University of Alberta Computer Process Control Group

Univariate Controller Performance Assessment

Limited Trial Version

Written by: CPC Group, University of Alberta

Version 2.0

Table of Contents

List of Figures

List of Tables

Introduction

The univariate 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 Filtering and Correlation (FCOR) Algorithm.

A "Quick Start" approach to using this toolbox will be presented, along with a detailed section containing a full explanation 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_uvpa`**" to start the toolbox. The GUI shown in Figure 1 should appear.
4) Press the "UVPA" menu. A new GUI will appear that is shown in Figure 2.

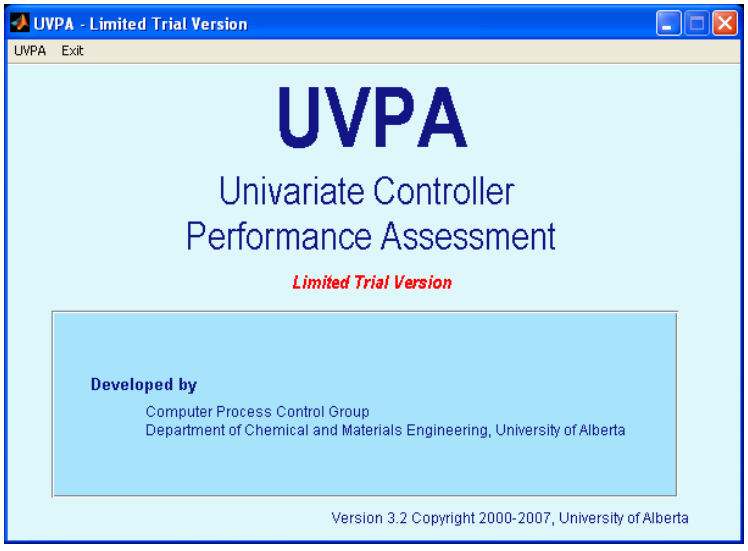

Figure 1: The First GUI that appears

