University of Alberta Computer Process Control Group

Multivariate Controller Performance Assessment

Limited Trial Version

Written by: CPC Control Group, University of Alberta

Version 2.1

Table of Contents

List of Figures

List of Tables

Introduction

The multivariate controller performance assessment toolbox was developed by the Computer Process Control Group at the University of Alberta to allow performance assessment to be performed using MATLAB using the Filtering and Correlation (FCOR) Algorithm.

A "Quick Start" approach to using this toolbox will be presented, along with a detailed section containing full explanations and examples for using this toolbox.

System Requirements

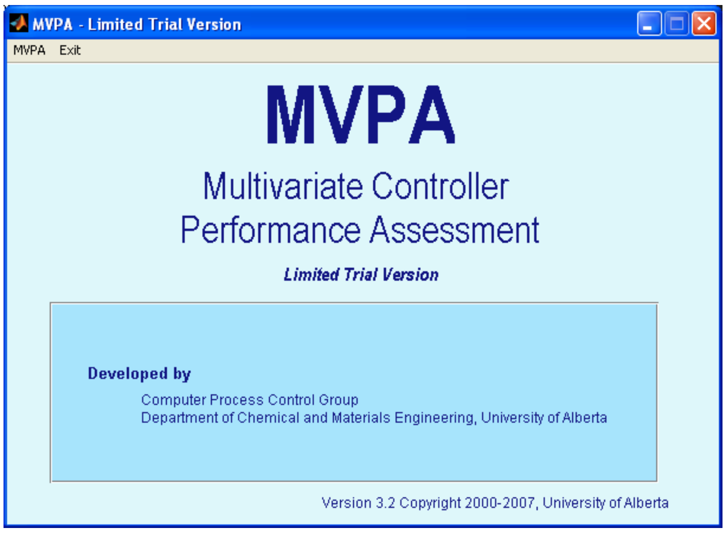
In order to run this toolbox properly, the following programmes are required:

1) MATLAB 2006a (MATLAB 7.1) or better. It should be noted that the newest version of MATLAB (MATLAB 2008a) makes the toolbox run slower.
2) The SYSTEM IDENTIFICATION TOOLBOX from MATLAB is required.

Quick Start

For quickly using the toolbox, the following steps should be followed:

1) Unzip the files to the desired location.
2) Start MATLAB, and point the current directory to the location of the unzipped files.
3) At the command prompt, type "**`>> main_mvpa`**" to start the toolbox. The GUI shown in Figure 1 should appear.
4) Press the "MVPA" menu. A new GUI will appear that is shown in Figure 2.


MVPA - Limited Trial Version

MVPA Exit

MVPA

Multivariate Controller Performance Assessment

Limited Trial Version

Developed by

Computer Process Control Group
Department of Chemical and Materials Engineering, University of Alberta

Version 3.2 Copyright 2000-2007, University of Alberta

Figure 1: The First GUI that appears

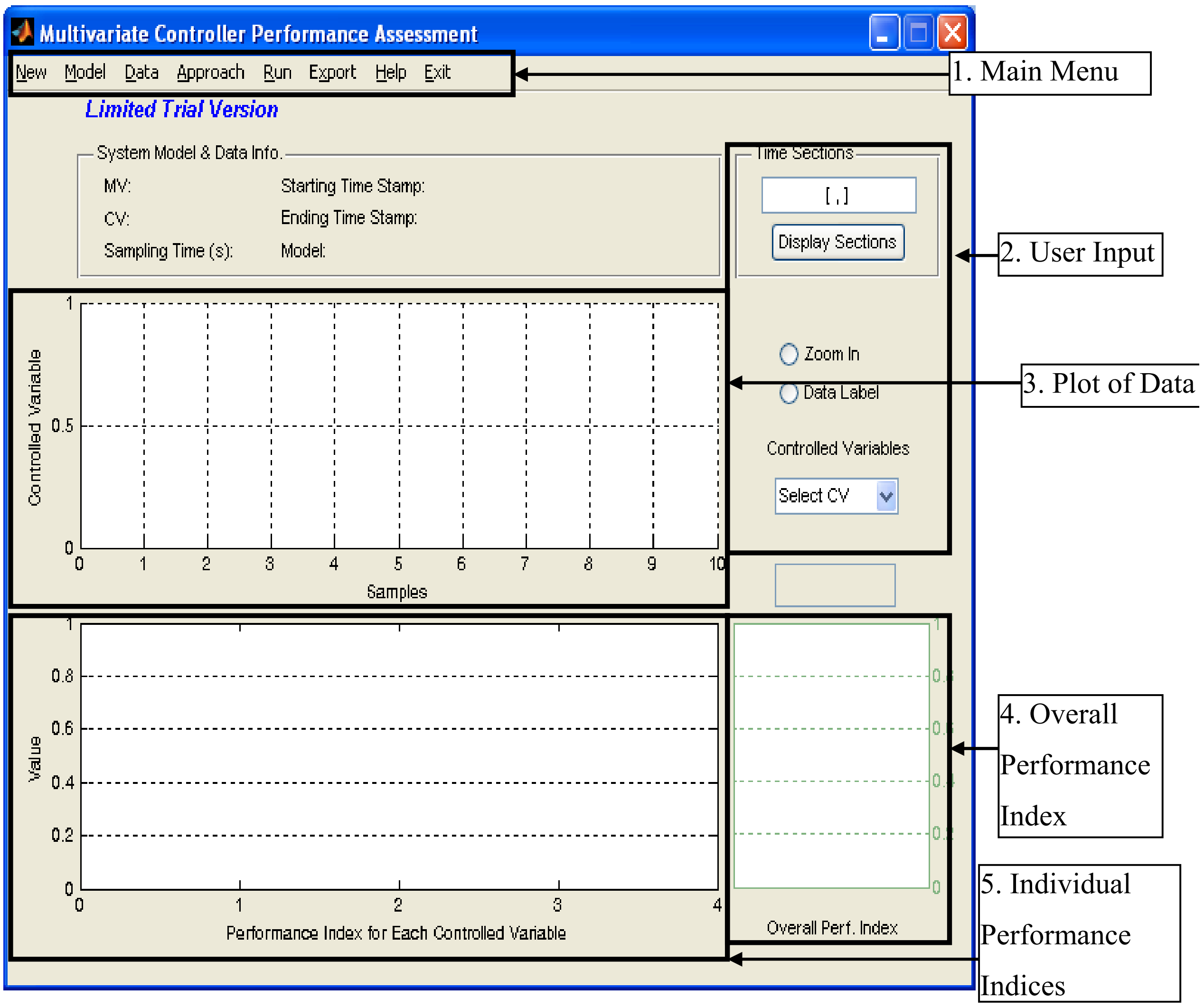
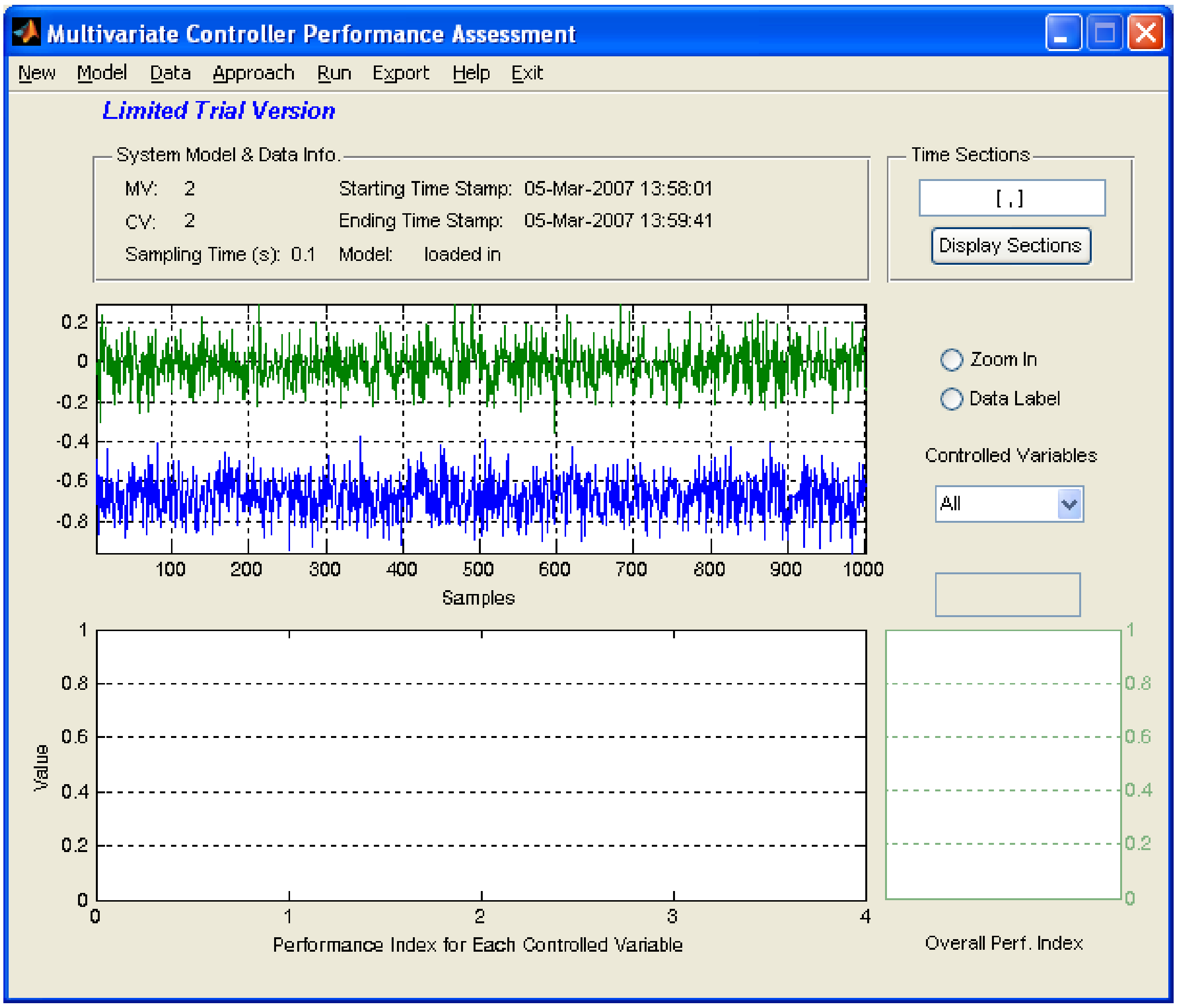


Figure 2: The main GUI for this toolbox, with the main regions highlighted

5) For the purpose of this quick start, we will be using the sample data provided in the zip file. Go to the "Model" Menu located in the Main Menu area, which is denoted as 1 in Figure 2. Select the "Case_Model.mat" file. Be patient as the file is loaded. This file contains the **open-loop model** for the process. Then go to the "Data" Menu located in the same area. Select the "Compact" option and, finally, select the "Case_Compact_Data.mat" file. Wait patiently as the file is loaded.
6) After a few seconds, the data should be loaded and a graph of the data should appear in region 5 of Figure 2. The resulting screenshot is shown in Figure 3.


Multivariate Controller Performance Assessment
New Model Data Approach Run Export Help Exit
Limited Trial Version
Time Sections
[,]
Display Sections
Zoom In
Data Label
Controlled Variables
All
Samples
Value
Performance Index for Each Controlled Variable
Overall Perf. Index

System Model & Data Info.

