}$$

and

$$\boldsymbol{e}_i^T = \begin{pmatrix} e_{i1} & e_{i2} & e_{i3} & e_{i4} \end{pmatrix} \tag{8.47}$$

so

$$\boldsymbol{A} = \left[\sum_{i=1}^{N} \boldsymbol{X}_i \boldsymbol{X}_i^T \right]^{-1} \left[\sum_{i=1}^{N} \boldsymbol{X}_i \boldsymbol{e}_i^T \right] \tag{8.48}$$

In this application, protection against a singularity in the inversion of the $\boldsymbol{X}$ covariance matrix is obtained using

$$\Lambda = \begin{bmatrix} 10^{-10} & 0 & 0 & 0 & 0 \\ 0 & 10^{-10} & 0 & 0 & 0 \\ 0 & 0 & 10^{-10} & 0 & 0 \\ 0 & 0 & 0 & 10^{-10} & 0 \\ 0 & 0 & 0 & 0 & 10^{-10} \end{bmatrix} \tag{8.49}$$

so that

$$\boldsymbol{A} = \left[\sum_{i=1}^{N} \boldsymbol{X}_i \, \boldsymbol{X}_i^T + \boldsymbol{\Lambda} \right]^{-1} \left[\sum_{i=1}^{N} \boldsymbol{X}_i \, \boldsymbol{e}_i^T \right] \tag{8.50}$$

The correlation matrix presented on the screen is in the form $\boldsymbol{A}^{\mathrm{T}}$:

$$\begin{array}{cccccccc} & & x_1 & x_2 & x_3 & x_4 & 1 \\ & & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow \\ e_1 & \leftarrow & \alpha_{11} & \alpha_{21} & \alpha_{31} & \alpha_{41} & \alpha_{51} \\ e_2 & \leftarrow & \alpha_{12} & \alpha_{22} & \alpha_{32} & \alpha_{42} & \alpha_{52} \\ e_3 & \leftarrow & \alpha_{13} & \alpha_{23} & \alpha_{33} & \alpha_{43} & \alpha_{53} \\ e_4 & \leftarrow & \alpha_{14} & \alpha_{24} & \alpha_{34} & \alpha_{44} & \alpha_{54} \end{array} \tag{8.51}$$

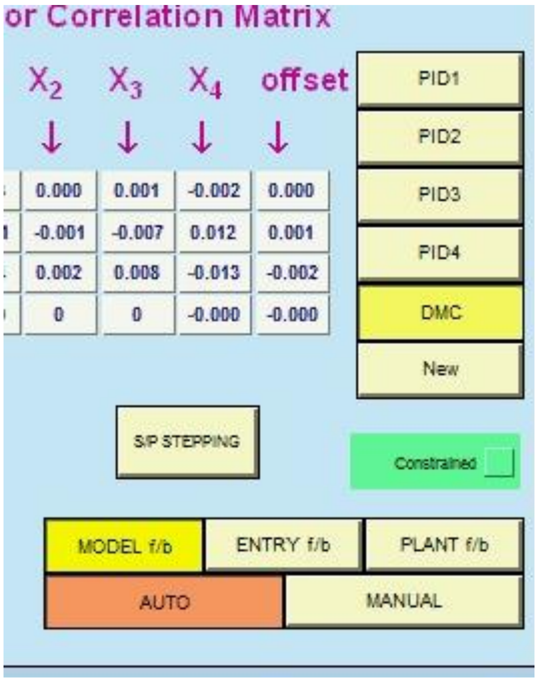
An efficiency is included in this moving-window application by adjusting only terms affected by the new point on each time-step, and subtracting out the contribution of the oldest point which is being lost from the window.

8.3.3 Tasks

In this exercise, the intention is to detect when the apparatus starts to behave differently to the model. Not only that, but the correlation matrix which is continuously updated in time will hopefully give us enough information to locate just *where* in this MIMO system the fault is likely to be. The exercise will be conducted by initially running with the matched model, then adjusting the "matched" model so that it differs from the "plant" being measured. Of course, in off-line MODEL mode, the "plant" will just be the original unchanged model.The updated correlation matrix seeks whether errors observed in the four level measurements (compared to expected levels) can be attributed to any of the 4 valves. For the purpose of the exercise, the behaviour error will be introduced by altering the K_V of a valve.

8.3.4 Method

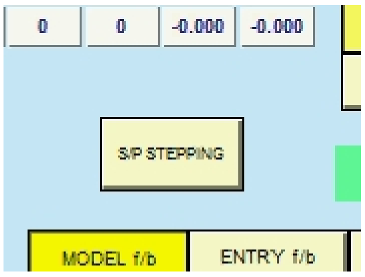
(1) Start the MED application. It should be in MODEL mode (*f*_8.20). The level setpoints are to be 40,30,80,30 for L_{SP1} through to L_{SP4}. Select the DMC control, and then switch the system to AUTO. Wait for the system to come to steady-state at these setpoints.


or Correlation Matrix

X_2	X_3	X_4	offset
0.000	0.001	-0.002	0.000
-0.001	-0.007	0.012	0.001
0.002	0.008	-0.013	-0.002
0	0	-0.000	-0.000

*f*_8.20

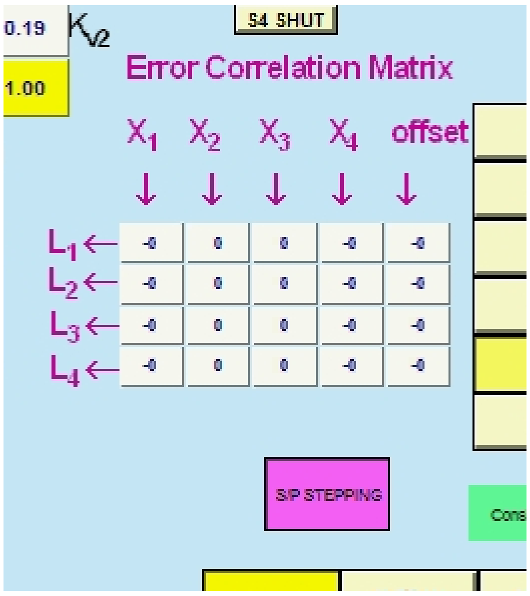
(2) For the purpose of identification, few of the relationships in the system can be observed at steady-state, so frequent time-changes will be forced on the system by moving the level setpoints in a random fashion. This is arranged so that the setpoints move up 10%, then back down 10% at random points in time, to avoid saturation of valves or levels. To start this process, press the S/P STEPPING button (*f*_8.21).