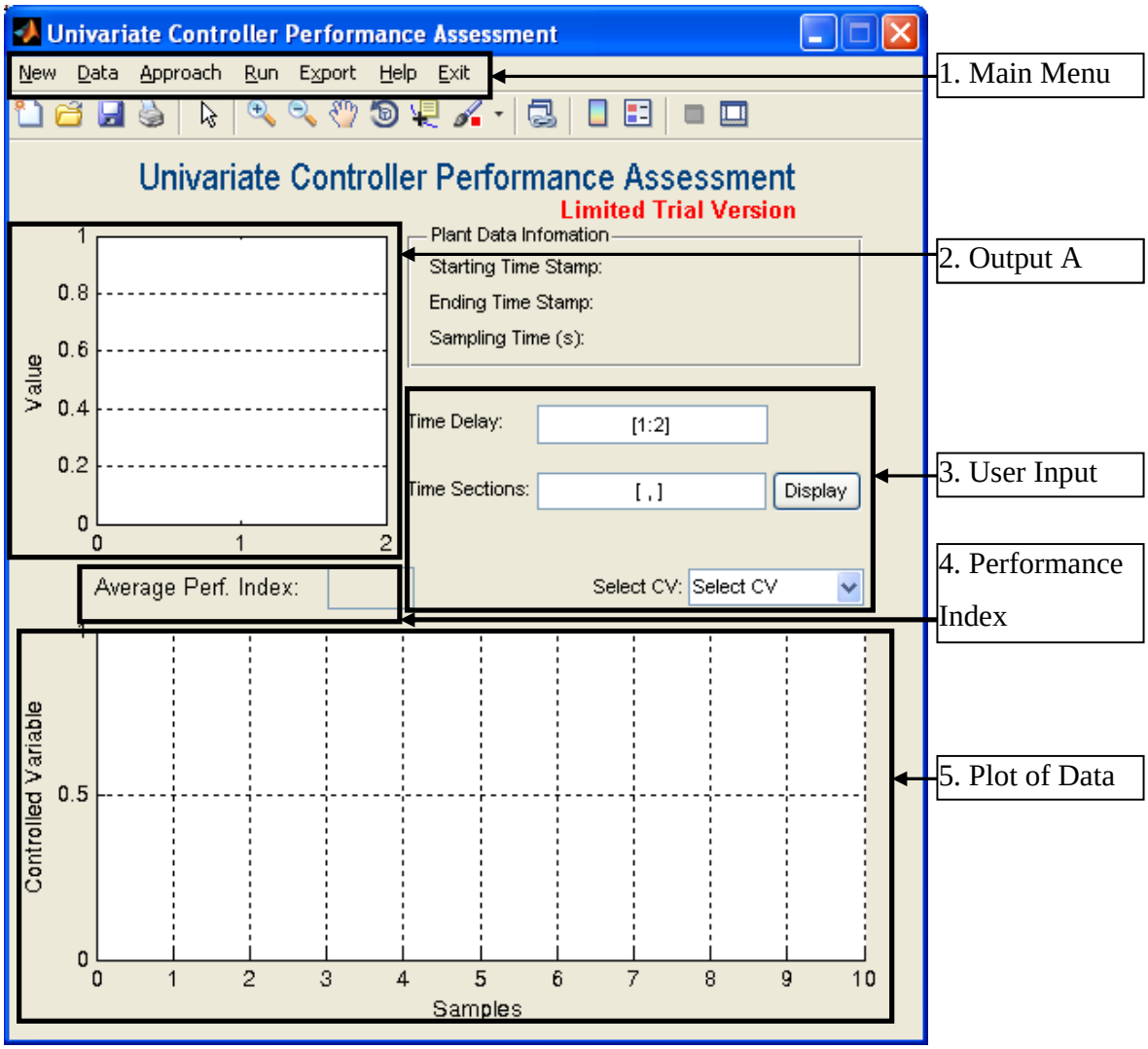

Figure 2: The main GUI for this toolbox, with the main regions highlighted

5) For the purpose of this quick start, we will be using the sample data provided in the zip file. Go to the “Data” Menu located in the Main Menu area, which is denoted as 1 in Figure 2. Select the “Compact” Option and select the “Ex_Compact_Data.mat” file.
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.

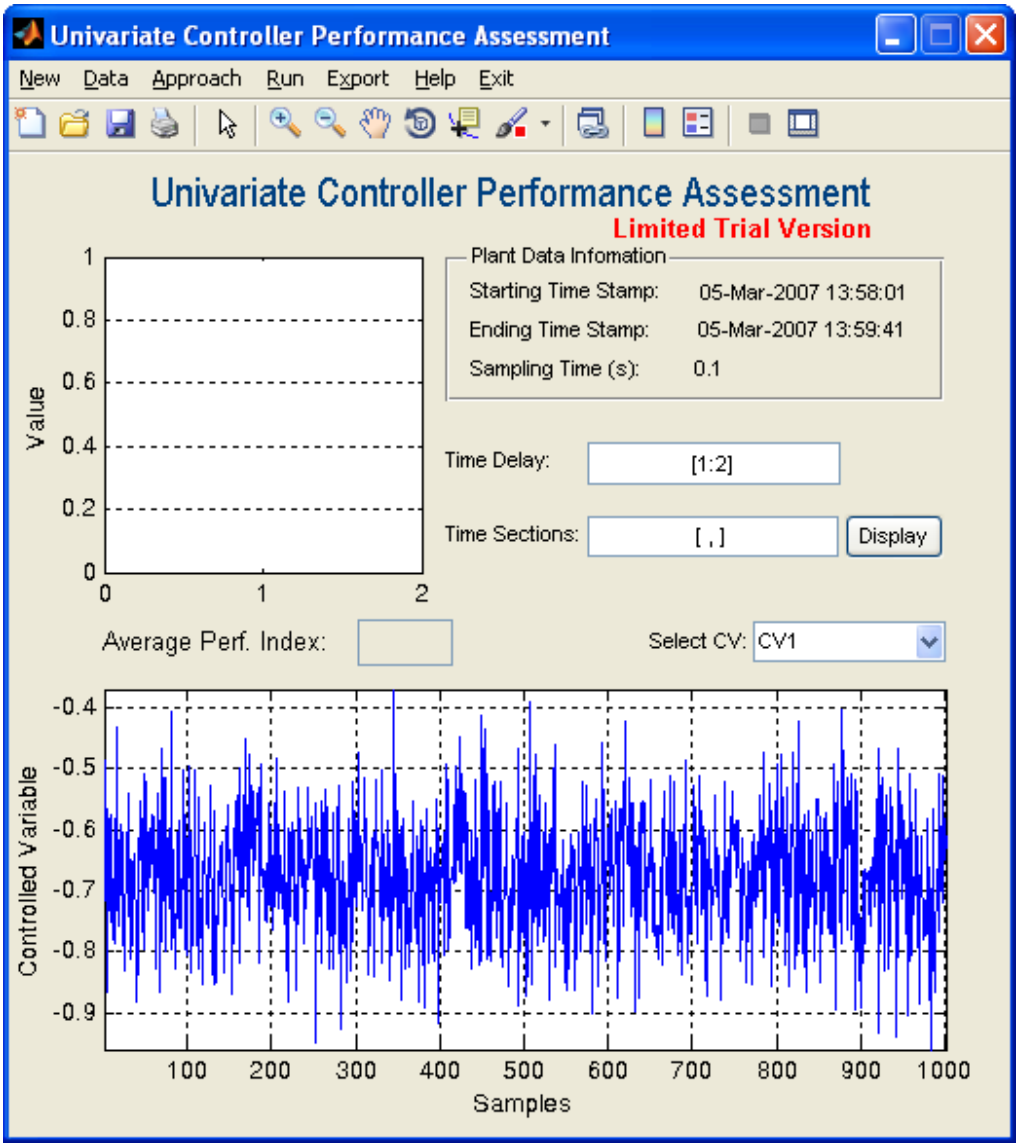

Figure 3: Screen shot after loading the data

7) Before the data can be analysed, the user needs to complete the section that is given as region 2 in Figure 2. In this example, multiple times, [1:3], will be selected and the Time Sections will be left as they are.
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.

