A systematic approach to the tuning of multivariable Dynamic Matrix Control (DMC) controllers

by

GZ Gous

A dissertation submitted in partial fulfilment
of the requirements for the degree

Master of Engineering (Control Engineering)

in the

Department of Chemical Engineering

Faculty of Engineering, the Built Environment
and Information Technology

University of Pretoria

Pretoria

September 2011

Synopsis

Traditionally the tuning of DMC-type multivariable controllers is done by trial and error. The APC engineer would choose arbitrary starting values and test the performance on a simulated controller. The engineer would then either increase the values to suppress movement more, or decrease them to have the manipulated variables move faster. When the controller performs acceptably in simulation, then the tuning is improved during the commissioning of the controller on the plant. This is a time consuming and unscientific exercise and therefore often does not get the required attention, leading to unacceptable controller behaviour during commissioning and sub-optimal control once commissioning is completed.

This dissertation presents a new method to obtain move suppression factors for DMC type multivariable controllers. The challenge in choosing move suppressions lies in the multivariable nature of the controller. Changing the move suppression on one manipulated variable will not only change the performance of that manipulated variable, it will also change the performance of every other manipulated variable with models to the same controlled variables. In the same way, changing the steady state cost of a manipulated variable or the equal concern error of a controlled variable will also affect the behaviour of every other manipulated variable with shared models.

There have been attempts to calculate the required move suppression factors mathematically. Some methods used an approach that is based on the premise that move suppression factors that present a well-conditioned controller matrix will provide a well behaved controller in terms of tuning. Some other methods focussed on providing parameters that will cause desirable controlled variable response, either by determining tuning parameters offline, or by re-tuning the controller in real time.

The method described in this paper uses a Nelder Mead (Nelder and Mead, 1965) search algorithm to search for move suppressions that will provide acceptable control behaviour. Acceptable behaviour is defined by characterising the dynamic move plan calculated by the controller for each of the manipulated variables, or by characterising the controlled variable path that will result from the manipulated variable moves. The search algorithm can change the move suppressions, the steady state costs, or the move suppression multipliers as used in DMC type controllers.

List of Symbols

δCV	-	Change in controlled variable
A	-	Dynamic unit step response model
ΔMV	-	Change in manipulated variable
r	-	Residual error
e	-	Controlled variable error (target – predicted)
a	-	Elements of A matrix
K	-	Move suppression of a manipulated variable
Wt	-	Controlled variable error weight
L	-	Large number
SSCost	-	Steady State or LP cost of a manipulated variable
MaxProfit	-	The maximum amount that the optimisation function Θ can be minimised if no constraints existed on the controlled variables.

1. Introduction

Model based multivariable control using the method now known as DMC or MPC was developed in industry in the early 1970's with the first application in 1973 (Cutler, 1983, Qin & Badgwell, 2003) and Prett and Gilette(1980) presenting the first application of DMC on a FCC unit. Numerous advances have been made such as non-linear controllers, matrix conditioning, real time gain scheduling etc. (Qin & Badgwell, 2003). The one part of the technology that is still largely without a scientific base has been the selection of move suppressions to tune the controller behaviour. To this day the acceptable approach in industry is to use trial and error to find tuning values that will provide acceptable controller behaviour in an offline simulation. These values are then refined during the commissioning of the online controller. Using trial and error on a small (4 manipulated variable) controller is already time consuming, and there are controllers in the field with more than 100 manipulated variables. Using this approach is bound to lead to oversights that will lead to difficulty during the commissioning of the controller. During the commissioning, the problems that do occur are often addressed by over-suppressing the manipulated variable movement, leading to a badly tuned controller.

The challenge in choosing or guessing move suppressions lies in the multivariable nature of the controller. Changing the move suppression on one manipulated variable will not only change the behaviour of that manipulated variable, it will also change the behaviour of every other manipulated variable with models to the same controlled variables. In the same way, changing the equal concern error of a controlled variable will also affect the behaviour of every manipulated variable with shared models. This makes the iterative approach as used throughout industry even more time consuming and unscientific.

Another challenge is that the "right" behaviour in a DMC controller is very ill-defined and is often based on the practitioner's personal experience and preferences. More or less aggressive controller behaviour is the outcome of the engineer's aversion to risk. Should the engineer make the controller too aggressive, circumstances such as model error may cause the controller to produce unwanted cycles on the plant. On the other hand, too little aggression in tuning may result in a controller that is sluggish and does not control the process properly.

A further level of complexity is added by the model-based nature of a DMC controller. If the model is an exact match of the plant behaviour, very aggressive tuning may be used. Should factors like non-linearity, process response changes or process noise lead to substantial plant/model mismatch, aggressive tuning will once again lead to undesired controller behaviour and performance.

In order to address these complexities several metrics were defined to describe desirable controller behaviour. These metrics are based on either the dynamic move plan calculated for the manipulated variables over the control horizon of the controller, or they are based on the dynamic response of the controlled variables that result from the manipulated variables' move plan. Subsequently a method was developed to find tuning parameters that will provide behaviour that will satisfy the metrics in an unconstrained DMC controller.

2. Background

DMC controllers are also called model predictive controllers. The technology is based on linear models that describe the process behaviour. At the heart of DMC lies the equation:

$\delta CV = A * \Delta MV$ …………………. 2.1

which is used to calculate how the controlled variables will change, based on controller input changes, or to calculate how to manipulate the manipulated variables in order to obtain the desired controlled variable response.

This study is concerned with the dynamics of the controller. In calculating the dynamic move plan for the manipulated variables, δCV is the desired controlled variable response, or the change in controlled variable value for each prediction interval of the controller. ΔMV is the dynamic move plan to be calculated or the change in manipulated variable for each control interval calculated by the controller, as shown in equation 2.7. A is the controller model expressed as the unit step response model for each manipulated variable, controlled variable pair, shown in equation 2.6.

A DMC type controller executes periodically. At every execution cycle the controller will:

- Calculate the open loop prediction for each controlled variable.
- Decide on steady state values for all manipulated variables and controlled variables.
- Calculate a move plan for each manipulated variable.
- Implement the first move of the move plan

2.1. Open loop prediction

During the open loop prediction, the controller will update a vector holding the future open loop prediction for each controlled variable by considering the changes in all controller inputs (manipulated variables and feedforward variables). The controller model will be used as in equation 2.1, multiplied by the most recent changes in inputs, and superimposed on the previous prediction of each controlled variable.

Feedback is applied by comparing the current value of each controlled variable with the predicted value from the previous execution cycle, and the new prediction will be shifted to start from the current controlled variable value.

2.2. Steady state values

Knowing what the open loop behaviour of the controlled variables will be, the controller must now calculate what to do with the manipulated variables to get the controlled variables to their desired values. The next step is therefore to use the:

- steady state gain information from the controller matrix
- the controlled variable setpoints or high and low limits
- manipulated variable high and low limits
- economic cost factors on manipulated variables and/or controlled variables

to determine the desired end values for all manipulated variables and controlled variables.

These optimisation considerations impact control action and control tuning by maximising or minimising manipulated variables, moving them in directions that can have a negative impact on control in the short term, and often running manipulated variables against high or low limits. Therefore this tuning method assumes that all manipulated variables are unconstrained and can participate in the control action. It also only considers control action, not the effect of superimposed optimisation action as well. Because this study assumes that we are dealing with an unconstrained multivariable controller, the intricacies of determining end values will not be discussed.

Because of these assumptions, the tuning provided by the method will need to be improved during commissioning, but as the method is aimed at providing initial tuning values pre-commissioning, this is acceptable. The tuning provided will have to be improved on the online controller to accommodate issues like non-linearity, valve sticktion and measurement noise regardless of how the starting values were obtained.

2.3. Move plan calculation

Next the controller must determine how to move the manipulated variables from their current positions to the end positions. This path that the manipulated variables will follow is called the move plan of the controller. This move plan will be determined by calculating a number of future moves for each manipulated variable.

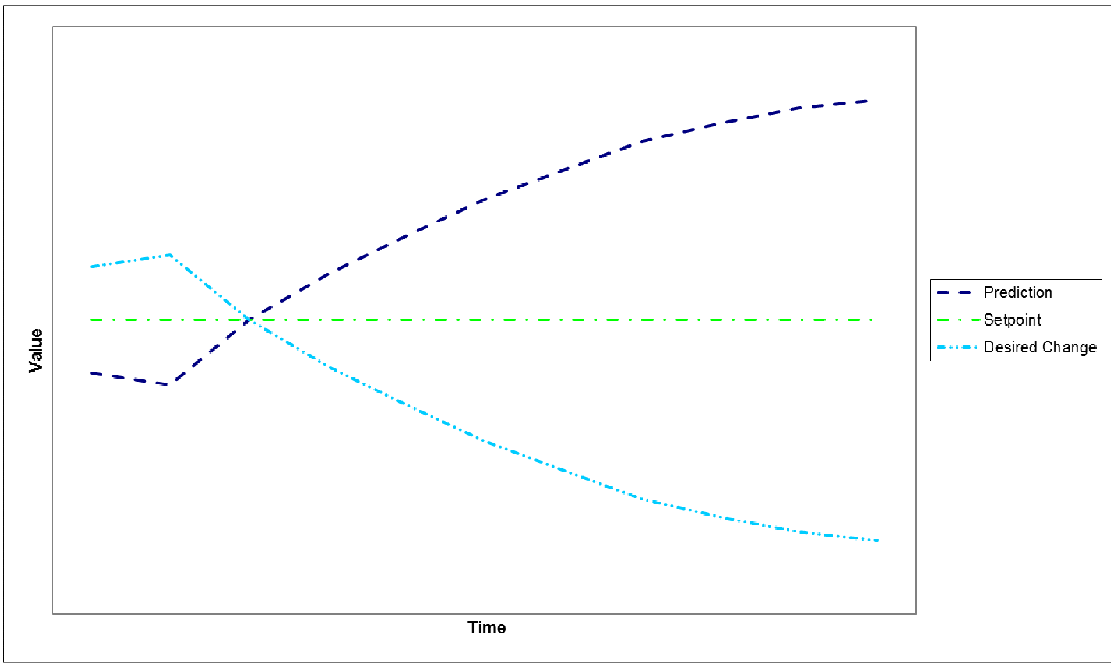


Figure 2.1 Open loop prediction for a controlled variable

If a single manipulated variable/ controlled variable pair is considered, with an open loop prediction for the controlled variable as shown by the prediction curve in figure 2.1, then the controller must find the optimal move plan for the manipulated variable to change the controlled variable behaviour from the open loop prediction to the setpoint. A move plan must therefore be found that will cause the controlled variable to respond like the mirror image of the open loop prediction around the setpoint (the desired change curve in figure 2.1) (Cutler, 1982). If this desired change can be induced in the controlled variable, and this change is added to the open loop prediction, a vector that is equal to the setpoint will result.

2.3.1. Minimising controlled variable error

As such a move plan is often physically impossible, a move plan must be found that causes that closest fit with this behaviour. The controlled variable error (e) is defined as the difference between setpoint and the open loop prediction. As the change affected by the manipulated variable movement is A*dMV, the residual error will be

$r = A * \Delta MV - e$ …………………………………………….. 2.2

The best possible setpoint tracking will be found at the minimum amount of residual error. In order to find the absolute minimum the residual error is squared.

$r^T r = (A * \Delta MV - e)^T (A * \Delta MV - e)$ …………………………..2.3

The minimum will then be found at

$$\frac{\partial\left(r^T r\right)}{\partial\,\Delta MV} = z \quad \text{………………………………………………………..2.4}$$

Where z is a vector of zeros with the same dimensions as ΔMV

Using the matrix identities $(AB)^T = B^T A^T$, $(A + B)^T = A^T + B^T$ and

$\frac{\partial\left(A^T B\right)}{\partial\, B} = \frac{\partial\left(B^T A\right)}{\partial\, B} = A$ this leads to the minimum being at

$\Delta MV = [A^T A]^{-1} A^T e$ …………………………………………………2.5

In equation 2.5, using a 2x2 controller with 4 model coefficients, that will calculate 2 manipulated variable moves at execution cycles 1 and 3 as example:

$$A = \begin{bmatrix} a_{1,1,1} & 0 & a_{1,2,1} & 0 \\ a_{1,1,2} & 0 & a_{1,2,2} & 0 \\ a_{1,1,3} & a_{1,1,1} & a_{1,2,3} & a_{1,2,1} \\ a_{1,1,4} & a_{1,1,2} & a_{1,2,4} & a_{1,2,2} \\ a_{1,1,4} & a_{1,1,3} & a_{1,2,4} & a_{1,2,3} \\ a_{1,1,4} & a_{1,1,4} & a_{1,2,4} & a_{1,2,4} \\ a_{2,1,1} & 0 & a_{2,1} & 0 \\ a_{2,1,2} & 0 & a_{2,2,2} & 0 \\ a_{2,1,3} & a_{2,1,1} & a_{2,2,3} & a_{2,2,1} \\ a_{2,1,4} & a_{2,1,2} & a_{2,2,4} & a_{21,2,2} \\ a_{2,1,4} & a_{2,1,3} & a_{2,2,4} & a_{2,2,3} \\ a_{2,1,4} & a_{2,1,4} & a_{2,2,4} & a_{2,2,4} \end{bmatrix} \quad \text{…………………………………………………..2.6}$$

where $a_{i,j,k}$ is the unit step response between controlled variable i and manipulated variable j at time interval k.

$$\Delta MV = \begin{bmatrix} \Delta MV_{1,1} \\ \Delta MV_{1,2} \\ \Delta MV_{2,1} \\ \Delta MV_{2,2} \end{bmatrix} \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots 2.7$$

where $\Delta MVi,j$ is the movement calculated for manipulated variable i at control interval j.

and

$$e = \begin{bmatrix} e_{1,1} \\ e_{1,2} \\ e_{1,3} \\ e_{1,4} \\ e_{1,5} \\ e_{1,6} \\ e_{2,1} \\ e_{2,2} \\ e_{2,3} \\ e_{2,4} \\ e_{2,5} \\ e_{2,6} \end{bmatrix} \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots 2.8$$

2.3.2. Minimising manipulated variable movement

Finding this minimum will calculate the most aggressive setpoint tracking possible. It will make the largest manipulated variable moves needed to theoretically keep the controlled variables as close to setpoint as possible. In practise, this is a bad idea as the smallest amount of model error will lead to the controller starting a cycle on the plant. Even if the models are perfect, or adapted to prevent this, physical constraints and final control element wear make such aggressive control undesirable.

For this reason the equation

$K^* \Delta MV = 0$ ……………………………………………………………2.9

is added to the A matrix in equation 2.5. This adds an opposing objective, namely to minimise manipulated variable movement to the existing objective which is to minimise residual controlled variable error.

In our 2x2 example, equation 2.5 remains the same, but the components change as follows:

$$A = \begin{bmatrix} a_{1,1,1} & 0 & a_{1,2,1} & 0 \\ a_{1,1,2} & 0 & a_{1,2,2} & 0 \\ a_{1,1,3} & a_{1,1,1} & a_{1,2,3} & a_{1,2,1} \\ a_{1,1,4} & a_{1,1,2} & a_{1,2,4} & a_{1,2,2} \\ a_{1,1,4} & a_{1,1,3} & a_{1,2,4} & a_{1,2,3} \\ a_{1,1,4} & a_{1,1,4} & a_{1,2,4} & a_{1,2,4} \\ a_{2,1,1} & 0 & a_{2,2,1} & 0 \\ a_{2,1,2} & 0 & a_{2,2,2} & 0 \\ a_{2,1,3} & a_{2,1,1} & a_{2,2,3} & a_{2,2,1} \\ a_{2,1,4} & a_{2,1,2} & a_{2,2,4} & a_{2,2,2} \\ a_{2,1,4} & a_{2,1,3} & a_{2,2,4} & a_{2,2,3} \\ a_{2,1,4} & a_{2,1,4} & a_{2,2,4} & a_{2,2,4} \\ K_1 & 0 & 0 & 0 \\ 0 & K_1 & 0 & 0 \\ 0 & 0 & K_2 & 0 \\ 0 & 0 & 0 & K_2 \end{bmatrix} \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots 2.10$$

where K_i is the move suppression for manipulated variable i.

$$e = \begin{bmatrix} e_{1,1} \\ e_{1,2} \\ e_{1,3} \\ e_{1,4} \\ e_{1,5} \\ e_{1,6} \\ e_{2,1} \\ e_{2,2} \\ e_{2,3} \\ e_{2,4} \\ e_{2,5} \\ e_{2,6} \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots 2.11$$

If equation 2.5 is solved with these components a much more conservative move plan will result, depending on the size of K_i. This is because now a new objective, namely minimise manipulated variable movement, is added to the existing objective, minimise residual controlled variable error. By increasing the value of K_i, more emphasis is placed on minimising manipulated variable movement, increasing move suppression at the expense of minimisation of controlled variable error.

2.3.3. Relative controlled variable importance

Some controlled variables are more important than others. For instance, violating a tubeskin temperature on a furnace may cause the tube to fail, resulting in loss of equipment or loss of life. On the other hand, violating a product specification will cause additional cost to rework the product, or the product will have to be sold as second grade. It is quite obvious that it is more important to ensure that the tubeskin temperature stays below the high limit than keeping product in specification.

For this reason, the error on all controlled variables is multiplied by a weight, making the error more or less, depending on the importance of the variable. In DMC, this weight is the inverse of a tuning parameter called the equal concern error if a linear program is used to minimise error. If a quadratic program is used, the weight is equal to the inverse of the square of the equal concern error. Therefore the error matrix becomes:

$$e = \begin{bmatrix} e_{1,1} * Wt_1 \\ e_{1,2} * Wt_1 \\ e_{1,3} * Wt_1 \\ e_{1,4} * Wt_1 \\ e_{1,5} * Wt_1 \\ e_{1,6} * Wt_1 \\ e_{2,1} * Wt_2 \\ e_{2,2} * Wt_2 \\ e_{2,3} * Wt_2 \\ e_{2,4} * Wt_2 \\ e_{2,5} * Wt_2 \\ e_{2,6} * Wt_2 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \qquad \ldots\ldots 2.12$$

where Wt_i is the inverse or the square of the inverse of controlled variable i.

2.3.4. Enforcing steady state values

The last consideration in determining the move plan for a controller is that nominal stability results if the manipulated variables are forced to move to their steady state values as determined by the steady state module that executes before the dynamic control module. (Genceli and Nikolau, 1993). This is done by once again adding to A and e in the following manner.

$$A = \begin{bmatrix} a_{1,1,1} & 0 & a_{1,2,1} & 0 \\ a_{1,1,2} & 0 & a_{1,2,2} & 0 \\ a_{1,1,3} & a_{1,1,1} & a_{1,2,3} & a_{1,2,1} \\ a_{1,1,4} & a_{1,1,2} & a_{1,2,4} & a_{1,2,2} \\ a_{1,1,4} & a_{1,1,3} & a_{1,2,4} & a_{1,2,3} \\ a_{1,1,4} & a_{1,1,4} & a_{1,2,4} & a_{1,2,4} \\ a_{2,1,1} & 0 & a_{2,1} & 0 \\ a_{2,1,2} & 0 & a_{2,2,2} & 0 \\ a_{2,1,3} & a_{2,1,1} & a_{2,2,3} & a_{2,2,1} \\ a_{2,1,4} & a_{2,1,2} & a_{2,2,4} & a_{21,2,2} \\ a_{2,1,4} & a_{2,1,3} & a_{2,2,4} & a_{2,2,3} \\ a_{2,1,4} & a_{2,1,4} & a_{2,2,4} & a_{2,2,4} \\ K_1 & 0 & 0 & 0 \\ 0 & K_1 & 0 & 0 \\ 0 & 0 & K_2 & 0 \\ 0 & 0 & 0 & K_2 \\ L & L & 0 & 0 \\ 0 & 0 & L & L \end{bmatrix} \qquad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots 2.13$$

where L is a very large number. The error matrix becomes:

$$e = \begin{bmatrix} e_{1,1} * Wt_1 \\ e_{1,2} * Wt_1 \\ e_{1,3} * Wt_1 \\ e_{1,4} * Wt_1 \\ e_{1,5} * Wt_1 \\ e_{1,6} * Wt_1 \\ e_{2,1} * Wt_2 \\ e_{2,2} * Wt_2 \\ e_{2,3} * Wt_2 \\ e_{2,4} * Wt_2 \\ e_{2,5} * Wt_2 \\ e_{2,6} * Wt_2 \\ 0 \\ 0 \\ 0 \\ 0 \\ \Delta MV_1 \\ \Delta MV_2 \end{bmatrix} \quad \text{..........}2.14$$

This will force the move plan to take the manipulated variables to the steady state values calculated by the steady state optimisation module, ensuring nominal stability.

2.3.5. Other DMC tuning constants

The investigation was expanded to include other tuning constants than just move suppressions. Other tuning constants that will have an impact on a DMC controller are move suppression multipliers.

Move suppression multiplier

A move suppression multiplier is a factor that is multiplied to the last couple of move suppressions in the A matrix. If k is used for the move suppression multiplier then, from the sixth move to the final move in the plan, the move suppression is increased in a linear fashion from the value of K to a value of K * k. For a SISO controller that will calculate 8 control moves, equation 2.13 will be replaced by:

$$A = \begin{bmatrix}
a_{1,1,1} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
a_{1,1,2} & a_{1,1,1} & 0 & 0 & 0 & 0 & 0 & 0 \\
a_{1,1,3} & a_{1,1,2} & a_{1,1,1} & 0 & 0 & 0 & 0 & 0 \\
a_{1,1,4} & a_{1,1,3} & a_{1,1,2} & a_{1,1,1} & 0 & 0 & 0 & 0 \\
a_{1,1,4} & a_{1,1,4} & a_{1,1,3} & a_{1,1,2} & a_{1,1,1} & 0 & 0 & 0 \\
a_{1,1,4} & a_{1,1,4} & a_{1,1,4} & a_{1,1,3} & a_{1,1,2} & a_{1,1,1} & 0 & 0 \\
a_{1,1,4} & a_{1,1,4} & a_{1,1,4} & a_{1,1,4} & a_{1,1,3} & a_{1,1,2} & a_{1,1,1} & 0 \\
a_{1,1,4} & a_{1,1,4} & a_{1,1,4} & a_{1,1,4} & a_{1,1,4} & a_{1,1,3} & a_{1,1,2} & a_{1,1,1} \\
a_{1,1,4} & a_{1,1,4} & a_{1,1,4} & a_{1,1,4} & a_{1,1,4} & a_{1,1,4} & a_{1,1,3} & a_{1,1,2} \\
a_{1,1,4} & a_{1,1,4} & a_{1,1,4} & a_{1,1,4} & a_{1,1,4} & a_{1,1,4} & a_{1,1,4} & a_{1,1,3} \\
a_{1,1,4} & a_{1,1,4} & a_{1,1,4} & a_{1,1,4} & a_{1,1,4} & a_{1,1,4} & a_{1,1,4} & a_{1,1,4} \\
K_1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & K_1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & K_1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & K_1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & K_1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & K_1 * (\frac{2}{3} + \frac{1}{3}k) & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & K_1 * (\frac{1}{3} + \frac{2}{3}k) & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & K_1 * k \\
L & L & L & L & L & L & L & L
\end{bmatrix} \quad ..2.15$$

The default of the move suppression multiplier is 2. As an example, if 8 control moves are calculated, and a move suppression of 1 is used with a move suppression multiplier of 5 is used, the K_1 values as shown in equation 2.13 will be 1,1,1,1,1,2.33,3.67,5. This will place a higher penalty on moves later in the control horizon.

Steady state costs
There are two types of manipulated variables in DMC, Min Cost and Min Move variables. If a manipulated variable is defined as Min Cost, the optimisation algorithm will either minimise or maximise this variable, depending on the LP cost that is set for that specific variable. If the manipulated variable is defined as Min Move, the optimisation algorithm will minimise the movement of that variable.

If a manipulated variable is defined as min cost, the steady state optimisation module in DMC will minimise a LP with a linear objective function:

$$\Theta = \Sigma(SSCost_i * \Delta MV_i) \quad \text{.......}2.16$$

or a QP with a quadratic function:

$$\Theta = (\Sigma(SSCost_i * \Delta MV_i) - MaxProfit)^2 \quad \text{.......}2.17$$

MaxProfit is the maximum amount by which the LP or QP function Θ can be minimised if controlled variable limits are ignored and only manipulated variable limits are enforced.

Using an LP or QP will ensure that the controller minimises or maximises the variables according to the relative steady state costs, and the sign of the costs of the manipulated variables.

If the manipulated variables are defined as Min Move manipulated variables, the LP objective function is changed to:

$$\Theta = \Sigma(SSCost_i * \Delta MV_i)_{MinCost} + \Sigma(SSCost_j * abs(\Delta MV_j))_{MinMove} \quad \text{.......}2.18$$

and to

$$\Theta = (\Sigma(SSCost_i * \Delta MV_i)_{MinCost} + \Sigma(SSCost_j * abs(\Delta MV_j))_{MinMove} - MaxProfit)^2 \quad \text{....}2.19$$

if a QP is used. Therefore, if a manipulated variable is defined as a Min Move variable, the steady state cost becomes a penalty on the total movement of the manipulated variable to steady state.