0	0	-0.000	-0.000

*f*_8.21

(3) Whilst the system continues to track the moving setpoints, keep an eye on the Error Correlation Matrix. All correlation terms between valves and errors in expected levels should be remaining zero, because the expected Levels are based on an identical model to the plant (*f*_8.22).

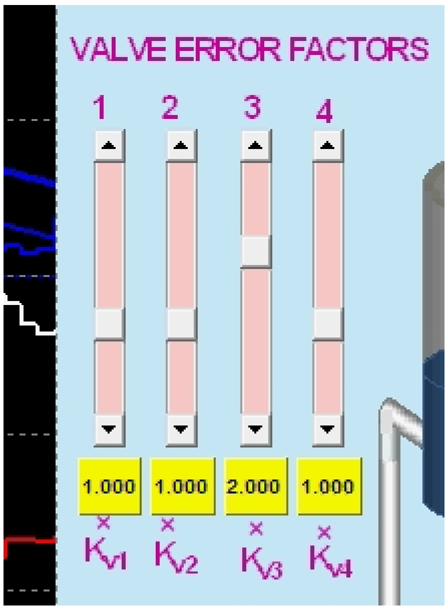


Error Correlation Matrix

	X_1	X_2	X_3	X_4	offset
L_1	-0	0	0	-0	-0
L_2	-0	0	0	-0	-0
L_3	-0	0	0	-0	-0
L_4	-0	0	0	-0	-0

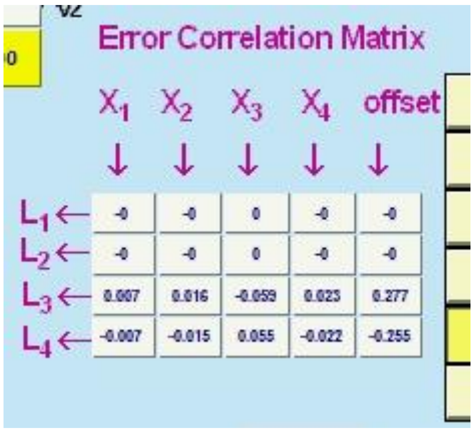
*f*_8.22

(4) Now create a plant-model mismatch by moving the error factor applied to K_{V3} (ie. the coefficient of valve 3) from 1.0 to 2.0 (*f*_8.23). Because this is applied to the *assumed* plant model, in practice it is equivalent to halving the actual K_{V3} on the plant (eg. partial blockage).



*f*_8.23

(5) Observe how the Error Correlation Matrix changes (*f*_8.24). Because of the system structure, one expects that the errors should only be identified in L_3 and L_4, and the strongesst correlation should be with valve 3.


Error Correlation Matrix

	X_1	X_2	X_3	X_4	offset
$L_1 \leftarrow$	-0	-0	0	-0	-0
$L_2 \leftarrow$	-0	-0	0	-0	-0
$L_3 \leftarrow$	0.007	0.016	-0.059	0.023	0.277
$L_4 \leftarrow$	-0.007	-0.015	0.055	-0.022	-0.255

*f*_8.24

(6) Now eliminate the error on K_{V3} by moving the slider back, and do some tests by applying errors to $K_{V1.}$ K_{V2} and $K_{V4.}$ Report on your observations and attempt to account for them.

8.3.5 Code extract

• A prediction error in tank levels is found by going back to measured tank levels $T_P = 10$ seconds ago, and using the assumed model to run forward from there 10 seconds (10 time-steps) up to present time, where the predicted levels are compared with measured levels to get the four errors. This modelling calculation is done immediately before the code listed below.

This set of errors E is associated with the set of 4 valve positions at the present time, X, and this (X,E) pair is fed into a (X_S,E_S) stack of MaxStack=200 such pairs gathered over the past sequence of time-points, with the oldest such pair being discarded when the stack reaches capacity. When the stack is full, the calculation is made verry efficient by just adjusting the variance and covariance totals for the additional contribution of the new time, and subtracting the contribution of the oldest time.

```
// Now we are back at the present
for (j=1;j<=Ntanks;j++)
{
        Lp[j]=Lm[j];         // the predictions of the present levels
        Ep[j]=Lm[j]-m_L[j];    // the prediction errors
}

// Store current predictive errors for correlation analysis
for (j=1;j<=Ntanks;j++)
{
        Elost[j]=Es[Sp][j];  // about to be overwritten
        Es[Sp][j]=Ep[j];
}

// Do correlation if enough data..........................................
if ((!b_Enough_Data) && (Sp==MaxStack))
{
        // time to initialise recursive moving window
        EX.Init(Ntanksp1);     // Correlation factors from X's to E's
        Xcov.Init(Ntanksp1,Ntanksp1);  // X covariances
        Mat.Init(Ntanksp1,Ntanksp1);   // working matrix
        EXprod.Init(Ntanksp1,Ntanks);  // EX products for correlation
        X.Init(Ntanksp1);
        EE.Init(Ntanks);
        Xcov=0;
        EXprod=0;
        Lamda_Suppression.Identity(Ntanksp1,MaxStack*1e-10,0.0);

        for (j=1;j<=Ntanks;j++)
        {
                Xsum[j]=0;
                Esum[j]=0;
        }
        X(Ntanksp1)=1;
        for (i=1;i<=MaxStack;i++)
        {
                for (j=1;j<=Ntanks;j++)
                {
                        Xsum[j]+=Xs[i][j];
                        Esum[j]+=Es[i][j];
                }
        }
        for (i=1;i<=MaxStack;i++)
        {
                for (j=1;j<=Ntanks;j++)
                {
                        Xdev[i][j]=Xs[i][j]-Xsum[j]/MaxStack;   //####MM100322 ....Test!!!
                        Edev[i][j]=Es[i][j]-Esum[j]/MaxStack;   //####MM100322 ....Test!!!
                        X(j)=Xdev[i][j];
                        EE(j)=Edev[i][j];
                }
                Xcov=Xcov+X*(~X);                       //####      /MaxStack;
                for (k=1;k<=Ntanks;k++)
                {
                        for (j=1;j<=Ntanksp1;j++)
                        {
                                EXprod(j,k)=EXprod(j,k)+EE(k)*X(j);
                        }
                }
        }
        b_Enough_Data=TRUE;
}
else
{
        if (b_Enough_Data)
        {
                X(Ntanksp1)=1;
                for (j=1;j<=Ntanks;j++)
                {
                        Xsum[j]+=Xs[Sp][j]-Xlost[j];
                        Esum[j]+=Es[Sp][j]-Elost[j];
                }
                for (j=1;j<=Ntanks;j++)
                {
                        X(j)=Xdev[Sp][j];
                        EE(j)=Edev[Sp][j];
                }
                Xcov=Xcov-X*(~X);          // dump oldest contribution
                for (k=1;k<=Ntanks;k++)
                {
                        for (j=1;j<=Ntanksp1;j++)
                        {
                                EXprod(j,k)=EXprod(j,k)-EE(k)*X(j);   // dump oldest contribution
                        }
                }
                // onto the latest measurements
                for (j=1;j<=Ntanks;j++)
                {
                        Xdev[Sp][j]=Xs[Sp][j]-Xsum[j]/MaxStack;
                        Edev[Sp][j]=Es[Sp][j]-Esum[j]/MaxStack;
                        X(j)=Xdev[Sp][j];
                        EE(j)=Edev[Sp][j];
                }
                Xcov=Xcov+X*(~X);           // bring in newest contribution  (####slight error due to mixture of averages)
                for (k=1;k<=Ntanks;k++)
                {
                        for (j=1;j<=Ntanksp1;j++)
                        {
                                EXprod(j,k)=EXprod(j,k)+EE(k)*X(j);   // bring in newest contribution
                        }
                }

                //updated, so now find coefficients
                Mat=(Xcov+Lamda_Suppression).InvNoError();
                for (k=1;k<=Ntanks;k++)
                {
                        for (j=1;j<=Ntanksp1;j++)
                        {
                                X(j)=EXprod(j,k);
                        }
                        EX=Mat*X;             // Solution! : Will give ZEROS if Xcov is singular!!!!!!  ####MM100325
                        for (j=1;j<=Ntanksp1;j++)
                        {
                                XtoEcoeff[k][j]=EX(j);
                        }
                }

        }
}
```

