AN INTEGRATED, REAL-TIME COMPUTING ENVIRONMENT

For Advanced Process Control Development

JAMES H. VAN DER LEE, DONALD G. OLSEN, BRENT R. YOUNG, WILLIAM Y. SVRCEK
University of Calgary • Calgary, Alberta, Canada T2N 1N4

Today's process control field is such that control techniques that were considered advanced even ten to twenty years ago are now becoming commonplace.[1] Model predictive control (MPC) in all it's incarnations is a good example—today there are well over two thousand MPC controllers reported to be in operation industrially.[2] Despite this abundance of MPC technology, however, commercial simulation software packages have been slow to incorporate MPC algorithms. Even when they are included, the algorithms are prescribed and the software does not allow for customization of the algorithm(s) by users such as process engineers. This can be attributed to the fact that there are many MPC algorithms and it would take large development teams to incorporate them all; but even if this were possible, it would not be particularly useful for the testing of a new algorithm.

This limitation must be accepted unless you decide to program your own code to simulate your own process and control algorithm, using a programming language such as C++ or Visual Basic for Applications (VBA). This approach is time-consuming, however, and is typically attempted only by process engineers with prior experience in such an exercise.

This lack of both the flexibility of commercial packages and the experience or education required to build one's own simulator and control algorithm can cause process engineers to steer clear of MPC, even though it may provide the solution they are looking for and despite its relative abundance and growing acceptance in a number of industries. This barrier to understanding and implementation also exists for many other related advanced process control (APC) technologies that are not as widespread as MPC.

From an educational perspective, this barrier to implementation has also largely prevented the facile inclusion of MPC and other APC technologies even in senior, advanced undergraduate process control technical electives, *e.g.*, as advocated by Edgar.[3,4] Even some excellent graduate courses emphasize the fundamentals and steer clear of APC,[5] leaving those students who wish to enter the field of process control devoid of practical experience with MPC and APC algorithmns.

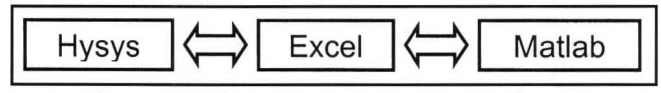


Figure 1. *Software communication pathways.*

This paper presents a simulation environment, composed of three readily available commercial software packages, that allows for quick and easy development of custom APC schemes on a wide variety of processes. This effectively removes the implementation barriers described above and allows the advanced undergraduate and graduate student an opportunity to study and implement APC schemes of

James van der Lee *is a full-time graduate student in Chemical and Petroleum Engineering at the University of Calgary. He received his BS (1999) in chemical engineering from the University of Calgary. His research focuses on the model predictive control of a pilot amine absorption/stripping plant.*
Don Olsen *is a part-time graduate student in Chemical and Petroleum Engineering at the University of Calgary and an applications engineer with Hyprotech Ltd. in Calgary. He received his BS in chemical engineering from the University of British Columbia, and his research focuses on model predictive control within the framework of a vinyl acetate process design.*
Brent Young *is Associate Professor of Chemical and Petroleum Engineering at the University of Calgary. He received his BE (1986) and his PhD (1993) degrees in chemical and process engineering from the University of Canterbury in New Zealand. His teaching and research interests center on process control and design.*
William Svrcek *is Professor of Chemical and Petroleum Engineering at the University of Calgary. He received his BSc (1962) and his PhD (1967) degrees in chemical engineering from the University of Alberta. His teaching and research interests center on process control and design.*

varying complexity, thus allowing for a level of understanding of APC that only a "learning-by-doing" approach can provide.

METHODOLOGY

The methodology behind this project is similar to that used by an ever-increasing number of chemical process control authors and educators in that it uses commercial software in order to perform tasks that, while important, would to a certain extent impede specific control education objectives. There are several successful examples of this approach[6,7] that quite effectively use Matlab to handle modeling and solution methods to clearly demonstrate a variety of fundamental control concepts. But given the nature of the problem of being able to overcome the implementation issues associated with APC for educational purposes in a way that would provide easy implementation in advanced technical elective or graduate classrooms, the following characteristics were determined to be important:

- *The need for an interactive dynamic simulation environment capable of using a wide variety of process models.*
- *A means for interacting with this environment so that custom algorithms can be implemented.*
- *An ability to be able to see what steps are occurring as they happen, while the simulation is running.*
- *Tools that contain many of the standard operations required for APC applications.*

Any software package(s) that meets these requirements has the potential to remove the implementation barrier. We found that Hyprotech's Hysys, Microsoft's Excel, and Mathwork's Matlab could be configured in such a manner that all these requirements were met. Excel is the industry standard for spreadsheets, and as a result many programs include provisions for two-way communication with Excel. Both Hysys and Matlab contain such links, and it is possible to use both links simultaneously in order to exploit the strengths of all three programs for use in APC applications. Figure 1 illustrates the basic communication pathway when the three programs are linked.