2.4. Tuning DMC controllers

2.4.1. Traditional DMC tuning

Traditional DMC tuning is done by first choosing the equal concern errors for all controlled variables. If a LP is used in the optimisation, the control weight (W) used in eq.2.12 and 2.14. will be set to the inverse of the equal concern errors. If a QP is used, then the weight is set to the inverse of the square of the equal concern error. The equal concern errors are values that indicate a comparable value of controlled variable error. It is a good way of comparing controlled variables with dissimilar units of measure, like temperatures and pressures. The values are often chosen by asking the question “What magnitude of controlled variable violation would cause operator distress?" and using this value as the equal concern error. In other words, if a violation of 2 kPa on a pressure would get the operators' attention, and a violation of 5 ^{o}C on a temperature would have a similar effect, these are good values for equal concern errors. The values are chosen in an arbitrary fashion, but represent the comparative value of keeping the controlled variables within limits.

After step testing control engineers often have a very good feel for these values, as they have probably caused severe operator discomfort during the entire step testing process.

Next the move suppression values (K_i in 2.13 and 2.15) must be chosen for all manipulated variables. Traditionally this is a non-scientific, laborious affair of trial and error (Iglesias, Sanjuán and Smith, 2006) where initial move suppression values are chosen and the controller response to upsets and setpoint changes simulated. The move suppressions are then adapted until the engineer is satisfied with the rate of change on the manipulated variables when controlled variable error exists in the simulation. Next the controller is commissioned on the plant and the tuning parameters refined by observing the controller response (Qin & Badgwell, 2003).

There are two major shortcomings with this approach. Firstly the definition of a "reasonable" response has never been defined and is left to the judgment of the control engineer. This will lead to the result that two engineers will tune the same controller and come up with very different tuning parameters that are based on experience and personality.

Secondly, the methodology is made difficult due to the multivariable nature of the tuning. If the move suppression of one manipulated variable is increased, this will cause the controller to allow less movement in that manipulated variable, which will inevitably lead to more controlled variable error as well as more movement on all manipulated variables that have models to the same controlled variables.

These points were proven by asking eight control engineers to tune the same two controllers and by comparing the chosen move suppression values. Details regarding the processes are shown in Appendix C. They all followed the traditional trial and error approach. No guidance was given regarding acceptable tuning, they had to use their own discretion and experience. The experience of the engineers varied from 2 to more than 10 years in APC as shown in table 2.1

Table 2.1 Experience of participating engineers

	Years APC Experience	Number of controllers implemented	Largest number of manipulated variables
Engineer 1	10+	25	25+
Engineer 2	2-5	25+	25+
Engineer 3	2-5	10-25	10-25
Engineer 4	5-10	10-25	10-25
Engineer 5	2-5	5-10	10-25
Engineer 6	2-5	5-10	10-25
Engineer 7	5-10	10-25	10-25
Engineer 8	2-5	20	10-25

The different tuning results are shown in table 2.2 and 2.3.

Table 2.2 Move Suppression values chosen for Distillation plant

	Engineer 1	Engineer 2	Engineer 3	Engineer 4	Engineer 5	Engineer 6	Engineer 7	Engineer 8	Average	Standard deviation
SupMov1	10.0	10.0	7.0	100	3	0.02	1	1	16.5	36.1
SupMov2	1.0	5.0	8.0	5	10	0.25	0.1	1	3.8	3.9
SupMov3	5.0	5.0	5.0	5	20	0.25	0.1	5	5.6	6.7

Table 2.3 Move suppression values chosen for Reactor

	Engineer 1	Engineer 2	Engineer 3	Engineer 4	Engineer 5	Engineer 6	Engineer 7	Engineer 8	Average	Standard deviation
SupMov1	6.0	2.5	5.0	3	5	6	4	8.5	5.0	1.4
SupMov2	12.0	5.0	6.0	3	12	12	1	14	8.1	4.7
SupMov3	15.0	5.0	4.0	3	12	12	1	15	8.0	5.1
SupMov4	5.0	2.0	3.0	3	6	6	1	11	4.8	2.2
SupMov5	1.0	2.0	5.0	9	7	7	1	20	7.1	4.3

The variation between the different move suppression values obtained by trial and error is shown clearly in graphical format in figures 2.2 and 2.3.

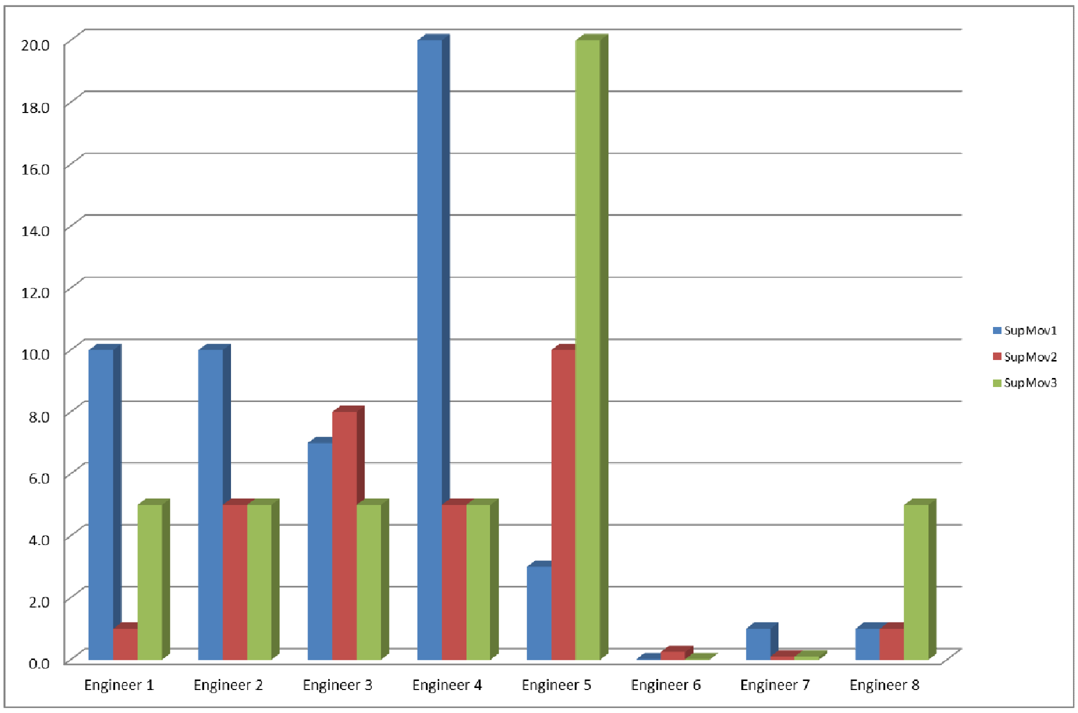


Figure 2.2 Tuning constants for distillation plant

This is an indication of the current situation in industry, with initial controller tuning varying according to the experience and personality of the control practitioner.

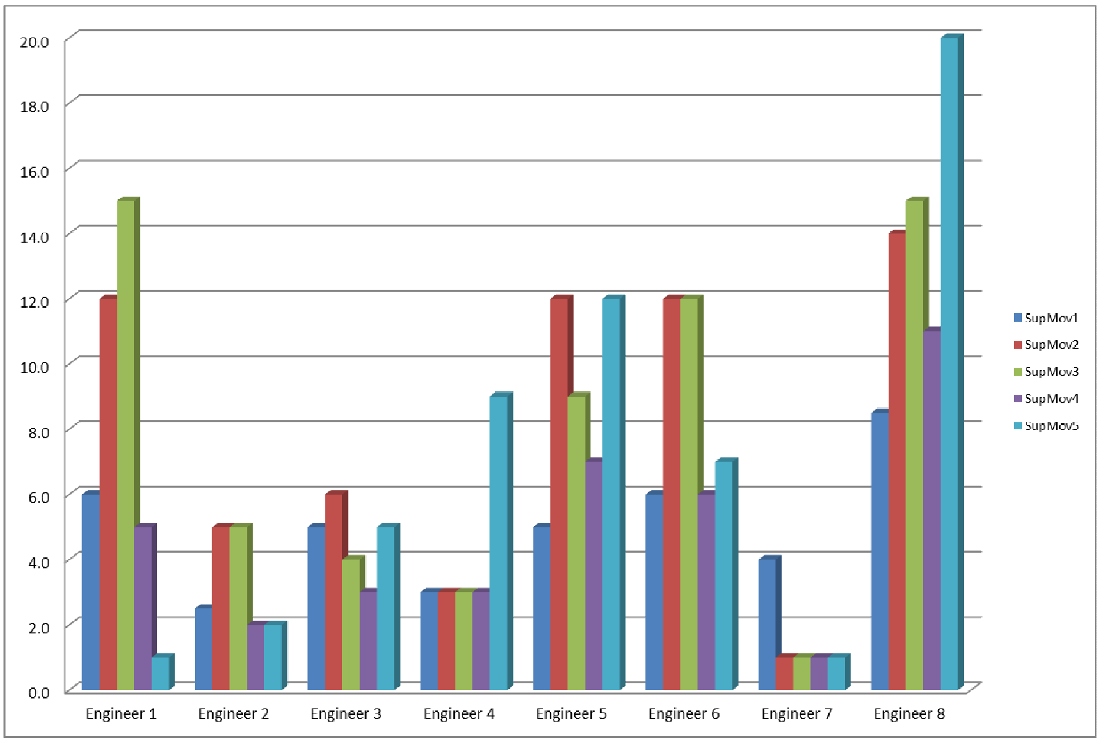


Figure 2.3 Tuning for Reactor

The different graphs clearly show a wide variance in the tuning constants chosen by the engineers. As previously discussed, this is because there is no clear definition of how a well performing controller should behave, as well as the personal preferences and experience of the engineers.

2.4.2. Recent developments in DMC tuning

There have been attempts to calculate the required move suppression factors mathematically. Shridar and Cooper (1998) noted that move suppression factors serve a dual purpose in DMC. Increasing the move suppressions will decrease manipulated variable movement, but will also decrease the matrix conditioning number. They used an approach that is based on the premise that these two effects are interrelated. They deduced that move suppression factors that present a well-conditioned controller matrix will provide a well behaved controller in terms of tuning and developed a tuning strategy that will calculate move suppressions to provide a predefined matrix conditioning number.

Other authors (Iglesias, Sanjuán and Smith, 2006) report that this method leads to unacceptably aggressive tuning. They developed a method of simulating the control behaviour and minimising a cost function. The cost function is the integral of the controlled variable error added to the integral of the manipulated variable movement multiplied by a weighting factor or:

$$PP = \int_0^\infty |e(t)|\,dt + \Gamma\int_0^\infty |m(t)|\,dt \;\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots 2.1$$

where:

- PP = Cost function
- e(t) is the controlled variable error
- m(t) is the manipulated variable moves

Γ is the weighting factor placed on the manipulated variable movement .

Increasing the weighting factor will decrease manipulated variable movement. Analysis of variance was then used to find the significant variables to calculate a tuning equation. The tuning equation developed provides significantly larger values of move suppression, with the shortcoming that it will only provide tuning values for SISO systems with first order models, severely limiting its use in industry.

Kai Han et al (2006) proposed a min-max algorithm that will select tuning parameters that will cause controlled variables to move sharply to steady state values with slightly oscillatory behaviour as shown in figure 2.4.

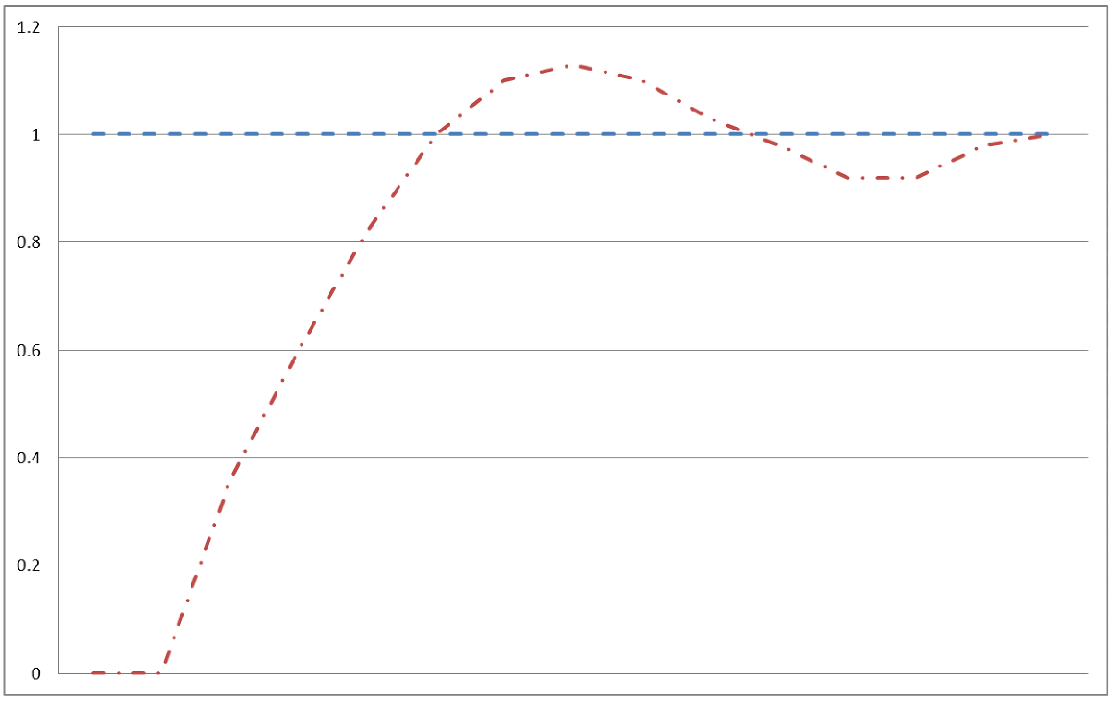


Figure 2.4 Han method controlled variable response

Ghazzawi, A et al. (2010) introduced an online tuning strategy that will re-tune the controller in real time, based on the predicted closed loop controlled variable response. The tuning will be based on the dynamic response to setpoint changes or how well disturbances are rejected. Acceptable dynamic limits are set on setpoint changes and disturbance rejection as shown in figure 2.5. The authors claim that this approach will work in a constrained multi-variable controller.

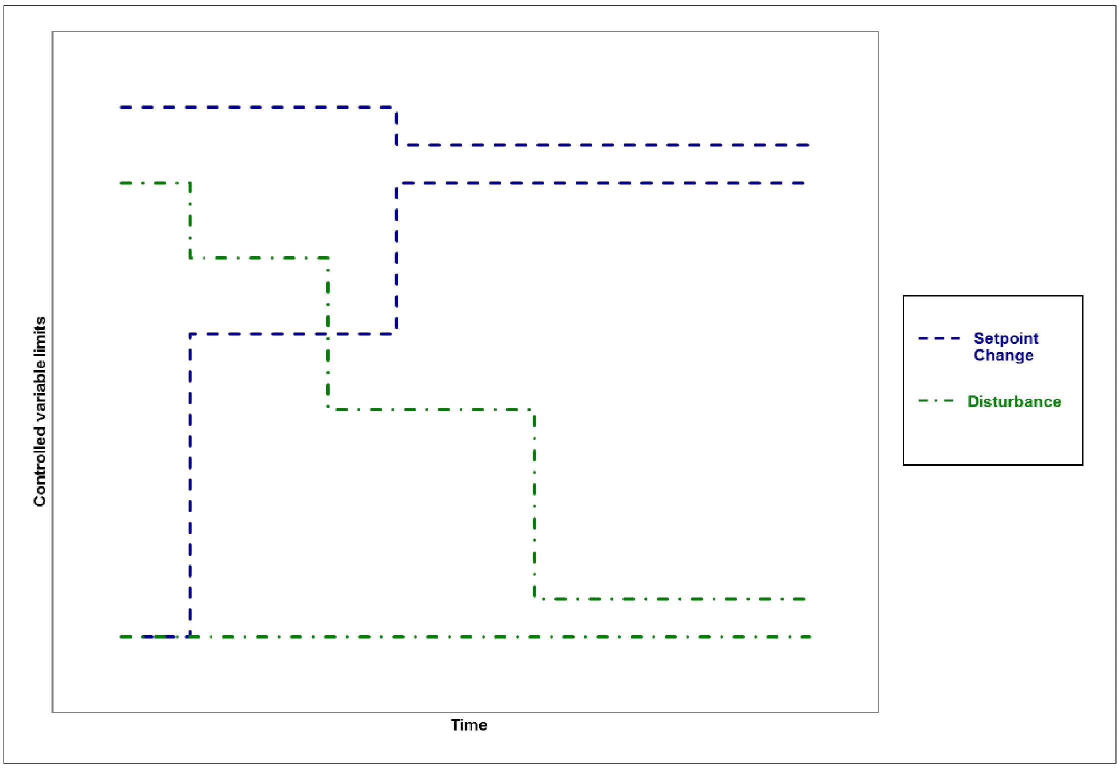


Figure 2.5 Examples of time domain performance specifications

While it is true that this approach will keep retuning the controller to handle controlled variable deviations consistently even when manipulated variables are constrained, this also introduces the risk that the control action will become excessively aggressive when all primary manipulated variables are constrained and the controller must use secondary relationships to bring a controlled variable to setpoint. As it is often difficult to obtain good models for these weaker relationships, this approach may then lead to unacceptable controller behaviour due to model error.

This risk is mitigated by placing upper and lower bounds on the move suppressions. Choosing the upper and lower bounds on the move suppressions brings the engineer back to the arbitrary choice of move suppression values and could significantly decrease the value obtained by using this approach.

3. Problem Statement

As should be apparent in the overview presented in the previous chapter, choosing move suppressions for DMC is not a trivial exercise. Because of the multi-variable nature, the different tuning parameters all interact with each other, making it a difficult exercise (Garcia, E. and Morari, M., 1985). If the tuning is not done properly, the controller will behave unacceptably on a live plant, with possible loss of production time, product, equipment, or in the worst case, loss of life.

3.1. Multivariable tuning

Changing the move suppression of one manipulated variable will change the rate at which the controller will manipulate this variable. Increasing the move suppression will slow it down, leading to less manipulated variable movement, with the obvious trade-off of more controlled variable error.

Because of the formulation of the DMC algorithm, changing one move suppression value will not only influence the movement of that manipulated variable, it will also affect the movement of all other manipulated variables that have models to the same controlled variables. Thus slowing down one manipulated variable will place a larger burden on the other manipulated variables to get rid of the controlled variable error. It will also lead to a slower rejection of disturbances or a slower change to a new setpoint on controlled variables.

This leads to a well-known truth, jokingly called the law of conservation of variability. Control engineers cannot get rid of variability; they can only move it between variables. If you suppress movement on manipulated variables, more variability will remain in the controlled variables and vice versa. If you remove variability on one manipulated variable by increasing the move suppression, it will increase variability on the other manipulated variables and all associated controlled variables.

3.2. Non-linear nature of DMC tuning.

In tuning a typical DMC, the control engineer will find that if increasing a move suppression by a certain amount will make the manipulated variable move slower, then increasing the move suppression again by the same amount will then have much less of an impact on the response. DMC tuning seems to be quite non- linear.

In figure 3.1 the move plan for one manipulated variable is shown, with move suppressions ranging from 1 to 10. This is one manipulated variable in a 6 by 6 controller. The move suppressions on the other manipulated variables were kept constant.

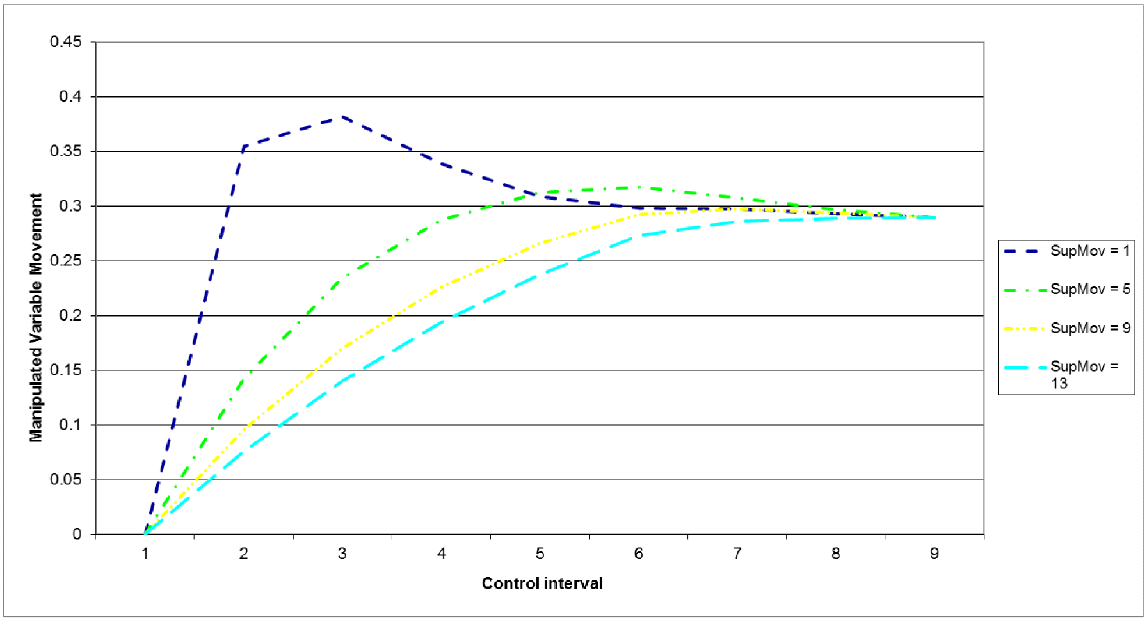


Figure 3.1 Effect of move suppression on manipulated variable movement

Figure 3.2 shows the response of the controlled variable on the difference in tuning.

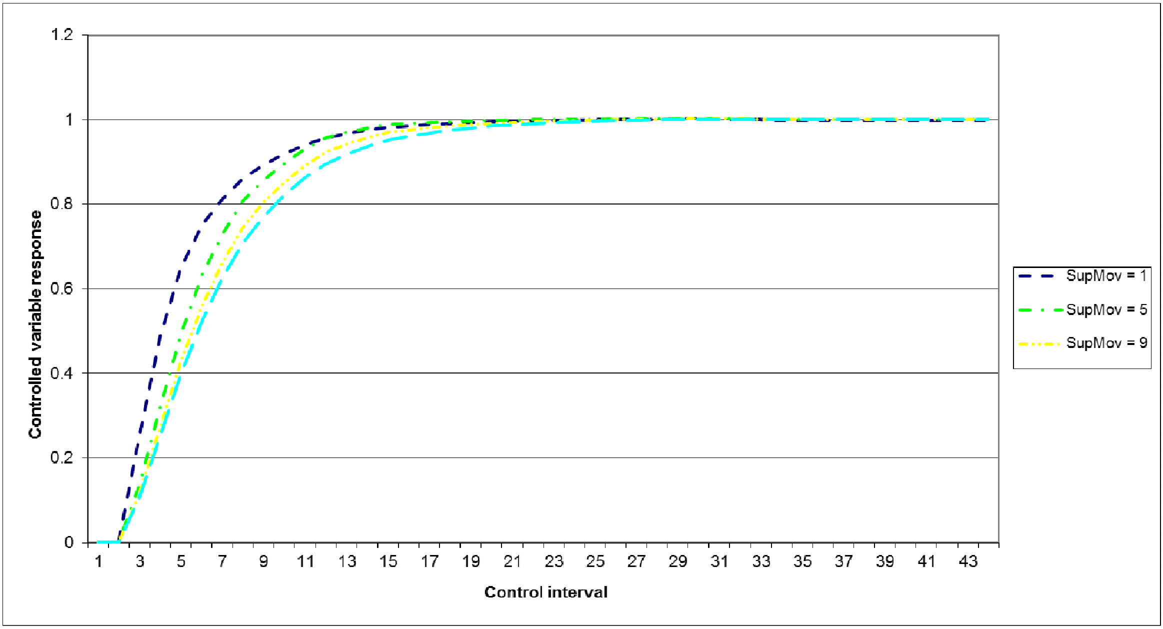


Figure 3.2 Effect of move suppression on controlled variable response

This clearly shows that changing the move suppression from 1 to 5 has a much larger impact on the manipulated variable than changing it from 9 to 13. The effect on the controlled variable is similar, but less pronounced as the other manipulated variables are called on to do more or less work.

3.3. The effect of model error

If perfect models exist, the tuning exercise will merely be a choice of where to place the process variability. Manipulated variables may be tuned so that valves move as fast as is acceptable, load and energy changes happen at a rate that operators are comfortable with, and that will suffice. The controller should control the plant well and instability should not occur.

However, perfect models are generally not the norm (Garcia & Morari, 1985). Process non-linearity is a major source of model/plant mismatch. Plants also change constantly, heat exchangers foul, catalyst degrade, pumps and valves wear out. (Cutler and Perry, 1983) There is often also noise and process drift as well as unmeasured disturbances present during step testing that will cause model error.