References

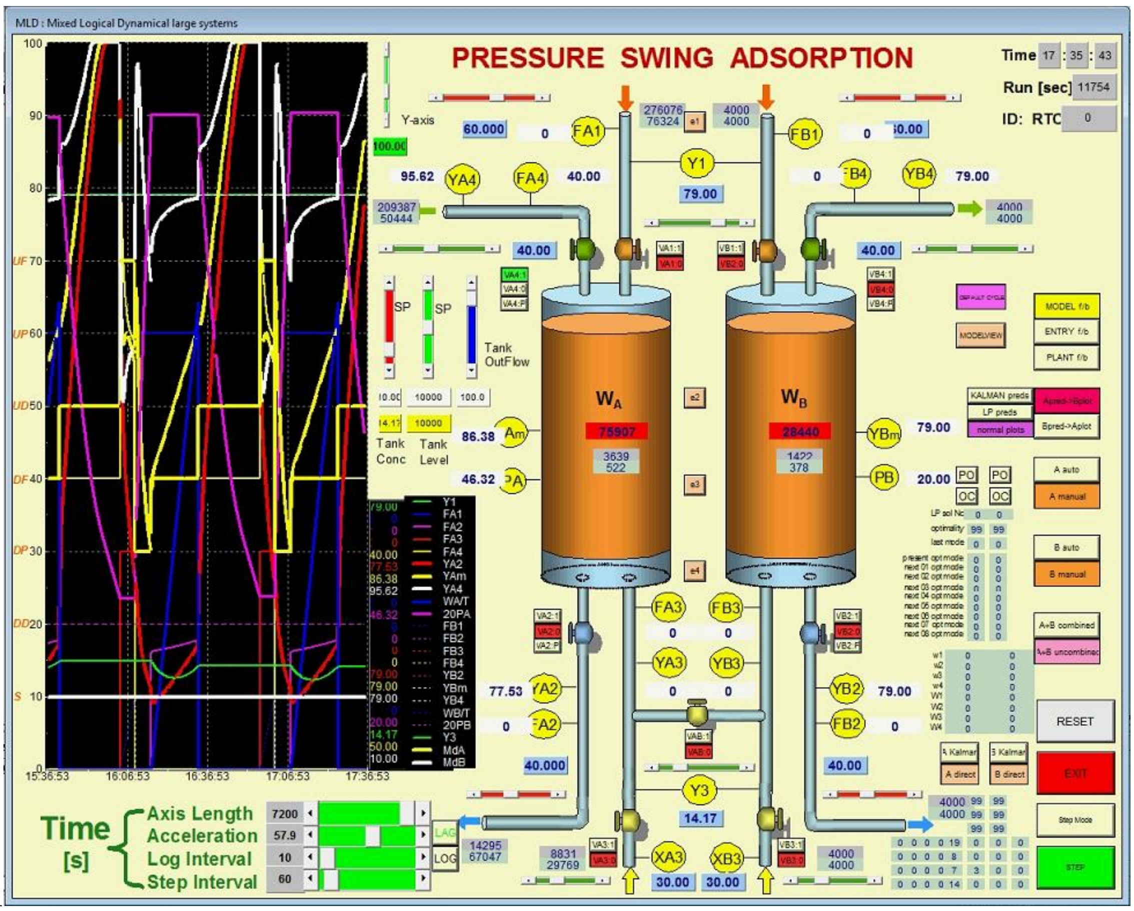
Kannie, N. and U. Managalparsad "Model error detection for a MIMO system*", Final Year Laboratory Research Project*, Chemical Engineering, University of KwaZulu-Natal, Durban, (2010).

Chapter 9 Simulations - Hybrid systems

.

9.1 MLD - Predictive control of pressure swing adsorption by optimal scheduling of the Skarstrom cycle

A four-phase cyclical scheduling problem, requiring a decision on when to end each phase in order to optimise production:



*f*_9.1

9.1.1 Typical settings

Two identical adsorbers are shown. Though this offers scope for shared use of purge gas, and for creation of a steadier supply of product, the focus at this point will be on adsorber A. Adsorber B will have the same settings.

FA1	pressurisation flow	60
FA2	depressurisation flow	40
FA3	purge flow	30
FA4	vent flow	40
Y1	% nitrogen (air feed)	79
Y2sp	setpoint % nitrogen in product tank	10

<u>Default Cycle:</u>

(1) **Pressurisation:** Pressurise by feeding FA1=60 through VA1 until P reaches 95% of Pmax (Pmax=4.5barg)

(2) **Adsorption at high pressure:** Continue to feed at FA1=60, opening VA2 to the product tank to hold pressure at Pmax. When a mass of 60000 of feed has been admitted, stop the feed by closing VA1. Also close VA2 to the product tank.

(3) **Depressurisation:** Open VA4 to vent (FA4=40) and allow the pressure to drop to Pmin.