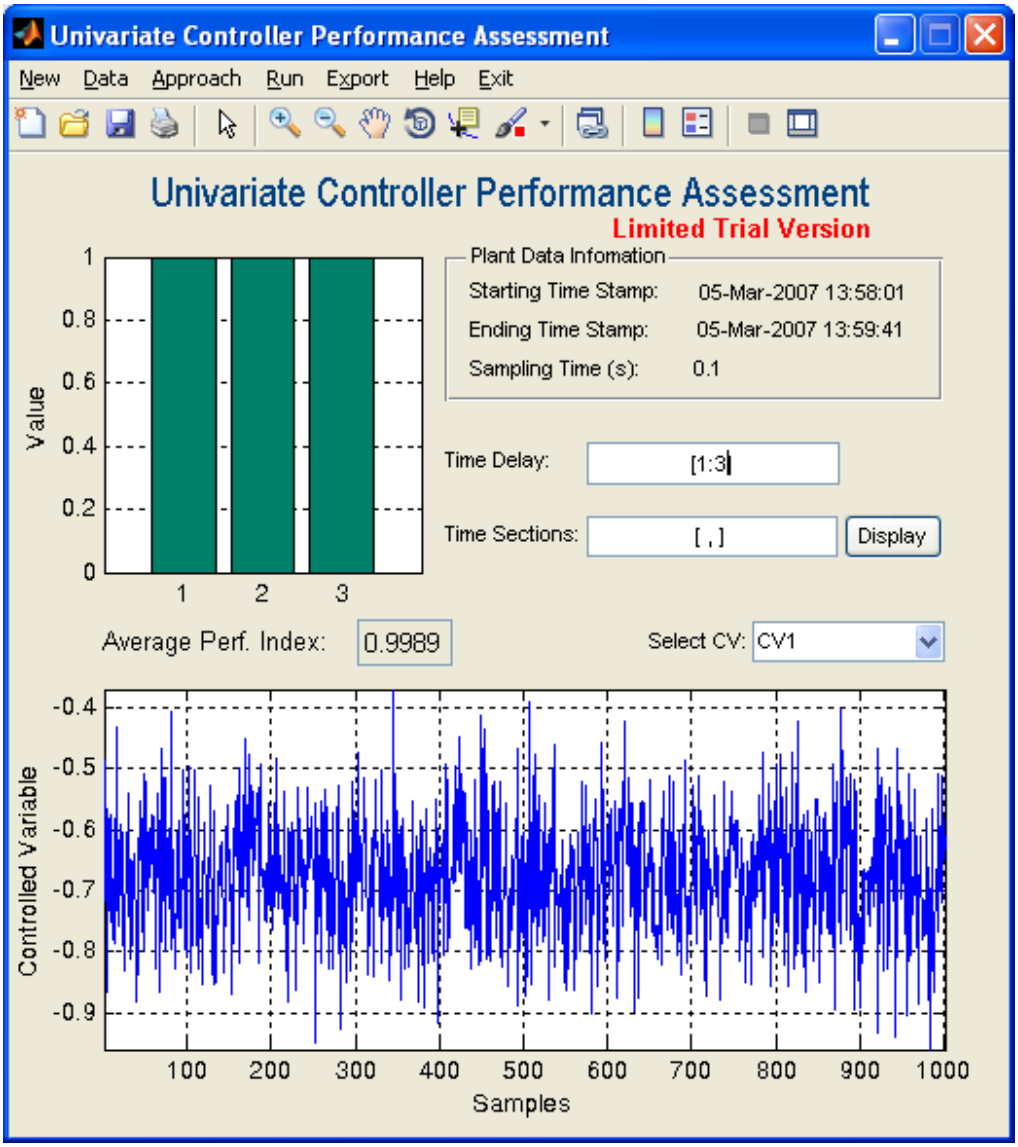

Figure 4: Screen shot after clicking "Run"

10) Figure 4 shows that using the given controller, the performance index is approximately 1 regardless of the time delay. This implies that the given controller, compared to a minimum variance controller, is almost the same, which implies that the control is optimal.

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

Univariate controller performance assessment is based on determining the variance of the given process compared with some "ideal" or best value. In the Filter and Correlation (FCOR) algorithm, the key performance variable is defined as

$$\eta = \frac{\sigma_{mv}^2}{\sigma_y^2} \tag{1}$$

where η is the performance index, that is, how much the process can potentially be improved with control, σ_y^2 is the variance of the data, and σ_{mv}^2 is the minimum variance for the process, which can be calculated as follows:

$$\sigma_{mv}^2 = \left(\sum_{i=0}^{d-1} f_i^2 \right) \sigma_e^2 \tag{2}$$

where σ_e^2 is the variance of the white noise (residuals or error if the model is being fitted), *d* is the time delay for the process, and f_i are the co-efficients of the infinite response model for the closed-loop process. The infinite response model, which is a consequence of Wold's Representation Theorem, can be written as:

$$y_t = \left(\sum_{i=0}^{\infty} f_i \; z^{-i} \right) e_t \tag{3}$$

The model given by Equation (3) can be obtained from the standard process control transfer function,

$$y_t = \frac{G_l}{1 + G_p G_c} e_t \tag{4}$$

using long division or MATLAB's *impulse* command. Equation (4) is estimated using time-series analysis of routine process data.

The following algorithm can be followed in analysing the performance of a process (Huang & Kadeli, 2008):

1) Obtain operating data from the desired, closed-loop system, when there is no setpoint 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 closed-loop original process is itself stable.

3) Obtain the time delay for the process.
4) Obtain the infinite response form of the closed-loop model.
5) Obtain the variance of the residuals.
6) Calculate the minimum variance using Equation (2).
7) Estimate the actual variance.
8) Calculate the performance index using Equation (1).