Model error can exist in the dynamics as well as the model gains. If the model gain is too big, the controller will make initial manipulated variable moves that are too small. Feedback will show that the moves were too small and in time the controller will increase the moves until the controlled variables reach their steady state targets. This will lead to a sluggish controller that will move manipulated variables too slowly and excessive controlled variable error will be seen.

Should the model gains be smaller than the real plant response, a much more serious situation will exist. The initial move plan will contain manipulated variable moves that are too large, and should the model mismatch be large enough, the first couple of moves may be enough to cause the controlled variables to overshoot their targets. If they overshoot by the same amount as the initial error, a sustained cycle will result. If the overshoot is larger than the initial error, the cycle will grow. This is an example of controller induced instability or input-output instability. (Nikolaou, 2001)

The same is true of error in the dynamics of the model. If the controlled variables respond faster than the controller expects them too, it will decrease the manipulated variable movement accordingly, leading to a sluggish response. If the plant reacts slower than the controller predicts, the controller will increase manipulated variable movement that may lead to cycling if the model error is large enough.

The effect of model error that may start a cycle on the plant is moderated by correct tuning. If very aggressive manipulated variable movements are made by the controller, feedback will come too late and a cycle is likely. Should the controller make slower moves, a cycle may be avoided.

On the other hand, if model error exists that will lead to a sluggish controller, this may be exaggerated by slow controller tuning. It must be stressed that it is far better for the control engineer to err on the sluggish side, rather than to have to explain why a cycle was started on the plant.

3.4. Lack of definition of acceptable control performance

During a tuning exercise, comments like "That manipulated variable is moving too fast" or "This manipulated variable is making moves that are too big" is often heard. These comments are based on the experience and personal likes and dislikes of the individual engineer, with no common language that can be used to compare the level of aggressiveness of the chosen tuning variable values. This will result in controllers that are tuned aggressively or not, depending on who did the project initially.

It is also not possible to compare the tuning between different controllers or even different manipulated variables in one controller with each other. Because the relative size of the equal concern errors that were chosen for the controlled variables is played off against the relative size of the move suppressions on the manipulated variables, there can be no absolute guideline of an acceptable range for move suppression values.

Many control engineers focus on magnitude of manipulated variable movement when tuning a controller and are indeed taught this way in advanced control courses in industry. The problem with this approach is that the magnitude of the manipulated variable movement is dependant on the size of the controlled variable error. As is intuitively clear, if the controlled variable is far from the desired steady state value, the manipulated variable will have to move over a larger range to make this possible. Therefore the magnitude of the controlled variable error must first be set before evaluating the manipulated variable movement. Choosing the magnitude of manipulated variable movement as indication of tuning aggression can therefore be misleading. In figure 3.3 it is shown that doubling the size of the controlled variable error will double the size of the manipulated variable movement in a linear fashion.

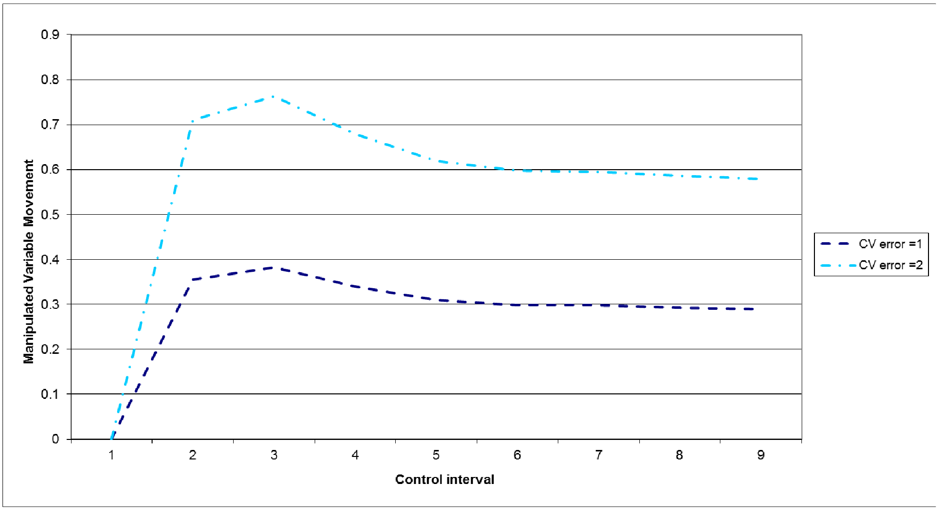


Figure 3.3 Effect of increasing controlled variable error

In the days of 8 bit controlled variable and manipulated variable values, it was common practise to scale all manipulated variables and controlled variables to get model gains that fall within a narrow range of each other. This was done to prevent loss of resolution on manipulated variable moves, but more importantly, to prevent LP errors in the DMC calculations. This practise is not enforced in most new DMC implementations as higher resolution variables are available. The effect of not scaling input variables also has a large impact on the choice of tuning variables, making a guideline for an acceptable range for tuning variables even more unattainable.

For these reasons, and also the non-linear nature of DMC tuning as discussed in section 3.2, it cannot be said that a controller with move suppressions below 1.0 is tuned fast and a controller with move suppressions above 10.0 will be slow.

4. Proposed Solution

4.1. Description of method developed

4.1.1. Characterisation of acceptable control behaviour – manipulated variable overshoot

As stated in section 3, many engineers place a large emphasis on magnitude of manipulated variable movement when tuning a controller. This will lead to variable results as the magnitude of the manipulated variable movement will be a function of controlled variable error. What is clear in figure 3.3 though is that even if the controller has to double the magnitude of the manipulated variable movement in order to address the controlled variable error, the shape of the move plan does not change. This remains true for different values of move suppressions. This fact can be exploited to define aggressiveness of DMC tuning.

If a manipulated variable is tuned very slowly, it will rise (or fall) steadily over the control horizon, almost in a linear fashion. The controlled variable will typically respond by moving slowly to the desired steady state value. More aggressive tuning will cause the manipulated variable to rise quite sharply to the steady state value, causing the controlled variable to rise quicker to its steady state target. If the move suppression is decreased more, the manipulated variable will tend to go beyond its steady state value, then return to it at the end of the control horizon. This will typically cause the controlled variable to go to its steady state target quickly, in some cases even crossing it before settling to it at steady state.

The effect of a manipulated variable to go beyond its steady state value to then change direction to settle at the steady state value will be called manipulated variable overshoot. The magnitude of the overshoot will be calculated as shown in figure 4.1.

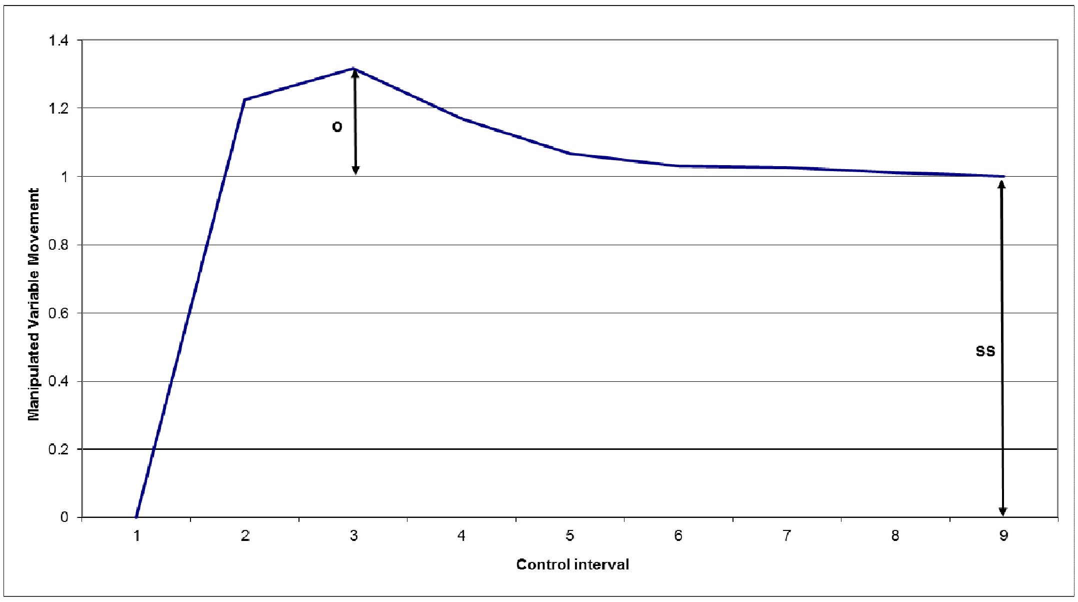


Figure 4.1 Calculating manipulated variable overshoot

If overshoot is defined as the magnitude of movement of the manipulated variable beyond the steady state value, shown by o in figure 4.1, with the steady state value shown as ss, then:

$$Percentage\,MV\,Overshoot = (o*100)/ss \quad \text{4.1}$$

As shown before the percentage overshoot is not dependent on the controlled variable error, but it is a good indication of the level of aggression of tuning.

If the 6 by 6 controller example in section 3.2 is considered again, it can be seen how the level of manipulated variable overshoot changes with the different move suppression values.

Table 4.1 Manipulated variable overshoot as function of move suppression

Move suppression	**Manipulated variable overshoot**
1	30.7%
5	7.0%
9	1.3%
13	0.0%

Table 4.1 clearly shows how manipulated variable overshoot decreases when the move suppression values are increased. It is also apparent that this is not a linear relationship.

The response in table 4.1 is not only dependent on the move suppression of the manipulated variable, it also depends on the specific model and the move suppressions of the other manipulated variables. It is also interesting to note that not only does increasing the move suppression value decrease the amount of overshoot, it also moves the occurrence of the overshoot to later in the calculated move plan.

The use of manipulated variable overshoot is equivalent to the use of manipulated variable overshoot in PID or SISO tuning (King, 2011). It is used in Lambda tuning first developed by Dahlin (1968). Lambda tuning is a form of internal model control and Chien (1988) and later Chien and Fruehauf (1990) developed tuning methods using direct synthesis. In PID tuning manipulated variable overshoot is also seen as a measurement of acceptable tuning aggression.

It is therefore suggested that in multivariable control manipulated variable overshoot can also be used as an indication of aggression of tuning and the metric that can be used to quantify this behaviour is simply the manipulated variable overshoot as shown in equation 4.1.

4.1.2. Characterisation of acceptable controller behaviour – first order manipulated variable movement

Table 4.1 shows that at large values of move suppression, manipulated variable overshoot goes to zero, and increasing move suppression more will not affect this, even though manipulated variable movement will still slow down. If a controller is desired with tuning that is less aggressive than tuning that will cause manipulated variable overshoot, another indication of tuning is if the chosen parameters cause the manipulated variables to gently go to their steady state values, moving along a first order path.

In a typical DMC controller the control horizon is 0.5 times the time to steady state chosen for the controller model. This is the time over which the manipulated variable must move to its steady state value. A controller that is tuned to follow a first order manipulated variable move plan will be quite robust when there is significant model uncertainty, but will still react fast enough to provide reasonable control performance.

The metric that was developed to characterise a manipulated variable that follows a first order path is to calculate the sum of the squared error between a first order move plan and the actual manipulated variable movement. The time constant for the first order response must be chosen so that the manipulated variable reaches its steady state value at the end of the control horizon. The software for this study was designed with a control interval of 15 controller cycles as explained in section 4.3, therefore the time constant was set at 3.5 controller intervals.

$$First\ order\ metric\ error = \sum^{control\ moves}(first\ order\ MV\ \ path - actual\ MV\ \ path)^2 \ \ldots\ldots 4.2$$

In the 6 by 6 controller example of section 3.2, a move suppression of 17 will cause the manipulated variable to follow a first order path to its steady state value. Many simulations have consistently shown that move suppression values must be increased to go from behaviour that shows overshoot to first order behaviour.

4.1.3. Characterisation of acceptable controller behaviour – first order controlled variable movement

Another possibility of characterising controller behaviour is to attempt to pick move suppressions that will cause controlled variables to follow a first order response to their steady state targets. This is usually a quite desirable way for a control system to behave and should be a reasonable measurement of successful tuning. This is opposed to previous attempts to use some measure of overshoot or overshoot and decay in the controlled variable response as this will lead to a very aggressively tuned controller that will misbehave when there is model error or non-linearity present.

However, factors like deadtime or inverse response in the controlled variable response make first order controlled variable movement a difficult goal to attain. Simply calculating the squared error between a typical first order response and the actual controlled variable movement will fail if deadtime or inverse response exists.

For this reason it was decided to characterise the controlled variable response only by firstly choosing at which prediction interval the controlled variable should be at 75% of the steady state value (PI_{75}). The squared error between the actual value at PI_{75} and 75% of the setpoint value and the squared error between the actual steady state value and the setpoint is added to calculate the metric that will characterise the success of the tuning to produce first order movement in the controlled variable.

$$First\ order\ CV\ error = (CVactual_{PI75} - 75\% * SP)^2 + (CVactual_{SS} - SP)^2 \quad \ldots\ldots 4.3$$

where:

First order CV error	=	metric to characterise deviation of controlled variable from first order response.
$CVactual_{PI75}$	=	Actual controlled variable value at PI_{75}
SP	=	Setpoint of controlled variable
$CVactual_{SS}$	=	Actual controlled variable value at steady state

In widely used DMC algorithms the prediction horizon is 1.5 times the time to steady state chosen for the controller model. If the manipulated variable under consideration tends to have models that reach steady state in a shorter time than the chosen controller time to steady state, it will be impossible to use tuning to let the controlled variables reach their steady state values over the full control horizon. For this reason a move suppression value of 50 for the 6 by 6 controller example was unable to slow down the controlled variable response enough to attain the goal of moving along a first order path to steady state. Simulations have consistently shown that larger move suppressions are required to cause a first order controlled variable response than those required to cause a first order manipulated variable move plan.

4.2. Description of method developed

Once a decision regarding which metric to use was made, a search algorithm can be used to find the multiple tuning constants that will provide the desired behaviour in the control action.

A Visual Basic macro was written in Microsoft Excel that will read in an existing DMC controller configuration file and the model file. The user can then choose which metric of tuning behaviour must be used.

Measuring manipulated variable overshoot is simply done as described in equation 4.1. The square of the difference between the desired overshoot and the actual overshoot is taken as the metric error. In appendix B the measurement and definition of manipulated variable overshoot is explained and expanded to manipulated variables that do not move as shown in figure 4.1.

If the desired behaviour is a first order manipulated variable path, a first order move from zero to the manipulated variable steady state value is calculated. The squared difference over all control intervals is then taken as the metric error.

Because the model deadtime or inverse response plays a large role in controlled variable response, the difference between actual controlled variable response and a first order response to the controlled variable target value does not provide a meaningful measure of success. For this reason the user is given the option to determine at which time interval the controlled variable must have risen to 75% of the steady state value. The metric error was then defined as the squared difference in the controlled variable response at the user defined time and 75% of the steady state value. To this was added the squared difference between the controlled variable at steady state, and the steady state target of the controlled variable.

For the purpose of this study, the Nelder-Mead or downhill simplex method (Nelder and Mead, 1965) was used as search algorithm. Once the metric error has been chosen and measured, Nelder-Mead is used to minimise the sum of the metric error over all manipulated and/or controlled variables by adjusting the move suppression, and optionally the move suppression multipliers.

4.2.1. Implementing Nelder-Mead using an Excel spreadsheet

The Excel spreadsheet contains multiple worksheets:

- Main
- Model
- MV detail
- CV Detail
- AMatrix
- Conditions
- MVMoves
- CVResponse
- ErrorMatrix
- SS
- Results
- Notes
- NelderSetup
- Nelder

Main
The different macro modules can be started from this worksheet. There are five different command buttons that can be selected to run the different modules. These will be described in detail in the next section.

Model
The DMC model file is read into this worksheet. It is displayed in matrix form. This is done mostly for troubleshooting and to visually inspect the model matrix.

MV detail
All pertinent manipulated variable details are stored in here when the DMC configuration file is read. These include:

- move suppressions
- typical manipulated variable move sizes
- move suppression multipliers
- steady state costs

These values can be edited before the calculation modules are run.

CV Detail
The high and low equal concern errors are read in from the controller configuration page and displayed here.

AMatrix
During the DMC calculation, the A matrix is constructed and displayed here. This is done mostly for troubleshooting purpose.

Conditions
During macro execution, the user must specify under which conditions the program must run. This user input is stored here. The residual error that will be minimised to search for the optimal tuning values is based on these input values and is also calculated in this worksheet.

MVMoves
The manipulated variable moves calculated by the DMC calculation will be stored in this worksheet. Several metrics that are used to characterise the manipulated variable move plan are also calculated here. The manipulated variable move plan can also be represented and inspected graphically.

CVResponse
The controlled variable response calculated by the DMC calculation is stored in this worksheet. A metric is calculated to define the rise time of all the controlled variables. The controlled variable response can also be represented and inspected graphically.

ErrorMatrix
The e matrix as used by the DMC calculation is stored here.

SS
A small part of the DMC steady state optimisation module is duplicated here. The end values for all manipulated variables are calculated based on the steady state costs and the controlled variable error. This is required to enforce the steady state end values on the manipulated variables and controlled variables in calculating the manipulated variable move plan.

Results
This worksheet is used to store different tables when a macro module is used to find different metric errors over a wide range of move suppression values.

Notes
This worksheet was used to keep developers notes.

NelderSetup
This worksheet is used when a macro module finds the minimum metric error while stepping through a range of move suppression values. This minimum may be used as a starting position for the search algorithm later.

Nelder
The Nelder-Mead search algorithm uses this worksheet to store its output.

4.2.2. Visual Basic macro

The macro execution is done using common building blocks that are used by the different modules. The modules are:

- Read ccf
- Run Once
- Run Nelder-Mead
- Get SupMov Table
- Get Starting Values

Read ccf

In this module, Excel will open a DMC controller configuration file to obtain:

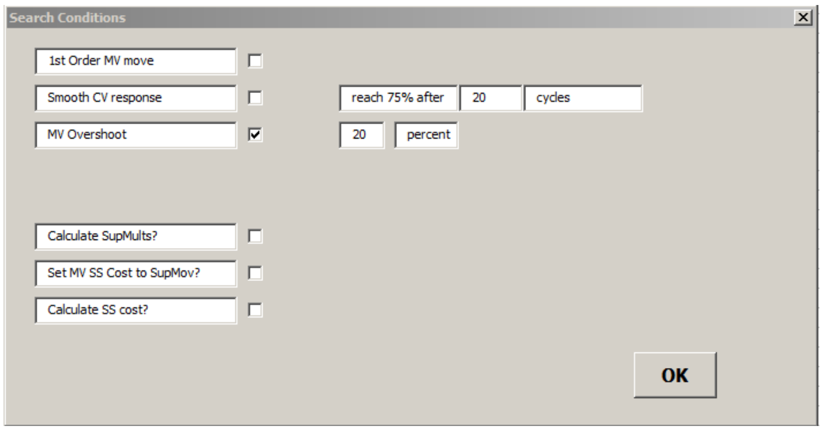
- The number of manipulated variables
- The number of controlled variables
- The existing tuning values
    - Manipulated variable move suppression
    - Manipulated variable move suppression multiplier
    - Manipulated variable steady state cost
    - Manipulated variable typical move size
    - Controlled variable equal concern errors
- Number of model coefficients
- Name of the model file

These values are stored in the different worksheets as described above.

It will then open the corresponding model file and read the models into a worksheet.

Run Once

In this module Excel will read the parameters that were loaded from the files or edited in Excel into Visual Basic variables. An input window will be displayed as shown in figure 4.2. In the top half, it will then ask the user which metric for the characterisation of acceptable behaviour must be used, and what amount of overshoot would be required. In the bottom half, the user may specify which tuning parameters may be adjusted to attempt to decrease the metric error.


Search Conditions

1st Order MV move ☐

Smooth CV response ☐ reach 75% after 20 cycles

MV Overshoot ☑ 20 percent

Calculate SupMults? ☐

Set MV SS Cost to SupMov? ☐

Calculate SS cost? ☐

OK

Figure 4.2 Excel macro input

The user is also able to specify if the search algorithm must:

- search for optimal values of the move suppression multipliers
- set the steady state cost to be the same value as the move suppression
- search for the optimal values of the steady state cost.

Even though these options are chosen, only the option to set the steady state cost to the same value as the move suppressions will influence the program execution in Run Once mode. The other options are only used when the solver is run.

The macro will then run the DMC calculations (equation 2.5) to find the manipulated variable move plan and the corresponding controlled variable responses. It will first determine the steady state values for all manipulated variables and controlled variables, based on the steady state costs. It will then calculate the manipulated variable move plan based on the move suppressions and move suppression multipliers.

The metrics and metric error will be calculated in the worksheets based on the response chosen.

Run Nelder-Mead

In this module Excel would read the parameters that were loaded from the files or edited in Excel into Visual Basic variables. It would then ask for the definition of behaviour that must be measured, and what amount of overshoot would be required as shown in figure 4.2.

The user is also able to specify if the search algorithm must:

- search for optimal values of the move suppression multipliers
- set the steady state cost to be the same value as the move suppression
- search for the optimal values of the steady state cost.

Based on which variables are selected to be optimised, the macro will then use the current values for move suppression, move suppression multipliers and steady state cost as starting values for the Nelder-Mead search algorithm. It will run the DMC calculations to find the manipulated variable move plan and the corresponding controlled variable responses. It will calculate the error based on the response chosen and pass the error to Nelder-Mead, which will search for the minimum error. Once Nelder-Mead has converged, it stops and places the optimised variables in the correct worksheet.

Get SupMov table

This module was developed to obtain multiple error values over a wide range of move suppressions. It will step through different move suppression values, run the DMC calculation, calculate the associated error and place it in a table. The table can be used to visually inspect the search plane that Nelder-Mead must optimise on.

Get Starting Values

If a Nelder-Mead search is started far away from the optimum point, it will take very long to converge. This module was written to step through multiple move suppressions, calculate the associated error and find a good starting position for

Nelder-Mead. It will then place the move suppressions that provide the smallest error value into the correct worksheet.

This approach is especially necessary if manipulated variable overshoot is desired. If the search is started where the overshoot is zero, the search algorithm will fail because there will be no change in error providing a flat search plane with no improvement in any direction.

4.3. Current limitations of software

As the software was developed to illustrate a concept, it was decided to not cater for all possible variations of controllers. Therefore certain software limitations were deemed acceptable.

4.3.1. Integrators

Currently the DMC calculation in the software is not capable of handling integrating models. This is definitely a suggested enhancement that will be considered in future.

4.3.2. Model coefficients

The current software will only accept models with 30 model coefficients and 8 control intervals, calculated at intervals 1,2,3,4,5,7,11 and 14.

5. Analysis of performance of method

5.1. *Comparison of different definitions of optimal tuning*

5.1.1. Controller performance with no model error

To compare the performance of the different tuning constants obtained by using the definitions of acceptable control, two simulated plants and controllers were used. Details regarding the processes, a distillation plant and reactor, are shown in Appendix C. The different metrics for acceptable DMC tuning as described in section 4.1 were applied and the program shown in Appendix A found move suppression values that will cause the behaviour as defined by the metrics.

The different move suppression values were loaded in the online controller and controller setpoints were changed. Data was collected on all controller variables to compare the performance. To demonstrate the change in manipulated variable move plans and controlled variable responses, one manipulated variable and one controlled variable of the reactor controller are shown in figures 5.1 and 5.2 and a manipulated variable and controlled variable of the distillation process are shown in figures 5.3 and 5.4.