The benefits of this type of system are rigorous steady state and dynamic plant simulation (*e.g.,* Hysys), a large library of functions useful for APC applications (*e.g.,* Matlab), and powerful data handling and visualization tools (*e.g.,* Excel) in software packages that are already familiar to many chemical engineers. The steps involved in creating a simulation using these three programs in conjunction with each other are:

- *A dynamic simulation case of the process required. It is necessary to make note of the values that will need to be read from and written to the process for the algorithm to work effectively.*
- *Add the necessary read/write variable on the variable page of the 'data book.' Create a new 'Process Data Table' (PDT) by pressing add on the PDT page in the 'data book,' and then add the desired variables by checking the showbox on the PDT page. Then press the view button, add 'tag names,' and select the 'access mode' (read, write, or read/write) for each variable. It should be noted that the order in which the show boxes are selected is the order in which the variables appear in the 'PDT set up' page (accessed by pressing the view button). Because of this property, it has been our experience that it is easiest to group read, write, and read/write variables and add them to the PDT in blocks, making note of the order in which each variable is added.*
- *Save the updated simulation case and make note of its file name and location.*
- *Start Excel and open the Visual Basic for Applications (VBA) Editor. Make sure the Excel link, Matlab Automation Server Type Library, and Hysys Type Library references are selected by selecting References in the Tools menu list in the VBA editor. This ensures that VBA will recognize the Matlab and Hysys object types.*
- *Insert a 'Module' into VBA and add the code as the numerical order of the following steps indicate.*

• **Step 1.** Variable declaration (see Figure 2).

```
Option Explicit 'Ensures that undefined variables are not allowed in code
'Objects required to bind Hysys/Process Data Table(PDT) to Excel/VBA
Public HyApp As HYSYS.Application
Public SimCase As SimulationCase
Public dt As HYSYS.DataTable

'Variables that are used in communicating information from the PDT to Excel/VBA
Public Wtags As Variant
Public Wvalues() As Double
Public Rtags As Variant
Public Rvalues As Variant
Public RWtags As Variant
Public RWvalues As Variant

'Special Flags used for error checking, in regards to establishment of
HYSYS/'Excel/VBA
Public dtValid As Boolean
Public simCaseValid As Boolean
Public hyAppValid As Boolean

'counter variable used to keep track of when the control algorithm should execute
Public i As Integer

'Excel worksheet Initialization
Public xlSheet As Excel.Worksheet

'Variables used in Registering of HYSYS/Excel/VBA Interface
Public notifyevent1 as EventSink


'used to control the integrator from Excel/VBA
Public integrator As Variant
```

Figure 2. *Step 1 - Variable declaration.*

```
Sub Main()
'lines of code in italic type are used for error handling
'and will allow easier debugging and to ensure the HYSYS/
'Excel/VBA is terminated in the event of an error
On Error GoTo mainerror
'Initialization
Init
'Binds Excel/VBA to the PDT
BindData Table

Form.Show 'Shows Form
 Exit Sub
main error:
MsgBox "Error in Main" & Err.Description
Cleanup
End Sub
```

Figure 3. *Step 2 - Main function.*

```
Sub Init()
On Error Go To InitError
'Regular HYSYS case Initialisation
Set HyApp = CreateObject("HYSYS.Application")
HyApp.Visible = True
hyAppValid = True
Set SimCase = GetObject("c:\your Hysyscase.hsc")
SimCase.Visible = True
simCaseValid = True
'initialises the active Excel worksheet
Set xlSheet = Sheets("Sheet1")
'the following space could be used to initialised other VBA variables,
'transfer data to or from Matlab or perform initialisation using Matlab functions
i = 0          'sets itegrator step counter to zero

Exit Sub
Init Error:
 MsgBox "Error initializing" & Err.Description
End Sub

Sub BindData Table()
On Error Go To Bind Error
Dim result as Boolean
'Bind datatable to object
Set dt = SimCase.DataTables.Item(0)
'Bind object to EBSink class
Set notifyevent1 = New EventSink
'register instance with datatable
result = dt.AddNotifyEventSink("LBSink", notifyevent1)
'Enable Data Transfer
dtValid = True
dt.StartTransfer

Wtags = dt.WriteTags          'object linked to write only variables in PDT
Rtags = dt.ReadTags           'object linked to read only variables in PDT
RWtags = dt.ReadWriteTags     'object linked to read/write variables in PDT

Exit Sub
Bind Error:
 MsgBox "Error in Binding Data Table" & Err.Description
 Cleanup
End Sub
```