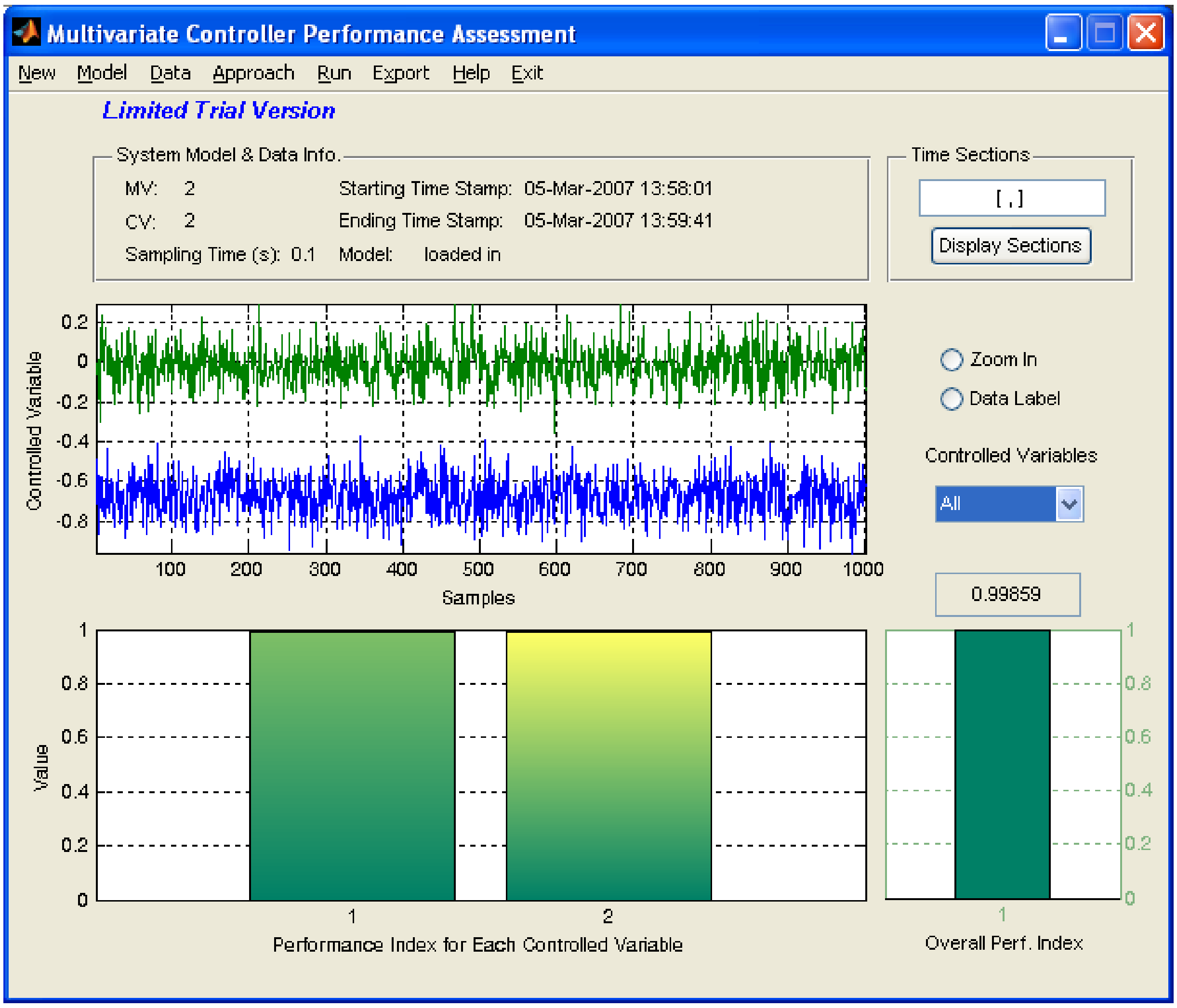
MV: 2 Starting Time Stamp: 05-Mar-2007 13:58:01

CV: 2 Ending Time Stamp: 05-Mar-2007 13:59:41

Sampling Time (s): 0.1 Model: loaded in

Figure 3: Screen shot after loading the data

7) At this point, the data can be zoomed in, but not out and a portion of the graph selected. This can be accomplished by changing the values in section 2 of Figure 2. For this case, nothing will be done.
8) Once this has been entered, click the “Run” menu, which is found in the Main Menu region of Figure 2.
9) After a few minutes, the screen should become similar to that shown in Figure 4.



System Model & Data Info.

MV: 2 Starting Time Stamp: 05-Mar-2007 13:58:01

CV: 2 Ending Time Stamp: 05-Mar-2007 13:59:41

Sampling Time (s): 0.1 Model: loaded in

0.99859

Figure 4: Screen shot after clicking "Run"

10) Figure 4 shows that using the given controllers, the overall performance index is approximately 1. This implies that the given controller, compared to a minimum variance controller, is almost the same, which implies that the control is optimal. Furthermore, the performance indices for each of the output are also close to 1. This would also suggest that these variables are running optimally.

11) To save the current model, click the “Export” Menu located in the Main Menu area and choose an appropriate name for the file.

12) To exit the toolbox, click the “Exit” Menu. This will only close the second GUI that appeared. To close the first GUI that appeared (Figure 1), click its “Exit” Menu. An error message may appear, but it can safely be ignored.

Detailed Instructions

Theory

The basic theory for multivariate controller performance assessment is similar to the univariate controller performance assessment, except that now more than one variable is being controlled and could potentially be assessed. The overall performance index is defined as:

$$\eta = \frac{\operatorname{trace}\left(\tilde{\Sigma}_{mv}\right)}{\operatorname{trace}\left(\tilde{\Sigma}_{y}\right)} \tag{2.1}$$

where Σ_y is the actual covariance of the process determined based on the process data or through time series modelling and $\tilde{\Sigma}_{mv}$ is the covariance matrix under minimum variance control, which is defined as

$$\tilde{\Sigma}_{mv} = \sum_{i=0}^{d-1} F_i \Sigma_e F_i^T \tag{2.2}$$

where Σ_e is the covariance matrix of the error (or residuals or noise if the data is fitted), *d* is the time delay, and F_i are the Markov parameter (infinite impulse response model) of the process, that is,

$$q^{-d} D G_{cl} = \sum_{i=0}^{\infty} F_i q^{-i} \tag{2.3}$$

with *D* being the interactor matrix. The interactor matrix is calculated using either the open-loop process transfer function or determined from closed-loop experimental data.

If it is desired to determine the performance of each individual component of the process, then the following formulae can be used:

$$\eta_{1:m} = \operatorname{diag}\left(\operatorname{diag}\left(\operatorname{diag}\left(\Sigma_{mv}\right)\right)\operatorname{diag}\left(\operatorname{diag}\left(\Sigma_{y}\right)\right)^{-1}\right) \tag{2.4}$$

where "diag" is a MATLAB function that takes the diagonal of a matrix or creates a diagonal matrix from the given entries, and Σ_{mv} is calculated as

$$\Sigma_{mv} = \sum_{i=0}^{d-1} E_i \Sigma_e E_i^T \tag{2.5}$$

with

$$\left[E_0, E_1, E_2, \ldots, E_{d-1}\right] = \begin{bmatrix} D_0 \\ D_1 \\ \vdots \\ D_{d-1} \end{bmatrix}^T \begin{bmatrix} F_0 & F_1 & \cdots & F_{d-1} \\ F_1 & F_2 & \cdots & 0 \\ \vdots & & 0 & 0 \\ F_{d-1} & 0 & \cdots & 0 \end{bmatrix} \quad (2.6)$$

and

$$D = \sum_{i=0}^{d-1} D_i q^{d-i} \quad (2.7)$$

The following algorithm can be followed in analysing the performance of a process (Huang & Kadeli, Dynamic Modeling, Predictive Control and Performance Monitoring, 2008):

1) Obtain operating data from the desired, closed-loop system, when there is no set point change. The mean should be subtracted from the data.
2) Perform a time series analysis of the data to obtain an appropriate model of the system. The model should be stable, since the original closed-loop process is itself stable.
3) Obtain the interactor matrix, *D*, for the (open-loop) process.
4) Obtain the *F*'s using Equation (2.3).
5) Obtain the covariance matrix of the noise using the model residuals.
6) Calculate the minimum variance matrix using Equation (2.2).
7) The overall performance index is given by Equation (2.1).
8) Rewrite the interactor matrix using Equation (2.7).
9) Obtain the required *E*'s using Equation (2.6).
10) Obtain the minimum variance matrix of the original output y_t using Equation (2.5).
11) Obtain the individual performance indices using Equation (2.4).

Data Storage

A total of 2 files are required to store the data for the analysis. The first file stores the information required to describe the model, while the second file stores the data required to describe the data that is to be analysed using the model. For a quick generation of the compact data storage objects, the file "gen_cmpct_Data.p" can be used.

Model Storage Format

The model storage file contains information about the multivariate model that is being used to describe the system. It must be entered as a transfer function object (tf). This matrix can be created using the following steps:

1) Assume that a process with *n* inputs and *m* outputs is being analysed. The transfer function between the *j*th input and the *i*th output is a rational function of either *s* or *z*, given as A_{ij} / B_{ij}, where *A* is the numerator and *B* is the denominator. Let *A'* be a row vector containing in descending order of powers of *s* or *z* the corresponding co-efficients of the numerator *A*. Similarly, let *B'* be the corresponding row vector of co-efficients of *s* or *z* for the denominator.
2) It is now necessary to create 2 cell arrays that contain all the information about the process. The first cell array, $\mathbb{A}$, is created as follows:

$$\begin{aligned}\mathbb{A} = \{&A_{11}^{'}, A_{12}^{'}, \cdots, A_{1n}^{'};\\ &A_{21}^{'}, A_{22}^{'}, \cdots, A_{2n}^{'};\\ &\vdots \qquad\qquad \vdots \;;\\ &A_{m1}^{'}, A_{m2}^{'}, \cdots, A_{mn}^{'}\}\end{aligned}$$

   The first row contains the transfer function between the first output and each of the inputs, while the second row contains the transfer function between the second output and each of the inputs. It should be noted that the curly brackets, {}, are used to create a cell array. In order to access, the (*i*, *j*) entry of the array, the command would be $\mathbb{A}\{i, j\}$. A similar cell array containing the values of the denominator, denoted as $\mathbb{B}$, is also created.
3) The time delay for the model can be created by forming the matrix $\mathbb{D}$, such that the (*i*, *j*) entry contains the time delay associated with the transfer function for the *i*th output and *j*th input. The array simply contains the numeric values.
4) The continuous transfer object can then be created using the following MATLAB command: "**`>>Atf=tf(A,B,'ioDelay',D)`**", where **`A`** is the cell array $\mathbb{A}$, **`B`** is the cell array $\mathbb{B}$, and **`D`** is the time-delay matrix, $\mathbb{D}$. To create a discrete time transfer function, the MATLAB command would be "**`>>Atf=tf(A,B,Ts,'ioDelay',D)`**", where **`Ts`** is the sampling time. If it is desired to create a discrete model that contains

powers of "z^{-1}", then the command would be

"`>>Atf=tf(A,B,Ts,'ioDelay',D,'Variable','z^-1')`"

Output Data Storage Formats

There are 2 different methods to store the output data that is required for the toolbox: comprehensive and compact.

Comprehensive

Assume that there are p samples of output data with a total of t controlled variables and s manipulated variables with a total of q tags. In the comprehensive data storage method, the data is stored as an object containing the following entries:

1) **controller_Status**: This is a p-by-1 double matrix that contains the status of each of the controllers, where 1 represents a controller that is "on" and 0 represents a controller that is "off."
2) **cell_char_TagList**: This is a q-by-1 cell matrix that contains the name of each of the tags that are presented in the process.
3) **cell_char_TimeStamp**: This is a p-by-1 cell matrix that contains the time stamp for each of the samples.
4) **dbl_Comprehensive_Data**: This is a p-by-q double matrix that contains the values for each of the tags and sample periods.
5) **dbl_SamplingTime**: This is a scalar double that contains the sampling time for the process.
6) **int_CVNumber**: This is a scalar integer that contains the number of controlled variables in the process, that is, t.
7) **int_MVNumber**: This is a scalar integer that contains the number of manipulated variables in the process, that is, s.
8) **status**: This is a p-by-($s + t$) double matrix that stores the data in the following manner: The first t columns contain the status of the controller variables, while the remaining s columns contain the status of the manipulated variables. A value of 1 signifies that the data is good.