Data Storage

There are 2 different methods to store the data that is required for the toolbox: comprehensive and compact. For a quick generation of the compact data storage object, the file "gen_cmpct_data_SISO.p" can be used.

Comprehensive

Assume that there are *p* samples of 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 compact data storage can be generated using the *p*-file "gen_cmpct_data_SISO.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 binary file.
2) At the command prompt create the following matrix, which contains the values of 3 samples of the process for the 2 controlled variables: ""**>> W=[1 2;3 4;5 6]**".
3) Type at the command prompt: ""**>> gen_cmpct_data_SISO**". The following should appear

```
1. m inputs
2. p outputs
3. Output data (y): N x p matrix
4. Sampling time
m:
```

4) Type "2" and press enter. This is the number of inputs (manipulated variables) to the process.
5) Next, type "2" and press enter. This is the number of outputs (controlled variables) in the process.
6) Then, 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 file, say for example “test_compact_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) In order to view the results, type “**`>>load test_compact_data.mat`**”. This will load each of the entries into the workspace of MATLAB. Entering the name of each of the entries given above should give the same results as are presented in Table 1.

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

Entry	Value
Controller_status	[1 1 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_uvpa`**". 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 "UVPA" 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 are highlighted and will be discussed separately.

Figure 5: The First GUI that appears

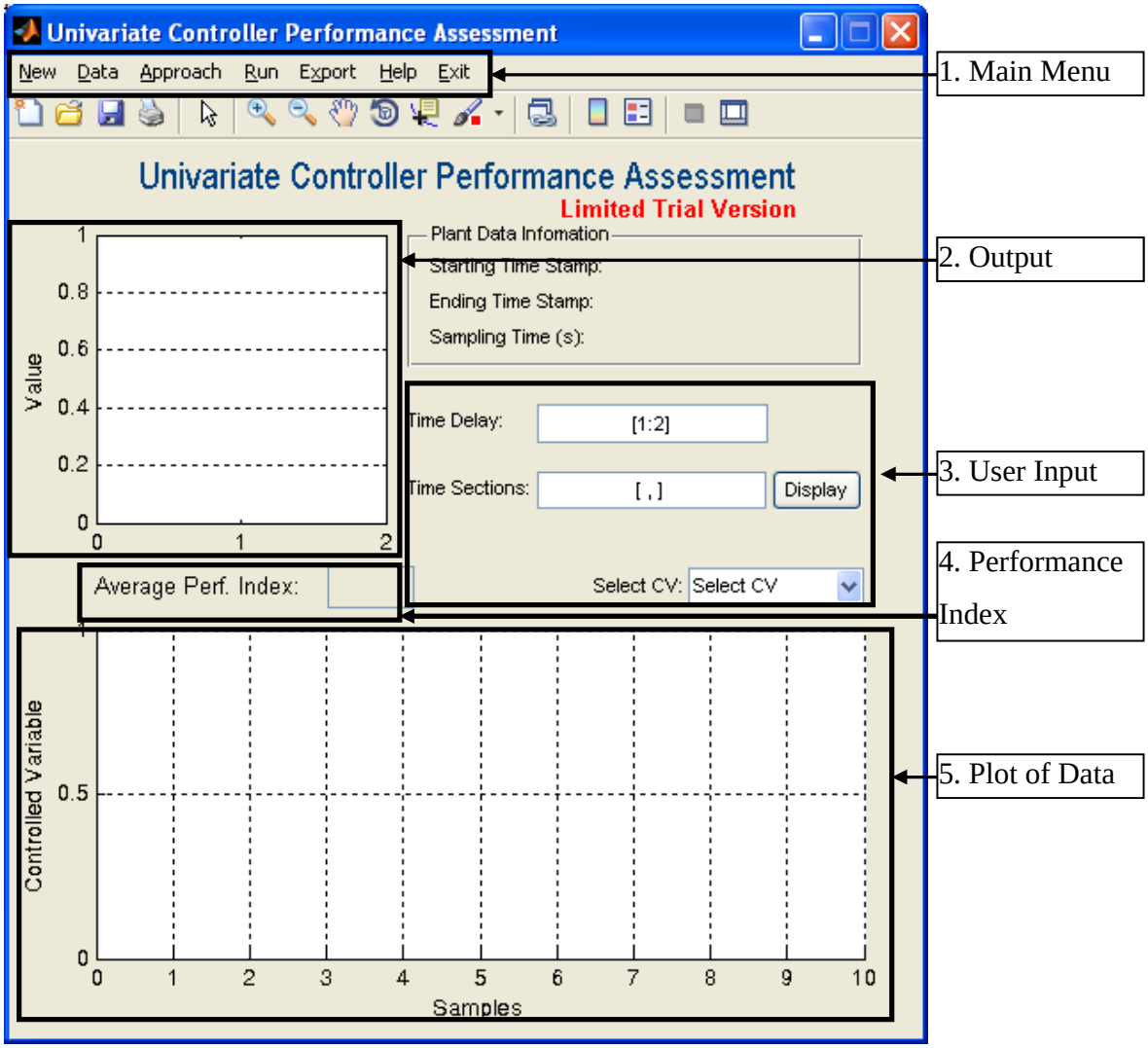

Figure 6: The main GUI for this toolbox, with the main regions highlighted

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) **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.