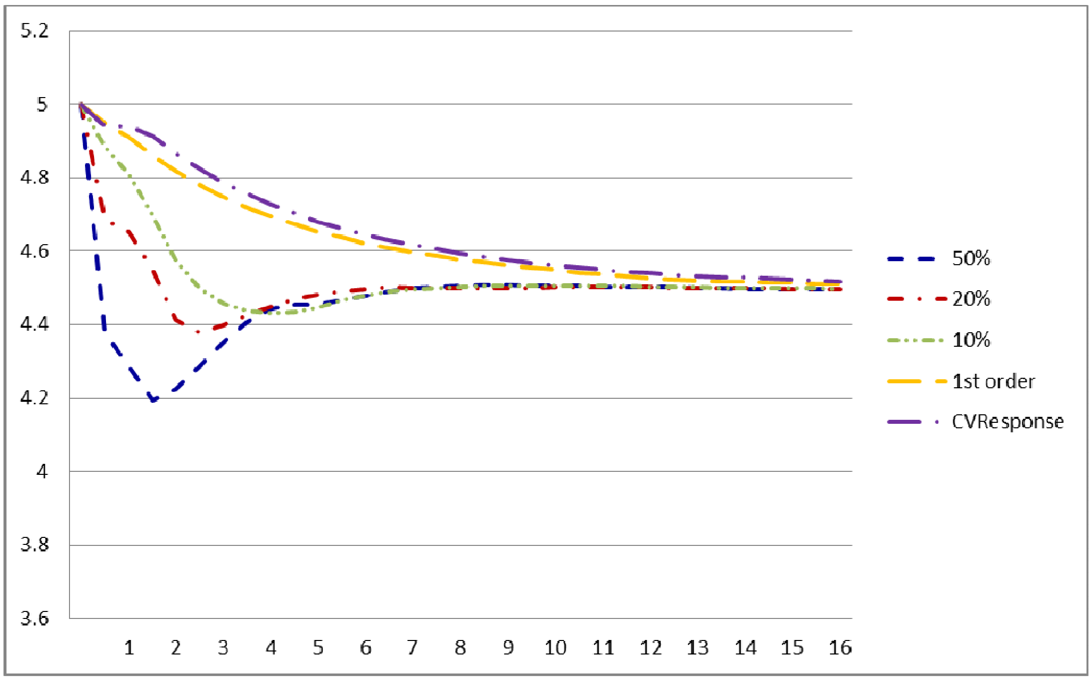


Figure 5.1 Reactor MV1 movement without model error

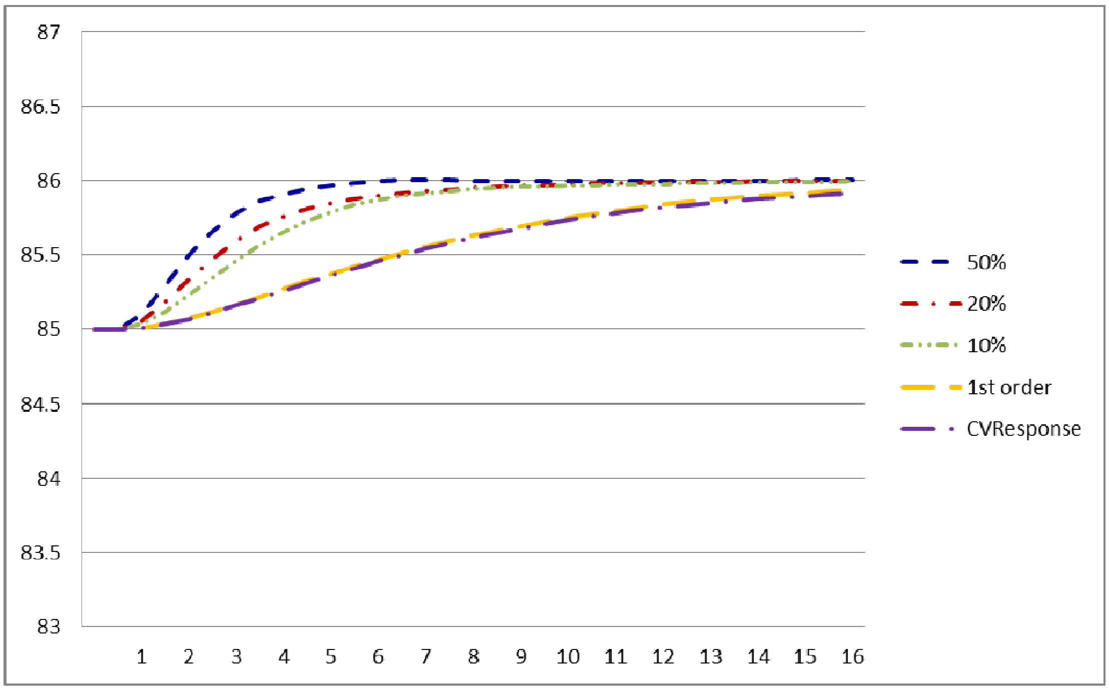


Figure 5.2 Reactor CV1 movement without model error

Figure 5.1 shows how the manipulated variables of the reactor controller move to their steady state value faster when the more aggressive tuning metrics like 50% overshoot are used. The controlled variables also move to setpoint faster as seen in figure 5.2. Using a first order move plan for the manipulated variables as metric and using a first order response path for the controlled variables provide much slower tuning as shown.

In figure 5.3 and 5.4 the effect of the ill-conditioned controller matrix can be seen, especially with the more aggressive tuning parameters. Because all controlled variables are controlled to a setpoint, slight numerical differences between the controller prediction and the simulation response cause the cycle.

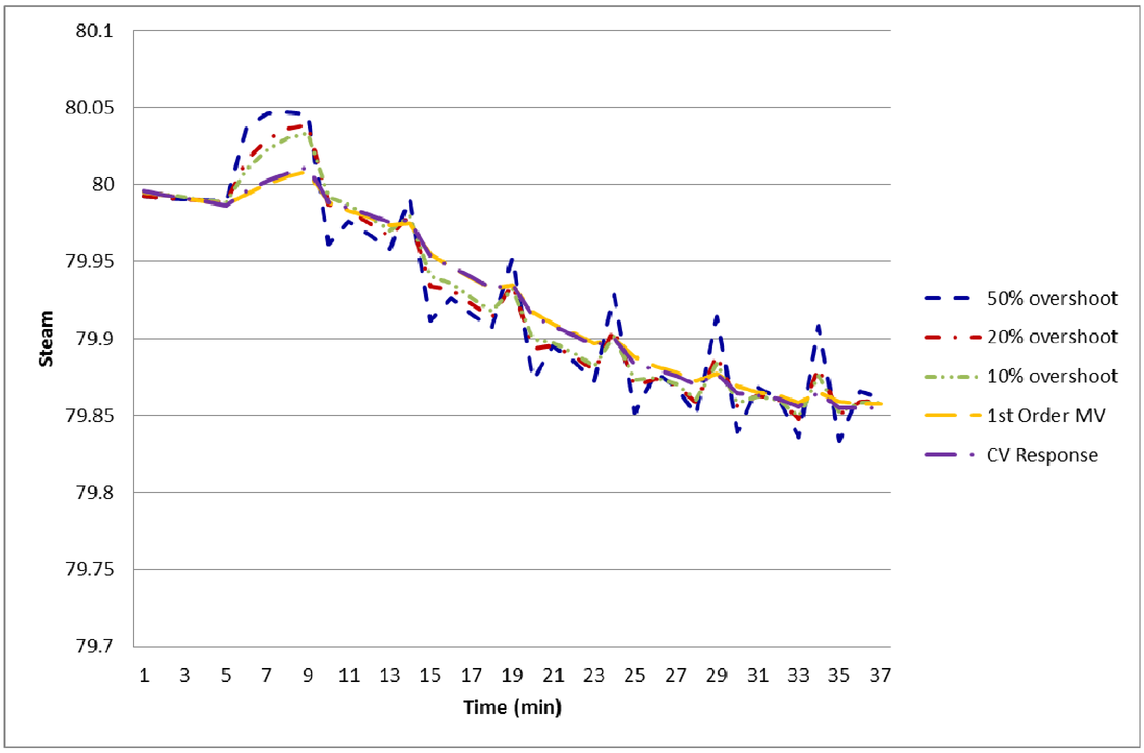


Figure 5.3 Distillation plant MV1 movement with no model error

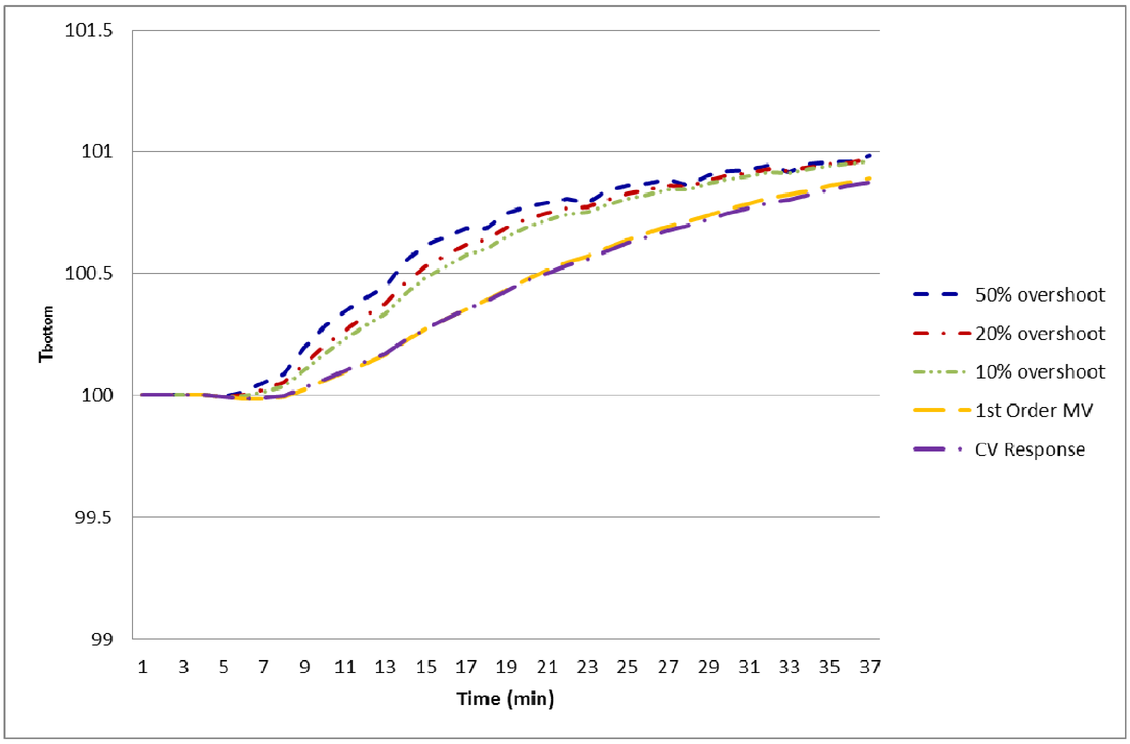


Figure 5.4 Distillation plant CV1 response with no model error

5.1.2. Controller performance with model error

Model error was created by changing the model gains on the simulations. If the plant or simulator response is smaller than the prediction created by the controller models, this will lead to a sluggish controller. While this is not optimal, it is much less destructive than when the plant or simulator response is much larger than the prediction. This will lead to the controller cycling, especially if aggressive move suppressions were used.

For this reason all gains on the reactor simulator were increased by 100% to simulate and test the controller response with different tuning constants under worst case conditions. These responses are demonstrated by showing the movement of the first manipulated variable and controlled variable of the reactor in figures 5.5 and 5.6.

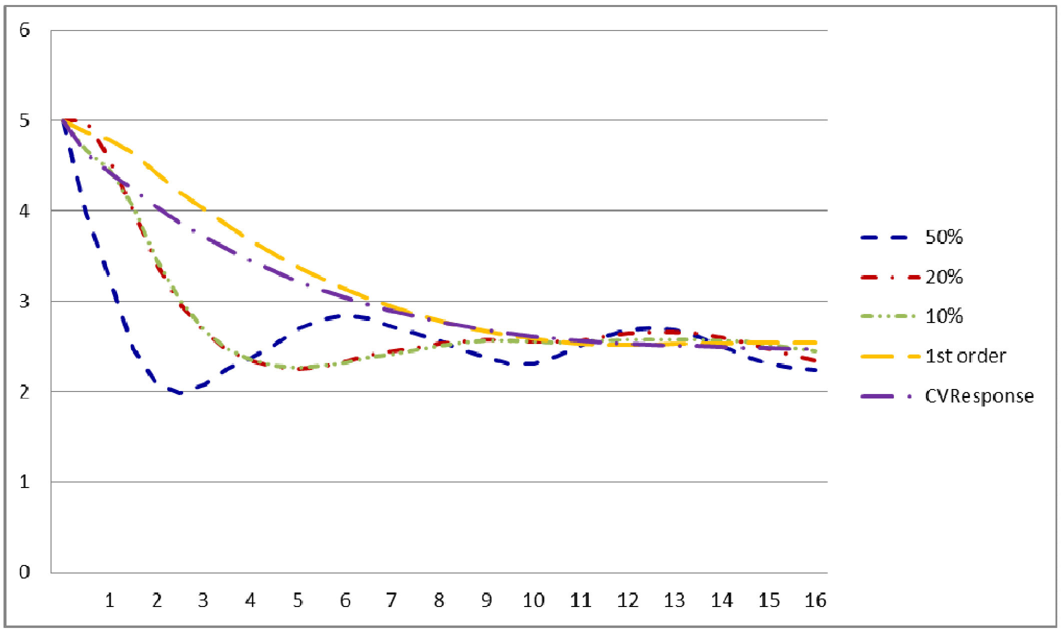


Figure 5.5 Reactor MV1 movement with model error

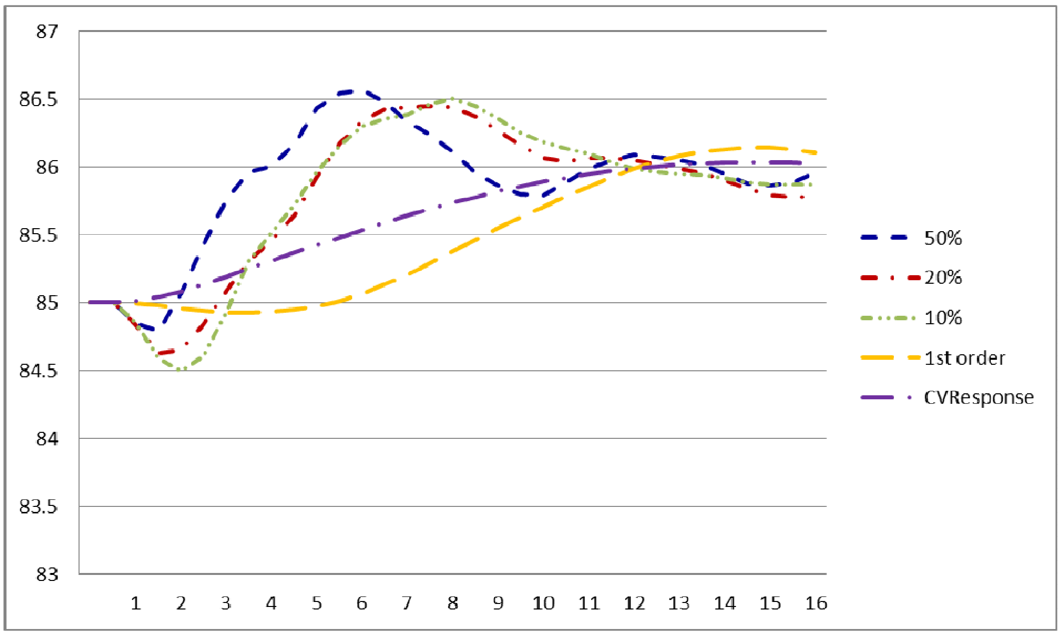


Figure 5.6 Reactor CV1 movement with model error

The data clearly shows that the more aggressive tuning has much more of a tendency to start a cycle on the process, with the less aggressive tuning not cycling at all. What is interesting to note is that the absolute movement over the control horizon of the manipulated variable also changes when model error is introduced. This is because the controller starts compensating for the model error by also moving the other manipulated variables, causing all manipulated variables to move away from the steady state values initially predicted.

Even with no model error, the distillation controller already had performance issues caused by ill-conditioning as shown in section 5.1.1. All gains for the bottoms temperature on the distillation simulator were increased by 100%. This further hampered controller performance as shown in figures 5.7 and 5.8.

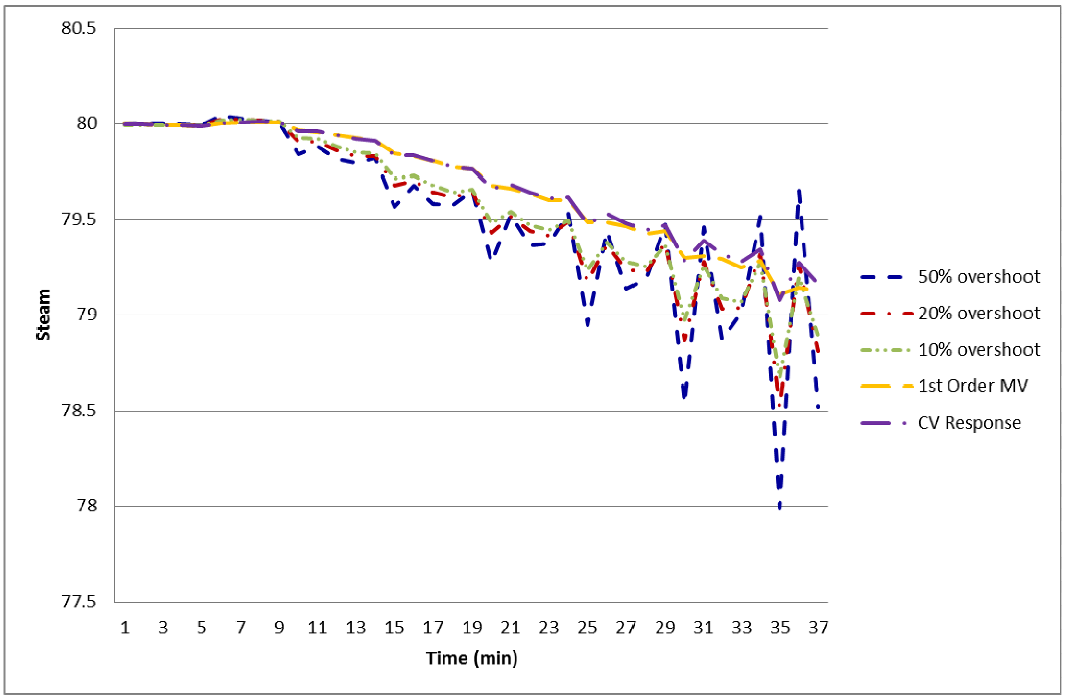


Figure 5.7 Distillation plant MV1 movement with model error

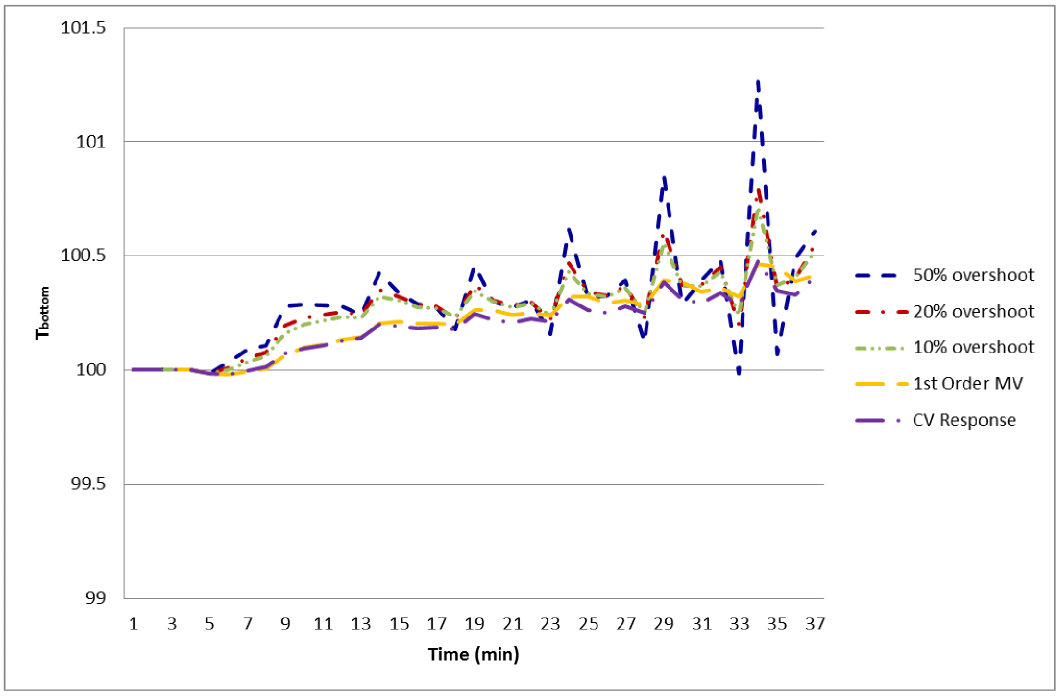


Figure 5.8 Distillation plant CV1 response with model error

Once again the slower tuning constants showed less of a tendency to cause unstable behaviour.

5.2. Analysis of search plane

The success of this method rests very strongly on the assumption that a global minimum will exist for varying move suppressions for each of the metrics. To determine if this holds, an array of metric errors was calculated for various move suppression values. In order to visually display the search plane, the reactor plant from section 5.1 was used and only the first two move suppressions were varied over a range. The other move suppression values were held at the value which would provide the lowest error for each metric.

5.2.1. Overshoot metric

In figure 5.9 it can be seen that the 20% overshoot metric does show a global minimum if move suppressions 1 and 2 are varied. The same result is found if any 2 other move suppressions are used, or if a 50% or 10% overshoot is used as target for the search algorithm.

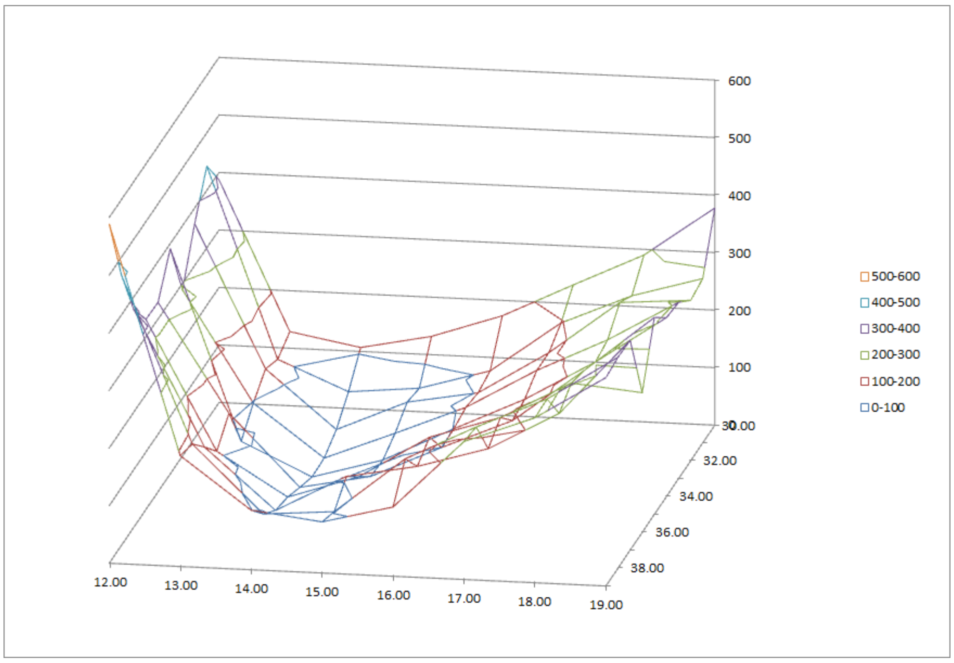


Figure 5.9 Residual error from 20% overshoot

5.2.2. First order manipulated variable move plan

If the same is done with the first order manipulated variable move plan metric, the result shown in figure 5.10 demonstrates that this metric also has a global minimum.

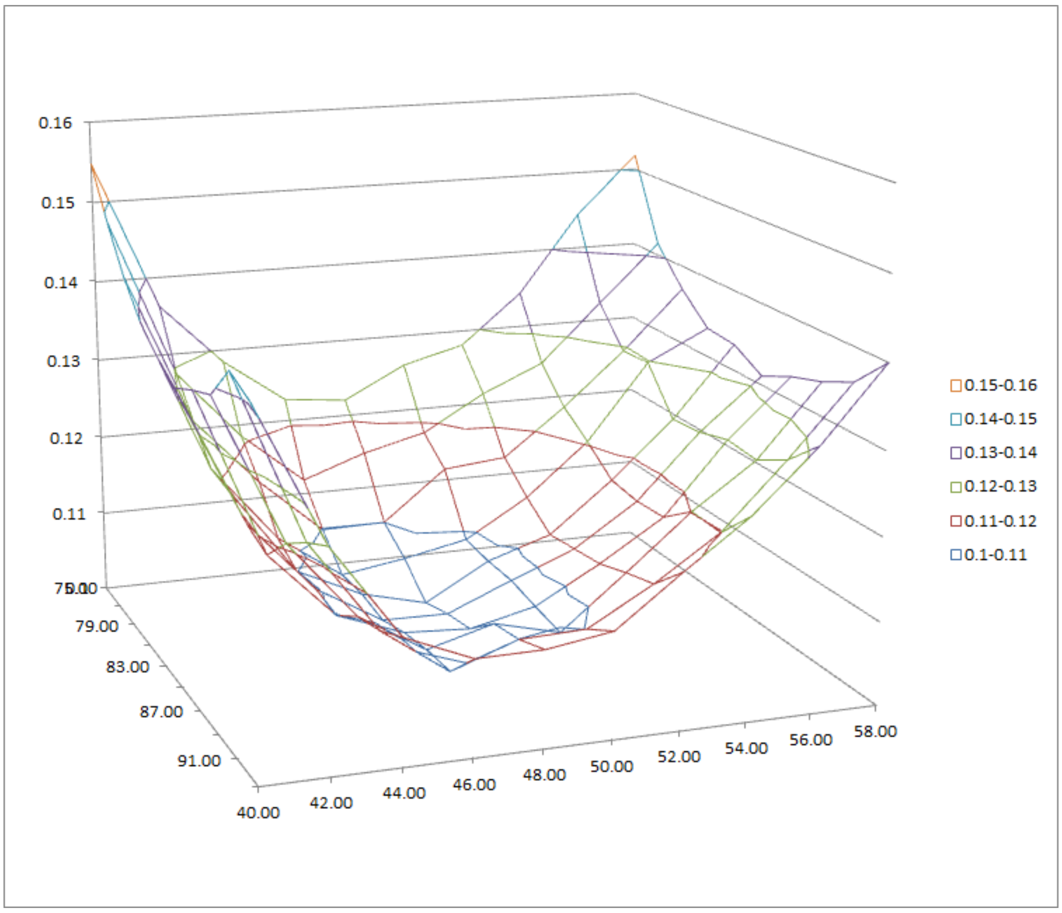


Figure 5.10 Residual error from first order manipulated variable movement

5.2.3. Smooth controlled variable response

The search plane for the smooth controlled variable response showed a problem. The error rapidly decreases as the move suppression values increase, causing the controlled variables to move slower towards the steady state values. At a certain stage, the error does not decrease significantly, but it also does not start increasing again. This behaviour is caused by the imposition of the steady state values on the manipulated variable move plan. Because the steady state values are enforced, the controlled variable movement cannot go much slower than a first order path towards the steady state value, even if very large move suppression values are chosen.