Compact

The compact data storage method is very similar to the comprehensive method except that instead of storing all the tags for a given process, only the tags for the controlled variables are stored. This implies that q is equal to t. The same fields as for the comprehensive method are used, except that "dbl_Comprehensive_Data" is now appropriately called "dbl_Compact_Data".

Data Generation

The data storage files for multivariate analysis can be generated using the *p*-file "gen_cmpct_data.p". The following steps should be followed to create a compact data set.

1) Start MATLAB and point the directory to the location of the above binary file.
2) At the command prompt create the following data matrix, which contains the 3 sample values of the process for the 2 controlled variables: "“**`>> W=[1 2;3 4;5 6]`**".
3) It will be assumed for the sake of this example that the process to be entered can be described as

$$y=\begin{bmatrix} \frac{2.3}{s^2+1}e^{-5s} & \frac{2s+1}{s^2+3s+1}e^{-1s} \\ \frac{5}{s+1}e^{-1s} & \frac{4s}{s^2+3s+1}e^{-3s} \end{bmatrix}$$

   Thus, the required arrays and matrices for the transfer function will be created as follows:

   a) Form the 4 *A*-matrices as "**`>>A11=[2.3]; A12=[2 1]; A21=[5]; A22=[4 0];`**".
   b) Form the 4 *B*-matrices as "**`>>B11=[2 0 1]; B12=[1 3 1]; B21=[1 1]; B22=[1 3 1];`**".
   c) Create the *A*-cell array as "**`>>A={A11,A12;A21,A22};`**".
   d) Create the *B*-cell array as "**`>>B={B11,B12;B21,B22};`**".
   e) The time delay matrix can be created as "**`>>D=[5 1;1 3];`**". It should be noted that the negative signs in the time delay have been ignored.
   f) The transfer function object is then created using the following command: "**`Atf=tf(A,B, 'ioDelay',D)`**". The resulting transfer function should match the one given above.

4) Type at the command prompt: "“**`>> gen_cmpct_Data`**". The following should appear

```
Consider a process with m inputs and p outputs
1. p x m plant continuous transfer
matrix/discrete transfer function
2. Output data (y): N x p matrix
3. Sampling time
Continuous/discrete transfer function matrix:
```

5) Type “Atf” or the name of the transfer function object that was created above, and press enter. This is the transfer function object that contains the open-loop plant model.
6) Next, type “W” and press enter. This is the sampled data matrix for the given process.
7) Finally, type “0.1” and press enter. It will be assumed that the sampling time is 0.1 seconds.
8) Enter a name for the output data storage file, say, for example, “test_data.mat” and press enter. **It should be noted that files that have the same name as those currently present in the directory will be overwritten without any warning**.
9) Enter a name for the model storage file, say, for example, “test_model.mat” and press enter. **It should be noted that files that have the same name as those currently present in the directory will be overwritten without any warning**.
10) In order to view the results, type “`>>load test_model.mat`” followed by ““`>>sys_MPC_Model`”. The results should be identical to that displayed below:

```
Transfer function from input 1 to output...
                        2.3
 #1:  exp(-5*s) * ------------
                     2 s^2 + 1
                         5
 #2:  exp(-1*s) * -----------
                       s + 1
Transfer function from input 2 to output...
                       2 s + 1
 #1:  exp(-1*s) * ------------------
                     s^2 + 3 s + 1
```

```
                       4 s
#2:  exp(-3*s) * ----------------
                   s^2 + 3 s + 1
```

11) In order to view the other file type "**`>>load test_data.mat`**". This will load each of the entries into the workspace of MATLAB. Entering the name of each of the entries given below should give the same results as are presented in Table 1.

Table 1: Summary of the Parameters Generated using the "gen_cmpct_Data" command

Entry	Value
Controller_status	[1 1 1]
cell_char_TagList	['CV1' 'CV2']
cell_char_TimeStamp	It should give the current time incremented by 0.1, 0.2, and 0.3
dbl_Compact_Data	[1 2 3 4 5 6]
dbl_SamplingTime	0.1000
int_CVNumber	2
int_MVNumber	2
status	[1 1 2 2 1 1 2 2 1 1 2 2]

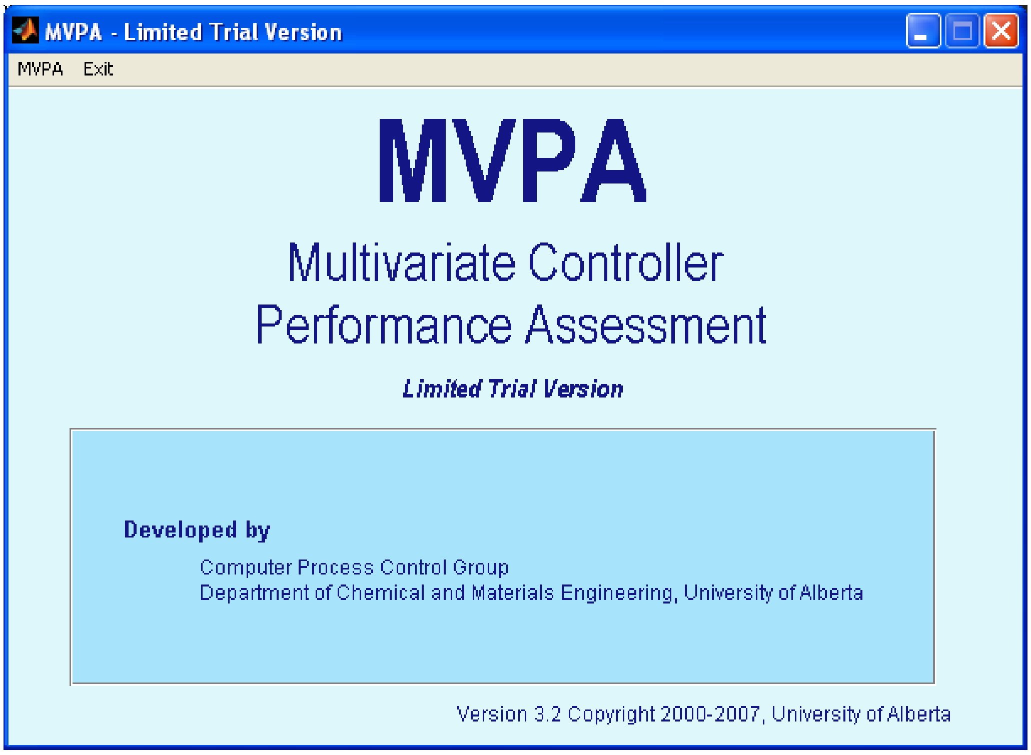
Using the Toolbox

Installation

The toolbox can be installed by simply unzipping the files to any desired location. In order for the toolbox to function properly, the System Identification Toolbox should be installed.

Starting the Toolbox

The toolbox can be accessed from MATLAB using the following sequence of commands. First MATLAB itself should be started and the directory pointed to the folder containing the files for this toolbox. Next, at the command prompt, type "**`>> main_mvpa`**". The GUI shown in Figure 5 should appear. This GUI is the main access to the toolbox. To start a session of the toolbox, click on the "MVPA" menu. This will bring up a new GUI, which is shown in Figure 6. In Figure 6, each of the main parts of the GUI is highlighted and will be discussed separately.



MVPA

Multivariate Controller Performance Assessment

Limited Trial Version

Developed by

Computer Process Control Group
Department of Chemical and Materials Engineering, University of Alberta

Version 3.2 Copyright 2000-2007, University of Alberta

Figure 5: The First GUI that appears

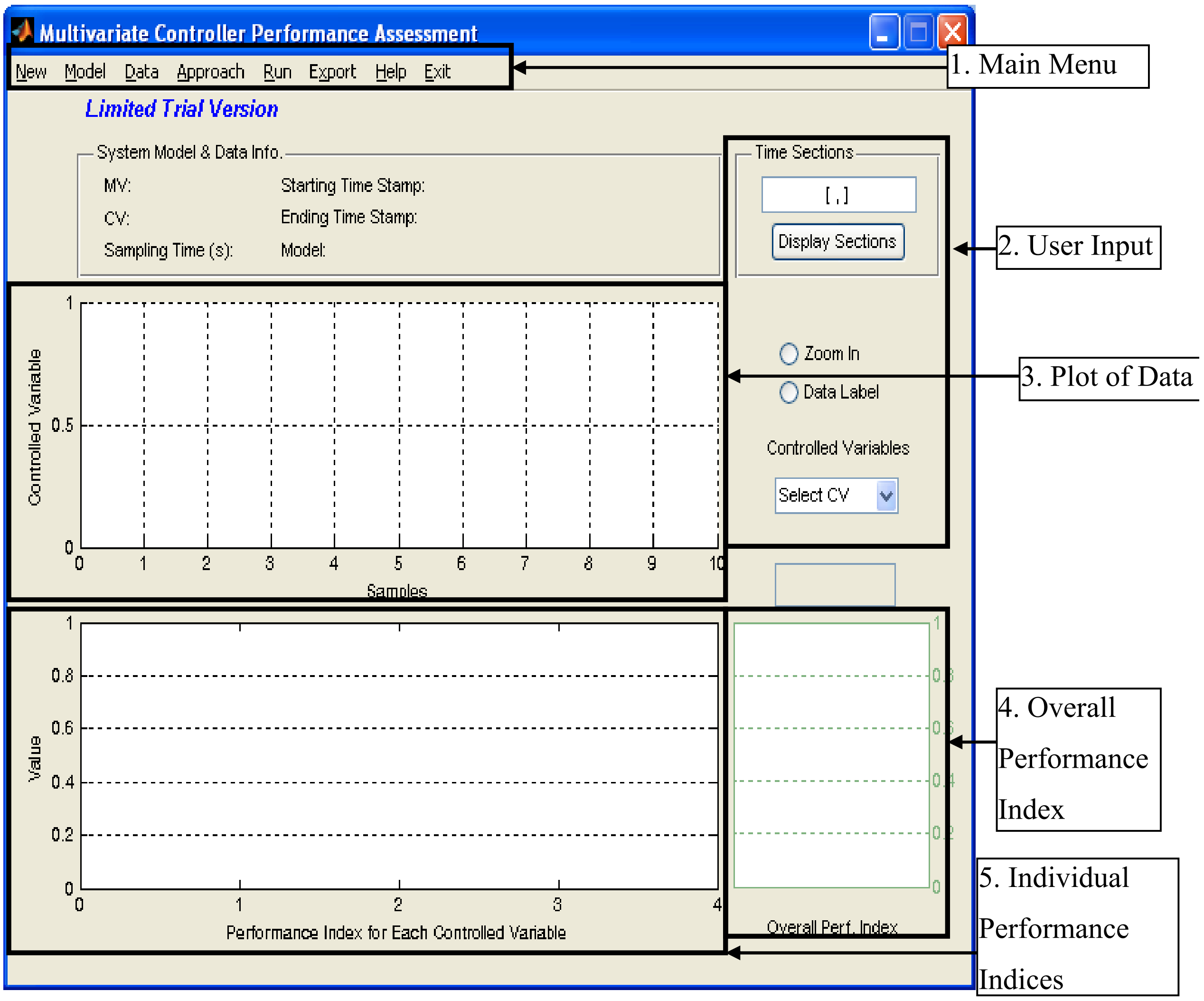


Figure 6: The main GUI for this toolbox, with the main regions highlighted

In-depth Discussion of the Toolbox

Section 1: Main Menu

A close-up of the Main Menu section is given in Figure 7. The Main Menu consists of the following 7 areas:

1) **New**: Clicking this menu will clear all the data from the current GUI and allow the user to restart the analysis from a clean layout.
2) **Model**: Clicking this menu will allow the user to select the open-loop model file that will be used in the analysis.
3) **Data**: Clicking this menu brings up a drop-down list that has 2 choices:

a. **Compact**: This assumes that the data to be used is stored using compact data storage.

b. **Comprehensive**: This assumes that the data to be used is stored using comprehensive data storage.