3) **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.
4) **Run**: Clicking this menu will cause the toolbox to analysis the data that has been selected.
5) **Export**: Clicking this menu will allow the current analysis to be saved as a ".mat" file.
6) **Exit**: Clicking this menu will cause the toolbox to close.

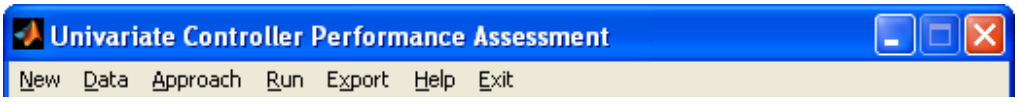

Figure 7: Close-up of the Main Menu

Section 2: Output

In this section, a bar graph is displayed showing the performance index as a function of the time delay.

Section 3: User Input

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

1) **Time Delay**: In this field, the user can enter any vector that contains the desired time delays that are to be analysed. The time delays need not be sequential, that is, "[1 4 7]" is an acceptable entry. In fact any of the following vectors are acceptable: "[3]", "[1:10]", and "[1:2:10]", which will only take every other number between the bounds 1 and 10.
2) **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).
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 4: Performance Index

This box displays the average of all the performance indices displayed in the graph above.

Section 5: Plot of Data

This graph displays a time series plot of the values for the current controlled variable.

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,1)`**: Creates discrete time transfer function given the numerator, **`num`**, and denominator, **`dem`**, 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.

The examples in this section will be based on the data available in the folder "Examples." Given the nature of the process, with 1,001 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.

For the example, the first control loop from the industry data provided with this toolbox will be used. It will be assumed that the time delay in the process is 2 samples.

The first step before any of the further examples can be done is to load this file into MATLAB. The following command will perform the task "**`>>load SISOexample.mat`**". The data will be stored in the vector t. Once the data has been loaded the following tasks can be performed.

Part I: 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.

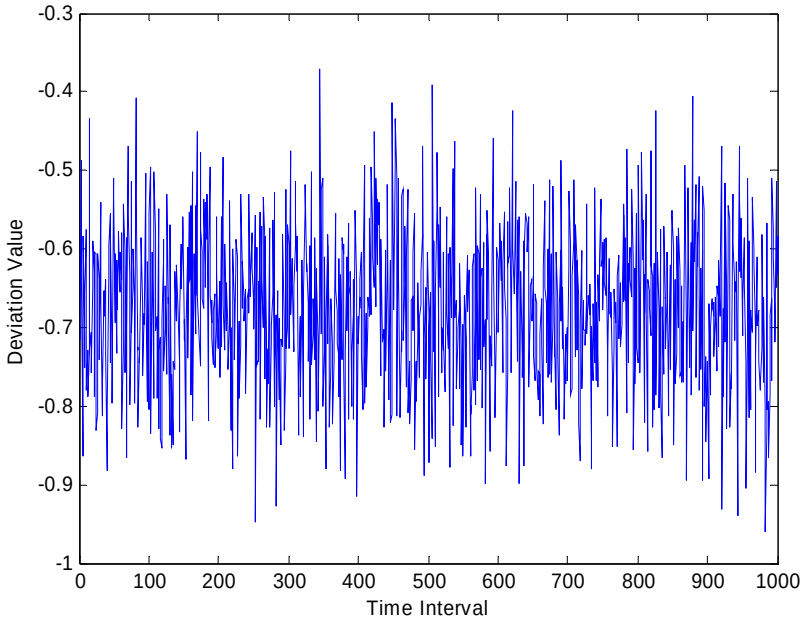

Figure 8: Deviation value of the first controller as a function of sampling interval

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 "**>>tdetrended = detrend(t,'constant')**".
2) Create a **iddata** object that stores the information about the process, "**>>z=iddata(tdetrended,[],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')**". MATLAB returns the following 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      0.33353     -0.72997   -0.0059398      0.74696      0.10091
x2      0.64084      0.30599     -0.68154     -0.09175     0.056415
x3     -0.24743     -0.58372     -0.49516     -0.52632      0.07519
x4      0.46209     -0.19759      0.27612     -0.41677     -0.69354
x5      0.18111    -0.083865      0.29574    0.0096886      0.18412
 C =
          x1           x2           x3           x4           x5
y1      0.75903       0.3281     0.065685       1.3079      0.39369
 K =
                    y1
           x1    0.0074553
           x2     0.013191
           x3  -0.00038048
           x4    -0.019743
           x5     0.020195
Estimated using N4SID from data set z
Loss function 0.00991873 and FPE 0.0102205
Sampling interval: 1
```

The above MATLAB results can be rewritten as

$$x_{t+1} = \begin{bmatrix} 0.334 & -0.730 & -0.006 & 0.747 & 0.101 \\ 0.641 & 0.306 & -0.682 & -0.092 & 0.056 \\ -0.247 & -0.584 & -0.495 & -0.56 & 0.075 \\ 0.462 & -0.198 & 0.276 & -0.417 & -0.694 \\ 0.181 & -0.084 & 0.296 & 0.010 & 0.184 \end{bmatrix} x_t + \begin{bmatrix} 0.007 \\ 0.013 \\ -0.000 \\ -0.020 \\ 0.020 \end{bmatrix}$$

$$y_t = \begin{bmatrix} 0.759 & 0.328 & 0.066 & 1.308 & 0.364 \end{bmatrix} x_t + e_t$$

Part II: Determining the Infinite Response Model

Manually

If it is desired to obtain the infinite response model co-efficients by hand, then the following 2 steps need to be followed in order to obtain the infinite response model:

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. The results are