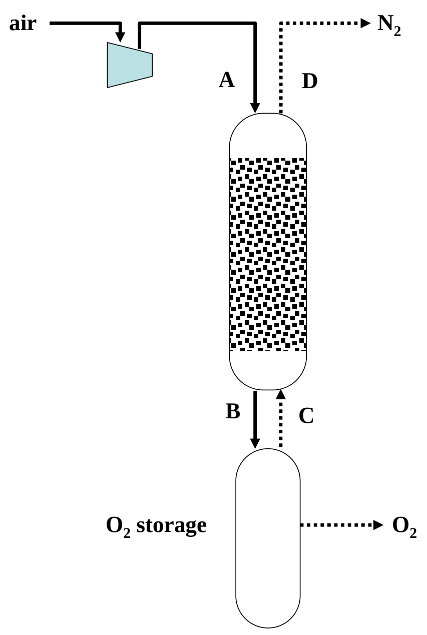
(4) **Purge at low pressure:** Now use vent valve VA4 to control the pressure at Pmin, whilst feeding back product gas as purge through VA3 at FA3=30. Once a mass of 10000 of product has been fed back, close VA3 and VA4 and repeat from (1) above.

(*Note:* The mimic diagram shows the purge flow returning independently of the product flow. This permits separate accumulators to be shown on these lines, from which the effectiveness of the process can be judged. The composition in the return purge line is set to that in the product tank.)

9.1.2 Theory

This example is taken from Mulholland and Latifi (2009). Pressure swing adsorption requires a repeated cycle of four steps. The periods of these steps, or other defined terminal conditions, determine the rate and quality of the product, and its cost. In transient situations such as upsets or grade changes, it is not intuitively obvious how the steps should be progressively altered to bring the plant to the desired operating point in an optimal fashion. The present work considers the problem of real-time maximization of the production of a single adsorber, and maintaining a setpoint concentration in its product receiving vessel. In a modelling exercise, these objectives have been met using predictive control based on completion of the present step, plus two full future cycles to reduce the end-effect. The approach sought to be fast and robust by suitable linearisation of the system. This allowed MILP solution in the mixed logical dynamical (MLD) framework as a mixed integer dynamic optimisation (MIDO). However, this problem was ultimately solved faster and more reliably by testing all combinations for constraint violations and the objective value.

An increasing range of adsorbent materials is extending the use of pressure swing adsorption (PSA) in the separation of gas mixtures. These materials are designed to selectively adsorb one component from a mixture. As in vapour-liquid equilibrium, the equilibrium quantity of this adsorbed component in the solid phase increases with its partial pressure in the gas phase. Thus the solid can be used to adsorb the component at high pressure, and it can be "regenerated" by expelling the adsorbed species at low pressure. In air separation, N_2 is selectively adsorbed, leaving an O_2-rich product stream. A number of adsorbers can be arranged to work in complementary cycles so as to smooth out production flow and the use of common resources. However, the present analysis will focus on a single adsorber with a product storage vessel. Figure *f*_9.2 shows a basic pressure swing adsorption configuration for air separation.



*f*_9.2

Four distinct steps, comprising the Skarstrom cycle, are required:

(1) **pressurisation:**
A open; B,C & D closed

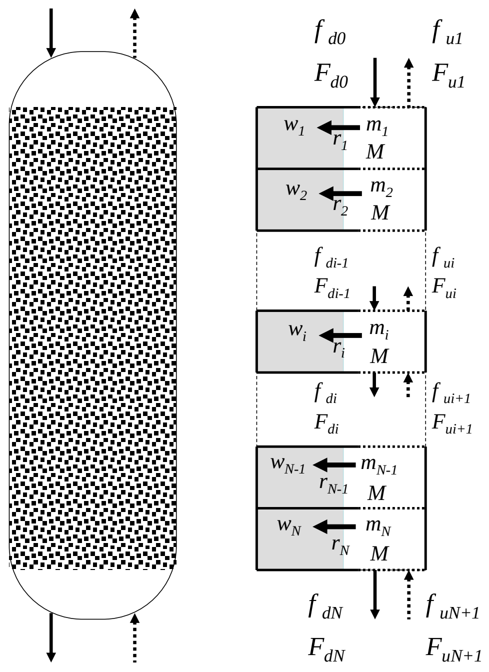
(2) **adsorption at high pressure:**
A & B open; C & D closed

(3) **depressurisation:**
D open; A,B & C closed

(4) **purge at low pressure:**
C & D open; A & B closed

During step 2, a high purity product can be obtained, particularly if some of the product itself is used in step 4 for purging, as is shown here. The mechanism by which a high-purity product is obtained is not entirely self-evident. It is in fact achieved by developing a suitable composition profile in the solid phase which acts to "screen" the down-ward moving air in step 2. That profile will of course oscillate through each full cycle of four steps, but the so-called "cyclic steady state" (CSS) is achieved once a fixed associated profile arises at the end of each step.

To date, most of the work aimed at optimising PSA operation has focused on the optimal "positioning" on the CSS cycle. The cycle can be positioned by choosing a particular set of four times, one for each of the Skarstrom steps. Alternatively, it can be positioned by choice of a particular set of heuristic rules, eg. based on pressure or flow. Figure *f*_9.3 sows a discrete representation of states in a pressure-swing adsorber.



$$r_i = \frac{dw_i}{dt} = k\left[m_i - cw_i\right]$$

*f*_9.3

In start-up, shut-down or recovery from an upset, heuristic rules are likely to be conservative and inefficient. What one seeks is an optimal strategy to bring the process from its current point to one which ensures product quality and rate, at minimum cost, possibly in coordination with other adsorbers.

The adsorber is modelled as a series of *N* mixed compartments as shown in *f*_9.3. In compartment *i*, w_i is the total number of moles of adsorbed N_2, m_i is the unadsorbed N_2, and *M* is the total unadsorbed gas, which will be the same for all compartments. Typical values are used for air separation, using a linear equilibrium relationship for the N_2 ($m_i^*=cw_i$) and ignoring the small amount of O_2 adsorbed. Pressure losses through the bed and thermal effects are likewise neglected. In the equations, M and F_i respectively represent the total gas inventory of a compartment, and the total gas flow, whilst f_i is the N_2 flow. Flows are divided into "downward" (*d*) and "upward" (*u*), of which one or the other will be zero depending on the step of the cycle.

$$\frac{dM}{dt} = \frac{1}{N}\left\{\begin{matrix} F_{d,0} + F_{u,N+1} - F_{d,N} - F_{u,1} \\ -\sum_{i=1}^{N} k\left[m_i - cw_i\right] \end{matrix}\right\} \tag{9.1}$$

$$\frac{dm_i}{dt} = f_{d,i-1} + f_{u,i+1} - f_{d,i} - f_{u,i} - k\left[m_i - cw_i\right] \tag{9.2}$$

$$\frac{dw_i}{dt} = k\left[m_i - cw_i\right] \tag{9.3}$$

The only nonlinearity arises as the requirement that the effluent composition from a compartment obeys the following equations for downward or upward flow respectively.