4) **Approach**: This allows the user to select what approach is going to be used to analysis the system. Currently, there is only a single choice: FCOR.
5) **Run**: Clicking this menu will cause the toolbox to analysis the data that has been selected.
6) **Export**: Clicking this menu will allow the current analysis to be saved as a ".mat" file.
7) **Exit**: Clicking this menu will cause the toolbox to close.

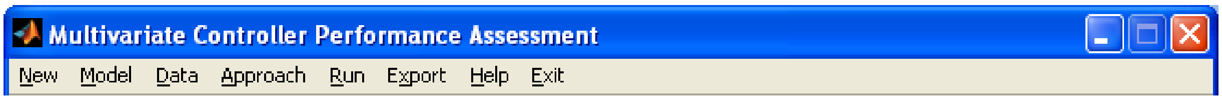

Multivariate Controller Performance Assessment
New Model Data Approach Run Export Help Exit

Figure 7: Close-up of the Main Menu

Section 2: User Input

In this section, the user can enter information about the process. There are 3 potential areas that can be changed:

1) **Time Sections**: In this filed, the user can specify what part of the total data is to be used in the analysis. The format for this entry is "[start sample value, end sample value]". A comma must separate the 2 values and the end value must be greater than the start value. As well, there must a sufficient number of data samples in order for the computer to estimate a model. It is possible to try more than 1 section simultaneously. Each sample section is separated by a semicolon (;). For example "[1,100; 40, 500]" will consider 2 sections; the first ranging from sample 1 to sample 100 and the second from sample 40 to sample 500. Clicking "Display" will display the selected samples in the data plot area (Section 5).
2) **Zoom In**: This allows the user to zoom in onto a certain section of the graph. However, there is no zoom-out button, so there is no way to undo the command.
3) **Select CV**: This allows the user to select which controlled variable will be used for the analysis. A drop-down menu will list all the possible controlled variables.

Section 3: Plot of Data

This graph displays a time series plot of the values for all (or only those that are selected) controlled variable.

Section 4: Overall Performance Index

This graph displays the overall performance index for the process.

Section 5: Individual Performance Indices

This section displays a bar graph that displays the performance index for each controller.

Examples

The following examples will examine how the toolbox using MATLAB functions to solve the given examples. As well, a detailed explanation of how to perform the same using the toolbox is presented. In order to perform the analysis using MATLAB, the following functions from the System Identification Toolbox are required:

a. **`model = n4sid(z, order)`**: Takes the "iddata" object, **`z`**, and the desired **`order`** for the model to return a state-space estimated model saved as an "iddata" object, **`model`**, for the data. If for **`order`** is it entered **`'best'`**, then the best model with orders between 1 and 10 will be determined.

b. **`model = pem(z,Mi)`**: Takes the "iddata" object, **`z`**, and the specifications for the model, **`Mi`**, to return an estimation for the model saved as an "iddata" object, **`model`**. If **`Mi = [na nb nc nd nf nk]`**, where **`nk`** is the time delay, and the remaining entries are the orders of the polynomials in the following expression:

$$A(na)y_t = \frac{B(nb)}{F(nf)}u_t + \frac{C(nc)}{D(nd)}e_t \tag{4}$$

c. **`[num,dem]=ss2tf(A,B,C,D,i)`**: Takes the state-space model,

$$\begin{aligned} x_{t+1} &= Ax_t + Bu_t \\ y_t &= Cx_t + Du_t \end{aligned} \tag{5}$$

and for the i^{th} input, calculates the numerator and denominator of the transfer function.

d. **`tfmodel=tf(num,dem,'iodelay',delay)`**: Creates a continuous time transfer function given the numerator, **`num`**, denominator, **`dem`**, and time delay, **`delay`**, and returns the results as a transfer function object, **`tfmodel`**.

e. **`[y]=impulse(model)`**: This calculates the impulse response co-efficients given either a transfer function object (**`model`**) or a state space object (**`model`**), and returns the appropriate co-efficients.

2) Other Required Functions:

a. `Y=cov(A)`: Determines the covariance of a matrix **`A`** and returns the result as the covariance matrix **`Y`**.

b. `W=diag(A)`: Takes the diagonal elements of a matrix, **`A`**, and returns the row vector, **`W`**, containing the diagonal entries. This function also will take a row vector, **`A`**, and return a diagonal matrix, **`W`**, whose entries are the sequential entries of **`A`**.

The examples in this section will be based on the data found in the example folder. Given the nature of the process, with 6,001 data points, and large calculations, most of the steps will be carried out with the help of the computer. A parallel analysis using the toolbox will also be presented.

The data used in this example comes from the example presented in (Huang & Kadeli, Dynamic Modeling, Predictive Control and Performance Monitoring, 2008). It will be assumed that the time delay in the process is 2 samples.

In order to simplify the process, it will be assumed that a discrete model is available. The programme will accept either a continuous or discrete models. In this example, it will be assumed that the discrete, open-loop, transfer function is given as:

$$G_p\left(z^{-1}\right)=\begin{bmatrix}\dfrac{z^{-1}}{1-0.4z^{-1}} & \dfrac{4z^{-2}}{1-0.1z^{-1}}\\ \dfrac{0.3z^{-1}}{1-0.1z^{-1}} & \dfrac{z^{-2}}{1-0.8z^{-1}}\end{bmatrix} \tag{2.8}$$

The first step before any of the further examples can be done is to load the transfer function object into MATLAB. All the required data is found in the Examples folder. The following command will load the required model "**`>>load data.mat`**". The data will be stored in the vector "**`data`**". Once the data has been loaded the following tasks can be performed.

Part I: Interactor Matrix for this Process

For the purpose of this explanation, it will be assumed that nilpotent interactor matrix will be desired. The algorithm presented by (Rogoziński, Papliński, & Gibbard, 1987) will be used to derive the nilpotent interactor matrix.

From (2.8), the process transfer function is given as

$$G_p\left(z^{-1}\right)=\begin{bmatrix} \dfrac{z^{-1}}{1-0.4z^{-1}} & \dfrac{4z^{-2}}{1-0.1z^{-1}} \\ \dfrac{0.3z^{-1}}{1-0.1z^{-1}} & \dfrac{z^{-2}}{1-0.8z^{-1}} \end{bmatrix} \tag{2.8}$$

In order to apply the algorithm it must be converted into a function of *z* and made a proper transfer function. Thus, multiplying each entry of (2.8) by *z* / *z* gives

$$G_p\left(z\right)=\begin{bmatrix} \dfrac{1}{z-0.4} & \dfrac{4z^{-1}}{z-0.1} \\ \dfrac{0.3}{z-0.1} & \dfrac{z^{-1}}{z-0.8} \end{bmatrix} \tag{2.8.1}$$

The remaining negative value of *z* in the numerator can be removed by extracting a factor of z^{-1} from (2.8.1).

This gives

$$G_p\left(z\right)=z^{-1}\begin{bmatrix} \dfrac{z}{z-0.4} & \dfrac{4}{z-0.1} \\ \dfrac{0.3z}{z-0.1} & \dfrac{1}{z-0.8} \end{bmatrix}=z^{-1}T(z) \tag{2.8.2}$$

In order to apply the algorithm, the process transfer function without the time delay must be factored into parts, which are denoted in (Rogoziński, Papliński, & Gibbard, 1987) as *N*(*z*) and *D*(*z*), such that

$$T(z)=N(z)\left(D(z)\right)^{-1} \tag{2.8.3}$$

and *D*(*z*) is a diagonal matrix. One possible diagonalisation can be determined as

$$\begin{aligned} N(z)&=\begin{bmatrix} z\left(z-0.1\right) & 4\left(z-0.8\right) \\ 0.3z\left(z-0.4\right) & \left(z-0.1\right) \end{bmatrix}=\begin{bmatrix} z^2-0.1z & 4z-3.2 \\ 0.3z^2-0.12z & z-0.1 \end{bmatrix} \\ D(z)&=\begin{bmatrix} \left(z-0.1\right)\left(z-0.4\right) & 0 \\ 0 & \left(z-0.1\right)\left(z-0.8\right) \end{bmatrix} \end{aligned} \tag{2.8.4}$$

From *D(z)*, it can be concluded that the highest ordered polynomial is 2. Thus, $n = 2$. Next, the N matrix must be constructed, which is a stacked matrix containing in decreasing powers of z the co-efficients of the polynomials in N. In this case, N has the following form:

$$N = \begin{bmatrix} N_0 \\ \vdots \\ N_n \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0.3 & 0 \\ \cdots & \cdots \\ -0.1 & 4 \\ -0.12 & 1 \\ \cdots & \cdots \\ 0 & -3.2 \\ 0 & -.1 \end{bmatrix} \tag{2.8.5}$$

From (2.8.5), it can be concluded that

$$N_0 = \begin{bmatrix} 1 & 0 \\ 0.3 & 0 \end{bmatrix} \tag{2.8.6}$$

A summary of some further values is presented in Table 2.

Table 2: Summary of the parameters required to calculate the interactor matrix

Parameter	Value
n	2
m (number of columns in $T(z)$)	2
p (number of rows in $T(z)$)	2
min(m,p)	2

According to the algorithm presented in (Rogoziński, Papliński, & Gibbard, 1987), the following algorithm was defined:

Iteration 1:

1) **Check whether $r_i = \operatorname{rank}(N_0) = \min(m, p)$. If this is the case, the algorithm terminates.** In this the rank of N_0 is 1, which is not equal to min(m,p). Thus, it is necessary to proceed to the next step.
2) **Factorise N_0 using any appropriate methods (QR-factorisation will be used in this example), so as to obtain a *p*-x-*p* non-singular matrix $Q^{(i)}$.** Using MATLAB's `qr` function gives:

```
>>[Q1,R1]=qr([1 0;0.3 0])
Q1 =
   -0.9578   -0.2873
   -0.2873    0.9578
R1 =
   -1.0440         0
    0              0
```

3) **Premultiply *N*(*z*) by *Q*$^{(i)}$ to obtain** $\bar{N}(z)$**.** Thus,

$$\begin{aligned}\bar{N} &= Q^{(i)}N(z)\\ &= \begin{bmatrix} -0.9578 & -0.2873 \\ -0.2873 & 0.9578 \end{bmatrix}\begin{bmatrix} z^2-0.1z & 4z-3.2 \\ 0.3z^2-0.12z & z-0.1 \end{bmatrix}\\ &= \begin{bmatrix} -1.044z^2+0.1303z & -4.119z-3.094 \\ -0.0862z & -0.1914z-0.8236 \end{bmatrix}\end{aligned} \tag{2.8.1.1}$$

4) **Obtain the row shift polynomial matrix (r.s.p.m) for the process.** In this iteration, the value of $k_i = p - r_i = 2 - 1 = 1$. Thus, *U* equals

$$U = \begin{bmatrix} U_0 \\ U_1 \end{bmatrix} = \begin{bmatrix} 0_{r_i} \\ I_p \\ 0_{k_i} \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix} \tag{2.8.1.2}$$

This implies that *U*(*z*) can be obtained as

$$U(z) = U_0 z + U_1 = \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix} z + \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ z & 0 \end{bmatrix} \tag{2.8.1.3}$$