```
>> [num,dem]=ss2tf(modeln4sid.A,modeln4sid.K,modeln4sid.C,1)
num =
    1.0000    0.0804   -0.5203    0.5426    0.6717   -0.3385
dem =
    1.0000    0.0883   -0.5035    0.5736    0.6816   -0.3483
```

This must then be converted into a transfer function object, that is,

```
>>tfmodel=tf(num,dem,1)
Transfer function:
z^5 + 0.08037 z^4 - 0.5203 z^3 + 0.5426 z^2 + 0.6717 z - 0.3385
-----------------------------------------------------------------
z^5 + 0.08828 z^4 - 0.5035 z^3 + 0.5736 z^2 + 0.6816 z - 0.3483
 Sampling time: 1
```

2) Now, long division can be performed in order to determine the infinite response model co-efficients. The long division gives

$$
\begin{array}{r}
1-0.00791z^{-1} \\
z^5+0.08828z^4-0.5035z^3+0.5736z^2+0.6816z-0.3483 \overline{)\; z^5+0.08037z^4-0.5203z^3+0.5426z^2+0.6717z-0.3385} \\
\underline{-z^5-0.08828z^4+0.5035z^3-0.5736z^2-0.6816z+0.3483} \\
-0.00791z^4-0.0168z^3-0.3100z^2-0.0099z+0.0098 \\
\underline{0.00791z^4+0.0007z^3+0.0040z^2+0.0045z-0.0053-0.0028z^{-1}} \\
-0.0161z^3-0.3060z^2-0.0054z+0.0045-0.0028z^{-1}
\end{array}
$$

Thus, the 2 required co-efficients are 1 and -0.00791.

In MATLAB

In MATLAB, the impulse response co-efficients can be found using the “impulse” function and the state space model. In this case, the results are

```
>> [y]=impulse(modeln4sid)
y =
         0
         0
         0
         0
    1.0000
   -0.0079
   -0.0161
   -0.0336
   -0.0105
    0.0084
    0.0215
    0.0257
   -0.0009
   -0.0087
   -0.0261
   -0.0116
    0.0024
    0.0146
```