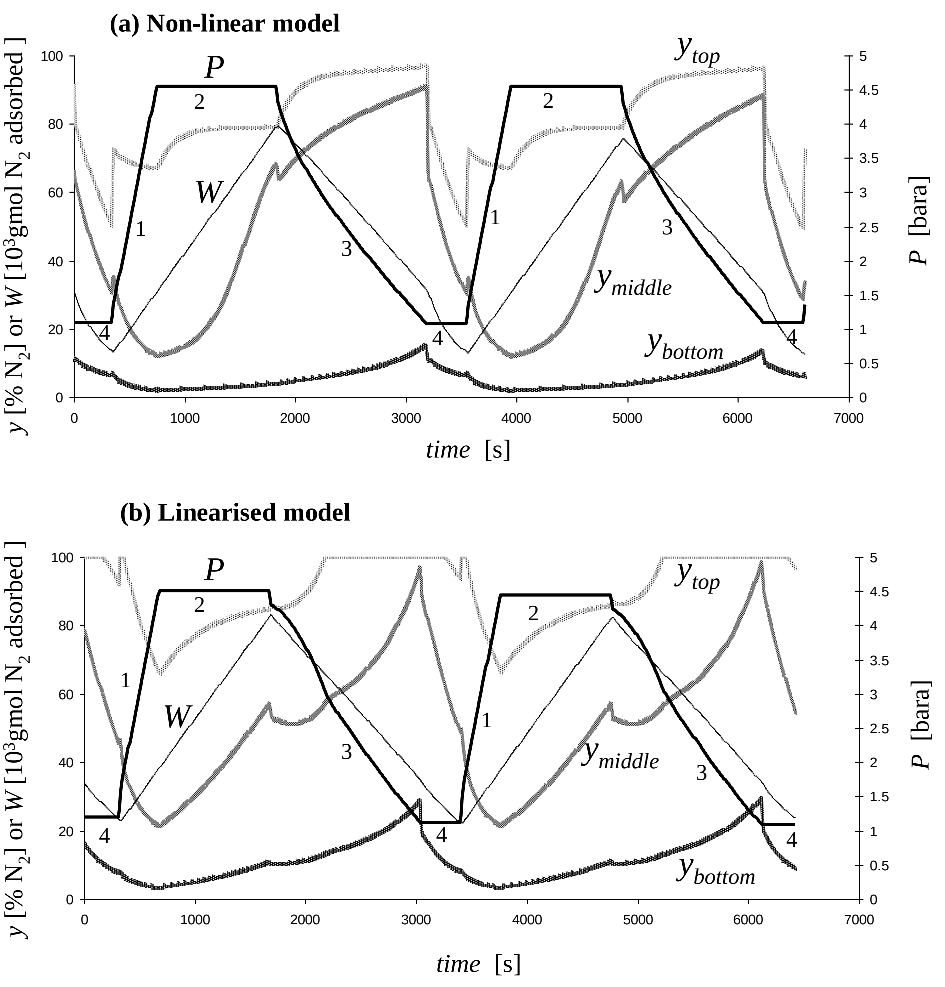
$$\frac{f_{d,i}}{F_{d,i}} = \frac{m_i}{M} = y_i \quad \text{or} \quad \frac{f_{u,i}}{F_{u,i}} = \frac{m_i}{M} = y_i \tag{9.4-9.5}$$

This was linearised using deviations (Δ) from an estimated operating point (')

$$\frac{f' + \Delta f}{F' + \Delta F} = \frac{m' + \Delta m}{M' + \Delta M} \tag{9.6}$$

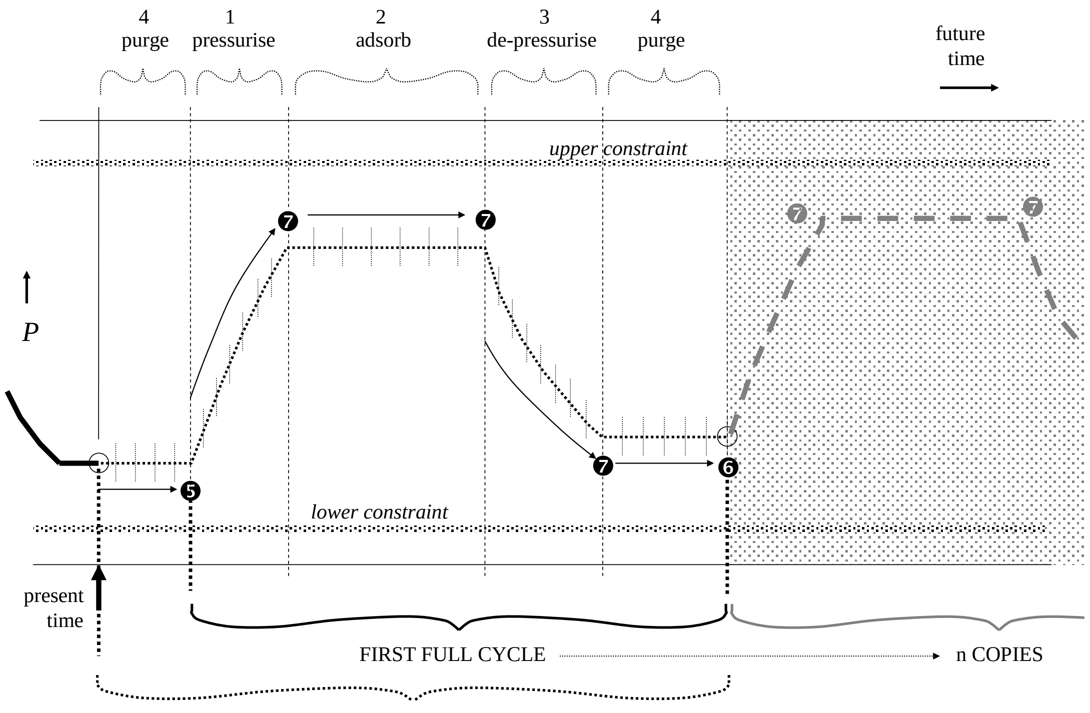
and neglect of the deviation products $\Delta f \Delta M$ and $\Delta F \Delta m$.

Figure *f*_9.4 compares *(a) Nonlinear Model* and *(b) Linearised Model* predictions for the Skarstrom cycles, under the same conditions. W is the total adsorbed mass of N_2. Though some errors are manifest in the linearised solution, they do not appear to disrupt the solution badly.