5) **Premultiply** $\bar{N}(z)$ **by *U* to obtain the *N*(z) matrix for the next iteration.** Thus,

$$\begin{aligned}N^{(i)}(z) &= U^{(i)}(z)\bar{N}(z)\\ &= \begin{bmatrix} 0 & 1 \\ z & 0 \end{bmatrix}\begin{bmatrix} -1.044z^2+0.1303z & -4.119z-3.094 \\ -0.0862z & -0.1914z-0.8236 \end{bmatrix}\\ &= \begin{bmatrix} -0.0862z & -0.1914z-0.8236 \\ -1.044z^3+0.1303z^2 & -4.119z^2-3.094z \end{bmatrix}\end{aligned} \tag{2.8.1.4}$$

Thus, $N^{(2)}$ is given as

$$N^{(1)} = \begin{bmatrix} 0 & 0 \\ 0.1303 & -4.119 \\ \cdots & \cdots \\ -0.0862 & -0.1914 \\ 0 & -3.094 \\ \cdots & \cdots \\ 0 & -0.8236 \\ 0 & 0 \end{bmatrix} \tag{2.8.1.5}$$

Therefore,

$$N_0^{(1)} = \begin{bmatrix} 0 & 0 \\ 0.1303 & -4.119 \end{bmatrix} \tag{2.8.1.6}$$

6) **The value of *S*(*z*) for the current iteration is found by multiplying *U*(*z*) and *Q*.** This gives,

$$\begin{aligned} S^{(i)}(z) &= U^{(i)}(z)Q^{(i)} \\ &= \begin{bmatrix} 0 & 1 \\ z & 0 \end{bmatrix} \begin{bmatrix} -0.9578 & -0.2873 \\ -0.2873 & 0.9578 \end{bmatrix} \\ &= \begin{bmatrix} -0.2873 & 0.9578 \\ -0.9578z & -0.2873z \end{bmatrix} \end{aligned} \tag{2.8.1.7}$$

Iteration 2:

1) **Check whether $r_i = \operatorname{rank}(N_0) = \min(m, p)$. If this is the case, the algorithm terminates.** Like before in iteration 1, the rank of N_0 is 1, which is not equal to min(*m*,*p*). Thus, it is necessary to proceed to the next step.
2) **Factorise N_0 using any appropriate methods (QR-factorisation will be used in this example), so as to obtain a *p*-x-*p* non-singular matrix $Q^{(i)}$.** Using MATLAB's `qr` function gives:

```
>>[Q2,R2]=qr([0 0;0.1303 -4.119])
Q2 =
     0    -1
    -1     0
R2 =
   -0.1303    4.1190
```

0 0

3) **Premultiply *N*(*z*) by *Q*$^{(i)}$ to obtain** $\bar{N}(z)$**.** Thus,

$$\begin{aligned}\bar{N} &= Q^{(i)}N(z)\\ &= \begin{bmatrix} 0 & -1 \\ -1 & 0 \end{bmatrix}\begin{bmatrix} -0.0862z & -0.1914z-0.8236 \\ 0.1303z^2 & -4.119z^2-3.094z \end{bmatrix}\\ &= \begin{bmatrix} 0.1303z^2 & 4.119z^2+3.094z \\ 0.0862z & 0.1914z+0.8236 \end{bmatrix}\end{aligned} \quad (2.8.2.1)$$

4) **Obtain the row shift polynomial matrix (r.s.p.m) for the process.** In this iteration, the value of $k_i = p - r_i = 2 - 1 = 1$. Thus, *U* equals

$$U = \begin{bmatrix} U_0 \\ U_1 \end{bmatrix} = \begin{bmatrix} 0_{r_i} \\ I_p \\ 0_{k_i} \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix} \quad (2.8.2.2)$$

This implies that *U*(*z*) can be obtained as

$$U(z) = U_0 z + U_1 = \begin{bmatrix} 0 & 0 \\ 1 & 0 \end{bmatrix} z + \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ z & 0 \end{bmatrix} \quad (2.8.2.3)$$

5) **Premultiply** $\bar{N}(z)$ **by *U* to obtain the *N*(z) matrix for the next iteration.** Thus,

$$\begin{aligned}N^{(i)}(z) &= U^{(i)}(z)\bar{N}(z)\\ &= \begin{bmatrix} 0 & 1 \\ z & 0 \end{bmatrix}\begin{bmatrix} 0.1303z^2 & 4.119z^2+3.094z \\ 0.0862z & 0.1914z+0.8236 \end{bmatrix}\\ &= \begin{bmatrix} 0.0862z & 0.1914z+0.8236 \\ 0.1303z^3 & 4.119z^3+3.094z^2 \end{bmatrix}\end{aligned} \quad (2.8.2.4)$$

Thus, $N^{(2)}$ is given as

$$N^{(2)} = \begin{bmatrix} 0 & 0 \\ 0 & 3.094 \\ \cdots & \cdots \\ 0.0862 & 0.1914 \\ 0 & 0 \\ \cdots & \cdots \\ 0 & 0.8236 \\ 0 & 0 \end{bmatrix} \quad (2.8.2.5)$$

Therefore,

$$N_0^{(2)} = \begin{bmatrix} 0 & 0 \\ 0 & 3.094 \end{bmatrix} \tag{2.8.2.6}$$

6) **The value of *S*(*z*) for the current iteration is found by multiplying *U*(*z*) and *Q*.** This gives,

$$\begin{aligned} S^{(i)}(z) &= U^{(i)}(z)Q^{(i)} \\ &= \begin{bmatrix} 0 & 1 \\ z & 0 \end{bmatrix}\begin{bmatrix} 0 & -1 \\ -1 & 0 \end{bmatrix} \\ &= \begin{bmatrix} -1 & 0 \\ 0 & -z \end{bmatrix} \end{aligned} \tag{2.8.2.7}$$

Iteration 3:

1) **Check whether $r_i = \text{rank}\left(N_0\right) = \min\left(m,p\right)$. If this is the case, the algorithm terminates.** Like before in iteration 2, the rank of N_0 is 1, which is not equal to min(*m*,*p*). Thus, it is necessary to proceed to the next step.
2) **Factorise N_0 using any appropriate methods (QR-factorisation will be used in this example), so as to obtain a *p*-x-*p* non-singular matrix $Q^{(i)}$.** Using MATLAB's `qr` function gives:

```
>> [Q2,R2]=qr([0 0;0 3.094])
Q2 =
     1     0
     0     1



R2 =
         0         0
         0    3.0940
```

3) **Premultiply *N*(*z*) by $Q^{(i)}$ to obtain $\overline{N}(z)$.** Thus,

$$\begin{aligned}\bar{N} &= Q^{(i)}N(z)\\ &= \begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix}\begin{bmatrix}0.0862z & 0.1914z+0.8236\\ 0 & 3.094z^2\end{bmatrix}\\ &= \begin{bmatrix}0.0862z & 0.1914z+0.8236\\ 0 & 3.094z^2\end{bmatrix}\end{aligned} \tag{2.8.3.1}$$

4) **Obtain the row shift polynomial matrix (r.s.p.m) for the process.** In this iteration, the value of $k_i = p - r_i = 2 - 1 = 1$. Thus, U equals

$$U = \begin{bmatrix}U_0\\ U_1\end{bmatrix} = \begin{bmatrix}0_{r_i}\\ I_p\\ 0_{k_i}\end{bmatrix} = \begin{bmatrix}0 & 0\\ 1 & 0\\ 0 & 1\\ 0 & 0\end{bmatrix} \tag{2.8.3.2}$$

This implies that $U(z)$ can be obtained as

$$U(z) = U_0 z + U_1 = \begin{bmatrix}0 & 0\\ 1 & 0\end{bmatrix}z + \begin{bmatrix}0 & 1\\ 0 & 0\end{bmatrix} = \begin{bmatrix}0 & 1\\ z & 0\end{bmatrix} \tag{2.8.3.3}$$

5) **Premultiply $\bar{N}(z)$ by U to obtain the N(z) matrix for the next iteration.** Thus,

$$\begin{aligned}N^{(i)}(z) &= U^{(i)}(z)\bar{N}(z)\\ &= \begin{bmatrix}0 & 1\\ z & 0\end{bmatrix}\begin{bmatrix}0.0862z & 0.1914z+0.8236\\ 0 & 3.094z^2\end{bmatrix}\\ &= \begin{bmatrix}0 & 3.094z^2\\ 0.0862z^2 & 0.1914z^2+0.8236z\end{bmatrix}\end{aligned} \tag{2.8.3.4}$$

Thus, $N^{(3)}$ is given as

$$N^{(2)} = \begin{bmatrix}0 & 3.094\\ 0.0862 & 0.1914\\ \cdots & \cdots\\ 0 & 0\\ 0 & 0.8236\\ \cdots & \cdots\\ 0 & 0\\ 0 & 0\end{bmatrix} \tag{2.8.3.5}$$

Therefore,

$$N_0^{(3)} = \begin{bmatrix}0 & 3.094\\ 0.0862 & 0.1914\end{bmatrix} \tag{2.8.3.6}$$

6) **The value of *S*(*z*) for the current iteration is found by multiplying *U*(*z*) and *Q*.** This gives,

$$\begin{aligned} S^{(i)}(z) &= U^{(i)}(z)Q^{(i)} \\ &= \begin{bmatrix} 0 & 1 \\ z & 0 \end{bmatrix}\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \\ &= \begin{bmatrix} 0 & 1 \\ z & 0 \end{bmatrix} \end{aligned} \tag{2.8.3.7}$$

Iteration 4:

1) **Check whether $r_i = \text{rank}(N_0) = \min(m, p)$. If this is the case, the algorithm terminates.** In this case, it can be seen that the rank of N_0 is 2, which is equal to min(*m*,*p*). Thus, all the iterations have been completed.

Obtaining the Interactor Matrix

Based on the criteria stated in (Rogoziński, Papliński, & Gibbard, 1987), the interactor matrix can be calculated as a product of all the $S^{(i)}$ that have been calculated above, that is,

$$K = \prod_{i=1}^{j-1} S^{(i)}(z) \tag{2.9}$$

where *j* is the total number of iterations completed. In the example above, 4 iterations where completed. Thus, the interactor matrix can be calculated as

$$\begin{aligned} K &= \prod_{i=1}^{j-1} S^{(i)}(z) \\ &= \begin{bmatrix} 0 & 1 \\ z & 0 \end{bmatrix}\begin{bmatrix} -1 & 0 \\ 0 & -z \end{bmatrix}\begin{bmatrix} -0.2873 & 0.9578 \\ -0.9578z & -0.2873z \end{bmatrix} \\ &= \begin{bmatrix} 0 & -z \\ -z & 0 \end{bmatrix}\begin{bmatrix} -0.2873 & 0.9578 \\ -0.9578z & -0.2873z \end{bmatrix} \\ &= \begin{bmatrix} -0.9578z & 0.2873z \\ 0.2873z^2 & 0.9578z^2 \end{bmatrix} \end{aligned} \tag{2.10}$$

Since the unitary interactor matrices are not unique, the final interactor matrix that will be used is given by Equation (2.11). Internally, the programme calculates the interactor matrix as

$$D = \begin{bmatrix} -0.9578z & -0.2873z \\ 0.2873z^2 & -0.9578z^2 \end{bmatrix} \tag{2.11}$$

It should be noted that the calculated interactor matrices are not unique. For a given transfer function matrix, 2 interactor matrices, D_1 and D_2, are “equivalent” if **and only if** (Huang & Shah, Performance Assessment of Control Loops: Theory and Applications, 1999),

$$D_1 = \Gamma D_2 \tag{2.12}$$

where Γ is an n-by-n, unitary, real matrix. The interactor matrices obtained using (2.10) and (2.111) have

$$\Gamma = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \tag{2.13}$$

and, thus, can be considered to be “equivalent.” For the purposes of this example, the interactor matrix obtained using (2.11) will be used. This will simplify comparisons between the programme and the manual results.

Part II: State-Space Model for the Process Data

Figure 8 shows a plot of the original data from which it is desired to obtain a model using time series analysis. Since there is no input, it is desired to model the process using an autoregressive, moving average (ARMA) model.