Figure 4. *Step 3.*

```
Sub APC_Algorithm()
On Error GoTo APC_AlgorithmError

'Place Advanced Process Control Algorithm here

Rvalues = dt.GetValues(Rtags)      'reads data from read tags in PDT
dt.SetValues Wtags, Wvalues'writes to write tags in PDT

Exit Sub

APC_AlgorithmError
MsgBox "Error in APC_Algorithm" & Err.Description
End Sub

Sub Cleanup()
On Error GoTo CleanupError
'This procedure terminates the HYSYS/Excel/VBA
If dtValid = True then
 'Clean up code for Notify Event
 Dim result As Boolean
 result = dt.RemoveNotifyEventSink(notifyevent1)
 Set notifyevent1 = Nothing

 dt.EndTransfer
 Set dt = Nothing
End If

If simCaseValid = True Then
 SimCase.Close
 Set SimCase = Nothing
End If
 If hyAppValid = True Then
 HyApp.Quit
 Set HyApp = Nothing
End

Exit Sub
Cleanup Error:
 MsgBox "Error in Quit" & Err.Description
End Sub
```

Figure 5. *Step 4.*

```
'Dispatch Interface
Dim instanceName As Variant

Private Sub Class_Initialize()
'Initiation Steps related to notify event interface if needed are placed
'here

End Sub

Public Function AdviseEvent()

'increments time counter with every solver event
i = i + 1

'flashes counter to Excel worksheet
xlSheet.Range("b22") = i

'after a predetermined # of integrator steps the contents of the if
'statement are implemented

If i = Switch Then
        APC_Algorithm
        i = 0
End If
End Function
```

Figure 6. *Event Sink Class Module.*

```
Public Sub QuitBttn_Click()
On Error GoTo QuitError
'allows user to terminate transfer using VBA GUI
 Cleanup
Exit Sub
QuitError:
 MsgBox "Error in Quit" & Err.Description
End Sub

Public Sub IntegratorStart_Click()
On Error GoTo StartError
'allows user to start Hysys integrator using VBA GUI
Set integrator = SimCase.Solver.Integrator
 integrator.IsRunning = True
Exit Sub
StartError
 MsgBox "Error in Start" & Err.Description
End Sub

Public Sub IntegratorStop_Click()
On Error GoTo StopError
'allows user to stop Hysys integrator using VBA GUI
Set integrator = SimCase.Solver.integrator
integrator.IsRunning = False
Exit Sub
StopError:
 MsgBox "Error in Stop" & Err.Description
End Sub
```

Figure 7. *MPC control form.*

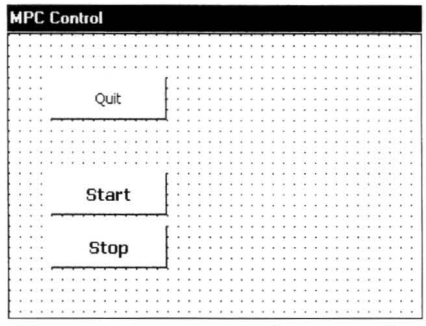


Figure 8. *Visual Basic button code.*

TABLE 1
Summary of the Excel-Matlab Link Commands

Matlab Function Syntax	*Function*
mlevalstring"Matlab command"	*Performs the string of Matlab enclosed in the quotes*
mlputmatrix mlvar *"worksheetRange"*	*Copies the matrix defined by 'worksheetRange' in the Excel worksheet to Matlab variable 'mlvar'*
mlgetmatrix mlvar "worksheetCell"	*Copies the contents of the Matlab variable 'mlvar' to the Excel worksheet. 'worksheetcell' represents the location of the upper left-hand cell of the matrix.*
mlputvar mlvar, VBvar	*Copies the contents of the Visual Basic variable 'VBvar' to the Matlab variable 'mlvar'*
Mlgetvar mlvar, VBvar	*Copies the contents of the Matlab variable 'mlvar' to the Visual Basic variable 'VBvar'*

- **Step 2.**

  Main function (see Figure 3). This function calls functions that initialize the Hysys-Excel link, binds the PDT to Excel variables, and causes the 'Form' GUI to be shown.

- **Step 3.**

  Functions that initialize the Hysys Excel/VBA link and bind the PDT to VBA variables (see Figure 4).