*f*_9.4

In the real-time situation, the controller is cycling asynchronously at its own time-interval. In the present case this is 10s. It does not need to match the Δt_j of any of the Skarstrom steps *j* because the entire optimisation calculation is repeated *a priori* on each controller time-step. What is important to the control algorithm is to know the system *state* at this time. A first step was thus to develop a *state observer*. A Kalman filter based on the linearised models in equations above, changing in sequence, was able to provide good estimates of the $2{\times}N{+}1$ state values using just three "measurement" outputs of the original *non-linear* model [(i) pressure *P*; (ii) product outflow composition during step 2; and (iii) purge outflow composition during steps 3 and 4].



*f*_9.5

Apart from these state values, the predictive control algorithm of course needs to know which of the four possible steps of the Skarstrom cycle is presently being conducted. (Historical information - eg. how long it has been in this step - is not required). Figure *f*_9.5 shows the concept of future Skarstrom step length optimisation for a system found (for example) to be in a purge mode on the controller time-step. A look-up table indicates the required future sequence for completion of an entire Skarstrom cycle (constrained), followed by a repeat full cycle with the same step lengths (unconstrained):

[1] **pressurisation:**
complete 1 then do 2→3→4→1 , 2→3→4→1
[2] **adsorption at high pressure:**
complete 2 then do 3→4→1→2 , 3→4→1→2
[3] **depressurisation:**
complete 3 then do 4→1→2→3 , 4→1→2→3
[4] **purge at low pressure:**
complete 4 then do 1→2→3→4 , 1→2→3→4

The identified future sequence is then the basis of the optimisation. It amounts to a choice of the number of intervals Δt_j to spend in each of the Skarstrom steps *j* (figure 4). The result is five separate interval counts. Steps occurring *after* the production step 4 would appear to play a neutral or negative role (eg. use of Product for purging). Thus the objective function used here is based on one further repetition of the cycle (*n*=1) to reduce such “end-effects”. The computational load is reduced by forcing the “copies” to use the Skarstrom step lengths of the first full cycle.

The main predictive control interest is in whether the intervals left in the first (partial) step add up to less than the controller time-step. In that case the controller must take action *now* by switching to the next Skarstrom step.

For each of the Skarstrom steps *j*=1,…,4 a range of transition models is pre-prepared, one for each of the possible number of intervals $1 \le i \le n_{\max}$ that could be used for that step:

$$\boldsymbol{x}\left(t+i\Delta t_j\right)=\boldsymbol{A}_j^{(i)}\,\boldsymbol{x}\left(t\right)+\boldsymbol{b}_j^{(i)} \tag{9.7}$$

The new arrays $\boldsymbol{A}_j^{(i)}$ and $\boldsymbol{b}_j^{(i)}$ are obtained by individually recursing equation 9.7 *i* times for each case *j* Now if the particular choices of *i* made to complete the present step *j* and the next 4 complete steps are

$i[j],\ i[j+1], i[j+2], i[j+3], i[j+4]$,

where it is understood that the index values will "wrap" around in the range 1,2,3,4, then it is these choices that must be made optimally in determining the future state sequence. Representing $\boldsymbol{x}\left(t+i[j]\Delta t_j\right)$ by $\boldsymbol{x}_j$ one has

$$\boldsymbol{x}_j=\boldsymbol{A}_j^{(i[j])}\,\boldsymbol{x}_{j-1}+\boldsymbol{b}_j^{(i[j])} \quad \text{(partial step)} \tag{9.8}$$

$$\left.\begin{aligned}\boldsymbol{x}_{j+k}&=\boldsymbol{A}_{j+k}^{(i[j+k])}\,\boldsymbol{x}_{j+k-1}+\boldsymbol{b}_{j+k}^{(i[j+k])}\\ \boldsymbol{x}_{j+k+4}&=\boldsymbol{A}_{j+k}^{(i[j+k])}\,\boldsymbol{x}_{j+k+3}+\boldsymbol{b}_{j+k}^{(i[j+k])}\end{aligned}\right\}\ k=1,\dots,4 \tag{9.9-9.10}$$

After completion of the present partial step, two whole cycles are executed, with the second cycle re-using the same number of intervals in each step as in the first cycle.

In this form, the problem lends itself to solution in the *mixed logical dynamical* (MLD) framework of Morari and co-workers (Bemporad and Morari, 1999; Morari *et al.*, 2000; Morari, 2002). Furthermore, the use of linear dynamic models allows solution by *mixed integer linear programming* (MILP). The selection of the optimal number of steps is facilitated by binary variables *δ*, eg. for equation 9.8 one requires the constraints

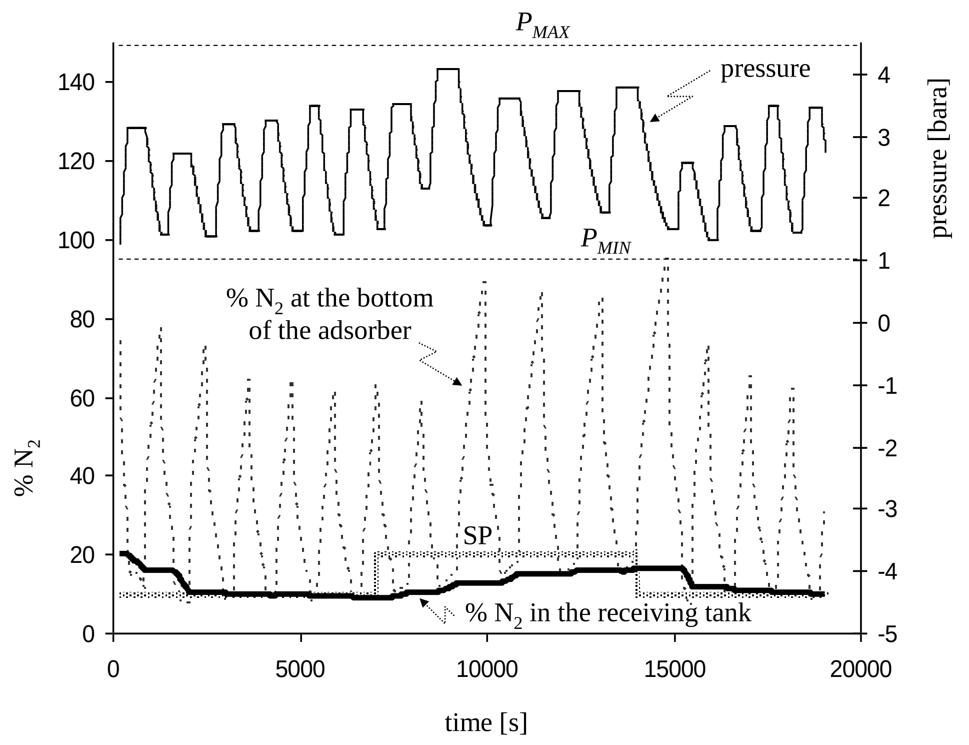
$$\boldsymbol{x}_j+\delta_{ij}\boldsymbol{e}_{\max}\le\boldsymbol{A}_j^{(i)}\,\boldsymbol{x}_{j-1}+\boldsymbol{b}_j^{(i)}+\boldsymbol{e}_{\max} \tag{9.11}$$