Once the search plane levels off, figure 5.11 shows that small local minima and maxima form. The search algorithm will then find one of these local minima, and will not find a repeatable solution as the starting values will determine which local minimum will be found. For this reason it is suggested that the smooth controlled variable response not be used.

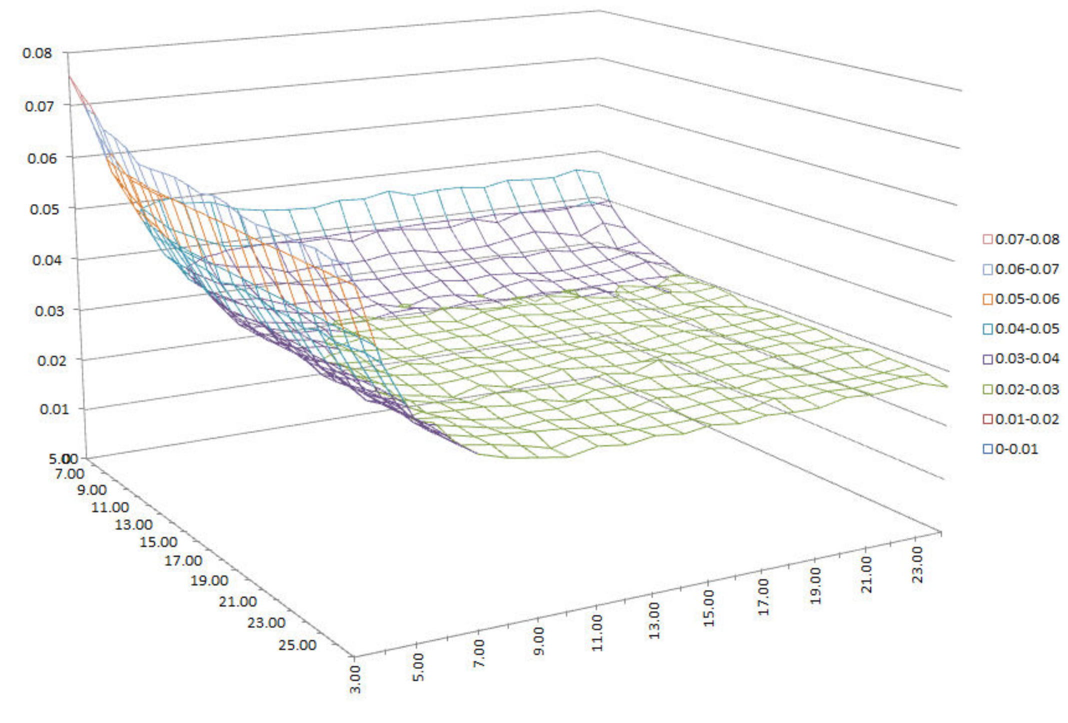


Figure 5.11 Residual error from smooth controlled variable response

5.3. Comparison with traditional tuning

As stated in section 2.4.1, several APC engineers in industry were asked to tune 2 simple plants in simulation mode in order to compare the tuning metric results with traditional tuning. Details regarding the processes are shown in Appendix C. They all followed the traditional trial and error approach. No guidance was given regarding acceptable tuning, they had to use their own discretion and experience. The experience of the engineers varied from 2 to more than 10 years in APC as shown in table 5.1

Table 5.1 Experience of participating engineers

	Years APC Experience	Number of controllers implemented	Largest number of manipulated variables
Engineer 1	10+	25	25+
Engineer 2	2-5	25+	25+
Engineer 3	2-5	10-25	10-25
Engineer 4	5-10	10-25	10-25
Engineer 5	2-5	5-10	10-25
Engineer 6	2-5	5-10	10-25
Engineer 7	5-10	10-25	10-25
Engineer 8	2-5	20	10-25

The initial tuning values that the engineers decided on using trial and error as shown in section 2.4.1 were compared with the results obtained from applying the different metrics and using Nelder Mead to minimise metric error as described in section 4.1. The different results are shown in table 5.2 and 5.3.

Table 5.2 Results of different tuning methods for Distillation plant

	50% MV Overshoot	20% MV Overshoot	10% MV Overshoot	1st Order MV Response	Smooth CV Response	Engineer 1	Engineer 2	Engineer 3	Engineer 4	Engineer 5	Engineer 6	Engineer 7	Engineer 8
SupMov1	1.2	2.5	3.4	17.6	21.6	10.0	10.0	7.0	100	3	0.02	1	1
SupMov2	9.2	14.4	16.7	442	34.8	1.0	5.0	8.0	5	10	0.25	0.1	1
SupMov3	7.9	14.2	18.4	58.2	51.4	5.0	5.0	5.0	5	20	0.25	0.1	5

Table 5.3 Results of different tuning methods for Reactor

	50% MV Overshoot	20% MV Overshoot	10% MV Overshoot	1st Order MV Response	Smooth CV Response	Engineer 1	Engineer 2	Engineer 3	Engineer 4	Engineer 5	Engineer 6	Engineer 7	Engineer 8
SupMov1	1.8	3.0	3.8	57	115	6.0	2.5	5.0	3	5	6	4	8.5
SupMov2	9.6	11.4	12.8	100	39.9	12.0	5.0	6.0	3	12	12	1	14
SupMov3	11	13.5	15.6	76	7.2	15.0	5.0	4.0	3	12	12	1	15
SupMov4	6.6	8.8	9.7	404	69.0	5.0	2.0	3.0	3	6	6	1	11
SupMov5	1.0	1.8	1.9	9.8	21.8	1.0	2.0	5.0	9	7	7	1	20

The variation between the different tuning metrics and the trial and error approach is shown clearly in graphical format in figures 5.1 and 5.2.

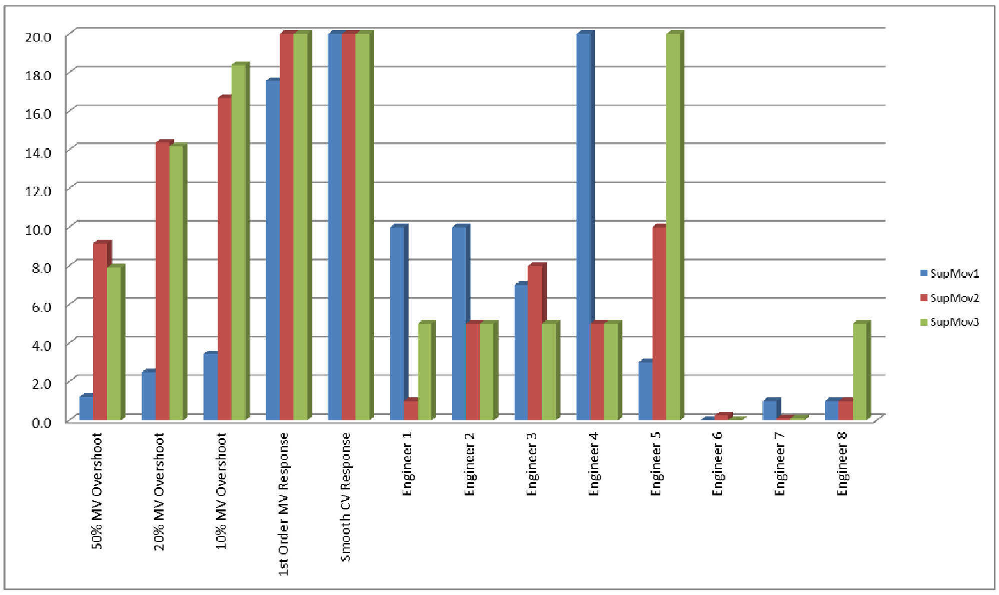


Figure 5.1 Tuning constants for distillation plant

On both graphs the y axis was stopped at 20 as the first order and especially the smooth controlled variable response metrics tend to provide ridiculously large move suppression values.

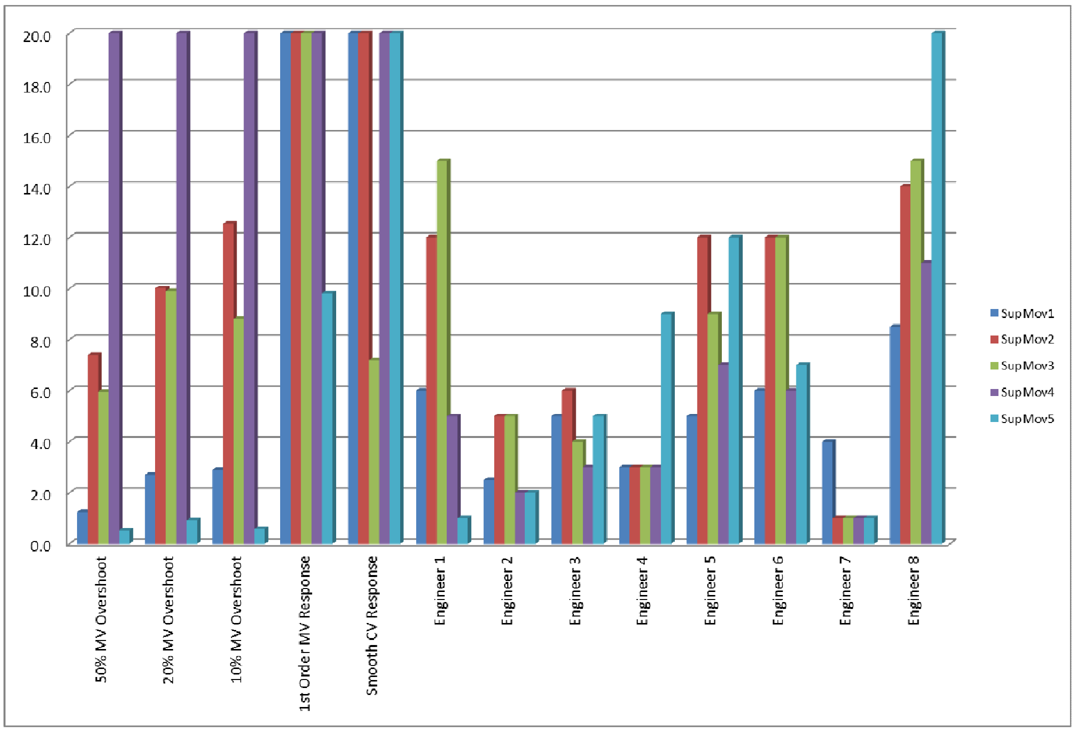


Figure 5.2 Tuning constants for reactor

As expected, the tuning method results show less aggressive behaviour as the amount of manipulated variable overshoot is decreased. Tuning for a first order movement in the manipulated variables or a smooth controlled variable response yields much larger move suppressions.

6. Conclusion

Historically the tuning of multi-variable or dynamic matrix controllers have been a matter of personal taste of the engineer and trial an error methods were used to tune controllers that influence multi-million dollar processes. The same goes for controllers that have an impact on environmental and safety issues. The problem was compounded by the lack of agreement of what acceptable controller behaviour is, with the level of aggressiveness of controller tuning depending on the judgement of the engineer. It also meant that comparing different tuning constants was vague and unscientific.

Using the shape of the predicted manipulated variable move plan or the shape of the controlled variable response as an indication of tuning aggressiveness addresses this problem.

It was found that using the controlled variable response may lead to convergence issues with the solver algorithm. Enforcing the steady state values on all manipulated variables will cause the controlled variable movement to follow a path that will be very close to a first order, regardless of larger move suppressions. This leads to the search plane levelling off at large move suppressions, instead of the metric error increasing.

Using the amount of manipulated variable overshoot as an indication of tuning aggressiveness provides more satisfying results. The amount of overshoot can vary from zero if a first order path is chosen, to any positive value chosen by the engineer. Using a solver algorithm to find the desired amount of overshoot in a manipulated variable can be used to find good starting values for tuning parameters.

Using the amount of manipulated variable overshoot is a valid way to characterise aggressiveness of tuning. By using it as a metric it is possible to compare different tuning values.

Even though the different metrics that have been introduced have the potential to enable APC practitioners to compare the aggressiveness of tuning parameters, this will by no means close the book on the subject. Other major considerations when choosing move suppressions are:

- Conditioning number of the controller matrix
- Quality of process models
- Non-linear process responses
- Severity of disturbances
- Consequences of controlled variable limit violation.
- The number of controller execution cycles from the time when a disturbance manifests and when the effect of the disturbance causes unacceptable behaviour in the controlled variables.

Taking these factors into consideration, it is recommended that the APC engineer obtains move suppression values that will provide 20% manipulated variable overshoot and tuning that will provide a first order move plan for the manipulated variables. Interpolation between these values can be done to use 20% manipulated variable overshoot for aggressive tuning, and moving towards the first order manipulated variable moves if less aggressive tuning is required if issues like ill-conditioning or model uncertainty exist. As per best practises, these tuning values will have to be refined by observing the actual controller response during the project commissioning phase.

The value in using manipulated variable overshoot is not that it will provide ideal tuning values for a live controller. It will firstly provide a way to characterise aggression of tuning, and a way to compare different sets of tuning values. Secondly it will provide a good set of initial tuning values for online controllers that will save the engineer time and process upsets during commissioning.

Appendix A - Excel Macro

Module 1

```
Option Base 1
'  DMC Tuner by GZ Gous 2008

Option Explicit
Global ModelFile As String
Global CCFile As String
Global Coefficient() As Double
Global dGain() As Double
Global IndependentTag() As String
Global DependentTag() As String
Global NumberOfIndependents As Integer
Global NumberOfMVs As Integer
Global NumberOfDependents As Integer
Global NumberOfCoefficients As Integer
Global SteadyStateTime As Double
Global NumberOfMoves As Integer
Global RampStatus()
Global SupMov() As Single
Global SupMlt() As Single
Global Cst() As Single
Global dMV() As Single
Global TypMov() As Single
Global CVError() As Single
Global ErrorMatrix() As Single
Global HiECE() As Single
Global LoECE() As Single
Global AMatrix() As Single
Global FileRead As Boolean
Global ATAIAT() As Single
Global fmovt() As Integer
Global EndWeight As Single
Global MVMoves() As Single
Global SolveSupMults As Boolean
Global DoOnce As Boolean
Global CVPath() As Single
Global MovTgt() As Single
Global SolveMV1st As Single
Global SolveRatio As Single
Global SolveCVSmooth As Single
Global SolveMVOvershoot As Single
Global MVOvershoot As Single
Global CVTime As Single
Global UseMV() As Single
Global UseCV() As Single
Global SetCstToSupMov As Boolean

Sub Master()
    ReadCCF
    ReadModelFile
    DisplayModelMatrix
End Sub
Sub GetSS()

Dim SolvTarget As String
Dim SolvChange As String

'SetCstToSupMov = True
If SetCstToSupMov Then
    For i = 1 To NumberOfIndependents
        Cst(i) = SupMov(i)
        Worksheets("MV Detail").Cells(5 + i, 5) = Cst(i)
    Next i
Else
    For i = 1 To NumberOfIndependents
        Cst(i) = Worksheets("MV Detail").Cells(5 + i, 5)
    Next i
End If
Worksheets("SS").Select
Range("A1:Z300").ClearContents
Cells(1, 1) = "CV"
Cells(1, 2) = "MV"
Cells(1, 3) = "Gain"
Cells(1, 4) = "dMV"
Cells(1, 5) = "dCV"
Cells(1, 6) = "dCVtot"
Cells(1, 7) = "Target"
Cells(1, 8) = "Err^2"
For i = 1 To NumberOfDependents
    Cells((i - 1) * NumberOfIndependents + 2, 1) = DependentTag(i)
    For j = 1 To NumberOfIndependents
        Cells((i - 1) * NumberOfIndependents + j + 1, 2) = IndependentTag(j)
        Cells((i - 1) * NumberOfIndependents + j + 1, 3) = Worksheets("Model").Cells(8 + (i - 1) *
NumberOfIndependents + j, 33)
        Cells((i - 1) * NumberOfIndependents + j + 1, 5).FormulaR1C1 = "=RC[-2]*RC[-1]"
    Next j
    Cells(i * NumberOfIndependents + 1, 6) = "=sum(E" & Format((i - 1) * NumberOfIndependents + 2, "##") & ":E" _
            & Format(i * NumberOfIndependents + 1, "##") & ")"
    Cells(i * NumberOfIndependents + 1, 7) = 1  'target for cv at ss
    Cells(i * NumberOfIndependents + 1, 8).FormulaR1C1 = "=(1000000*(RC[-1]-RC[-2]))^2"
Next i
Cells((i - 1) * NumberOfIndependents + 2, 1) = "Cost"
Cells(i * NumberOfIndependents + 1, 6) = "=sum(E" & Format((i - 1) * NumberOfIndependents + 2, "##") & ":E" _
            & Format(i * NumberOfIndependents + 1, "##") & ")"
Cells(i * NumberOfIndependents + 1, 7) = 0
Cells(i * NumberOfIndependents + 1, 8).FormulaR1C1 = "=(RC[-1]-RC[-2])^2"
For j = 1 To NumberOfIndependents
    Cells(j + 1, 4) = 1
    Cells((i - 1) * NumberOfIndependents + j + 1, 2) = IndependentTag(j)
    Cells((i - 1) * NumberOfIndependents + j + 1, 3) = Cst(j)
    Cells((i - 1) * NumberOfIndependents + j + 1, 5).FormulaR1C1 = "=RC[-2]*abs(RC[-1])"
Next j
Cells(NumberOfIndependents + 2, 4).Select
ActiveCell.Formula = "=D2"
Selection.Copy
For j = 1 To (NumberOfDependents) * NumberOfIndependents - 1
    Cells(ActiveCell.Row + 1, 4).Select
    ActiveSheet.Paste
Next j
Application.CutCopyMode = False
Cells(2, 8).FormulaR1C1 = "=sum(R[1]C:R[200]C)"

SolvTarget = "$H$2"
SolvChange = "$D$2:$D$" & Format(NumberOfIndependents + 1, "##")
SolverReset
SolverOptions MaxTime:=1000, Iterations:=5000, Precision:=0.000001, _
    AssumeLinear:=False, StepThru:=False, Estimates:=2, Derivatives:=1, _
    SearchOption:=1, IntTolerance:=5, Scaling:=False, Convergence:=0.0001, _
    AssumeNonNeg:=False
SolverOk SetCell:=SolvTarget, MaxMinVal:=2, ValueOf:="0", ByChange:=SolvChange
SolverSolve (True)
ReDim dMV(NumberOfIndependents)
For j = 1 To NumberOfIndependents
    dMV(j) = Cells(j + 1, 4)
Next j
End Sub

Sub ReadCCF()
    Dim iFile As Integer           ' Model file handle
    Dim sBuf As String              ' Temporary buffers
    Dim i As Integer, _
        j As Integer, _
        k As Integer, _
        p As Integer, _
        dependent As Integer, _
        independent As Integer
    Dim IsFF As Single


'Worksheets("MV Detail").Cells(1.1) = 0
    ' Get the name of the Model file
  CCFile = Application.GetOpenFilename("CC Files (*.ccf), *.ccf")

    ' Load the file... only if the user selected one
  If (ModelFile <> "False") Then

        ' Read & Store the model file header
```

```
 ' Open the model file
iFile = FreeFile()
Open CCFile For Input As #iFile


Do
        Line Input #iFile, sBuf
Loop Until Left(sBuf, 7) = ".IPMIND"
sBuf = Mid(sBuf, InStrRev(sBuf, "~~~", Len(sBuf) - 3) + 3, 20)
NumberOfIndependents = Val(Left(sBuf, Len(sBuf) - 3))
Do
        Line Input #iFile, sBuf
Loop Until Left(sBuf, 7) = ".IPNDEP"
sBuf = Mid(sBuf, InStrRev(sBuf, "~~~", Len(sBuf) - 3) + 3, 20)
NumberOfDependents = Val(Left(sBuf, Len(sBuf) - 3))
Do
        Line Input #iFile, sBuf
Loop Until Left(sBuf, 7) = ".IPXNCI"
sBuf = Mid(sBuf, InStrRev(sBuf, "~~~", Len(sBuf) - 3) + 3, 20)
NumberOfCoefficients = Val(Left(sBuf, Len(sBuf) - 3))
 Do
        Line Input #iFile, sBuf
Loop Until Left(sBuf, 7) = ".MDLNAM"
sBuf = Mid(sBuf, InStrRev(sBuf, "~~~", Len(sBuf) - 3) + 3, 20)
ModelFile = Left(sBuf, Len(sBuf) - 3)

ReDim SupMov(NumberOfIndependents)
ReDim SupMlt(NumberOfIndependents)
ReDim Cst(NumberOfIndependents)
ReDim TypMov(NumberOfIndependents)
ReDim HiECE(NumberOfDependents)
ReDim LoECE(NumberOfDependents)
ReDim IndependentTag(NumberOfIndependents)
ReDim DependentTag(NumberOfDependents)
ReDim Coefficient(NumberOfIndependents, NumberOfDependents, NumberOfCoefficients)
ReDim RampStatus(NumberOfDependents)
ReDim dGain(NumberOfIndependents, NumberOfDependents)



Worksheets("MV Detail").Select
Worksheets("MV Detail").Range("a1", "IV65536").Select
Selection.ClearContents
Selection.Interior.ColorIndex = xlNone
Worksheets("MV Detail").Range("a1").Select
Worksheets("MV Detail").Range("a5") = "MV Name"
Worksheets("MV Detail").Range("b5") = "SUPMOV"
Worksheets("MV Detail").Range("c5") = "TYPMOV"
Worksheets("MV Detail").Range("d5") = "SUPMLT"
Worksheets("MV Detail").Range("e5") = "SS Cost"

For independent = 1 To NumberOfIndependents
    SupMlt(independent) = 2
     Do
        Line Input #iFile, sBuf
    Loop Until Left(sBuf, 4) = ".CST"
        sBuf = Mid(sBuf, InStrRev(sBuf, "~~~", Len(sBuf) - 3) + 3, 20)
        Cst(independent) = Val(Left(sBuf, Len(sBuf) - 3))
    Do
        Line Input #iFile, sBuf
    Loop Until Left(sBuf, 5) = ".ISFF"
    IsFF = 1 - Val(Mid(sBuf, 25, 1))
    If IsFF = 1 Then
        Do
            Line Input #iFile, sBuf
            If Left(sBuf, 7) = ".SUPMLT" Then
                sBuf = Mid(sBuf, InStrRev(sBuf, "~~~", Len(sBuf) - 3) + 3, 20)
                SupMlt(independent) = Val(Left(sBuf, Len(sBuf) - 3))
            End If
        Loop Until Left(sBuf, 7) = ".SUPMOV"
            sBuf = Mid(sBuf, InStrRev(sBuf, "~~~", Len(sBuf) - 3) + 3, 20)
            IsFF = Val(Left(sBuf, Len(sBuf) - 3))
        Do
            Line Input #iFile, sBuf
```

```
            Loop Until Left(sBuf, 7) = ".TYPMOV"
              sBuf = Mid(sBuf, InStrRev(sBuf, "~~~", Len(sBuf) - 3) + 3, 20)
              TypMov(independent) = Val(Left(sBuf, Len(sBuf) - 3))
          End If
          Worksheets("MV Detail").Range("a5").Offset(independent, 1) = IsFF
          Worksheets("MV Detail").Range("a5").Offset(independent, 2) = TypMov(independent)
          Worksheets("MV Detail").Range("a5").Offset(independent, 3) = SupMlt(independent)
          Worksheets("MV Detail").Range("a5").Offset(independent, 4) = Cst(independent)
          SupMov(independent) = IsFF
      Next independent

      Worksheets("CV Detail").Select
      Worksheets("CV Detail").Range("a1", "IV65536").Select
      Selection.ClearContents
      Selection.Interior.ColorIndex = xlNone
      Worksheets("CV Detail").Range("a1").Select
      Worksheets("CV Detail").Range("a5") = "CV Name"
      Worksheets("CV Detail").Range("b5") = "Hi ECE"
      Worksheets("CV Detail").Range("c5") = "Lo ECE"
          For dependent = 1 To NumberOfDependents
          Do
            Line Input #iFile, sBuf
          Loop Until Left(sBuf, 7) = ".ECECML"
          sBuf = Mid(sBuf, InStrRev(sBuf, "~~~", Len(sBuf) - 3) + 3, 20)
          LoECE(dependent) = Val(Left(sBuf, Len(sBuf) - 3))
          Do
            Line Input #iFile, sBuf
          Loop Until Left(sBuf, 7) = ".ECECMU"
          sBuf = Mid(sBuf, InStrRev(sBuf, "~~~", Len(sBuf) - 3) + 3, 20)
          HiECE(dependent) = Val(Left(sBuf, Len(sBuf) - 3))
          Worksheets("CV Detail").Range("a5").Offset(dependent, 1) = HiECE(dependent)
          Worksheets("CV Detail").Range("a5").Offset(dependent, 2) = LoECE(dependent)
      Next dependent


      Close #iFile
    End If

  End Sub

  Sub ReadModelFile()
      Dim iFile As Integer           ' Model file handle
      Dim bDMCplus As Boolean         ' True = DMCplus

      Dim sComment As String         ' Model Comment
      Dim sTag As String             ' Tag name
      Dim sUnits As String           ' Engineering units

      Dim sBuf As String             ' Temporary buffers
      Dim i As Integer, _
          j As Integer, _
          k As Integer, _
          p As Integer, _
          dependent As Integer, _
          independent As Integer, _
          interval As Integer        ' Loop counters
      Dim holdMVs As Integer
      Dim bNewStyle As Boolean

      holdMVs = NumberOfIndependents

      ' Get the name of the Model file
      'ModelFile = Application.GetOpenFilename("Model Files (*.mdl), *.mdl")

      ' Load the file... only if the user selected one
      If (ModelFile <> "False") Then

          ' Read & Store the model file header

       ' Open the model file
      iFile = FreeFile()
      Open ModelFile For Input As #iFile

      ' Comment
      Line Input #iFile, sBuf
```

```
sComment = Trim(sBuf)

' Number of files and the file names
Line Input #iFile, sBuf
For i = 1 To Val(sBuf)
   Line Input #iFile, sBuf
Next i

' Number of independents
Line Input #iFile, sBuf
NumberOfIndependents = Val(sBuf)

' Number of dependents
Line Input #iFile, sBuf
'NumberOfDependents = Val(sBuf)

' Number of coefficients
Line Input #iFile, sBuf
'NumberOfCoefficients = Val(sBuf)

' Time to steady-state
Line Input #iFile, sBuf
Line Input #iFile, sBuf
SteadyStateTime = Val(sBuf)

' DMCplus model style flag
Line Input #iFile, sBuf
If Val(sBuf) = 9896# Then bNewStyle = True

' Re-dimension global arrays as necessary

ReDim IndependentTag(NumberOfIndependents)
'ReDim DependentTag(NumberOfDependents)
ReDim Coefficient(NumberOfIndependents, NumberOfDependents, NumberOfCoefficients)
'ReDim RampStatus(NumberOfDependents)
ReDim dGain(NumberOfIndependents, NumberOfDependents)

' read tagnames
For independent = 1 To NumberOfIndependents
   Line Input #iFile, sBuf
   IndependentTag(independent) = Trim(Mid$(sBuf, 37, 12))
Next independent

For dependent = 1 To NumberOfDependents
   Line Input #iFile, sBuf
   DependentTag(dependent) = Trim(Mid$(sBuf, 37, 12))
Next dependent

For dependent = 1 To NumberOfDependents

   ' Read & thrash the next dependent variable header: save the ramp status
     Line Input #iFile, sBuf
     RampStatus(dependent) = Val(Mid$(sBuf, 27, 9))
     For i = 1 To 10
       Line Input #iFile, sBuf
     Next i


     For independent = 1 To NumberOfIndependents

       ' Read & Store the next independent variable curve
       ' Tagname, Eng Units, and double precision SS gain
       Line Input #iFile, sBuf
       If bDMCplus Then
          dGain(independent, dependent) = Val(Mid$(sBuf, 27, Len(sBuf)))
       End If

       ' Model coefficients
       sBuf = ""
       For interval = 1 To NumberOfCoefficients
         If sBuf = "" Then
           Line Input #iFile, sBuf
           sBuf = Trim(sBuf)
         End If
```

```
              p = InStr(sBuf, " ")
              If p = 0 Then
                Coefficient(independent, dependent, interval) = Val(sBuf)
                sBuf = ""
              Else
                Coefficient(independent, dependent, interval) = Val(Left(sBuf, p))
                sBuf = Trim(Mid(sBuf, p + 1, Len(sBuf)))
              End If
            Next interval

            If Not bDMCplus Then
              If RampStatus(dependent) = 0 Then
                dGain(independent, dependent) = Coefficient(independent, dependent, NumberOfCoefficients)
              Else
                dGain(independent, dependent) = Coefficient(independent, dependent, NumberOfCoefficients) -
Coefficient(independent, dependent, NumberOfCoefficients - 1)
              End If
            End If

          Next independent

        Next dependent

        NumberOfIndependents = holdMVs
        ' Close the model file
        Close #iFile
        FileRead = True
      End If

      For independent = 1 To NumberOfIndependents
        Worksheets("MV Detail").Range("a5").Offset(independent, 0) = IndependentTag(independent)
      Next independent
      For dependent = 1 To NumberOfDependents
              Worksheets("CV Detail").Range("a5").Offset(dependent, 0) = DependentTag(dependent)
      Next dependent

    End Sub



Sub DisplayModelMatrix()

  Dim independent As Integer, dependent As Integer, interval As Integer
  Dim NumberOfFeedforwards As Integer
  Dim FirstMV As Integer
  Dim FeedforwardRange As String
  Dim FeedforwardsNotEntered As Boolean
  Dim tempplace As Integer


  FirstMV = 1
'----- Clear gain matrix

  Worksheets("Model").Select
  Worksheets("Model").Range("a1", "IV65536").Select
  Selection.ClearContents
  Selection.Interior.ColorIndex = xlNone
  Range("a5").Select



'----- Store numbers of independents and dependents in spreadsheet

  Worksheets("Model").Range("a4") = ModelFile
  Worksheets("Model").Range("a5") = NumberOfIndependents
  Worksheets("Model").Range("b5") = "independents"
  Worksheets("Model").Range("a6") = NumberOfDependents
  Worksheets("Model").Range("b6") = "dependents"
  Worksheets("Model").Range("a7") = NumberOfCoefficients
  Worksheets("Model").Range("b7") = "coefficients"
  Worksheets("Model").Range("a8") = "CV Name"
  Worksheets("Model").Range("b8") = "Ramp Flag"
  Worksheets("Model").Range("C8") = "MV Name"

  For dependent = 1 To NumberOfDependents
```

```
    tempplace = (dependent - 1) * NumberOfIndependents
    Worksheets("Model").Range("a9").Offset(tempplace, 0) = DependentTag(dependent)
    Worksheets("Model").Range("b9").Offset(tempplace, 0) = RampStatus(dependent)
    For independent = FirstMV To NumberOfIndependents
      Worksheets("Model").Range("c9").Offset(tempplace + independent - 1, 0) = IndependentTag(independent)
      For interval = 1 To NumberOfCoefficients
        Worksheets("Model").Range("c9").Offset(tempplace + independent - 1, interval) = Coefficient(independent,
dependent, interval)
      Next interval
    Next independent
  Next dependent


End Sub
```