- **Step 4.**

  Functions that contain the APC algorithm and terminate the Hysys-Excel link (see Figure 5).

- **Step 5.**

  Insert a 'Class Module,' change its name to 'Event Sink,' and add the following code (see Figure 6).

- **Step 6.**

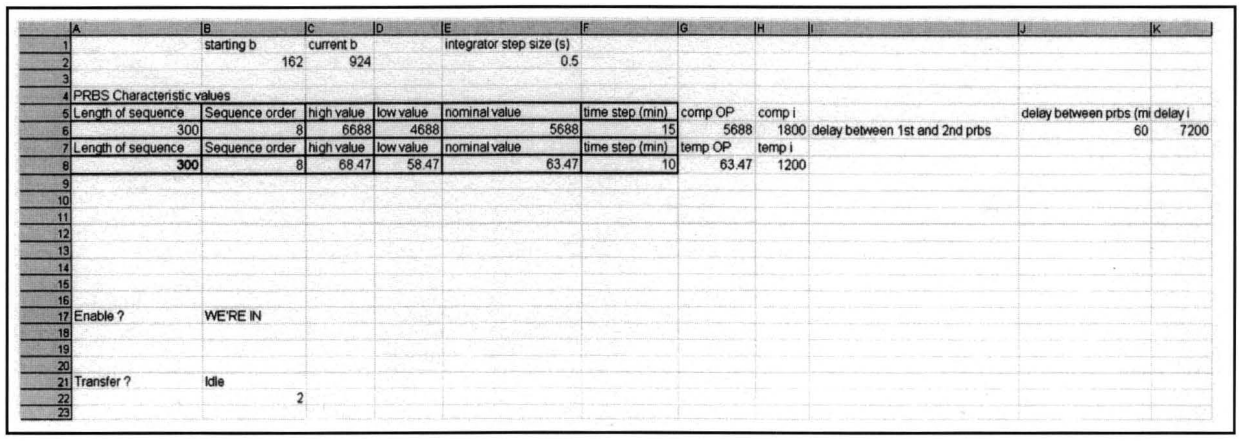
  Insert and create a 'Form' of similar structure to that in Figure 7 and insert the code in Figure 8 for the appropriate buttons.

The previous steps result in the basic structure of a Hysys-to-Excel link that will recognize when the simulation case undergoes a solver event, which may be either the steady-state solver updating the solution for a change in operating conditions or when the dynamics solver completes a time step.

At this point it would be possible to fill areas as indicated in the code in Figure 8 with an APC algorithm, using VBA and Excel alone, and then run an APC-enabled simulation case. But this would typically involve writing a substantial amount of code for routine matrix manipulation procedures and data handling, etc., effectively overshadowing the APC algorithms if the user is inexperienced. This is where the "power" of the Excel-Matlab link is most apparent. By allowing direct access to all of Matlab's functions and the ability to read and write values to both the Excel worksheet and VBA variables through function calls (summarized in Table 1) in the VBA code, the majority of the student's time can be spent developing and testing various APC algorithms. In fact, Matlab's toolbox functions, such as those from the MPC toolbox could also be used in this environment if one wished to implement Matlab's algorithms for APC on a case-study plant.

The following case study is an example of how to fill in some of the blanks in the code above to obtain a useful algorithm. It also is provided to give examples of PDT format, Matlab calls via the Excel link, and the associated VBA code.

The example details one way of implementing a series of pseudo random-binary sequences (PRBS) used in the indentification of a distillation column. The distillation is one column of the Dimethyl Ether production described in Turton, *et al.*[8] The column separates a stream primarily composed of methanol (25-40 wt%) and water (75-60 wt%) ranging in



	A	B	C	D	E	F	G	H	I	J	K
1		starting b	current b		integrator step size (s)						
2		162	924		0.5						
3											
4	PRBS Characteristic values										
5	Length of sequence	Sequence order	high value	low value	nominal value	time step (min)	comp OP	comp i		delay between prbs (mi	delay i
6	300	8	6688	4688	5688	15	5688	1800	delay between 1st and 2nd prbs	60	7200
7	Length of sequence	Sequence order	high value	low value	nominal value	time step (min)	temp OP	temp i			
8	300	8	68.47	58.47	63.47	10	63.47	1200			
9											
10											
11											
12											
13											
14											
15											
16											
17	Enable ?	WE'RE IN									
18											
19											
20											
21	Transfer ?	Idle									
22		2									
23											

▲ ***Figure 9.*** *Format of Excel worksheet used in PRBS example.*

► ***Figure 11.*** *Additions/modifications to the variable declarations and init() for the PRBS example.*

flow from 6000 to 12000 lb/h. The distillation is performed at 35-40 psia and produces high-purity water of lower than 220 ppm methanol in a column of 17 theoretical stages. The best conventional PI control configuration was determined to be LV[9] using reflux (L) to control for top-tray temperature and boil-up (V) via reboiler duty to control for bottoms methanol composition in water.

