Using Maple for Classroom and Laboratory Instruction in Process Control

P. A. Ralston, M. J. Maron
University of Louisville

Abstract

The ability to simulate control systems is necessary for many undergraduate and graduate engineering courses and laboratories. In particular, process control instruction is enhanced by tools that provide convenient graphical support and allow reliable numerical computation. Computer Algebra Systems (CAS) provide such support, and rapid accurate symbolic computation as well. This paper describes two ways that the capabilities of the CAS Maple have been used to enhance instruction in process-control lectures and laboratories. The first is the use of built-in Maple commands, the second is the use of a package of procedures written for particular applications. These procedures serve as "higher level" Maple commands that make useful information accessible to students and faculty who have limited Maple expertise. The package is currently being used with considerable student and teacher satisfaction, and improvements are planned.

Background

Recent articles have described how computer tools like Mathematica, Maple, Mathcad, and Matlab were used to apply the principles of process control [Ogunye (1995,1996a,b); Munro and Tsapekis (1994); Ohtani, Fukuzawa, and Masubuchi (1994); Ralston and Maron (1997)], and a recent ASEE Frontiers in Education Conference [1996] dedicated an entire session to the use of Matlab in teaching process control. All of these software tools provide effective two and three dimensional graphs and a powerful programming environment, and each has particular strengths relative to the others.

Due to a variety of factors, including cost, performance, and familiarity, the University of Louisville purchased a global site license for MapleV, Release 4. This paper describes how Maple procedures were used to augment Maple's "built-in" capabilities in teaching process control.

Two Ways to Use Maple in Instruction

Computer Algebra Systems (CAS) like Maple or Mathematica have commands for symbolic, numeric, and graphical operations. These commands, generally entered interactively, are actually calls to built-in procedures that have the commands as their names. CAS commands are special in that they can accept and return symbolic or graphical objects as well as numerical ones. For example, the Maple command

```
plot( [x^2,sin(x)], x=-Pi..Pi,
                    color=[red,blue] );
```

has three calling parameters: one list of two algebraic expressions, x^2 and $\sin(x)$, and two equations, one to provide details about the horizontal plot range and one to provide details about the color. The command returns (and displays) a plot structure that can be stored for later re-use if desired. Terminating the command with a colon rather than a semicolon suppresses the display.

Built-in CAS commands enable rapid, flexible, neatly-displayed symbolic and graphical output that can be used to enhance instruction. In particular, they can be used to demonstrate the effects of varying system parameters and for a rapid review of prerequisite material (e.g. DE and Laplace transforms for a course such as process control).

In addition to built-in commands, modern CASs have extensive programming power. CAS users with advanced skills can write procedures that become "higher-level" *discipline-specific commands* that can be used just like built-in CAS commands. In this way, advanced CAS capabilities can be made available to novice students and instructors. In Maple, collections of related user-created procedures can be put into a *package* and loaded for use like any built-in Maple package. This paper describes a package that was developed for process control courses at the University of Louisville.

Motivation for a Process-Control Package

Chemical engineering faculty at UofL seek to use effective, reasonably-priced software tools in teaching lecture and laboratory courses in process control. After investigating several options, the decision was made to create a single, inexpensive CAS-based software package to facilitate understanding and application of process control concepts, and through its use, demonstrate this understanding. The chosen CAS was Maple, for which UofL has a multi-platform site license. Institutions with a similar license for Mathematica, Matlab, or Mathcad (some of which have a separate toolkit for control theory) can get comparable results with those tools.

Needed for chemical engineering process control assignments is the ability to simulate, at a minimum, open and closed loop continuous and sampled systems for processes that exhibit deadtime. In addition, basic systems concepts such as solving differential or difference equations using Laplace or z-transforms must be reviewed or demonstrated effectively. The process control laboratory course has a large simulation component, to reinforce single loop enhancements such as feedforward control, deadtime compensation, and multivariable control. In the past, students in these courses typically used a variety of computer tools, including Maple, spreadsheets, and some special-purpose control software. Students often spent considerable time learning to use the latter tools in order to complete the activities that required them.

In Fall 1996, students in the graduate process control course were asked to use Matlab with SIMULINK and the Control System Toolbox for some projects and Maple procedures as described above for others, and then to compare them. In most respects, students found the Matlab suite and Maple procedures equally easy to use. The only significant drawback of the Maple procedures was the lack of a graphical interface like the one SIMULINK provides. The ability to change a simulation by manipulating a block diagram instead of a procedure call was very appealing. This student response provided motivation to continue writing Maple procedures and plan to add a graphical interface. The final motivation came when the 1997 license renewal cost for one of the special-purpose simulation tools increased dramatically.