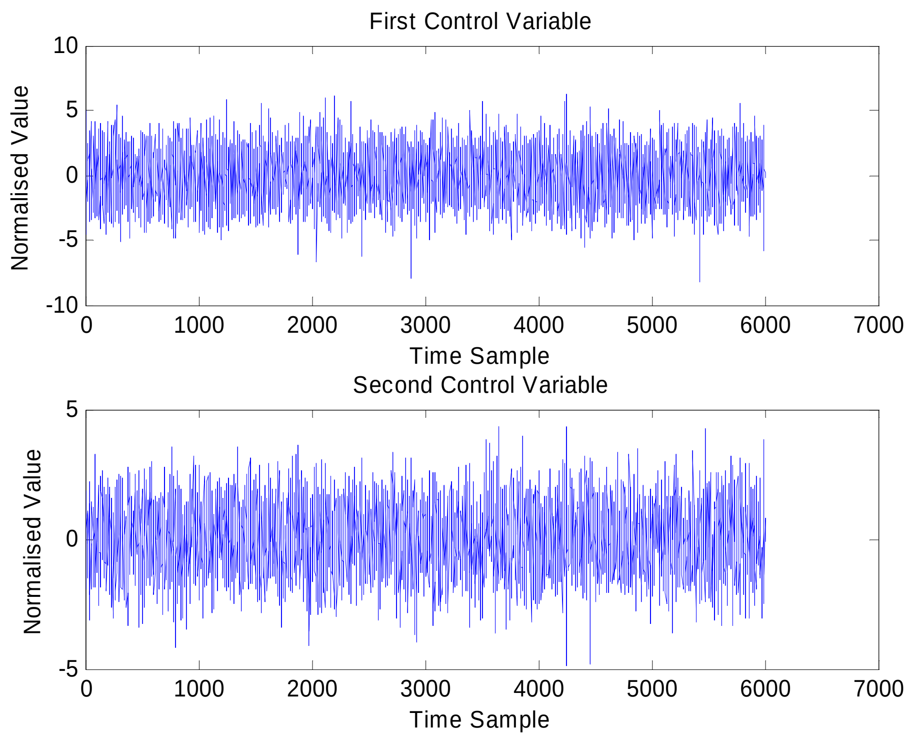


Figure 8: Deviation value of all of the controllers as a function of sampling interval

In MATLAB, this can be accomplished in the following steps:

1) First, the mean must be removed from the data. This can be accomplished using "**`>>outputde = detrend(data,'constant')`**".
2) Create an **iddata** object that stores the information about the process, "**`>>z=iddata(outputde,[],1)`**", since the output from the process is *t*, there are no inputs, and the *sampling time* is 1. This creates an object that can be used for time series analysis.
3) Since it is desired to learn how the MATLAB toolbox actually calculates the values, "**`n4sid`**" will be used for modelling the data. In general, the accuracy of the model is considered by examining the residuals. For easy of calculations, it will be assumed that the results from MATLAB can be trusted. This command will accomplish the desired results "**`>>modeln4sid=n4sid(z,'best')`**". The result can be improved by using the "**`pem`**" function. Thus, "“**`>>modelPEM=pem(z,modeln4sid)`**". MATLAB returns the following final results:

```
State-space model:  x(t+Ts) = A x(t) + K e(t)
                       y(t) = C x(t) + e(t)


A =
    x1      x2       x3      x4       x5
x1  0.151   -0.483   -0.222  0.735    0.160
x2  0.524   -0.606   0.275   -0.435   -0.002
x3  -0.409  -0.339   0.300    0.096   -0.441
x4  -0.090  -0.010   0.310    0.352   0.230
x5  -0.074  -0.081   -0.164   -0.110  0.173


C =
    x1       x2       x3       x4      x5
y1  103.51   75.83    28.22    13.03   -27.72
y2  -37.51   21.80    57.93    -58.91  -0.954
```

```
K =
                 y1            y2
          x1 -5.81x10^-5     -0.0096
          x2  -0.00097      0.0054
          x3  0.0035       -0.00017
          x4  -0.0028      -0.0006
          x5  -8.41x10^-5   9.19x10^-5
x(0) =

          x1           0
          x2           0
          x3           0
          x4           0
          x5           0

Estimated using PEM using SearchMethod = Auto from data set
z
Loss function 0.996583 and FPE 1.00323
Sampling interval: 1
```

The above MATLAB results can be rewritten as

$$x_{t+1} = \begin{bmatrix} 0.151 & -0.483 & -0.222 & 0.735 & 0.160 \\ 0.524 & -0.606 & 0.275 & -0.435 & -0.002 \\ -0.409 & -0.339 & 0.300 & 0.096 & -0.441 \\ -0.090 & -0.010 & 0.310 & 0.352 & 0.230 \\ -0.074 & -0.081 & -0.164 & -0.110 & 0.173 \end{bmatrix} x_t + \begin{bmatrix} -5.81\text{x}10^{-5} & -0.0096 \\ -0.00097 & 0.0054 \\ 0.0035 & -0.00017 \\ -0.0028 & -0.0006 \\ -8.41\text{x}10^{-5} & 9.19\text{x}10^{-5} \end{bmatrix} e_t$$

$$y_t = \begin{bmatrix} 103.51 & 75.83 & 28.22 & 13.03 & -27.72 \\ -37.51 & 21.80 & 57.93 & -58.91 & -0.954 \end{bmatrix} x_t + e_t$$

It should be noted that the output also provides the initial states (values) of the system when the sampling time is equal to zero.

Part II: Determining the Infinite Response Model

Manually

Before the infinite response model can be determined, it must be converted into a transfer function form in order to perform the calculations manually. This can be accomplished in the following 2 steps:

1) First, the model needs to be converted into transfer function form. This conversion can be performed manually by solving for y_t as a function of the e_t. The resulting conversion equation is then given as

$$\frac{y_t}{e_t} = C\left(zI - A\right)^{-1} K + 1 \tag{5}$$

However, given the large size of the matrices involved and the need to work in symbolic variables, MATLAB's build in function, "`ss2tf`", will be used. Unfortunately, this function will not convert a matrix of transfer function. Instead, it will only convert the transfer functions associated with a given input. The results are:

```
>> [num1,dem1]=ss2tf(modelPEM.A,modelPEM.K,modelPEM.C,[1 0;0
1],1)
num1 =
    1.0000   -0.3842    0.0060   -0.5367    0.5694   -0.1056
         0    0.3545    0.0840   -0.0664   -0.0084   -0.0110
dem1 =
    1.0000   -0.3697    0.0029   -0.1633    0.2029   -0.0774

>> [num2,dem2]=ss2tf(modelPEM.A,modelPEM.K,modelPEM.C,[1 0;0
1],2)
num2 =
         0   -0.5959   -0.8052    1.0799   -0.1655    0.0110
    1.0000    0.1332   -0.1155   -0.0376   -0.0498    0.0068
dem2 =
    1.0000   -0.3697    0.0029   -0.1633    0.2029   -0.0774
```

The closed loop transfer function can be obtained as:

```
>>Gcl=tf({num1(1,:) num2(1,:);num1(2,:) num2(2,:)},{dem1
dem2; dem1 dem2},1)
```

```
Transfer function from input 1 to output...
      z^5 - 0.3842 z^4 + 0.005959 z^3 - 0.5367 z^2 + 0.5694 z - 0.1056
 #1:  ---------------------------------------------------------------
      z^5 - 0.3697 z^4 + 0.002873 z^3 - 0.1633 z^2 + 0.2029 z - 0.0774

        0.3545 z^4 + 0.08401 z^3 - 0.06643 z^2 - 0.008448 z - 0.011
 #2:  ---------------------------------------------------------------
      z^5 - 0.3697 z^4 + 0.002873 z^3 - 0.1633 z^2 + 0.2029 z - 0.0774


Transfer function from input 2 to output...
          -0.5959 z^4 - 0.8052 z^3 + 1.08 z^2 - 0.1655 z + 0.01104
 #1:  ---------------------------------------------------------------
      z^5 - 0.3697 z^4 + 0.002873 z^3 - 0.1633 z^2 + 0.2029 z - 0.0774

      z^5 + 0.1332 z^4 - 0.1155 z^3 - 0.03757 z^2 - 0.04979 z + 0.006807
 #2:  -----------------------------------------------------------------
       z^5 - 0.3697 z^4 + 0.002873 z^3 - 0.1633 z^2 + 0.2029 z - 0.0774


Sampling time: 1
```

2) Now the multiplication $z^{-d}D\hat{G}_{cl}$ must be performed. This gives

$$= z^{-2}\begin{bmatrix} -0.9578z & -0.2873z \\ 0.2873z^2 & -0.9578z^2 \end{bmatrix}\begin{bmatrix} \dfrac{z^5-0.38z^4+0.01z^3-0.53z^2+0.57z-0.11}{z^5-0.37z^4+0.003z^3-0.16z^2+0.20z-0.08} & \dfrac{-0.60z^4-0.81z^3+1.08z^2-0.17z+0.01}{z^5-0.37z^4+0.003z^3-0.16z^2+0.20z-0.08} \\ \dfrac{0.36z^4+0.08z^3-0.07z^2-0.01z-0.01}{z^5-0.37z^4+0.003z^3-0.16z^2+0.20z-0.08} & \dfrac{z^5+0.13z^4-0.12z^3-0.04z^2-0.05z+0.01}{z^5-0.37z^4+0.003z^3-0.16z^2+0.20z-0.08} \end{bmatrix}$$

$$= z^{-2}\begin{bmatrix} \dfrac{-0.96z^6+0.26z^5-0.03z^4+0.53z^3-0.54z^2+0.11z}{z^5-0.37z^4+0.003z^3-0.16z^2+0.20z-0.08} & \dfrac{-0.29z^6+0.54z^5+0.81z^4-1.02z^3+0.18z^2-0.01z}{z^5-0.37z^4+0.003z^3-0.16z^2+0.20z-0.08} \\ \dfrac{0.29z^7-0.45z^6-0.07z^5-0.09z^4+0.17z^3-0.02z^2}{z^5-0.37z^4+0.003z^3-0.16z^2+0.20z-0.08} & \dfrac{-0.96z^7-0.30z^6-0.12z^5+0.35z^4-9.5\text{x}10^{-4}z^3-0.01z^2}{z^5-0.37z^4+0.003z^3-0.16z^2+0.20z-0.08} \end{bmatrix}$$

$$= \begin{bmatrix} \dfrac{-0.96z^4+0.26z^3-0.03z^2+0.53z-0.54+0.11z^{-1}}{z^5-0.37z^4+0.003z^3-0.16z^2+0.20z-0.08} & \dfrac{-0.29z^4+0.54z^3+0.81z^2-1.02z+0.18-0.01z^{-1}}{z^5-0.37z^4+0.003z^3-0.16z^2+0.20z-0.08} \\ \dfrac{0.29z^5-0.45z^4-0.07z^3-0.09z^2+0.17z-0.02}{z^5-0.37z^4+0.003z^3-0.16z^2+0.20z-0.08} & \dfrac{-0.96z^5-0.30z^4-0.12z^3+0.35z^2-9.5\text{x}10^{-4}z-0.01}{z^5-0.37z^4+0.003z^3-0.16z^2+0.20z-0.08} \end{bmatrix}$$

$$=\begin{bmatrix} \dfrac{-0.96z^{-1}+0.26z^{-2}-0.03z^{-3}+0.53z^{-4}-0.54z^{-5}+0.11z^{-6}}{1-0.37z^{-1}+0.003z^{-2}-0.16z^{-3}+0.20z^{-4}-0.08z^{-5}} & \dfrac{-0.29z^{-1}+0.54z^{-2}+0.81z^{-3}-1.02z^{-4}+0.18z^{-5}-0.01z^{-6}}{1-0.37z^{-1}+0.003z^{-2}-0.16z^{-3}+0.20z^{-4}-0.08z^{-5}} \\ \dfrac{0.29-0.45z^{-1}-0.07z^{-2}-0.09z^{-3}+0.17z^{-4}-0.02z^{-4}}{1-0.37z^{-1}+0.003z^{-2}-0.16z^{-3}+0.20z^{-4}-0.08z^{-5}} & \dfrac{-0.96-0.30z^{-1}-0.12z^{-2}+0.35z^{-3}-9.5\text{x}10^{-4}z^{-4}-0.01z^{-5}}{1-0.37z^{-1}+0.003z^{-2}-0.16z^{-3}+0.20z^{-4}-0.08z^{-5}} \end{bmatrix}$$

3) At this point long division would need to be performed for each of the transfer function to determine the infinite response model for each of the entries, which then can be combined to yield the overall model. Since only the first 2 terms are required for this example, it will be faster to use the remainder theorem and a bit of examination. The first thing to consider is that the first row of the transfer function matrix has the largest power in the numerator that is smaller than the denominator. Thus, the value of the constant term (z^0) will be 0. Next, note that the denominator in all cases starts with a 1. Thus, for the second row, the value of the constant term will be equal to the first term of the numerator. Thus, $F_0 = \begin{bmatrix} 0 & 0 \\ 0.29 & -0.96 \end{bmatrix}$. The first row of F_1 will be the first value in the numerator in the first row above. For the second row, the second co-efficient of the numerator less the second co-efficient of the denominator multiplied by either 0.29 for the first column or -0.96 for the second column will give the terms for the second row of F_1. Thus, $F_1 = \begin{bmatrix} -0.96 & -0.29 \\ -0.34 & -0.66 \end{bmatrix}$. It should be noted that this method works for this example solely due to the nature of the problem presented.