```
 0.0214
 0.0029
-0.0036
-0.0196
-0.0112
-0.0014
 0.0091
```

It should be noted that impulse function returns some "spurious" values that can be ignored. The first non-zero values are required. This issue does not occur if a transfer function model is used.

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

The variance of the residuals can be found from the model output. In this example, it has been bolded in Part I in the MATLAB output. The variance of the residuals is given as 0.009 991 873. This implies, by Equation (2) that the minimum variance is given as:

$$\begin{aligned}\sigma_{mv}^2 &= \left(\sum_{i=0}^{d-1} f_i^{\,2}\right)\sigma_e^2 \\ &= \left(1.00^2 + \left(-0.0079\right)^2\right)\left(0.009\,991\,873\right) \\ &= 0.009992\end{aligned}$$

The variance of the process, σ_y^2, can be determined as 0.0099.

Part IV: Determining the Efficiency

The performance index can be calculated as

$$\eta = \frac{\sigma_{mv}^2}{\sigma_y^2} = \frac{0.009992}{0.0099} = 1.00\,.$$

Thus, the computed efficiency is a remarkable 0.9939, which implies that the control of the process cannot be further improved by tuning.

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_uvpa`**", and click on "**`UVPA`**" in the window that appears.

Next, the data that will be used in this assessment must be loaded. Load the file, "**`Ex_Compact_Data.mat`**". Make sure the first control variable is selected. Set the time delay to "[2]". The screen should now resemble Figure 9.

Once it has been verified that this is the case, click on "Run" and patiently wait until the results are displayed. The screen should then resemble Figure 10.