Figure 9 shows the Excel workbook that is used to hold the data that is necessary to configure the individual PRBS signals, the delay between the two signals, the starting "b" value that allows the indentification process to start at any desired pont, the means to display the link status, and the progress of the number of steps since the last data exchange. Figure 10 shows what the PDT should look like for this case. Figures 11, 12, and 13 show the necessary additions/modification to the VBA code.

Figure 14 shows a typical result when the above additions/modifications are made and the simulation is run.

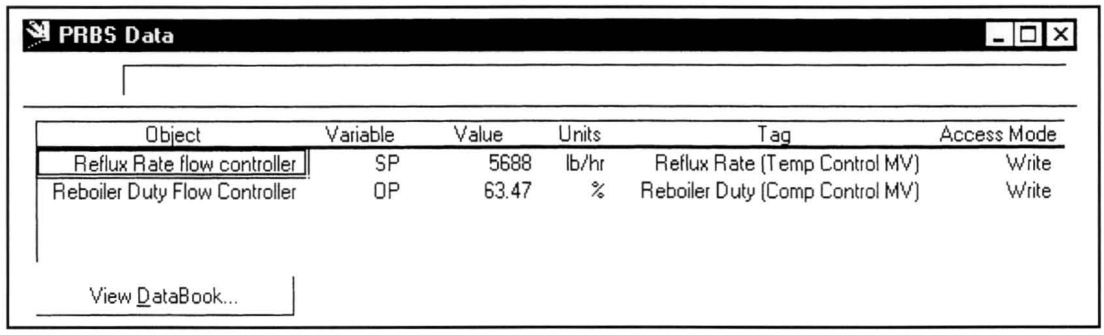
Although the environment is extremely flexible and provides easy set up, these benefits would be negated if this environment proved to dramatically slow the speed of the integrator. The performance of the simulation can be measured by comparing the real-time factor (RTF) [which is defined by (simulated time interval)/(actual time required to compute simulated time interval)] of a simulation us-

```
'The following changes should be made to the variable declarations
'remove variables associated with read tags and modify the Wvalues as follows
Public Wvalues (2) As Double
'add the following variables
Public b  As Integer 'variable that allows identification to start at given point
Public comptime As Double ' variable that holds number of integrator steps for time in comp PRBS
Public temptime As Double ' variable that holds number of integrator steps for time in temp PRBS

Public delayi As Double      ' variable that holds number of integrator steps for time delay between PRBS signals

'The following should replace Sub init()
Sub Init()
On Error GoTo InitError
 'Regular Initialization
 Set HyApp = Create Object("HYSYS.Application")
 HyApp.Visible = True
 hyAppValid = True
 Set SimCase = GetObject("c:\yourdirectories\yourfilename")
 SimCase. Visible = True
 simCaseValid = True
 Set xlSheet = Sheets("Sheet1")
 'initialization of counter and starting variables
 i = 1
 b = xlSheet.Range ("b2")
 inputs necessary values into Matlab
 mlputmatrix "compchar", xlSheet.Range ("a6:f6")
 mlputmatrix "tempchar", xlSheet.Range("a8:f8")
 mlputmatrix "delaytime", xlSheet.Range ("j6")
 mlputmatrix "integerstep", xlSheet.Range ("e2")
 calculates PRBS signals, and number of integration steps to for delay and time to next move(in terms of i)
 mlevalstring " compdist = idinput(compchar(1),'prbs',[compchar(2) 1],[compchar(4) compchar(3)])"
 mlevalstring " tempdist = idinput(tempchar(1),'prbs',[tempchar(2) 1],[tempchar(4) tempchar(3)])
 mlevalstring " comptime = round((60*compchar(6)/integerstep))"
 mlevalstring " temptime = round((60*tempchar(6)/integerstep))"
 mlevalstring " delayi = round((60*delaytime/integerstep))"
 'the following prints the above results to screen
 mlgetmatrix "comptime", "h6"
 Matlabrequest                'forces data transfer to occur between vba/Excel and matlab
 mlgetmatrix "temptime", "h8"
Matlabrequest
 mlgetmatrix "delayi", "k6"
 Matlabrequest
 Exit Sub
InitError:
 MsgBox "Error initializing" & Err.Description
End Sub
```