Module 2

```
Option Base 1
'  DMC Tuner by GZ Gous 2008

Option Explicit

Sub RunDMCOnce()
    Application.ScreenUpdating = False
    ReadCCFInfo
    ReadModelMatrix
    GetConditions
    SetError
    DMCCalc
    Application.ScreenUpdating = True
End Sub
Sub GetSupMovTable()

Dim sm1 As Integer, sm2 As Integer, sm3 As Integer, i As Integer

    Worksheets("Results").Select
    Range("A1", "IV65536").Select
    Selection.ClearContents
    Selection.Interior.ColorIndex = xlNone
    Cells(1, 1).Select
    Cells(1, 1) = "SupMov1"
    Cells(1, 2) = "SupMov2"
    Cells(1, 3) = "SupMov3"
    Cells(1, 4) = "Total Error"
    Cells(1, 5) = "Error from smooth CV Response"
    Cells(1, 6) = "Error from move size ratio"
    Cells(1, 7) = "Error from MV 1st order response"
    Cells(1, 8) = "Error from minimise CV Error"

    ReadCCFInfo
    ReadModelMatrix
    GetConditions
    SetError
    For sm1 = 1 To 15
        If sm1 = 1 Then SupMov(1) = 0.1 Else _
        SupMov(1) = SupMov(1) * 2
        Worksheets("MV Detail").Cells(6, 2).Value = SupMov(1)
        For sm2 = 1 To 15
            If sm2 = 1 Then SupMov(2) = 0.1 Else _
            SupMov(2) = SupMov(2) * 2
            Worksheets("MV Detail").Cells(7, 2).Value = SupMov(2)
            For sm3 = 1 To 15
                If sm3 = 1 Then SupMov(3) = 0.1 Else _
                SupMov(3) = SupMov(3) * 2
                Worksheets("MV Detail").Cells(8, 2).Value = SupMov(3)
                DMCCalc
                WriteResults
            Next sm3
        Next sm2
    Next sm1
End Sub
Sub GetStart()
    Application.ScreenUpdating = False
    ReadCCFInfo
    ReadModelMatrix
    GetConditions
    SetError
    NelderOne
    CycleSupMov
    Application.ScreenUpdating = True
End Sub
Sub ReSolver()
    ReadCCFInfo
    ReadModelMatrix
    GetConditions
    SetError
    SetUpNelder
    'Call stepSupMov(1)
    StartVertXs
    startNM
End Sub
Sub StartVertXs()
Dim i As Integer, j As Integer, k As Integer, temp As Integer

Application.ScreenUpdating = False
For i = 1 To searchDim
    For j = 1 To NumberOfIndependents
        VertX(i, j) = Worksheets("MV Detail").Range("B5").Offset(j, 0).Value
        SupMov(j) = Worksheets("MV Detail").Range("B5").Offset(j, 0).Value
    Next j
    If SolveSupMults Then
        For j = NumberOfIndependents + 1 To searchDim
            VertX(i, j) = Worksheets("MV Detail").Range("D5").Offset(j - NumberOfIndependents, 0).Value
            SupMlt(j - NumberOfIndependents) = Worksheets("MV Detail").Range("D5").Offset(j -
NumberOfIndependents, 0).Value
        Next j
    End If
    If i <= NumberOfIndependents Then
        VertX(i, i) = Worksheets("MV Detail").Range("B5").Offset(i, 0).Value * 1.2
        SupMov(i) = Worksheets("MV Detail").Range("B5").Offset(i, 0).Value * 1.2
    Else
        VertX(i, i) = Worksheets("MV Detail").Range("D5").Offset(i - NumberOfIndependents, 0).Value * 1.2
        SupMlt(i - NumberOfIndependents) = Worksheets("MV Detail").Range("D5").Offset(i - NumberOfIndependents,
0).Value * 1.2
    End If
    DMCCalc
    VertX(i, 0) = Worksheets("MV Detail").Range("F5").Value
Next i
For j = 1 To searchDim
     If j <= NumberOfIndependents Then
        VertX(i, j) = Worksheets("MV Detail").Range("B5").Offset(j, 0).Value * 1.2
        SupMov(j) = Worksheets("MV Detail").Range("B5").Offset(j, 0).Value * 1.2
    Else
        VertX(i, j) = Worksheets("MV Detail").Range("D5").Offset(j - NumberOfIndependents, 0).Value * 1.2
        SupMlt(j - NumberOfIndependents) = Worksheets("MV Detail").Range("D5").Offset(j - NumberOfIndependents,
0).Value * 1.2
    End If
Next j
DMCCalc
VertX(i, 0) = Worksheets("MV Detail").Range("F5").Value
Application.ScreenUpdating = True



'to be deleted
For i = 1 To NoVertXs
    temp = searchDim + 2
    For j = 0 To searchDim
          Worksheets("Nelder").Cells(5 + VertXOrder(i), temp) = VertX(VertXOrder(i), j)
          temp = j + 2
    Next j
Next i
End Sub
Sub CycleSupMov()
Dim i As Integer, j As Integer, k As Integer
Dim Smallest As Single, smallSupMov As Single

For i = 1 To NumberOfIndependents
    'For j = 1 To NumberOfIndependents
```

```
  '   UseMV(j) = False
  'Next j
  'UseMV(i) = True
  Smallest = 1000000
  smallSupMov = 0.1
  SupMov(i) = 0.05
  For j = 1 To 12
    SupMov(i) = SupMov(i) * 2
    Worksheets("MV Detail").Cells(5 + i, 2) = SupMov(i)
    DMCCalc
    Worksheets("NelderSetup").Select
    For k = 1 To NumberOfIndependents
      ActiveCell.Offset(0, k - 1) = SupMov(k)
    Next k
    ActiveCell.Offset(0, k - 1) = Worksheets("MV Detail").Range("F5").Value
    If Worksheets("MV Detail").Range("F5").Value < Smallest Then
      Smallest = Worksheets("MV Detail").Range("F5").Value
      smallSupMov = SupMov(i)
    End If
    Cells(ActiveCell.Row + 1, 1).Select
  Next j
  SupMov(i) = smallSupMov
  Worksheets("MV Detail").Range("B5").Offset(i, 0) = SupMov(i)
Next i
Worksheets("MV Detail").Select
End Sub
Sub stepSupMov(i As Integer)
Dim j As Integer

  For j = -1 To 2
    SupMov(i) = 10 ^ j
    'Worksheets("MV Detail").Cells(5 + i, 2) = SupMov(i)
    If i < NumberOfIndependents Then Call stepSupMov(i + 1)
    DMCCalc
    Worksheets("NelderSetup").Select
    For k = 1 To NumberOfIndependents
      ActiveCell.Offset(0, k - 1) = SupMov(k)
    Next k
    ActiveCell.Offset(0, k - 1) = Worksheets("MV Detail").Range("F5").Value
    Cells(ActiveCell.Row + 1, 1).Select
    If Worksheets("MV Detail").Cells(5, 5) < VertX(VertXOrder(1), 0) Then
      For k = 1 To NumberOfIndependents
        VertX(VertXOrder(1), k) = SupMov(k)
      Next k
      VertX(VertXOrder(1), 0) = Worksheets("MV Detail").Cells(5, 5)
    End If
    OrderVertXs
  Next j
  SupMov(i) = 10 ^ j
End Sub
Sub WriteResults()
  Worksheets("Results").Select
  Cells(ActiveCell.Row + 1, 1).Select
  ActiveCell.Value = SupMov(1)
  ActiveCell.Offset(0, 1).Value = SupMov(2)
  ActiveCell.Offset(0, 2).Value = SupMov(3)
  ActiveCell.Offset(0, 3).Value = Worksheets("conditions").Range("B30")
  ActiveCell.Offset(0, 4).Value = Worksheets("conditions").Range("B31")
  ActiveCell.Offset(0, 5).Value = Worksheets("conditions").Range("B32")
  ActiveCell.Offset(0, 6).Value = Worksheets("conditions").Range("B33")
  ActiveCell.Offset(0, 7).Value = Worksheets("conditions").Range("B34")

End Sub
Sub CVResponse()

Dim i As Integer

ReDim CVPath((1.5 * NumberOfCoefficients - 1) * NumberOfDependents, 1)


Call MultArray(AMatrix, MVMoves, CVPath, (1.5 * NumberOfCoefficients - 1) * NumberOfDependents,
NumberOfMoves * NumberOfIndependents, 1)
Worksheets("CVResponse").Activate
 Worksheets("CVResponse").Range("A1", "IV65536").Select
 Selection.ClearContents
 'Selection.Interior.ColorIndex = xlNone
 Range("a5").Select
 For i = 1 To (1.5 * NumberOfCoefficients - 1) * NumberOfDependents
      Worksheets("CVResponse").Range("a5").Offset(i, 1) = CVPath(i, 1)
 Next i
 For i = 1 To NumberOfDependents
      Worksheets("CVResponse").Range("a5").Offset(i * (NumberOfCoefficients * 1.5 - 1) - 44 + CVTime, 0) = 0.75
      Worksheets("CVResponse").Range("a5").Offset(i * (NumberOfCoefficients * 1.5 - 1) - 44 + CVTime,
3).FormulaR1C1 = "=((RC[-2]-RC[-3])/RC[-3])^2"
      Worksheets("CVResponse").Range("a5").Offset(i * (NumberOfCoefficients * 1.5 - 1), 0) = 1
      Worksheets("CVResponse").Range("a5").Offset(i * (NumberOfCoefficients * 1.5 - 1), 2).FormulaR1C1 =
"=((RC[-1]-RC[-2])/RC[-2])^2"
 Next i

Worksheets("CVResponse").Range("a5").Offset(0, 2).FormulaR1C1 = "=average(R[1]C:R[300]C)"
Worksheets("CVResponse").Range("a5").Offset(0, 3).FormulaR1C1 = "=average(R[1]C:R[300]C)"

Worksheets("Conditions").Range("A30") = "Total Error"
Worksheets("Conditions").Range("B30").Formula = "=B34"
If Worksheets("conditions").Range("B4") = 1 Then Worksheets("Conditions").Range("B30").Formula =
Worksheets("Conditions").Range("B30").Formula + "+B31"
If Worksheets("conditions").Range("B3") = 1 Then Worksheets("Conditions").Range("B30").Formula =
Worksheets("Conditions").Range("B30").Formula + "+B32"
If Worksheets("conditions").Range("B6") = 1 Then Worksheets("Conditions").Range("B30").Formula =
Worksheets("Conditions").Range("B30").Formula + "+B33"
If Worksheets("conditions").Range("B7") = 1 Then Worksheets("Conditions").Range("B30").Formula =
Worksheets("Conditions").Range("B30").Formula + "+B35"

Worksheets("Conditions").Range("A31") = "Error from smooth CV Response"
Worksheets("Conditions").Range("B31").Formula = "=Conditions!B4*CVResponse!D5"
Worksheets("Conditions").Range("A32") = "Error from move size ratio"
Worksheets("Conditions").Range("B32").Formula = "=Conditions!B3*MVMoves!G5"
Worksheets("Conditions").Range("A33") = "Error from MV 1st order response"
Worksheets("Conditions").Range("B33").Formula = "=Conditions!B6*MVMoves!H5"
Worksheets("Conditions").Range("A34") = "Error from minimise CV Error"
Worksheets("Conditions").Range("B34").Formula = "=Conditions!B4*CVResponse!C5"
Worksheets("Conditions").Range("A35") = "Error from optimise MV Overshoot"
Worksheets("Conditions").Range("B35").Formula = "=Conditions!B7*MVMoves!K5"

Worksheets("MV Detail").Range("F5").Formula = "=Conditions!B30"

End Sub

Sub WriteMoves()

Dim i As Integer, j As Integer
Dim movacc As Single
Dim CellStr As String
Dim currRow As Integer


ReDim MovTgt(NumberOfMoves * NumberOfIndependents, 1)

For i = 1 To NumberOfIndependents
    MovTgt((i - 1) * NumberOfMoves + 1, 1) = 0.249 * TypMov(i)
    MovTgt((i - 1) * NumberOfMoves + 2, 1) = 0.435 * TypMov(i)
    MovTgt((i - 1) * NumberOfMoves + 3, 1) = 0.575 * TypMov(i)
    MovTgt((i - 1) * NumberOfMoves + 4, 1) = 0.76 * TypMov(i)
    MovTgt((i - 1) * NumberOfMoves + 5, 1) = 0.865 * TypMov(i)
    MovTgt((i - 1) * NumberOfMoves + 6, 1) = 0.924 * TypMov(i)
    MovTgt((i - 1) * NumberOfMoves + 7, 1) = 0.968 * TypMov(i)
    MovTgt((i - 1) * NumberOfMoves + 8, 1) = 0.986 * TypMov(i)
Next i
 Worksheets("MVMoves").Select
 Worksheets("MVMoves").Range("a1", "V65536").Select
 Selection.ClearContents
 'Selection.Interior.ColorIndex = xlNone
 Range("a5").Select


 Worksheets("MVMoves").Range("A4") = "MVMoves"
 Worksheets("MVMoves").Range("B4") = "Typical Moves"
 Worksheets("MVMoves").Range("C4") = "1st Order"
 Worksheets("MVMoves").Range("D4") = "Abs(MV)"
 Worksheets("MVMoves").Range("E4") = "dMVx/dMV1"
```

```
  Worksheets("MVMoves").Range("F4") = "TypMovx/TypMov1"
  Worksheets("MVMoves").Range("G4") = "Err^2"
  Worksheets("MVMoves").Range("H4") = "1st Order Err^2"
  Worksheets("MVMoves").Range("I4") = "Overshoot"
  Worksheets("MVMoves").Range("J4") = "Err"
  Worksheets("MVMoves").Range("K4") = "Err^2"

  For i = 1 To NumberOfIndependents
  movacc = 0
    For j = 1 To NumberOfMoves
        movacc = movacc + MVMoves((i - 1) * NumberOfMoves + j, 1)
        Worksheets("MVMoves").Range("a5").Offset((i - 1) * NumberOfMoves + j, 0) = movacc
        Worksheets("MVMoves").Range("a5").Offset((i - 1) * NumberOfMoves + j, 1) = MovTgt((i - 1) * NumberOfMoves
+ j, 1)
        If Abs(movacc) > 0 Then Worksheets("MVMoves").Range("a5").Offset((i - 1) * NumberOfMoves + j,
7).FormulaR1C1 = "=((RC[-7]-RC[-5])/RC[-5])^2"
    Next j
    For j = 1 To NumberOfMoves
        Worksheets("MVMoves").Range("a5").Offset((i - 1) * NumberOfMoves + j, 2) = MovTgt((i - 1) * NumberOfMoves
+ j, 1) * movacc / TypMov(i)
    Next j
  Next i

  For i = 1 To NumberOfIndependents * NumberOfMoves
    Cells(5 + i, 4).FormulaR1C1 = "=abs(RC[-3])"
  Next i

  'calculate overshoot
  For i = 1 To NumberOfIndependents
    For j = 1 To NumberOfMoves
        currRow = 5 + (i - 1) * NumberOfMoves + j
        Cells(currRow, 9).Formula = "=max(D" & Format(currRow - j + 1, "##") & ":D" & Format(currRow, "##") _
            & ")-D" & Format(currRow, "##") & "+D" & Format(5 + i * NumberOfMoves, "##")
    Next j
  Next i

ActiveWorkbook.Names.Add Name:="FirstMV", RefersTo:="=MVMoves!$A$13"
'Move ratio of Typical Move
For i = 1 To NumberOfIndependents
    CellStr = "E" & Format(i * NumberOfMoves + 5, "##")
    Range(CellStr).FormulaR1C1 = "=abs(RC[-4]/FirstMV)"
Next i
For i = 1 To NumberOfIndependents
    CellStr = "F" & Format(i * NumberOfMoves + 5, "##")
    Range(CellStr).Formula = "='MV Detail'!C" & Format(i + 5, "##") & "/'MV Detail'!C6"
    CellStr = "G" & Format(i * NumberOfMoves + 5, "##")
    Range(CellStr).FormulaR1C1 = "=(RC[-1]-RC[-2])^2"
Next i
' Optimise MV Overshoot
For i = 1 To NumberOfIndependents
    If UseMV(i) Then
        CellStr = "I" & Format(i * NumberOfMoves + 5, "##")
        Range(CellStr).FormulaR1C1 = "=max(R[-7]C:R[-1]C)/RC[-5]"
        CellStr = "J" & Format(i * NumberOfMoves + 5, "##")
        Range(CellStr).FormulaR1C1 = "=(RC[-1]-1)*100"
        Range(CellStr).Formula = Range(CellStr).Formula + "-'Conditions'!B8"
        CellStr = "K" & Format(i * NumberOfMoves + 5, "##")
        Range(CellStr).FormulaR1C1 = "=(RC[-1])^2"
    End If
Next i


Worksheets("MVMoves").Range("G5").FormulaR1C1 = "=sum(R[1]C:R[2500]C)"
Worksheets("MVMoves").Range("H5").FormulaR1C1 = "=sum(R[1]C:R[2500]C)"
Worksheets("MVMoves").Range("K5").FormulaR1C1 = "=sum(R[1]C:R[2500]C)"


End Sub
Sub CalcMoves()

Dim AT() As Single
Dim ATA() As Single
Dim ATAI() As Single
```

```
ReDim AT(NumberOfMoves * NumberOfIndependents, NumberOfMoves * NumberOfIndependents + (1.5 *
NumberOfCoefficients - 1) _
       * NumberOfDependents + NumberOfIndependents)
ReDim ATA(NumberOfMoves * NumberOfIndependents, NumberOfMoves * NumberOfIndependents)
ReDim ATAI(NumberOfMoves * NumberOfIndependents, NumberOfMoves * NumberOfIndependents)
ReDim ATAIAT(NumberOfMoves * NumberOfIndependents, NumberOfMoves * NumberOfIndependents + (1.5 *
NumberOfCoefficients - 1) _
       * NumberOfDependents + NumberOfIndependents)
ReDim MVMoves(NumberOfMoves * NumberOfIndependents, 1)

Call Transpose(AMatrix(), AT(), NumberOfMoves * NumberOfIndependents + (1.5 * NumberOfCoefficients - 1) _
                    * NumberOfDependents + NumberOfIndependents, NumberOfMoves * NumberOfIndependents)
Call MultArray(AT(), AMatrix(), ATA(), NumberOfMoves * NumberOfIndependents, NumberOfMoves *
NumberOfIndependents + (1.5 * NumberOfCoefficients - 1) _
       * NumberOfDependents + NumberOfIndependents, NumberOfMoves * NumberOfIndependents)


Call InverseArray(ATA(), ATAI(), NumberOfMoves * NumberOfIndependents)
Call MultArray(ATAI(), AT(), ATAIAT(), NumberOfMoves * NumberOfIndependents, NumberOfMoves *
NumberOfIndependents, NumberOfMoves * _
                    NumberOfIndependents + (1.5 * NumberOfCoefficients - 1) * NumberOfDependents +
NumberOfIndependents)
Call MultArray(ATAIAT(), ErrorMatrix(), MVMoves(), NumberOfMoves * NumberOfIndependents, NumberOfMoves *
_
                    NumberOfIndependents + (1.5 * NumberOfCoefficients - 1) * NumberOfDependents +
NumberOfIndependents, 1)



End Sub
Sub CalcDMV()
Dim i As Integer, j As Integer, k As Integer, l As Integer, m As Integer
Dim SupMltJ As Single

NumberOfMoves = 8
' Current code only handles 8 moves, set at times 1,2,3,4,5,7,11,14
EndWeight = 1000 'Enforce end condition weighting

ReDim AMatrix(NumberOfMoves * NumberOfIndependents + (1.5 * NumberOfCoefficients - 1) _
     * NumberOfDependents + NumberOfIndependents, NumberOfMoves * NumberOfIndependents)
ReDim ErrorMatrix(Int(1.5 * NumberOfCoefficients - 1) * NumberOfDependents + NumberOfIndependents *
NumberOfMoves + NumberOfIndependents, 1)

'fill the errormatrix
  Worksheets("ErrorMatrix").Select
  Worksheets("ErrorMatrix").Range("a1", "IV65536").Select
  Selection.ClearContents
  Selection.Interior.ColorIndex = xlNone
  Range("a5").Select
For i = 1 To NumberOfDependents
   For j = 1 To Int(1.5 * NumberOfCoefficients - 1)
      ErrorMatrix(j + (i - 1) * Int(1.5 * NumberOfCoefficients - 1), 1) = CVError(i, j) / HiECE(i)
      Worksheets("ErrorMatrix").Range("a5").Offset(j + (i - 1) * Int(1.5 * NumberOfCoefficients - 1), 0) = CVError(i, j) /
HiECE(i)
   Next j
Next i

For i = 1 To NumberOfIndependents * NumberOfMoves
   ErrorMatrix(NumberOfDependents * Int(1.5 * NumberOfCoefficients - 1) + i, 1) = 0
   Worksheets("ErrorMatrix").Range("a5").Offset(NumberOfDependents * Int(1.5 * NumberOfCoefficients - 1) + i, 0) =
0
Next i

For i = 1 To NumberOfIndependents
   ErrorMatrix(NumberOfDependents * Int(1.5 * NumberOfCoefficients - 1) + NumberOfIndependents *
NumberOfMoves + i, 1) = dMV(i) * EndWeight
   Worksheets("ErrorMatrix").Range("a5").Offset(NumberOfDependents * Int(1.5 * NumberOfCoefficients - 1) +
NumberOfIndependents * NumberOfMoves + i, 0) = dMV(i) * EndWeight
Next i

Worksheets("AMatrix").Select
   Worksheets("AMatrix").Range("a1", "IV65536").Select
   Selection.ClearContents
   Worksheets("AMatrix").Range("a1").Select
```

```
For i = 1 To Int(NumberOfCoefficients * 1.5 - 1) * NumberOfDependents + NumberOfMoves *
NumberOfIndependents + NumberOfIndependents
    For j = 1 To NumberOfMoves * NumberOfIndependents
        AMatrix(i, j) = 0
        Worksheets("AMatrix").Range("a1").Offset(i - 1, j - 1) = 0
    Next j
Next i

' Current code only handles 8 moves, set at times 1,2,3,4,5,7,11,14
ReDim fmovt(8)
fmovt(1) = 1
fmovt(2) = 2
fmovt(3) = 3
fmovt(4) = 4
fmovt(5) = 5
fmovt(6) = 7
fmovt(7) = 11
fmovt(8) = 14

'controller models
For l = 1 To NumberOfDependents
    For i = 1 To NumberOfIndependents
        For j = 1 To NumberOfMoves
                For k = 1 To Int(NumberOfCoefficients * 1.5) - fmovt(j)
                    If k > NumberOfCoefficients Then
                        m = NumberOfCoefficients
                    Else
                        m = k
                    End If
                    AMatrix((l - 1) * (1.5 * NumberOfCoefficients - 1) + k + fmovt(j) - 1, j + (i - 1) * NumberOfMoves) =
UseMV(i) * Coefficient(i, l, m)
                    Worksheets("AMatrix").Range("a1").Offset((l - 1) * (1.5 * NumberOfCoefficients - 1) + k + fmovt(j) - 2, j +
(i - 1) * NumberOfMoves - 1) = UseMV(i) * Coefficient(i, l, m)
                Next k
        Next j
    Next i
Next l
'move suppressions
For i = 1 To NumberOfIndependents
    SupMltJ = 0
    For j = 1 To NumberOfMoves
        If j > 5 Then
            SupMltJ = 1 + (j - 5) * 0.333333333 * (SupMlt(i) - 1)
        Else
            SupMltJ = 1
        End If
        AMatrix(Int(1.5 * NumberOfCoefficients - 1) * NumberOfDependents + j + (i - 1) * NumberOfMoves, j + (i - 1) *
NumberOfMoves) = SupMov(i) * SupMltJ
        Worksheets("AMatrix").Range("a1").Offset(Int(1.5 * NumberOfCoefficients - 1) * NumberOfDependents + j + (i -
1) * NumberOfMoves - 1, j + (i - 1) * NumberOfMoves - 1) = SupMov(i) * SupMltJ
    Next j
Next i
'end condition
For i = 1 To NumberOfIndependents
    For j = 1 To NumberOfMoves
        AMatrix(NumberOfMoves * NumberOfIndependents + (1.5 * NumberOfCoefficients - 1) * NumberOfDependents
+ i, (i - 1) * NumberOfMoves + j) = EndWeight
        Worksheets("AMatrix").Range("a1").Offset(NumberOfMoves * NumberOfIndependents + (1.5 *
NumberOfCoefficients - 1) * NumberOfDependents + i - 1, (i - 1) * NumberOfMoves + j - 1) = EndWeight
    Next j
Next i

End Sub

Sub SetError()
Dim i As Integer, j As Integer

ReDim CVError(NumberOfDependents, Int(1.5 * NumberOfCoefficients))
For i = 1 To NumberOfDependents
    For j = 1 To Int(1.5 * NumberOfCoefficients)
        CVError(i, j) = HiECE(i)
    Next j
Next i
End Sub
Sub GetConditions()
```

```
Dim independent As Integer, dependent As Integer
Dim i As Integer
Dim rowi As Integer



ReDim UseMV(NumberOfIndependents)
ReDim UseCV(NumberOfDependents)

Worksheets("Conditions").Select
Range("a1").Select
For i = 1 To NumberOfIndependents
   Cells(1 + i, 4).Value = IndependentTag(i)
   Cells(1 + i, 5).Value = 1
   UseMV(i) = 1
Next i
For i = 1 To NumberOfDependents
   Cells(1 + i, 7).Value = DependentTag(i)
   Cells(1 + i, 8).Value = 1
   UseCV(i) = 1
Next i
If NumberOfIndependents >= 1 Then GetVars.MV1.Text = IndependentTag(1)
If NumberOfIndependents >= 2 Then GetVars.MV2.Text = IndependentTag(2)
If NumberOfIndependents >= 3 Then GetVars.MV3.Text = IndependentTag(3)
If NumberOfIndependents >= 4 Then GetVars.MV4.Text = IndependentTag(4)
If NumberOfIndependents >= 5 Then GetVars.MV5.Text = IndependentTag(5)
If NumberOfIndependents >= 6 Then GetVars.MV6.Text = IndependentTag(6)
If NumberOfIndependents >= 7 Then GetVars.MV7.Text = IndependentTag(7)
If NumberOfIndependents >= 8 Then GetVars.MV8.Text = IndependentTag(8)
If NumberOfIndependents >= 9 Then GetVars.MV9.Text = IndependentTag(9)
If NumberOfIndependents >= 10 Then GetVars.MV10.Text = IndependentTag(10)
If NumberOfIndependents >= 11 Then GetVars.MV11.Text = IndependentTag(11)
If NumberOfIndependents >= 12 Then GetVars.MV12.Text = IndependentTag(12)
If NumberOfDependents >= 1 Then GetVars.CV1.Text = DependentTag(1)
If NumberOfDependents >= 2 Then GetVars.CV2.Text = DependentTag(2)
If NumberOfDependents >= 3 Then GetVars.CV3.Text = DependentTag(3)
If NumberOfDependents >= 4 Then GetVars.CV4.Text = DependentTag(4)
If NumberOfDependents >= 5 Then GetVars.CV5.Text = DependentTag(5)
If NumberOfDependents >= 6 Then GetVars.CV6.Text = DependentTag(6)
If NumberOfDependents >= 7 Then GetVars.CV7.Text = DependentTag(7)
If NumberOfDependents >= 8 Then GetVars.CV8.Text = DependentTag(8)
If NumberOfDependents >= 9 Then GetVars.CV9.Text = DependentTag(9)
If NumberOfDependents >= 10 Then GetVars.CV10.Text = DependentTag(10)
If NumberOfDependents >= 11 Then GetVars.CV11.Text = DependentTag(11)
If NumberOfDependents >= 12 Then GetVars.CV12.Text = DependentTag(12)


If NumberOfIndependents < 2 Then GetVars.MV2.Visible = False
If NumberOfIndependents < 3 Then GetVars.MV3.Visible = False
If NumberOfIndependents < 4 Then GetVars.MV4.Visible = False
If NumberOfIndependents < 5 Then GetVars.MV5.Visible = False
If NumberOfIndependents < 6 Then GetVars.MV6.Visible = False
If NumberOfIndependents < 7 Then GetVars.MV7.Visible = False
If NumberOfIndependents < 8 Then GetVars.MV8.Visible = False
If NumberOfIndependents < 9 Then GetVars.MV9.Visible = False
If NumberOfIndependents < 10 Then GetVars.MV10.Visible = False
If NumberOfIndependents < 11 Then GetVars.MV11.Visible = False
If NumberOfIndependents < 12 Then GetVars.MV12.Visible = False
If NumberOfIndependents < 2 Then GetVars.MV2st.Visible = False
If NumberOfIndependents < 3 Then GetVars.MV3st.Visible = False
If NumberOfIndependents < 4 Then GetVars.MV4st.Visible = False
If NumberOfIndependents < 5 Then GetVars.MV5st.Visible = False
If NumberOfIndependents < 6 Then GetVars.MV6st.Visible = False
If NumberOfIndependents < 7 Then GetVars.MV7st.Visible = False
If NumberOfIndependents < 8 Then GetVars.MV8st.Visible = False
If NumberOfIndependents < 9 Then GetVars.MV9st.Visible = False
If NumberOfIndependents < 10 Then GetVars.MV10st.Visible = False
If NumberOfIndependents < 11 Then GetVars.MV11st.Visible = False
If NumberOfIndependents < 12 Then GetVars.MV12st.Visible = False

If NumberOfDependents < 2 Then GetVars.CV2.Visible = False
If NumberOfDependents < 3 Then GetVars.CV3.Visible = False
If NumberOfDependents < 4 Then GetVars.CV4.Visible = False
If NumberOfDependents < 5 Then GetVars.CV5.Visible = False
If NumberOfDependents < 6 Then GetVars.CV6.Visible = False
If NumberOfDependents < 7 Then GetVars.CV7.Visible = False
If NumberOfDependents < 8 Then GetVars.CV8.Visible = False
If NumberOfDependents < 9 Then GetVars.CV9.Visible = False
If NumberOfDependents < 10 Then GetVars.CV10.Visible = False
If NumberOfDependents < 11 Then GetVars.CV11.Visible = False
If NumberOfDependents < 12 Then GetVars.CV12.Visible = False
If NumberOfDependents < 2 Then GetVars.CV2st.Visible = False
If NumberOfDependents < 3 Then GetVars.CV3st.Visible = False
If NumberOfDependents < 4 Then GetVars.CV4st.Visible = False
If NumberOfDependents < 5 Then GetVars.CV5st.Visible = False
If NumberOfDependents < 6 Then GetVars.CV6st.Visible = False
If NumberOfDependents < 7 Then GetVars.CV7st.Visible = False
If NumberOfDependents < 8 Then GetVars.CV8st.Visible = False
If NumberOfDependents < 9 Then GetVars.CV9st.Visible = False
If NumberOfDependents < 10 Then GetVars.CV10st.Visible = False
If NumberOfDependents < 11 Then GetVars.CV11st.Visible = False
If NumberOfDependents < 12 Then GetVars.CV12st.Visible = False


GetCond.Show
SolveSupMults = GetCond.SupMult
SolveRatio = False 'GetCond.MVRatio
SolveCVSmooth = GetCond.SmoothCV
SolveMVOvershoot = GetCond.MVOver
SetCstToSupMov = GetCond.SSCost
MVOvershoot = GetCond.MVPct
CVTime = GetCond.CVTime
SolveMV1st = GetCond.MV1st

'If Not GetCond.UseAllVars Then
'    GetVars.Show
'End If

If Not GetVars.MV1st Then Cells(2, 5) = 0
If Not GetVars.MV2st Then Cells(3, 5) = 0
If Not GetVars.MV3st Then Cells(4, 5) = 0
If Not GetVars.MV4st Then Cells(5, 5) = 0
If Not GetVars.MV5st Then Cells(6, 5) = 0
If Not GetVars.MV6st Then Cells(7, 5) = 0
If Not GetVars.MV7st Then Cells(8, 5) = 0
If Not GetVars.MV8st Then Cells(9, 5) = 0
If Not GetVars.MV9st Then Cells(10, 5) = 0
If Not GetVars.MV10st Then Cells(11, 5) = 0
If Not GetVars.MV11st Then Cells(12, 5) = 0
If Not GetVars.MV12st Then Cells(13, 5) = 0
If Not GetVars.CV1st Then Cells(2, 8) = 0
If Not GetVars.CV2st Then Cells(3, 8) = 0
If Not GetVars.CV3st Then Cells(4, 8) = 0
If Not GetVars.CV4st Then Cells(5, 8) = 0
If Not GetVars.CV5st Then Cells(6, 8) = 0
If Not GetVars.CV6st Then Cells(7, 8) = 0
If Not GetVars.CV7st Then Cells(8, 8) = 0
If Not GetVars.CV8st Then Cells(9, 8) = 0
If Not GetVars.CV9st Then Cells(10, 8) = 0
If Not GetVars.CV10st Then Cells(11, 8) = 0
If Not GetVars.CV11st Then Cells(12, 8) = 0
If Not GetVars.CV12st Then Cells(13, 8) = 0
For i = 1 To NumberOfIndependents
    UseMV(i) = Cells(i + 1, 5)
Next i
 For i = 1 To NumberOfDependents
    UseCV(i) = Cells(i + 1, 8)
Next i

Range("A2").Value = "Solve SupMlts"
Range("A3").Value = "Solve for MV Move size ratio of TypMov"
Range("A4").Value = "Solve for smooth CV response"
Range("A5").Value = "Time for CV to reach 75%"
Range("A6").Value = "Solve for MV 1st order response"
Range("A7").Value = "Solve for Mv overshoot"
Range("A8").Value = "Percent Mv overshoot"
Range("B2").Value = SolveSupMults
Range("B6").Value = Abs(SolveMV1st)
Range("B3").Value = Abs(SolveRatio)
Range("B4").Value = Abs(SolveCVSmooth)
```

```
    Range("B5").Value = CVTime
    Range("B7").Value = Abs(SolveMVOvershoot)
    Range("B8").Value = MVOvershoot

    Worksheets("Conditions").Range("A11") = "FF Name"
    Worksheets("Conditions").Range("B11") = "Size"
    rowi = 0
    For i = 1 To NumberOfIndependents
        If SupMov(i) = 0 Then
            rowi = rowi + 1
            Worksheets("Conditions").Range("A11").Offset(rowi, 0) = IndependentTag(i)
        End If
    Next i
End Sub

Sub ReadCCFInfo()

     Dim dependent As Integer, _
         independent As Integer

    NumberOfIndependents = Worksheets("Model").Cells(5, 1)
    NumberOfDependents = Worksheets("Model").Cells(6, 1)
    NumberOfCoefficients = Worksheets("Model").Cells(7, 1)

    ReDim SupMov(NumberOfIndependents)
    ReDim SupMlt(NumberOfIndependents)
    ReDim TypMov(NumberOfIndependents)
    ReDim Cst(NumberOfIndependents)
    ReDim HiECE(NumberOfDependents)
    ReDim LoECE(NumberOfDependents)
    ReDim IndependentTag(NumberOfIndependents)
    ReDim DependentTag(NumberOfDependents)
    ReDim Coefficient(NumberOfIndependents, NumberOfDependents, NumberOfCoefficients)
    ReDim RampStatus(NumberOfDependents)
    ReDim dGain(NumberOfIndependents, NumberOfDependents)



    Worksheets("MV Detail").Select
    Worksheets("MV Detail").Range("a1").Select
    'Worksheets("MV Detail").Cells(1.1) = 0
    For independent = 1 To NumberOfIndependents
        IndependentTag(independent) = Worksheets("MV Detail").Range("a5").Offset(independent, 0)
        SupMov(independent) = Worksheets("MV Detail").Range("a5").Offset(independent, 1)
        TypMov(independent) = Worksheets("MV Detail").Range("a5").Offset(independent, 2)
        SupMlt(independent) = Worksheets("MV Detail").Range("a5").Offset(independent, 3)
        Cst(independent) = Worksheets("MV Detail").Range("a5").Offset(independent, 4)
    Next independent

    Worksheets("CV Detail").Select
    Worksheets("CV Detail").Range("a1").Select
        For dependent = 1 To NumberOfDependents
        DependentTag(dependent) = Worksheets("CV Detail").Range("a5").Offset(dependent, 0)
        HiECE(dependent) = Worksheets("CV Detail").Range("a5").Offset(dependent, 1)
        LoECE(dependent) = Worksheets("CV Detail").Range("a5").Offset(dependent, 2)
    Next dependent

End Sub

Sub ReadModelMatrix()

  Dim independent As Integer, dependent As Integer, interval As Integer
  Dim NumberOfFeedforwards As Integer
  Dim FeedforwardRange As String
  Dim FeedforwardsNotEntered As Boolean
  Dim tempplace As Integer


  Worksheets("Model").Select


  For dependent = 1 To NumberOfDependents
    tempplace = (dependent - 1) * NumberOfIndependents
    DependentTag(dependent) = Worksheets("Model").Range("a9").Offset(tempplace, 0)
    For independent = 1 To NumberOfIndependents
```

```
      IndependentTag(independent) = Worksheets("Model").Range("c9").Offset(tempplace + independent - 1, 0)
      For interval = 1 To NumberOfCoefficients
        Coefficient(independent, dependent, interval) = Worksheets("Model").Range("c9").Offset(tempplace +
independent - 1, interval)
      Next interval
    Next independent
  Next dependent


End Sub


Sub Transpose(ByRef AT() As Single, ByRef BT() As Single, i, j)
Dim k As Integer, l As Integer
' i = number of rows in AT, j = number of columns in AT
For k = 1 To i
    For l = 1 To j
        BT(l, k) = AT(k, l)
    Next l
Next k

End Sub

Sub MultArray(ByRef A() As Single, ByRef B() As Single, ByRef C() As Single, i As Integer, j As Integer, n As Integer)
' C = A*B, i,j = dim of A
'         j,n = dim of B
'         i,n = dim of C
Dim k As Integer, l As Integer, m As Integer

For k = 1 To i
    For l = 1 To n
        C(k, l) = 0
        For m = 1 To j
            C(k, l) = C(k, l) + A(k, m) * B(m, l)
        Next m
    Next l
Next k

End Sub

Sub InverseArray(ByRef A() As Single, ByRef B() As Single, i As Integer)
'B = A^-1

Dim k As Integer, l As Integer, m As Integer
Dim C() As Single
Dim temp As Single


ReDim C(i, i)

For k = 1 To i
    For l = 1 To i
        B(k, l) = 0
        C(k, l) = A(k, l)
    Next l
    B(k, k) = 1
Next k


For k = 1 To i
    temp = C(k, k)
    For l = 1 To i
        C(k, l) = C(k, l) / temp
        B(k, l) = B(k, l) / temp
    Next l
    For l = k + 1 To i
        temp = C(l, k)
        If temp <> 0 Then
            For m = 1 To i
                C(l, m) = C(l, m) / temp - C(k, m)
                B(l, m) = B(l, m) / temp - B(k, m)
            Next m
        End If
    Next l
Next k
```

```
For k = i To 1 Step -1
    For l = k - 1 To 1 Step -1
        temp = C(l, k)
        For m = 1 To i
            C(l, m) = C(l, m) - C(k, m) * temp
            B(l, m) = B(l, m) - B(k, m) * temp
        Next m
    Next l
Next k


End Sub
```