$$\boldsymbol{x}_j+\delta_{ij}\boldsymbol{e}_{\min}\ge\boldsymbol{A}_j^{(i)}\,\boldsymbol{x}_{j-1}+\boldsymbol{b}_j^{(i)}+\boldsymbol{e}_{\min} \tag{9.12}$$

$$\sum_{i=1}^{n_{\max}}\delta_{ij} \quad = \quad 1 \tag{9.13}$$

Here the vectors ***e*** contain the maximum and minimum deviation values when (all but one of) the *i*-models are not obeyed (large positive and negative numbers).

Whereas the task required was quite simple - *viz.* choose the best combination of interval numbers in the first five Skarstrom steps - it became clear that the linear program was an inefficient means of solving the problem. The numerous additional constraints required for model choice as in equations 9.11 to 9.13, and to deal with variable saturation, slowed down LPSOLVE (Michel Berkelaar - http://sourceforge.net/projects/lpsolve/), and caused failures. Even if continuous variables were included in the search, it would be quicker to evaluate *every apex* of the system for its objective value and compliance with constraints. Indeed, this was the procedure used to produce the results in figure *f*_9.6, for predictive control of the N_2 concentration in the storage vessel.



*f*_9.6

9.1.3 Tasks

Only operation of the single 'A' adsorber will be considered:

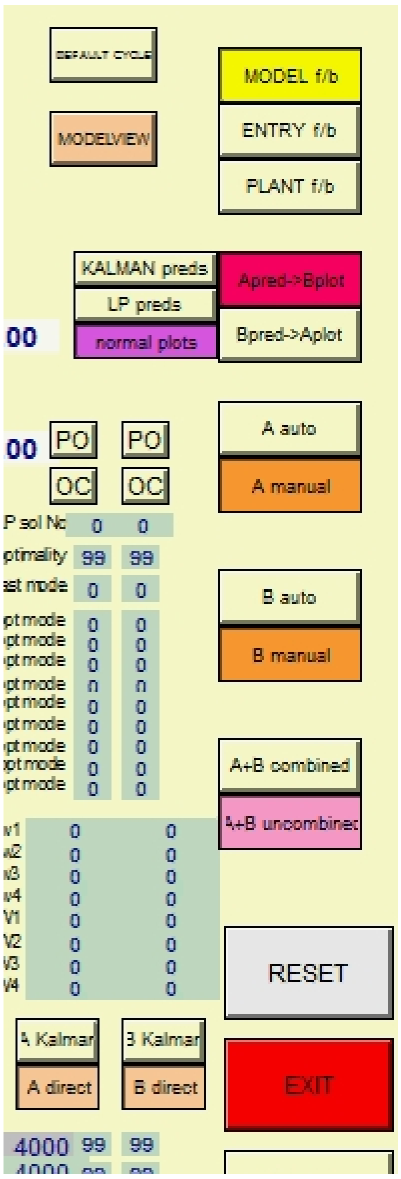
(1) Run the 'A' adsorber on the default cycle, until the concentration of N_2 in the product storage tank becomes steady

(2) Switch to optimal control of the 'A' adsorber and track changes in the production rate and composition.

(3) Once steady under optimal control, step the %O_2 setpoint, and observe the response

9.1.4 Methods

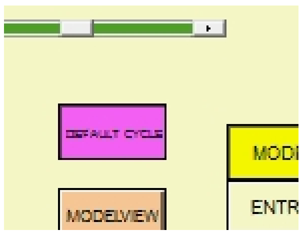
(1) When you start the MLD application, the buttons on the right should look like this (*f*_9.7):



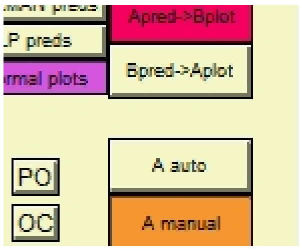
f_9.7

This means that the plots refer to the A adsorber, which is represented by the model. The full model results are being presented directly to the control algorithm (ie. all states, without having to use the Kalman filter to extend the three nominal measurements to the full state. Presently neither the A nor the B adsorber is on automatic control. All inlets and outlets are sealed.

(1) Press the DEFAULT CYCLE button to start a standard fixed cycle as detailed in section 9.1.1 (*f*_9.8). The ‘A’ adsorber should be on ‘A manual’ (*f*_9.9).

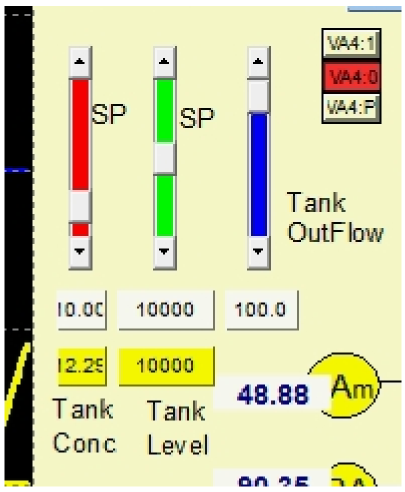


*f*_9.8



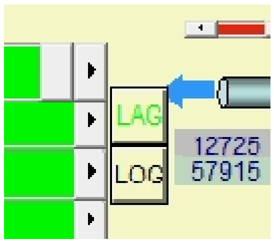
*f*_9.9

(2) Set the product O_2 tank composition setpoint to 10 %N_2, and set the total inventory of this tank (level) to a fixed mass of 10000 (setting the "Tank Outflow" scrollbar to 100 will ensure that the excess is "trimmed out" to consumers (*f*_9.10).



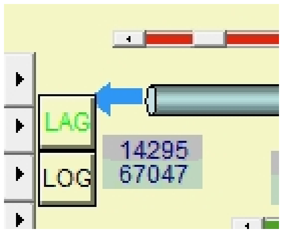
*f*_9.10

(3) Let the default cycle repeat a number of times until the system is reacting in a steady fashion to it, then start a LOG file to record changes (*f*_9.11). The %N_2 in the product storage tyank (Y3) will have dropped from its starting value of 20%.