In MATLAB

In MATLAB, the impulse response co-efficients can be found using the “impulse” function and the final transfer function matrix obtained above. In order to save some time and effort, the final model obtained by hand after multiplying in the delay and the interactor matrix has been saved as “ImpulseModel”. Thus, this needs to be loaded

```
>> load ImpulseModel
```

In this case, the results are

```
>> [y]=impulse(ImpulseModel)
y(:,:,1) =
         0    0.2873
   -0.9578   -0.3437
   -0.0880   -0.2067
   -0.0596   -0.1191
    0.3550    0.0138
   -0.2316    0.0439
   -0.0484    0.0121
    0.0460    0.0148
   -0.0973    0.0006
    0.0305   -0.0057
    0.0109    0.0013
   -0.0250   -0.0015
    0.0190   -0.0005
   -0.0048    0.0012
   -0.0058   -0.0005
    0.0069    0.0001
   -0.0040    0.0002
    0.0000   -0.0003
    0.0019    0.0001
   -0.0018    0.0000
    0.0007   -0.0001
    0.0003    0.0001
   -0.0006   -0.0000


y(:,:,2) =

         0   -0.9578
   -0.2873   -0.6529
    0.4262   -0.3594
    0.9628    0.0588
   -0.7157    0.1106
```

```
 0.0333    0.0370
 0.0504    0.0453
-0.2607   -0.0050
 0.1286   -0.0138
-0.0056    0.0034
-0.0527   -0.0059
 0.0583    0.0001
-0.0255    0.0030
-0.0071   -0.0016
 0.0172    0.0009
-0.0137    0.0003
 0.0034   -0.0007
 0.0036    0.0004
-0.0050   -0.0001
 0.0028   -0.0002
-0.0001    0.0002
-0.0013   -0.0001
 0.0013   -0.0000
```

It is important to understand how to read the values obtained from MATLAB properly. The first row of each of the 2 results (y(:,:,1) and y(:,:,2)) represent the 2 columns of the first F matrix, that is, F_0. The next row of each of the results represents the next 2 columns of the second F matrix, that is, F_1. Given the assumptions made above, both the hand and manual results are the same.

Part III: Determining the Variance of the Residuals and of the Model

The variance of the residuals can be found from the model output, using the command "**`>>sigmaE=modelPEM.noisevariance`**". In this case, the following matrix is returned

$$\Sigma_e = \begin{bmatrix} 0.992 & -0.002 \\ -0.002 & 1.01 \end{bmatrix}$$

This implies, by Equation (2.2) that the minimum variance is given as:

$$
\begin{aligned}
\tilde{\Sigma}_{mv} &= \sum_{i=0}^{d-1} F_i \Sigma_e F_i^T \\
&= F_0 \Sigma_e F_0^T + F_1 \Sigma_e F_1^T \\
&= \begin{bmatrix} 0 & 0 \\ 0.29 & -0.96 \end{bmatrix} \begin{bmatrix} 0.992 & -0.002 \\ -0.002 & 1.01 \end{bmatrix} \begin{bmatrix} 0 & 0 \\ 0.29 & -0.96 \end{bmatrix}^T \\
&\quad + \begin{bmatrix} -0.96 & -0.29 \\ -0.34 & -0.66 \end{bmatrix} \begin{bmatrix} 0.992 & -0.002 \\ -0.002 & 1.01 \end{bmatrix} \begin{bmatrix} -0.96 & -0.29 \\ -0.34 & -0.66 \end{bmatrix}^T \\
&= \begin{bmatrix} 0.998 & 0.516 \\ 0.516 & 1.569 \end{bmatrix}
\end{aligned}
$$

The covariance matrix for the process data can be obtained using the command "`>>cov(outputted)`". The resulting matrix is given as:

$$
\Sigma_Y = \begin{bmatrix} 3.214 & -0.250 \\ -0.250 & 1.487 \end{bmatrix}
$$

Part IV: Determining the Overall Performance Efficiency

The overall performance efficiency index efficiency can be calculated as

$$
\eta = \frac{\operatorname{trace}\left(\tilde{\Sigma}_{mv}\right)}{\operatorname{trace}\left(\Sigma_y\right)} = \frac{\operatorname{trace}\left(\begin{bmatrix} 0.998 & 0.516 \\ 0.516 & 1.569 \end{bmatrix}\right)}{\operatorname{trace}\left(\begin{bmatrix} 3.214 & -0.250 \\ -0.250 & 1.487 \end{bmatrix}\right)} = \frac{0.998 + 1.569}{3.214 + 1.487} = 0.546 .
$$

Thus, the computed performance index is 0.546.

Part V: Determining the Individual Performance Indices

The first step in calculating the individual performance indices is to calculate the *E* matrix. The *E* matrix can be calculated as follows:

$$
\begin{aligned}
\left[E_0, E_1, E_2, \ldots, E_{d-1}\right] &= \begin{bmatrix} D_0 \\ D_1 \\ \vdots \\ D_{d-1} \end{bmatrix}^T \begin{bmatrix} F_0 & F_1 & \cdots & F_{d-1} \\ F_1 & F_2 & \cdots & 0 \\ \vdots & & 0 & 0 \\ F_{d-1} & 0 & \cdots & 0 \end{bmatrix} \\
&= \begin{bmatrix} 0 & 0 \\ 0.2873 & -0.9578 \\ -0.9578 & -0.2873 \\ 0 & 0 \end{bmatrix}^T \begin{bmatrix} 0 & 0 & -0.96 & -0.29 \\ 0.29 & -0.96 & -0.34 & -0.66 \\ -0.96 & -0.29 & 0 & 0 \\ -0.34 & -0.66 & 0 & 0 \end{bmatrix} \\
&= \left[\begin{array}{cc|cc} 1.003 & 0.002 & -0.098 & -0.190 \\ -0.002 & 1.003 & 0.326 & 0.632 \end{array}\right]
\end{aligned}
$$

The minimum variance matrix of the original output can be calculated as

$$
\begin{aligned}
\Sigma_{mv} &= \sum_{i=0}^{d-1} E_i \Sigma_e E_i^T \\
&= \begin{bmatrix} 1.003 & 0.002 \\ -0.002 & 1.003 \end{bmatrix} \begin{bmatrix} 0.992 & -0.002 \\ -0.002 & 1.01 \end{bmatrix} \begin{bmatrix} 1.003 & 0.002 \\ -0.002 & 1.003 \end{bmatrix}^T \\
&\quad + \begin{bmatrix} -0.098 & -0.190 \\ 0.326 & 0.632 \end{bmatrix} \begin{bmatrix} 0.992 & -0.002 \\ -0.002 & 1.01 \end{bmatrix} \begin{bmatrix} -0.098 & -0.190 \\ 0.326 & 0.632 \end{bmatrix}^T \\
&= \begin{bmatrix} 1.044 & -0.155 \\ -0.155 & 1.524 \end{bmatrix}
\end{aligned}
$$

The performance indices can be then be calculated as

$$
\begin{aligned}
\eta_{1:m} &= \operatorname{diag}\left(\operatorname{diag}\left(\operatorname{diag}\left(\Sigma_{mv}\right)\right)\operatorname{diag}\left(\operatorname{diag}\left(\Sigma_y\right)\right)^{-1}\right) \\
&= \operatorname{diag}\left(\operatorname{diag}\left(\operatorname{diag}\left(\begin{bmatrix} 1.044 & -0.155 \\ -0.155 & 1.524 \end{bmatrix}\right)\right)\operatorname{diag}\left(\operatorname{diag}\left(\begin{bmatrix} 3.214 & -0.250 \\ -0.250 & 1.487 \end{bmatrix}\right)\right)^{-1}\right). \\
&= \begin{bmatrix} 0.325 \\ 1.025 \end{bmatrix}
\end{aligned}
$$

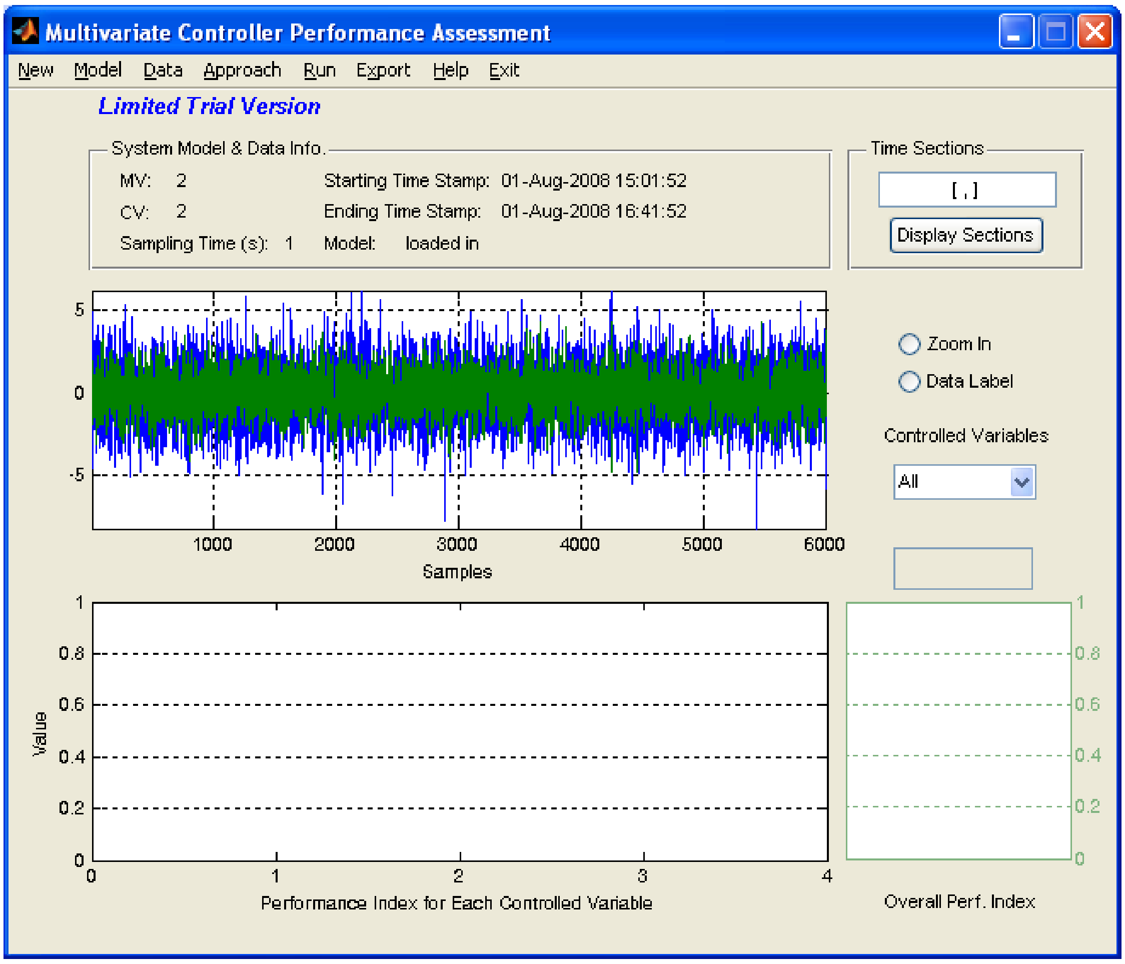
Thus, the computed performance index for the first controller, that is, the controller for the controller variable 1 (CV1) is 0.325, while it is 1.025 for the second controller. It should be noted that an individual controller can have a performance index greater than 1.