Module 3

```
Option Base 1
'  Nelder Mead solver by GZ Gous and B de Jongh 2010

Option Explicit

Global order() As Integer
Global VertX() As Double
Global VertXOrder() As Integer
Global searchDim As Integer, NoVertXs As Integer
Global i As Integer, j As Integer, k As Integer
Global TheCowsComeHome As Boolean
Global Alpha As Double
Global Gamma As Double
Global Rho As Double
Global Sigma As Double
Global Centroid() As Double
Global NewPoint() As Double
Global ExpandPoint() As Double
Global ContractPoint() As Double
Global ReducedPoint() As Double
Global Cycle As Integer
Sub SetUpNelder()

Alpha = 1
Gamma = 2
Rho = 0.5
Sigma = 0.5
Cycle = 0
searchDim = NumberOfIndependents
If SolveSupMults Then searchDim = searchDim * 2
NoVertXs = searchDim + 1 'no of points on simplex
TheCowsComeHome = False
ReDim VertXOrder(1 To NoVertXs)
ReDim VertX(1 To NoVertXs, 0 To searchDim) 'Number of vertex, number of dimension of vertex
ReDim NewPoint(0 To searchDim)
ReDim ExpandPoint(0 To searchDim)
ReDim ContractPoint(0 To searchDim)
ReDim ReducedPoint(0 To searchDim)

Worksheets("NelderSetup").Select
Range("A1:IV65000").Select
Selection.ClearContents
Range("A1").Select

For i = 1 To NoVertXs
      VertX(i, 0) = 1000000
      VertXOrder(i) = i
Next i
Worksheets("Nelder").Select
Range("A5", "Z50").Select
Selection.ClearContents
Range("A5").Select
Cells(5, 1) = "Vertex number"
For i = 1 To NoVertXs
    Cells(5, 1 + i) = "Dim" & Str(i)
    Cells(5 + i, 1) = i
Next i
```

```
Cells(6 + NoVertXs, 2).Formula = "=(max(B6:B" & Format(5 + NoVertXs, "##") & ")-min(B6:B" & Format(5 + NoVertXs,
"##") & "))/min(B6:B" & Format(5 + NoVertXs, "##") & ")"
Cells(6 + NoVertXs, 2).Select
Selection.Copy
For i = 2 To searchDim + 1
   Cells(6 + NoVertXs, ActiveCell.Column + 1).Select
   ActiveSheet.Paste
Next i
Cells(6 + NoVertXs, 1).Formula = "=max(B" & Format(5 + NoVertXs + 1, "##") & ":" & Chr(64 + searchDim + 1) &
Format(5 + NoVertXs + 1, "##") & ")"
Cells(5, 1 + NoVertXs) = "F(x)"
'Worksheets("Nelder").Cells(20, 1) = "Reflected Point"
'Worksheets("Nelder").Cells(21, 1) = "Expanded Point"
'Worksheets("Nelder").Cells(22, 1) = "Contracted Point"
Worksheets("Nelder").Cells(5, searchDim + 3) = "Order"
Worksheets("Nelder").Cells(5, searchDim + 4) = "Cycle"
End Sub
Sub NelderOne()
Alpha = 1
Gamma = 2
Rho = 0.5
Sigma = 0.5
Cycle = 0
searchDim = 1
NoVertXs = searchDim + 1 'no of points on simplex
TheCowsComeHome = False
ReDim VertXOrder(1 To NoVertXs)
ReDim VertX(1 To NoVertXs, 0 To searchDim) 'Number of vertex, number of dimension of vertex
ReDim NewPoint(0 To searchDim)
ReDim ExpandPoint(0 To searchDim)
ReDim ContractPoint(0 To searchDim)
ReDim ReducedPoint(0 To searchDim)

Worksheets("NelderSetup").Select
Range("A1:IV65000").Select
Selection.ClearContents
Range("A1").Select

For i = 1 To NoVertXs
      VertX(i, 0) = 1000000
      VertXOrder(i) = i
Next i
Worksheets("Nelder").Select
Range("A5", "Z50").Select
Selection.ClearContents
Range("A5").Select
Cells(5, 1) = "Vertex number"
For i = 1 To NoVertXs
   Cells(5, 1 + i) = "Dim" & Str(i)
   Cells(5 + i, 1) = i
Next i
Cells(6 + NoVertXs, 2).Formula = "=(max(B6:B" & Format(5 + NoVertXs, "##") & ")-min(B6:B" & Format(5 + NoVertXs,
"##") & "))/min(B6:B" & Format(5 + NoVertXs, "##") & ")"
Cells(6 + NoVertXs, 2).Select
Selection.Copy
For i = 2 To searchDim + 1
   Cells(6 + NoVertXs, ActiveCell.Column + 1).Select
   ActiveSheet.Paste
Next i
Cells(6 + NoVertXs, 1).Formula = "=max(B" & Format(5 + NoVertXs + 1, "##") & ":" & Chr(64 + searchDim + 1) &
Format(5 + NoVertXs + 1, "##") & ")"
Cells(5, 1 + NoVertXs) = "F(x)"
'Worksheets("Nelder").Cells(20, 1) = "Reflected Point"
'Worksheets("Nelder").Cells(21, 1) = "Expanded Point"
'Worksheets("Nelder").Cells(22, 1) = "Contracted Point"
Worksheets("Nelder").Cells(5, searchDim + 3) = "Order"
Worksheets("Nelder").Cells(5, searchDim + 4) = "Cycle"
End Sub
Sub startNM()

Dim temp As Integer
Dim Big As Single, Small As Single
'Dim DeltaSum As Single

For i = 1 To NoVertXs
```

```
    temp = searchDim + 2
    For j = 0 To searchDim
        Worksheets("Nelder").Cells(5 + VertXOrder(i), temp) = VertX(VertXOrder(i), j)
        temp = j + 2
    Next j
Next i

Do
Cycle = Cycle + 1
Worksheets("Nelder").Select
Application.ScreenUpdating = True
Worksheets("Nelder").Cells(6, searchDim + 4) = Cycle
Application.ScreenUpdating = False
OrderVertXs
FindCentroid
Reflect
'Evaluate
If (NewPoint(0) < VertX(VertXOrder(2), 0)) And (NewPoint(0) >= VertX(VertXOrder(NoVertXs), 0)) Then
    '  use reflected point
    temp = searchDim + 2
    For j = 0 To searchDim
       VertX(VertXOrder(1), j) = NewPoint(j)
       Worksheets("Nelder").Cells(5 + VertXOrder(1), temp) = NewPoint(j)
       temp = j + 2
    Next j
ElseIf NewPoint(0) < VertX(VertXOrder(NoVertXs), 0) Then
       ' expansion
       For j = 1 To searchDim
          ExpandPoint(j) = Centroid(j) + Alpha * (Centroid(j) - VertX(VertXOrder(1), j))
          If j <= NumberOfIndependents Then
             SupMov(j) = ExpandPoint(j)
          Else
             SupMlt(j - NumberOfIndependents) = ExpandPoint(j)
          End If
          'Worksheets("Nelder").Cells(21, j + 1) = ExpandPoint(j)
       Next j
       DMCCalc
       ExpandPoint(0) = Worksheets("MV Detail").Cells(5, 6)
       If ExpandPoint(0) < NewPoint(0) Then
          temp = searchDim + 2
          For j = 0 To searchDim
             VertX(VertXOrder(1), j) = ExpandPoint(j)
             Worksheets("Nelder").Cells(5 + VertXOrder(1), temp) = ExpandPoint(j)
             temp = j + 2
          Next j
       Else
          temp = searchDim + 2
          For j = 0 To searchDim
             VertX(VertXOrder(1), j) = NewPoint(j)
             Worksheets("Nelder").Cells(5 + VertXOrder(1), temp) = NewPoint(j)
             temp = j + 2
          Next j
       End If
    Else
       '  contraction
       For j = 1 To searchDim
          ContractPoint(j) = VertX(VertXOrder(1), j) + Rho * (Centroid(j) - VertX(VertXOrder(1), j))
          'Worksheets("Nelder").Cells(22, j + 1) = ContractPoint(j)
          If j <= NumberOfIndependents Then
             SupMov(j) = ContractPoint(j)
          Else
             SupMlt(j - NumberOfIndependents) = ContractPoint(j)
          End If
       Next j
       DMCCalc
       ContractPoint(0) = Worksheets("MV Detail").Cells(5, 6)
       If ContractPoint(0) < VertX(VertXOrder(1), 0) Then
          temp = searchDim + 2
          For j = 0 To searchDim                                    '  check 000
             VertX(VertXOrder(1), j) = ContractPoint(j)
             Worksheets("Nelder").Cells(5 + VertXOrder(1), temp) = ContractPoint(j)
             temp = j + 2
          Next j
       Else
          'reduction
```

```
      For i = 1 To searchDim
        For j = 1 To searchDim
          VertX(VertXOrder(i), j) = VertX(VertXOrder(NoVertXs), j) + Sigma * (VertX(VertXOrder(i), j) -
VertX(VertXOrder(NoVertXs), j))
          Worksheets("Nelder").Cells(5 + VertXOrder(i), j + 1) = VertX(VertXOrder(i), j)
          If j <= NumberOfIndependents Then
            SupMov(j) = VertX(VertXOrder(i), j)
          Else
            SupMlt(j - NumberOfIndependents) = VertX(VertXOrder(i), j)
          End If
        Next j
        DMCCalc
        VertX(VertXOrder(i), 0) = Worksheets("MV Detail").Cells(5, 6)
        Worksheets("Nelder").Cells(5 + VertXOrder(i), NoVertXs + 1) = VertX(VertXOrder(i), 0)
      Next i
    End If
End If

If (Worksheets("Nelder").Cells(6 + NoVertXs, 1) < 0.05) Or (Cycle > 2000) Then TheCowsComeHome = True
Loop Until TheCowsComeHome
For i = 1 To searchDim
  If i <= NumberOfIndependents Then
    Worksheets("MV Detail").Cells(5 + i, 2) = VertX(VertXOrder(NoVertXs), i)
    SupMov(i) = VertX(VertXOrder(NoVertXs), i)
  Else
    Worksheets("MV Detail").Cells(5 + i - NumberOfIndependents, 4) = VertX(VertXOrder(NoVertXs), i)
    SupMlt(i - NumberOfIndependents) = VertX(VertXOrder(NoVertXs), i)
  End If

Next i
DMCCalc
Worksheets("MV Detail").Select
Application.ScreenUpdating = True
End Sub

Sub OrderVertXs() 'from greatest to least

Dim temp As Integer

For i = 1 To NoVertXs
  VertXOrder(i) = i
Next i
For i = 1 To NoVertXs
  For j = i To NoVertXs
    If VertX(VertXOrder(i), 0) < VertX(VertXOrder(j), 0) Then
      temp = VertXOrder(i)
      VertXOrder(i) = VertXOrder(j)
      VertXOrder(j) = temp
    End If
  Next j
Next i

For i = 1 To NoVertXs
  Worksheets("Nelder").Cells(5 + VertXOrder(i), searchDim + 3) = i
Next i
End Sub

Sub FindCentroid()

ReDim Centroid(1 To searchDim)
' Set centroid at second worst point
For j = 1 To searchDim
   Centroid(j) = VertX(VertXOrder(2), j)
Next j
'calc centroid up in dimensions
If NoVertXs > 2 Then
  For i = 3 To NoVertXs
    For j = 1 To searchDim
      Centroid(j) = Centroid(j) + (VertX(VertXOrder(i), j) - Centroid(j)) / (i - 1)
    Next j
  Next i
End If
End Sub

Sub Reflect()
```

```
For j = 1 To searchDim
     NewPoint(j) = Centroid(j) + Alpha * (Centroid(j) - VertX(VertXOrder(1), j))
     If j <= NumberOfIndependents Then
        SupMov(j) = NewPoint(j)
    Else
        SupMlt(j - NumberOfIndependents) = NewPoint(j)
    End If
     'Worksheets("Nelder").Cells(20, j + 1) = NewPoint(j)
Next j
DMCCalc
NewPoint(0) = Worksheets("MV Detail").Cells(5, 6)

End Sub

Sub DMCCalc()
    GetSS
    CalcDMV
    CalcMoves
    WriteMoves
    CVResponse
End Sub
```