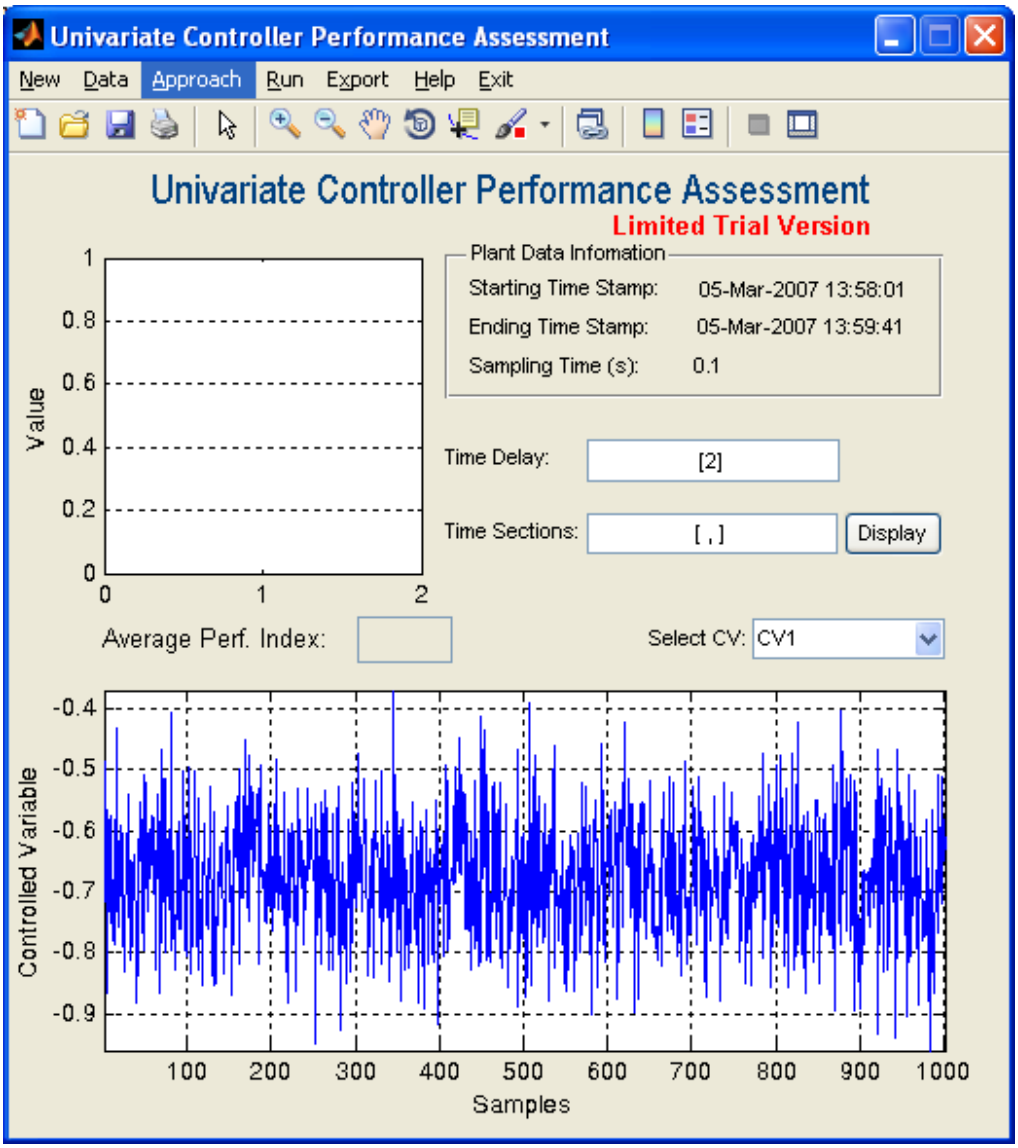

Figure 9: Screen shot 1 for the example

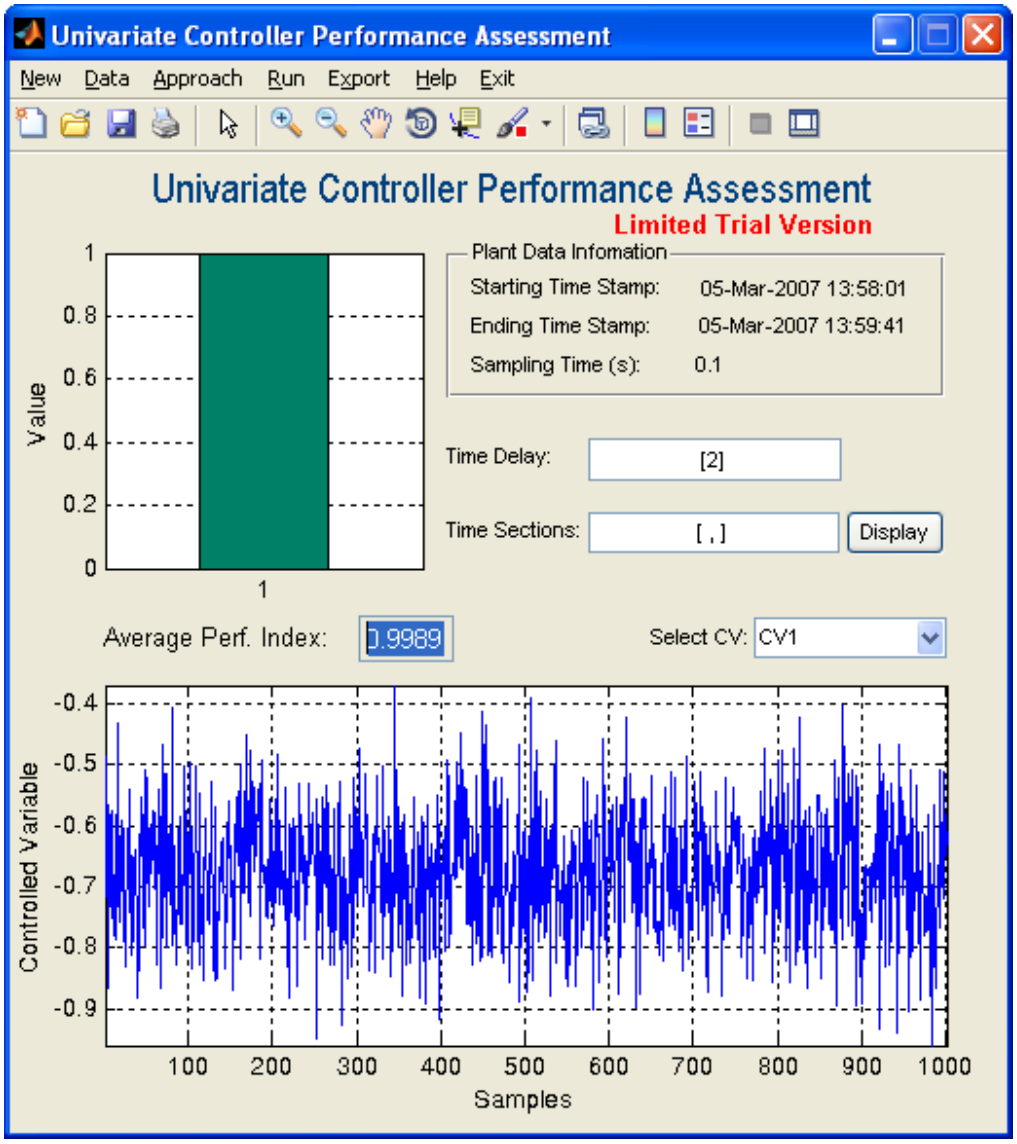

Figure 10: Screen shot 2 for the example

From Figure 10, it can be seen that the average performance index is calculated as 0.9989, which compares favourably with the values obtained by hand at 1.00. The same conclusion can be drawn as before that the control of this process is **optimal**.

Part VI: Some Other Issues Using the Toolbox

In the previous cases, it has been assumed that the time delay is known from experience, or *a prioiri*. In reality, this may not be the case. This issue can be circumvented by entering a range of time delays. In this case, the behaviour of the performance index as a function of time

delay can be examined to determine the effectiveness of the given control. In general, the faster the curve approaches a value of 1, the better the controller performance is.

Assume that we are using the same dataset as before. Now, let us enter a range of time delays as a vector "[1:10]". Enter a time section as "[450,600]". Once this has been entered, 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.

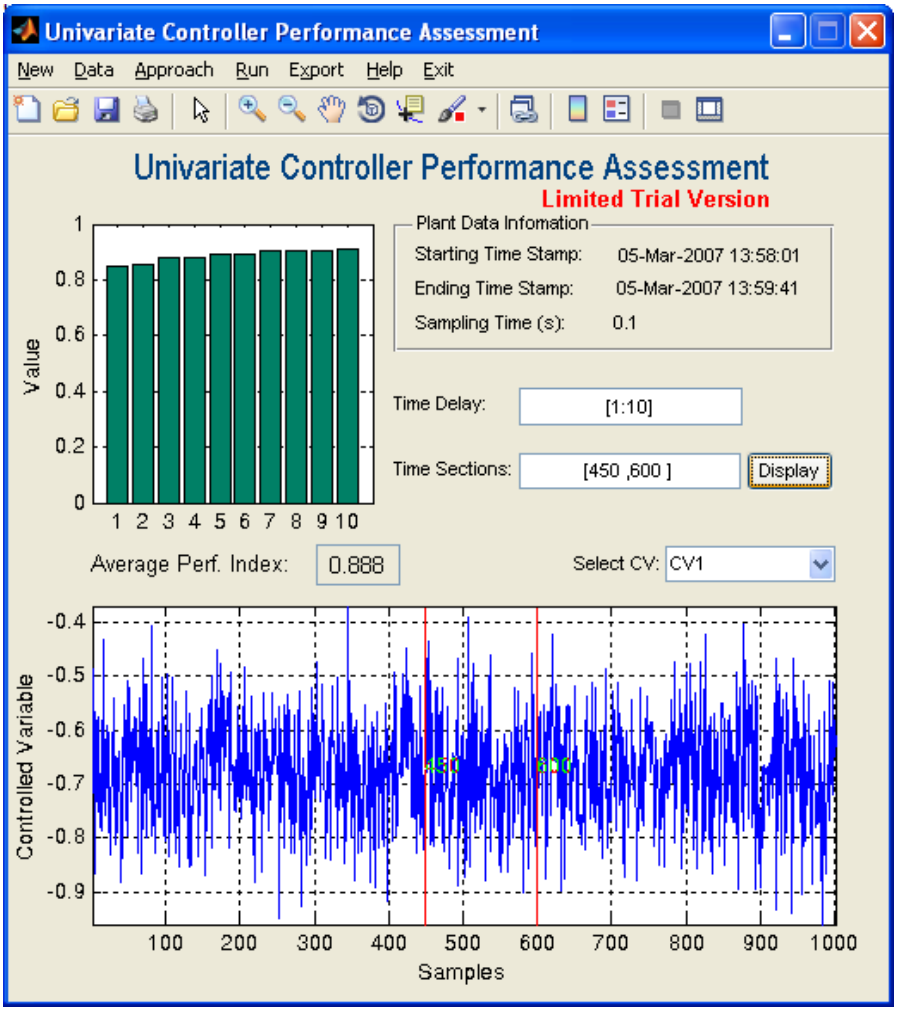

Figure 11: Results of using a range of time delays

Figure 11 shows that the there are ripples the performance index as a function of time delay graph. This would suggest that the original data in this given time section is oscillatory. This seems to be referred by the bottom figure, where in the region between the red lines, the data have a periodic behaviour.

References

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