Process-Control Package Description

The Maple procedures in the process-control package currently fall into three categories:

1. Single-Loop Control Simulation

- **PID( )** *Proportional Integral Derivative (PID)* control of a continuous or sampled system.
- **FF( )** *Feedforward* and/or feedback PID control of a sampled system.
- **PIDDTC( )** *Deadtime Compensation* using preliminary or final Smith predictor
- **IMC( )** *Internal Model Control* for a sampled system
- **SMPC( )** Simplified *Model Predictive Control* for a sampled system.

2. Multiple Input/Output PID Control Simulation

- **PI2( )** 2x2 sampled systems under PID control.

3. Analysis Tools

- **checkdata( )** *displays data* entered as lists.
- **linfit( )** *fits a straight line* to given data and displays the result graphically.
- **detind( )** calculates the *determination index* r^2 of a least squares fit.
- **bode1( )** *Bode plot* of individual factors of G(s) along with overall AR and ϕ.
- **bode2( )** *Bode plot* of several G(s)s and/or (ω, AR) data sets on a single graph.

Three Illustrative Examples

We now give one lengthy example to illustrate the instructional use of built-in Maple commands and two short examples that illustrate procedures **bode1( )** and **bode2( )** in the process-control package

EXAMPLE 1 Inverse Response and Overshoot

One of the more difficult concepts to describe in class is how a system with two first-order processes in a parallel structure (Figure 1) can exhibit inverse response and overshoot.

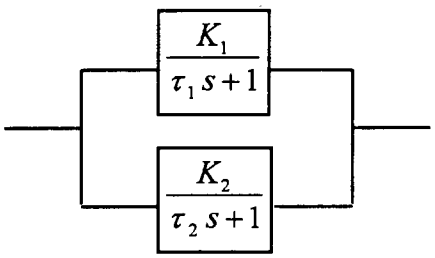


Figure 1 Two first order processes in parallel

The following Maple steps can be executed in class to provide a convincing demonstration of conditions under which inverse response and overshoot occur. Write the overall process transfer function *Gp* as the sum of two first order processes *G1* and *G2*:

```
> G1:=K1/(tau1*s+1): G2:=K2/(tau2*s+1):
> Gp := collect(normal(G1+G2),s):
> 'G1'=G1,`  G2`=G2; 'Gp'=Gp;
```

$$G1 = \frac{K1}{\tau 1\, s+1},\quad G2 = \frac{K2}{\tau 2\, s+1}$$

$$Gp = \frac{(K1\,\tau 2 + K2\,\tau 1)\,s + K1 + K2}{(\tau 1\, s+1)(\tau 2\, s+1)}$$

Note that although neither *G1* nor *G2* has a zero, $Gp = 0$ when numerator(Gp) = 0, i.e., when $s = -1/\tau 3$, where

$$\tau 3 = \frac{K1\,\tau 2 + K2\,\tau 1}{K1 + K2}. \qquad (*)$$

Set $K2 = 1 - K1$ (to normalize the gain of *Gp* to 1), then solve (*) for *K1* and store the resulting formula as *K1sol*:

```
> tau3=-1/solve( numer(Gp)=0, s ):   # (*)
> K1sol := solve(subs( K2=1-K1,"), K1);
```

$$K1sol = \frac{-\tau 3 + \tau 1}{-\tau 2 + \tau 1}$$

The step response is the inverse Laplace transform of *Gp/s*. To get it, use the **invlaplace( )** command, which is in Maple's **inttrans** package:

```
> inttrans[invlaplace](Gp/s,s,t):
> ystep := collect( " , exp );
```

$$ystep := -K1\, e^{\left(-\frac{t}{\tau 1}\right)} + (K1 - 1)\, e^{\left(-\frac{t}{\tau 2}\right)} + 1$$

Question: **How does *ystep* vary with $\tau 3$ (or *K1*) for fixed values of $\tau 1$ (say 2) and $\tau 2$ (say 1)?**

```
> tau1 := 2: tau2 := 1:   # easily changed
```