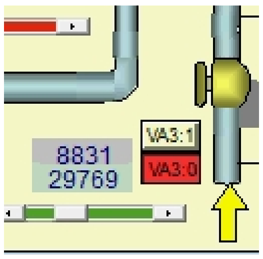
*f*_9.11

(4) In this analysis, the supply of gas to the nominal "storage tank" has been disconnected from the return purge gas as follows: The supply arrives as FA2 during the Skarstrom steps 2 and 3. This flow is represented ifigure *f*_9.12 by two "accumulators", of which the top one (14295) represents the mass of N_2 delivered by FA2 so far, and the bottom one (57915) is the total mass of gas (N_2 and O_2) delivered by FA2 so far.



*f*_9.12

Since in this analysis, the product storage tank volume has been set, the excess gas must leave the vessel for consumption/overflow. Regardless, the storage vessel is well mixed, so this creates a first-order response as far as % N_2 in the vessel is concerned. This computed %N2 is transferred to Y3, the % N2 in the returning purge gas flow FA3. However, the total purge gas used is otherwise unlimited, is kept independent of FA2 and the storage vessel, and is only recorded on two similar accumulators in figure *f*_9.13, one for the accumulated mass (8831) of N_2 returned as purge and the other for the total mass (29769) of gas (N_2 and O_2) returned as purge (but always at current Y3).



*f*_9.13

Thus the net production and cumulative composition of production over a period of time is given by the difference between these two sets of accumulators.

(5) Once the default cycle is steady, set the 'A' adsorber to optimal control by pressing the "A AUTO" button, and then immediately switch off the default cycle by pressing the DEFAULT CYCLE button again.

(6) Let the system run for some time to assess how well the controller brings the storage tank composition Y3 to its setpoint (10% N_2). Check that the "LAG" warning at the bottom of the screen does not switch on (*f*_9.12). This would mean that the computations are starting to take longer than the simulation clock (which has been left with its acceleration too great). If this happens, reduce the acceleration to allow the computations to catch up.

(7) Once the system has settled again, step the storage tank setpoint up to 20% N_2, and observe the response for a number of cycles.

(8) Close your log file by pressing LOG again (*f*_9.12). Exit the application and retrieve your log file. Process your data to track the storage tank %N_2 variations during the transitions, and how the total production rate (average per time) varied between the default cycle and optimal controller.

9.1.5 Code extracts

• The final part of the LP solution set-up, after the constraints are set, is the definition of the objective function, and execution of the LP.

```
(................after setting of all of the constraints for the LP, ...............continue with.........)

//[ALP] SET UP OBJECTIVE FUNCTION  (BUT IT IS SAID TO BE FASTER AHEAD OF CONSTRAINTS? #########)
ret+=set_add_rowmode(lp, FALSE); //[ALP] have to turn if off when stopped adding rows
j = 0;                          //[ALP]########first objective col is at 1 not 0 !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

for (iphase=1; iphase<=Nphase_TOTAL; iphase++)  //######################### was 2 to 5 !!!!
{
// add the absolute deviations in the "whole" part of the cycle (include "skip" s for the meantime!!!!!!!!!!!###############)

   //[ALP] OBJECTIVE FUNCTION : X_Tank_SP tracking incentive (LEVEL CONTROL)
   colno[j] = n*iphase+nAb+4*N+4;  // X_Tank_absdev
   row[j++] = -X_Tank_SP_wt;

   //[ALP] OBJECTIVE FUNCTION : m_Tank_SP tracking incentive (COMPOSITION CONTROL)
   colno[j] = n*iphase+nAb+4*N+5;  // m_Tank_absdev
   row[j++] = -m_Tank_SP_wt;
}

//[ALP] OBJECTIVE FUNCTION : Production rate incentive (as W2)
colno[j] = 1*n+2*N+1+6;                  // W2 at end of FIRST phase
row[j++] = -ProductionW2_wt;             // not really necessary but makes numbers more meaningful
colno[j] = Nphase_TOTAL*n+2*N+1+6; // W2 at end of LAST phase
row[j++] = +ProductionW2_wt;

//[ALP] OBJECTIVE FUNCTION : Consumption rate dis-incentive (as W3)
colno[j] = 1*n+2*N+1+7;                  // W3 at end of FIRST phase
row[j++] = +ConsumptionW3_wt;            // not really necessary but makes numbers more meaningful
colno[j] = Nphase_TOTAL*n+2*N+1+7; // W3 at end of LAST phase
row[j++] = -ConsumptionW3_wt;

//[ALP] Finally....
ret+=set_obj_fnex(lp, j, row, colno);         //[ALP]
set_maxim(lp);                    //[ALP] direction of optmisation

//[ALP] Now lpSOLVE the optimisation!
//[ALP] ============================
set_verbose(lp, IMPORTANT);         //[ALP] only important messages to be shown (##########how????)
set_timeout(lp, 40);              //[ALP] timeout in seconds############
set_scaling(lp,SCALE_EXTREME );           //[ALP] scaling - various choices ??########

LPresultA = solve(lp);                //[ALP]

//[ALP] Results (ret)
//[ALP]=============
//[ALP] Solver status values
//[ALP] UNKNOWNERROR            -5
//[ALP] DATAIGNORED            -4
//[ALP] NOBFP                 -3
//[ALP] NOMEMORY              -2
//[ALP] NOTRUN                -1
//[ALP] OPTIMAL                0
//[ALP] SUBOPTIMAL              1
//[ALP] INFEASIALE             2
//[ALP] UNBOUNDED               3
//[ALP] DEGENERATE              4
//[ALP] NUMFAILURE              5
//[ALP] USERABORT              6
//[ALP] TIMEOUT                7
//[ALP] RUNNING                8
//[ALP] PRESOLVED               9


//[ALP] get results
double objval=get_objective(lp);
get_variables(lp, row);


//[ALP] Interpret results
//[ALP] =================

LPsoln_numberA+=1;

if (LPresultA == 0) //############MM090113   ####((LPresultA == 0) || (LPresultA == 1))   //[ALP] or SUBOPTIMAL
{
   .
   .
   .  (.....implement optimal control action first move and step model.......)
```

References

Bemporad A. and Morari M. (1999) "Control of systems integrating logic, dynamics and constraints", *Automatica* **35**, 407-427

Morari M. (2002) "Hybrid system analysis and control via mixed integer optimisation", *Chemical process control VI, AIChE Symposium Series No.326, Vol.*

Morari M., Bemporad A. and Mignone, D. (2000), "A framework for control, state estimation, fault detection and verification of hybrid systems", *Automatisierungstechnik* **48**, 1-8.

Mulholland, M. and M.A. Latifi, "Predictive control of pressure swing adsorption", *AT&P Journal* PLUS **2**, ISSN 1336-5010, 43-50 (2009)