Figure 11. *Additions/modification to the variable declarations and init() for the PRBS example.*



▲ ***Figure 10.*** *PDT for the PRBS example.*

◄ ***Figure 12.*** *Additions to the VBA module for the PRBS example.*

```
'The following procedures need to be added to the module
Sub WriteOPcomp()
On Error GoTo WriteOPcomperror
'Procedure that writes values to PDT when prbs signal is on temperature is activated
'the following transfer Matlab data to Excel worksheet
mlevalstring "comp = compdist(a)"
mlgetmatrix "comp", "g6"  'calculated prbs value for temp disturbance
Matlabrequest
mlevalstring "temp = tempchar(5)"
mlgetmatrix "temp", "g8"  'nominal value for temp
Matlabrequest
"The following transfers data to PDT
Wvalues(0) = xlSheet.Range("g6")   'input to comp
Wvalues (1) = xlSheet.Range("g8")   'input to temp
dt.SetValue Wtags, Wvalues
Exit Sub
Write OPcomperror:
 MsgBox "Error in WriteOPcomp" & Err.Description
 Cleanup
End Sub
Sub Write OPtemp()
On Error GoTo Write OPtemperror
 Procedure that writes values to PDT when PRBS sigal is on temperature is activated
 the following transfer Matlab data to Excel worksheet
 mlevalstring "comp = compchar(5)"
 Matlabrequest
 mlevalstring "temp = tempdist((a-compchart(1)-1))"
 mlgetmatrix "temp", "g8"   'calculated prbs value for temp disturbance
 Matlabrequest
'The following transfers data to PDT
 Wvalues(0) = xlSheet.Range ("g6")   'input to comp
 Wvalues (1) = xlSheet.Range("g8")   'input to temp
 dt.SetValues Wtags, Wvalues
 Exit Sub
Write OPtemperror
 MsgBox "Error in Write OPtemp" & Err.Description
 Cleanup
End Sub
Sub WriteOPnom()
On Error GoTo Write OPnomerror
'Procedure that write nominal values to PDT
'the following transfer Matlab data to Excel worksheet
mlevalstring "comp = compchar(5)"
mlgetmatrix "comp", "g6"
Matlabrequest
mlevalstring " temp = tempchar(5)"
mlgetmatrix "temp", "g8"
Matlabrequest
'The following transfers data to PDT
Wvalues (0) = xlSheet.Range ("g6")
Wvalues (1) = xlSheet.Range ("g8")
dt.SetValues Wtags, Wvalues
Exit Sub
Write OPnomerror
MsgBox "Error in WriteOPnom" & Err Description
Cleanup
End Sub
```

▼ ***Figure 13.*** *Additions/modifications to AdviseEvent() for the PRBS example.*

```
'Replace the contents of public function events with the following
Public Function AdviseEvent()

i = i + 1        'i is a counter which counts the number of solver steps that occured
xlSheet.Range ("b22") = i   'flashes i to the worksheet
xlSheet.Range ("c2")                'flashes b to the worksheet
mlputmatrix "a", Cells (2,3)   'allows Matlab to see the progress in the indentification
                                    'sequence
'The contents of this if statement are responsible for the first prbs sequence
If i = xlSheet.Range("h6") And b <= xlSheet.Range ("a6") Then
  WriteOPcomp
  b = b + 1
  i = 0
End If
'The contents of this if statement are responsible for the delay between sequences
If i = xlSheet.Range("k6") And b =  (xlSheet.Range("a6") + 1) Then
  Write OPnom
  b = b + 1
  i = 0
End If
'The contents of this statement are responsible for second prbs sequence which follows the
                'delay
If i = xlSheet.Range("h8") and b>=(xlSheet.Range("a6")+2) And b<(xlSheet.Range("a6") +
xlSheet.Range("a8") + 2) Then
  Write OPtemp
  b = b + 1
  i = 0
End if
'Stop integrator when identification sequence ends
If b = (xlSheet.Range("a6") + xlSheet.Range ("x8") + 2) Then
  Set integrator = SimCase.Solver.Integrator
  integrator.IsRunning = False
  End If
End function
```