Worksheet Main

```
Private Sub CommandButton1_Click()
GetStart
End Sub

Private Sub GetSMTable_Click()
    DoOnce = True
    GetSupMovTable
End Sub

Private Sub resolve1_Click()
    DoOnce = True
    ReSolver
End Sub

Private Sub RunOnce_Click()
    DoOnce = True
    RunDMCOnce
End Sub

Private Sub Start_Click()
    DoOnce = False
    Master
End Sub
```

Appendix B – Different manifestations of manipulated variable overshoot.

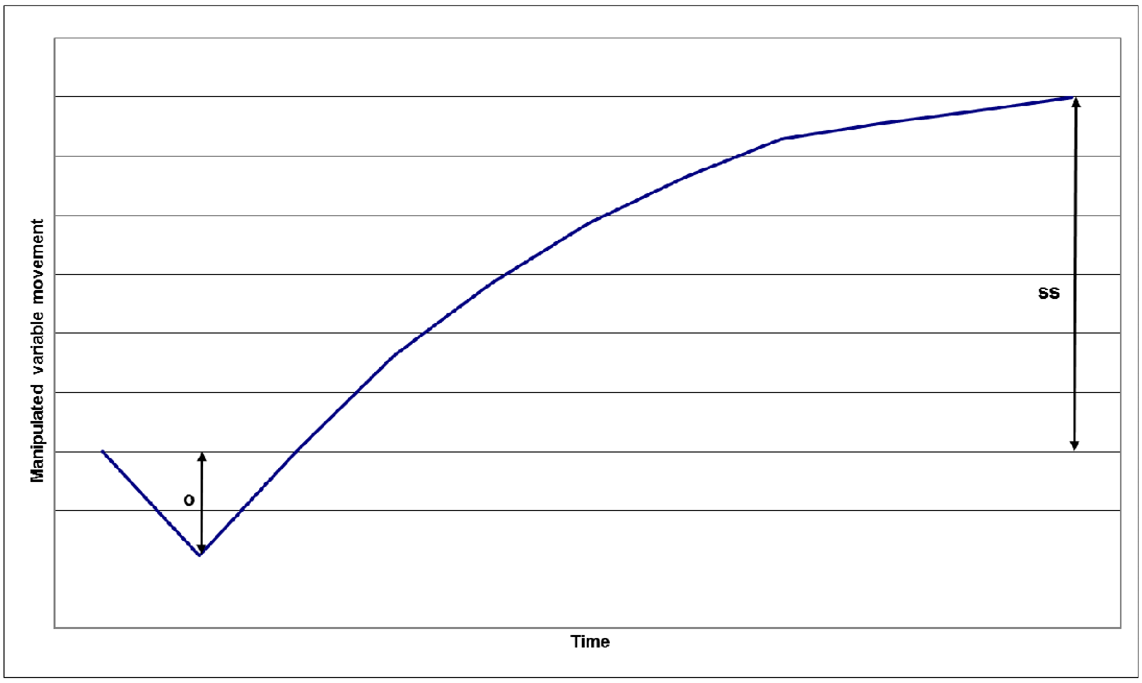


Figure B.1 Negative manipulated variable overshoot

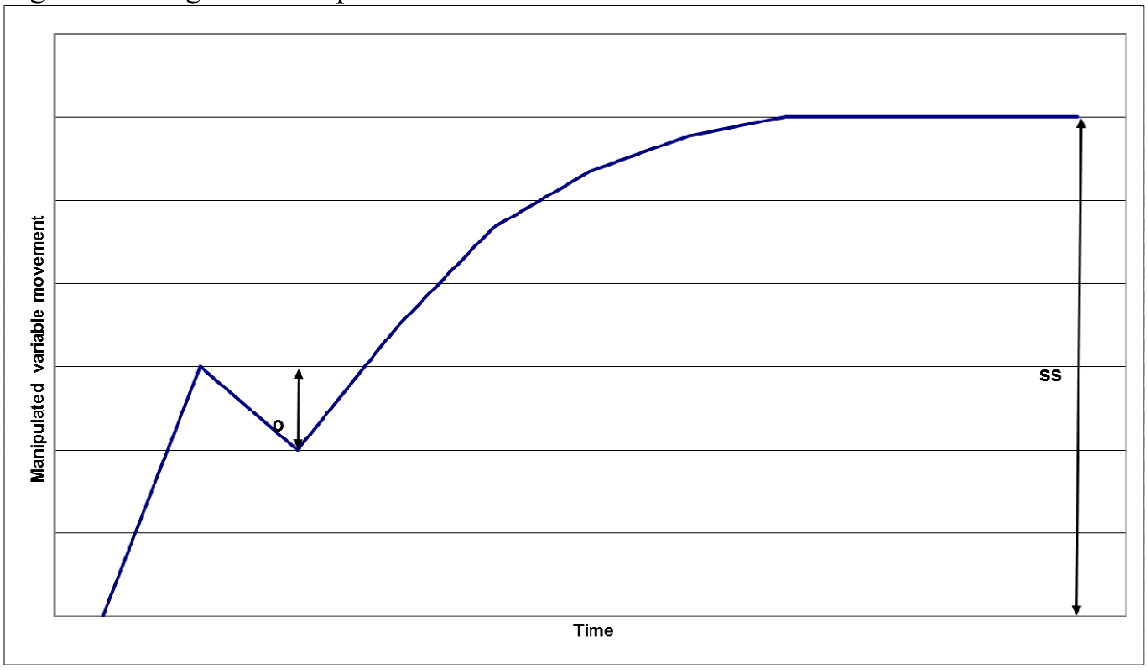


Figure B.2 Dynamic manipulated variable overshoot

Appendix C – Plants and models used in tuning exercises.

Reactor simulation

The plant consists of 4 continuous stirred tank reactors in series, or one reactor with 4 chambers, with a preheated feed undergoing an exothermic reaction. Each of the 4 chambers has a cooling water coil with an associated flow controller. The feed flows into the first chamber, and from there overflows into the next. Product is let out of the last chamber under level control.

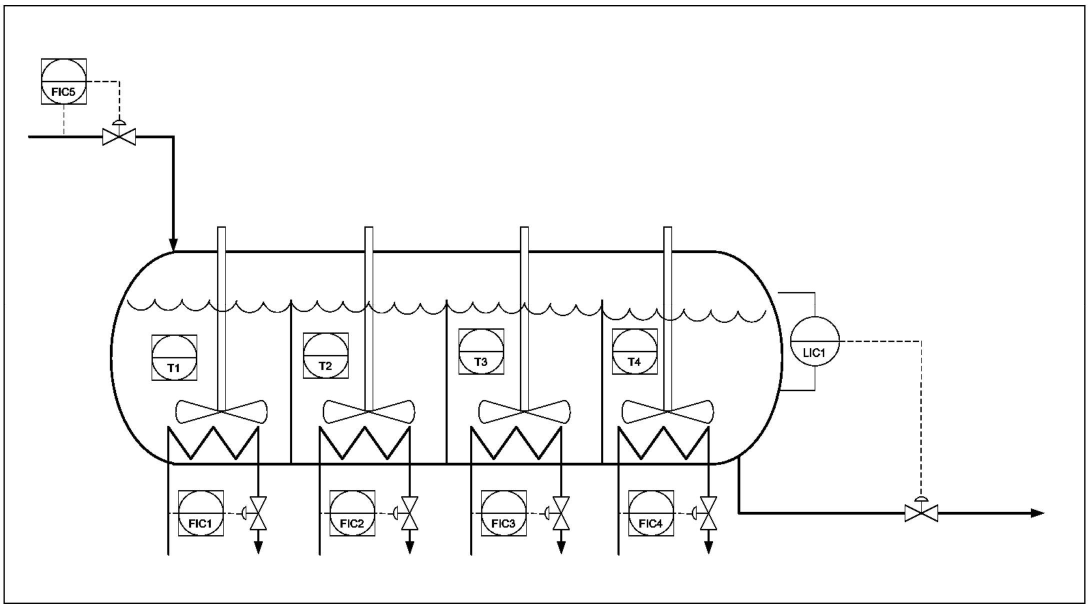


Figure C1 Reactor process flow diagram

An increase in feed into the reactor increases the exotherm.

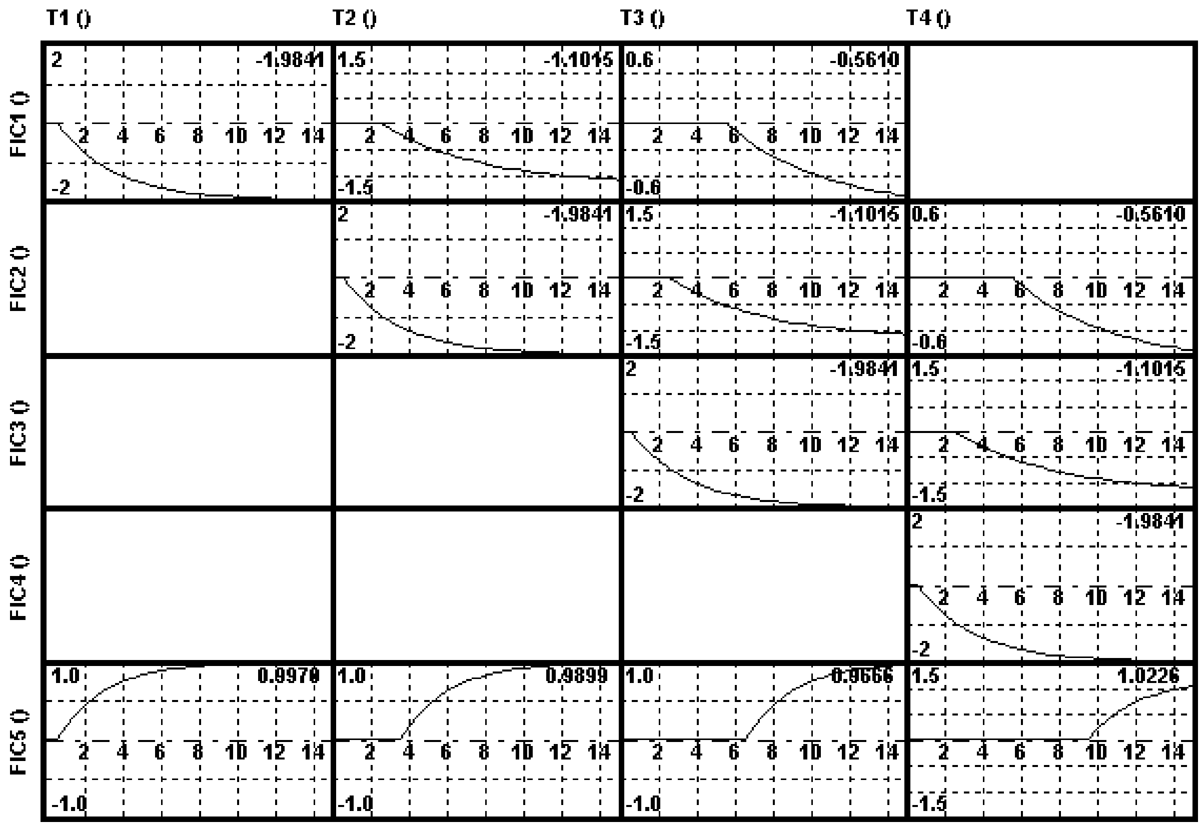


Figure C2. Reactor models

Distillation Simulation
This is a normal distillation column with 3 MVs and 3 CVs. MVs are feed, steam and reflux cooling water. CVs are top and bottom temperature, and column dP. The model is shown below.

This model was specifically chosen because of the ill conditioning that exists between the top and bottom temperature and the feed and steam manipulated variables. The topic of ill-conditioning will not be pursued here. For the purpose of this dissertation, it will suffice to say that the slightest model error will lead to very bad controller behaviour if ill-conditioning exists.

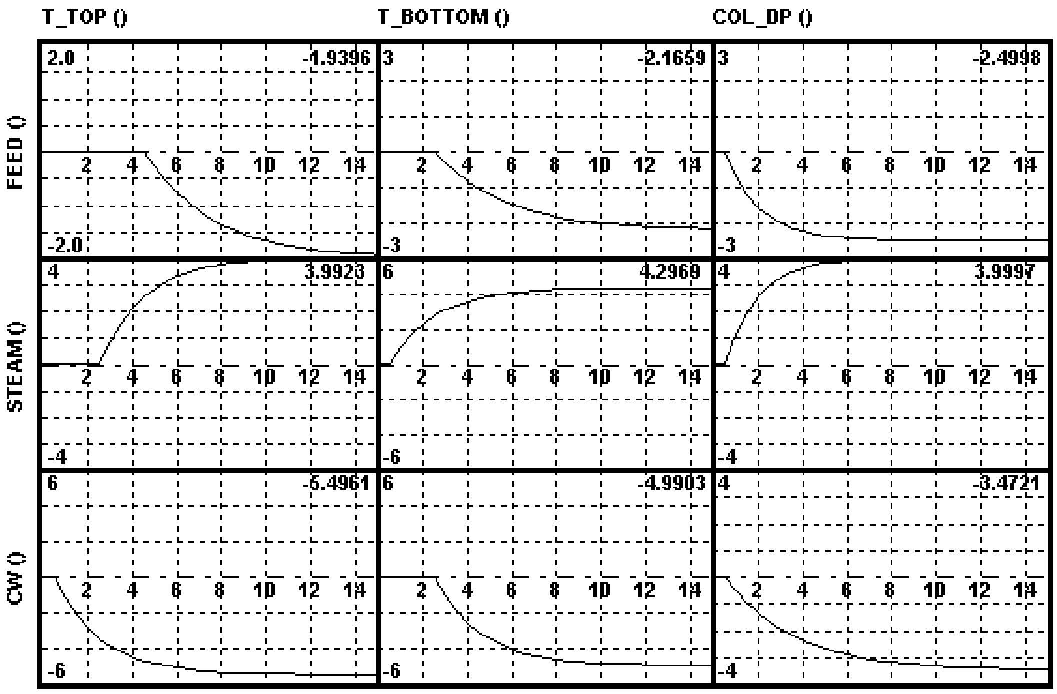


Figure C3 Distillation plant models

References

Chien, I.L. (1988) IMC-PID controller design – an extension. *Proceedings of the IFAC Adaptive Control of Chemical Processes Conference*, *Denmark*, pp.147-152

Chien, I.L. and Fruehauf, P.S. (1990) Consider IMC tuning to improve controller performance. Chemical Engineering Progress, 86, pp.33-41

Cutler, C.R. (1982) Dynamic Matrix Control of Imbalanced Systems. *ISA Transactions*, Vol. 21 No. 1 pp1-6

Cutler, C.R. and Perry, R.T. (1983) Real time optimization with multivariable control is required to maximize profits. *Computers and Chemical Engineering* Vol. 7, No. 5, pp. 663~67, 1983

Cutler, C.R., Haydel, J.J., Morshedi, A.M. (1983) An Industrial Perspective on Advanced Control *Annual Meeting - American Institute of Chemical Engineers*

Dahlin, E.B. (1968) Designing and tuning digital controllers*, Instruments and Control Systems*, 2(6), pp.77-83

Garcia, C.E. and Morari, M. (1985) Internal Model Control. 2. Design Procedure for Multivariable Systems, *Ind. Eng. Chem. Process Des. Dev.*, 24, pp. 472 - 484

Genceli, H. and Nikolau, M. (1993) Robust Stability Analysis of Constrained l1-norm Model Predictive Control. *AIChE J.*, December 1993, Vol. 39, pp. 1954-1965

Ghazzawi, A., Ali, E., Nouh, A. and Zafirou, E., (2010) On-line tuning strategy for model-predictive controllers, *Journal of Process Control 11*, pp.265-284

Iglesias, E.J., Sanjuán, M.E. and Smith, C.A (2006) *Ingenaria & Desarrollo*, Vol. 19, pp. 88-100

Kai Han, Jun Zhao, Jixin Qian (2006) A Novel Robust Tuning Strategy for Model Predictive Control *Proceedings of the 6th World Congress on Intelligent Control and Automation, June 21 - 23, 2006, Dalian, China*

King, M. (2011) Process Control A Practical Approach, Wiley, pp. 60-65

Nelder, J.A. and R. Mead, "A Simplex Method for Function Minimization", *The Computer Journal* (1965) 7 (4): pp. 308-313.

Nikolaou, M (2001) Model Predictive Controllers: A Critical Synthesis of Theory and Industrial Needs. *Advances In Chemical Engineering*, Vol. 26 pp. 131-204

Prett, D. M., & Gillette, R. D. (1980). Optimization and constrained multivariable control of a catalytic cracking unit. *Proceedings of the joint automatic control conference.*

Qin, S.J. and Badgwell, T.A. (2003) A survey of industrial model predictive control technology *Control Engineering Practise 11*, pp. 733-764

Shridar, R. and Cooper, D.J. (1998) A novel tuning strategy for multivariable model predictive control, *ISA Transactions*, Vol. 36, No. 4. pp. 273-280