Part V: Using the Programme to Obtain the Same Results

Now, that it has been explained how to perform the calculations manually with minimal computer help, the programme that was designed to do this will be explained. The first step is to load the programme, "**`>>main_mvpa`**", and click on "**`MVPA`**" in the window that appears.

Next, load the model into the programme. This can be accomplished by clicking on the Model Menu and selecting the "**`Dtf.mat`**" file.

After a few seconds, load the data by clicking the Data Menu and selecting the "Compact" option. Select the file entitled "**`Book_Data.mat`**". The screen should now look similar to that shown in Figure 9.


Multivariate Controller Performance Assessment

Limited Trial Version

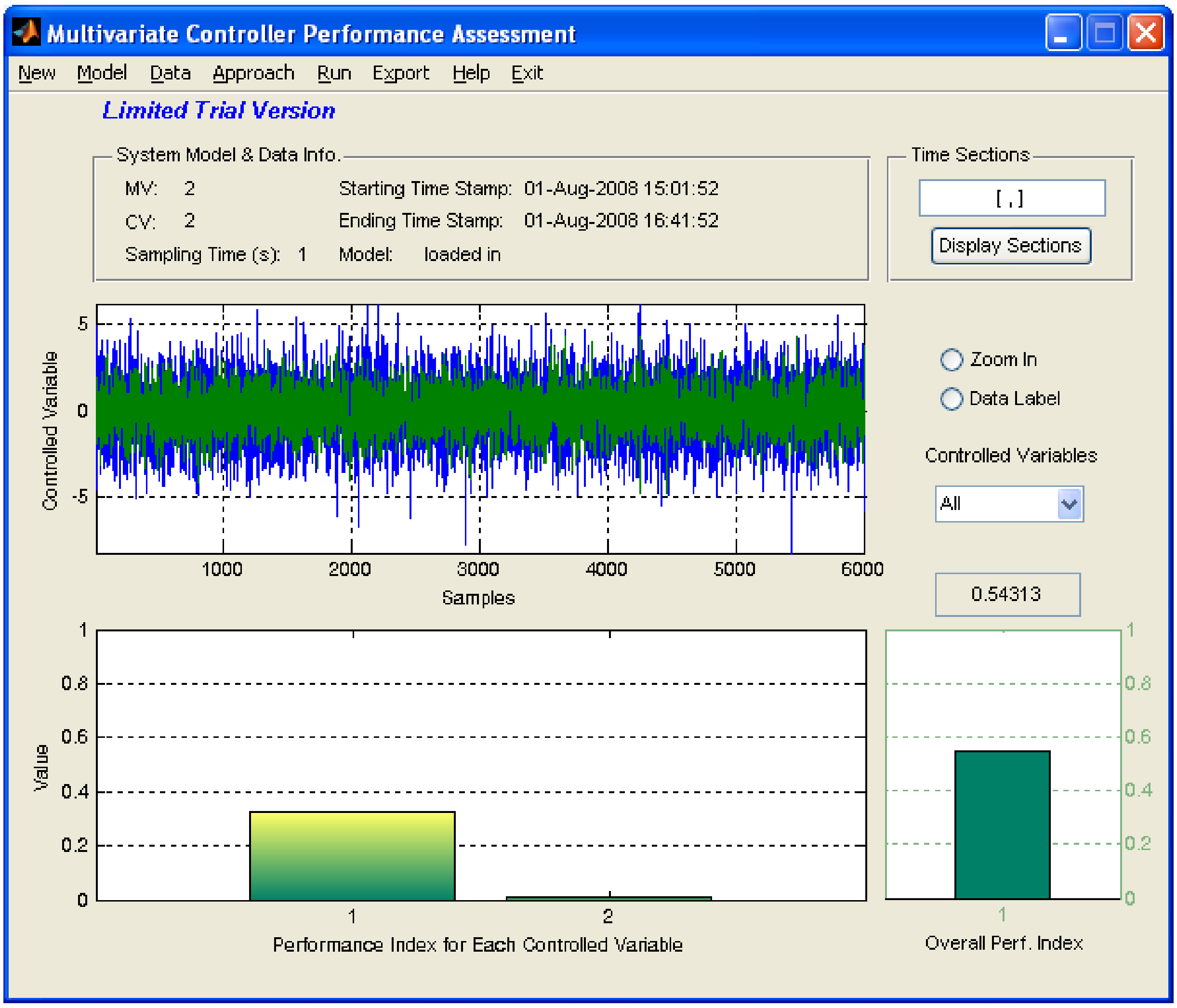
System Model & Data Info.

MV:	2	Starting Time Stamp:	01-Aug-2008 15:01:52
CV:	2	Ending Time Stamp:	01-Aug-2008 16:41:52
Sampling Time (s):	1	Model:	loaded in

Time Sections: [,]

Figure 9: Screen shot 1 for the example

Once it has been verified the Figure 9 matches the screen that was obtained, click on "Run" Menu and patiently wait until the results are displayed. The screen should then resemble Figure 10.


Multivariate Controller Performance Assessment
New Model Data Approach Run Export Help Exit
Limited Trial Version
System Model & Data Info.
Time Sections
[,]
Display Sections
Zoom In
Data Label
Controlled Variables
All
Controlled Variable
Samples
Value
Performance Index for Each Controlled Variable
Overall Perf. Index

MV: 2 Starting Time Stamp: 01-Aug-2008 15:01:52
CV: 2 Ending Time Stamp: 01-Aug-2008 16:41:52
Sampling Time (s): 1 Model: loaded in

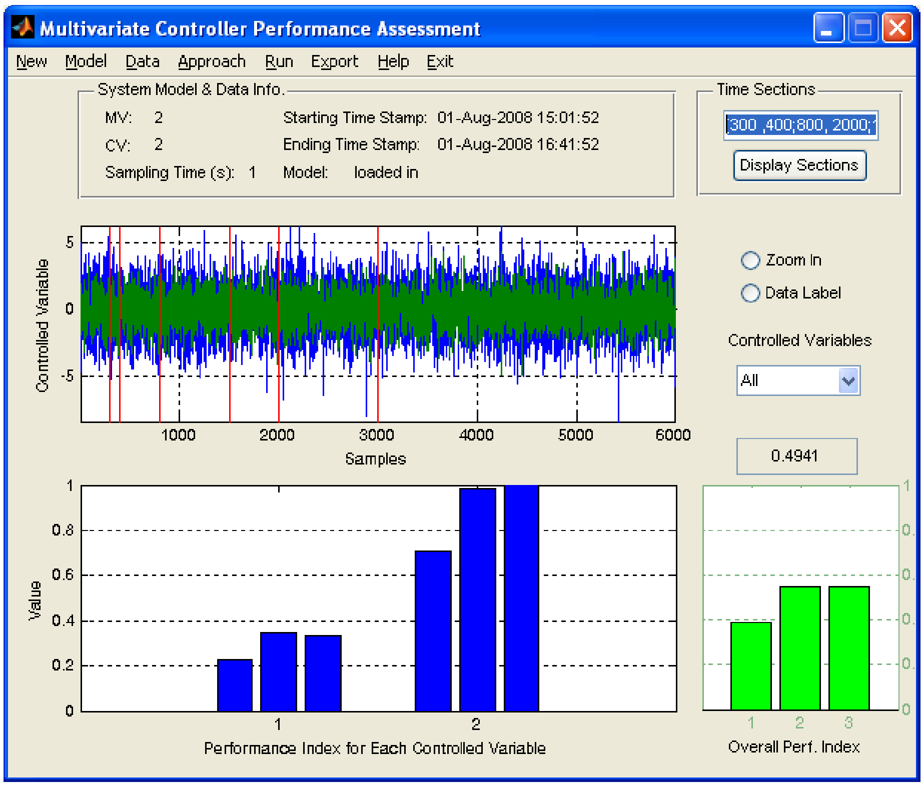
0.54313

Figure 10: Screen shot 2 for the example

From Figure 10, it can be seen that the overall performance index is calculated as 0.543, which compares favourably with the values obtained by hand at 0.546. The same conclusion can be drawn as before that the control of this process can be improved. For the first control variable, the performance index is about 0.30, which compares favourably with the previously obtained value of 0.325. On the other hand, for the second control variable, Figure 10 suggests that it is approximately equal to 1.02, while the previously obtained value was 1.02. For the first variable, the control is not very good, while for the second variable the control is quite good. In neither case are the individual controllers very good.

Part VI: Some Other Issues Using the Toolbox

Assume that we are using the same dataset as before. Now, let us enter a time section as "[300, 400; 800, 2000; 1500, 3000]", which allow the user to examine the behaviour of the control for 3 different sections: from the 300th to the 400th, from the 800th to the 2,000th, and from the 1,500th to 3,000th samples. Once the time section have been entered in Section 2 of the interface, press "Run" and wait for the results to be displayed. If you are unlucky, and there is no change in the results, then press "Run" again (There seems to be a quirk in the programme that causes it to improperly display the results). The correct screen should look like Figure 11.


Multivariate Controller Performance Assessment
New Model Data Approach Run Export Help Exit
System Model & Data Info.
Time Sections
Display Sections
Zoom In
Data Label
Controlled Variables
All
Controlled Variable
Samples
Value
Performance Index for Each Controlled Variable
Overall Perf. Index

MV: 2
CV: 2
Sampling Time (s): 1
Starting Time Stamp: 01-Aug-2008 15:01:52
Ending Time Stamp: 01-Aug-2008 16:41:52
Model: loaded in
300 ,400;800, 2000;
0.4941

Figure 11: Results of using a range of time sections

Figure 11 shows the results. In theory, all of the 3 bars should be at the same height as they were for all the data together. However, a smaller sample, as in the first time section, may lead to "inaccurate" results. In fact, the larger the sample size, the close will the values approach the results obtained above and the theoretical results. Also, it should be noted that the value displayed above the overall performance index bars is the average performance index for all 3 cases. It can be seen that for the last 2 bars, the overall performance index was close to the true value of 0.54. Similar results can be seen for the individual performance indices.

References

Huang, B., & Kadeli, R. (2008). *Dynamic Modeling, Predictive Control and Performance Monitoring.* London, United Kingdom: Springer Verlag.

Huang, B., & Shah, S. (1999). *Performance Assessment of Control Loops: Theory and Applications.* Springer.

Rogoziński, M. W., Papliński, A. P., & Gibbard, M. J. (1987, March). An Algorithm for the Calculation of a Nilpotent Interactor Matrix for Linear Multivariable Systems. *IEE Transactions on Automatic Control , AC-32* (3), pp. 234-237.