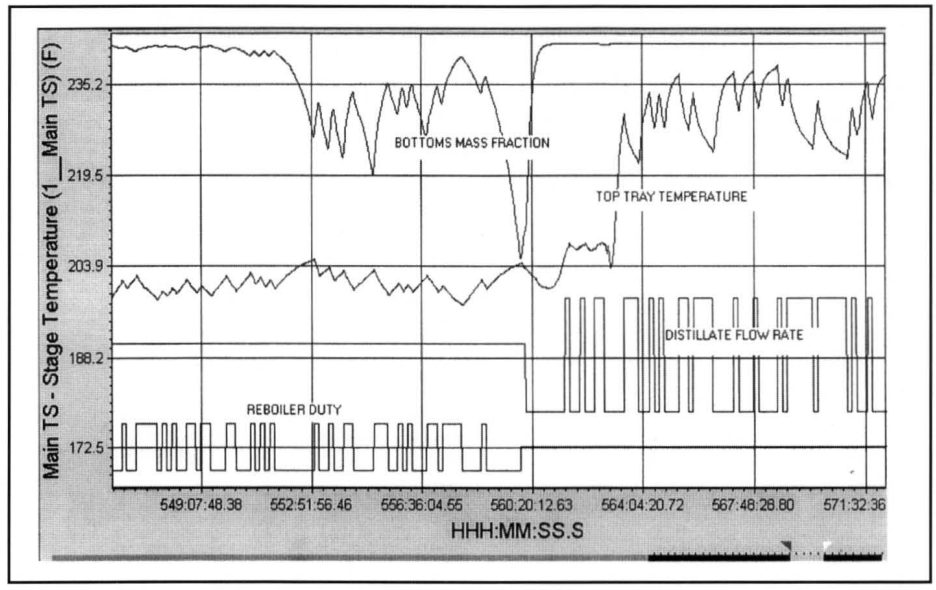


Figure 14.

Typical result using the environment for the PRBS example.

(The x-axis shows simulation time in hours, minutes, and seconds, and the y-axis shows the trends of various process variables.

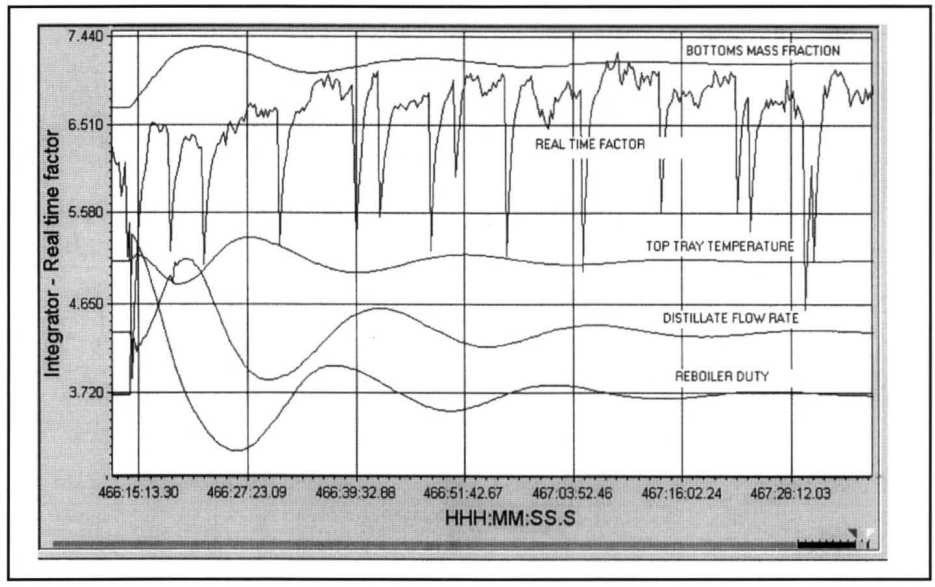


Figure 15.

Response to a bottoms methanol composition set point change using PID controllers.

(The x-axis shows simulation in hours, minutes, and seconds, and the y-axis shows the trends of various process variables.)

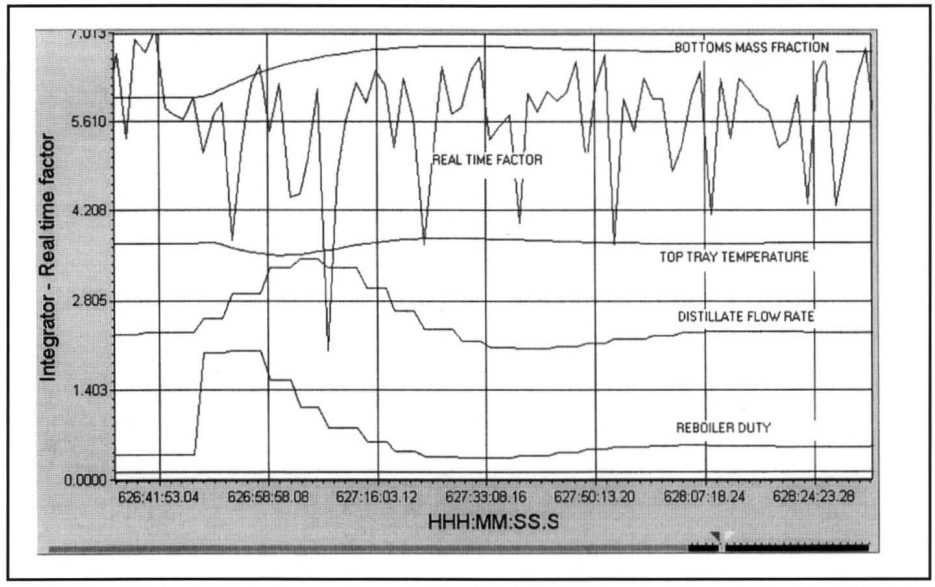


Figure 16.

Response to a bottoms methanol composition set point change using a linear 2x2 DMC algorithm developed using the environment.

(The x-axis shows simulation in hours, minutes, and seconds, and the y-axis shows the trends of various process variables.)

ing the environment to one that does not. Figure 15 shows the system response using PID controllers to control for bottom's composition and top-tray temperature. Figure 16 shows the same distillation process with control carried out using a linear 2x2 Dynamic Matrix Controller (DMC)[10,11] that has been implemented using the link described in this paper. It can be seen that the RTF for the 2x2 DMC controller case is comparable to that of the PID controller case, and also gives better performance in terms of controller movement and oscillation around the set point.

CONCLUSIONS

The development of an integrated, real-time computing environment for advanced process control development and education using Hysys, Excel, and Matlab linked with each other has been outlined in this article. The methodology used to develop the environment was detailed in order to enable the reader to substantially reduce the learning curve involved in developing the communication structure itself, thus allowing a means to focus the attention onto a large variety of APC algorithms. The potential of the environment has been demonstrated using an example that implements a PRBS identification sequence on a methanol water distillation column simulation.

REFERENCES

1. Ramaker, B.L., H.K. Lau, and E. Hernandez, "Control Technology Challenges for the Future," in *Proc. CPC V,* J.C. Kantor, C.E. Garcia, and B. Carnahan, eds, *AIChE Symp. Series No. 316,* **93,** 1 (1997)
2. Qin, S.J., and T.A. Badgwell, "An Overview of Industrial Model Predictive Control Technology," in *Proc. CPC V,* J.C. Kantor, C.E. Garcia, and B. Carnahan, eds, *AIChE Symp. Series No 316,* **93**, 232 (1997)
3. Edgar, T.F., "Process Control Education: Past, Present, and Future," in "Chemical Engineering Education: Curricula for the Future," *Proc. Indo-US Seminar,* D. Ramkrishna, P.B. Deshpande, R. Kumar, and M.M. Sharma, eds, Bangalore, India, pp. 117 (1998)
4. Edgar, T.F., "Process Control Education in the Year 2000: A Roundtable Discussion," *Chem. Eng. Ed.,* **24**(3), 72 (1990)
5. Rhinehart, R.R., S. Natarajan, and J.J. Anderson, "A Course in Process Dynamics and Control: An Experience to Bridge the Gap Between Theory and Industrial Practice," *Chem. Eng. Ed.,* **29**(4) 218 (1995)
6. Doyle, III, F.J., E.P. Gatzke, and R.S. Parker, "Practical Case Studies for Undergraduate Process Dynamics and Control Using Process Control Modules," *Comp Appl. in Eng. Ed.,* **6**, 181 (1998)
7. Doyle III, F.J., V. Venkatasubramanian, and T.A. Kendi, "Purdue Control Modules: A Flexible Set of Software Modules for an Undergraduate Process Dynamics and Control Lboratory," *Comp. Apps. Eng. Ed.,* **4**(3), 179 (1996)
8. Turton, R., R.C. Bailie, W.B. Whiting, and J.A. Shaeiwitz, *Analysis, Synthesis, and Design of Chemical Processes,* Prentice-Hall, Upper Saddle River, NJ (1998)
9. Shinskey, F.G., *Distillation Control for Productivity and Energy Conservation,* 2nd ed., McGraw Hill, New York, NY (1984)
10. Cutler, C.R., and B.L. Ramaker, "Dynamic Matrix Control: A Computer Control Algorithm," *AIChE National Meeting,* Houston, TX (1979)
11. Cutler, C.R., and B.L. Ramaker, "Dynamic Matrix Control: A Computer Control Algorithm," in *Proc. Joint Automatic Cont. Conf.,* paper WP5-B (1980) ❐