These assigned values of *tau1* and *tau2* are automatically substituted in *ystep* and *K1sol*. We show this for *ystep*:

```
> 'ystep'=ystep;
```

$$ystep := -K1\, e^{(-t/2)} + (K1 - 1)\, e^{(-t)} + 1$$

The next two commands create a list of $\tau 3$ values and then display a list of the corresponding *K1*values.

```
> tau3list := [4,3,2,3/2,1,0,-1,-2,-3]:
> K1list=map(unapply(K1sol,tau3),");
```

$$tau3list := \left[4, 3, 2, \frac{3}{2}, 1, 0, -1, -2, -3\right]$$

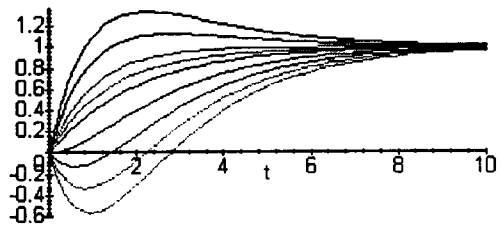
$$K1list = \left[-2, -1, 0, \frac{1}{2}, 1, 2, 3, 4, 5\right]$$

The following **for loop** stores a sequence of *ysteps* as *yseq* and a sequence of titled *plots* of *ystep* as *plseq*.

```
> yseq := NULL: plseq := NULL:
> for t3 in tau3list do
> cat( `tau1 = `,2,`, tau2 = `,1,
       `,tau3=`,convert(t3,string),
         K1 = `,convert("",string) ):
>   subs(K1=subs(tau3=t3,K1sol),ystep):
>   plseq:=plseq,plot(",t=0..10,title=""):
>   yseq := yseq,"":
> od:
```

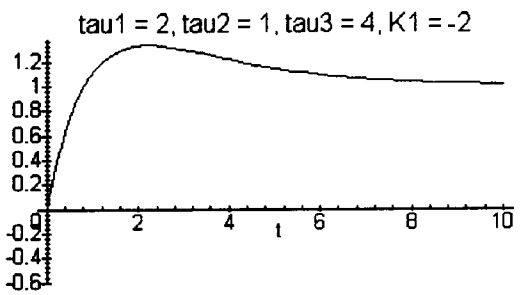
One way to display how *ystep* varies with $\tau 3$ (or *K1*) is to plot *ystep* for the selected values.

```
> plot( [yseq], t=0..10,
        color=[black,blue,green,tan,red,
               brown,magenta,cyan,pink] );
```



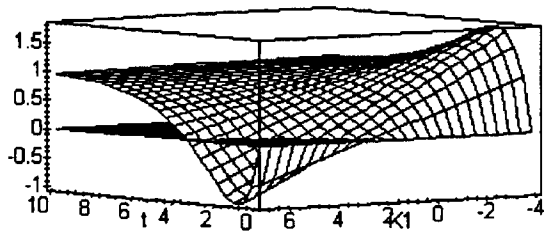
A second way is to "animate" the graph of *ystep*. The frame for $\tau 3 = 4$ is shown. The others are displayed by using the mouse to step through the remaining frames or to run the animation.

```
> display( [plseq], insequence=true );
```



A final way is to plot *ystep* as a function of (*t*, *K1*). The horizontal plane $y = 0$ is also plotted to see clearly when *ystep* changes sign (inverse response).

```
> plot3d( {ystep,0}, t=0..10, K1=-4..7 );
```



All three displays indicate overshoot for $K1 < 0$ and inverse response for $K1 > 2$. One can then examine the general expression for *ystep* (just before ***Question***: above) to see why this makes sense when $\tau_2 < \tau_1$.

EXAMPLE 2 Bode plots showing factors of G(*s*)

The *amplitude ratio AR* of a transfer function G(*s*) is the magnitude of G(*i* ω), that is, | G(*i* ω) |, and its *phase lag* ϕ is the argument of G(*i* ω). A *Bode plot* of G(*s*) is a plot of $\log_{10}(AR)$ and/or ϕ versus $\log_{10}(\omega)$. Procedure **bode1( )** can display Bode plots of individual factors of G(*s*) along with the overall *AR* and ϕ. The syntax for calling **bode1( )** is:

```
bode1( Gs, omegarng, hdrstr, dtails )
```

where **Gs** is the transfer function, assumed to be a function of *s*; **omegarng** is *a* . . *b*, the ω range to plot; and **hdrstr** is `**fraction**`, `**as is**`, or a positive integer giving the number of significant digits to display.

When **bode1( )** is called with **dtails = true**, Maple's algebraic capabilities are used to factor G(*s*) into the gain *K*, a delay $e^{(-\theta s)}$ (if any), and normalized numerator and denominator factors of the form

$$\tau s + 1 \quad \text{or} \quad \tau^2 s^2 + 2\tau\zeta s + 1,$$

and Bode plots for each factor are shown along with that for G(*s*) itself. We illustrate this for a G(*s*) that has one real pole, one real zero, and a delay.

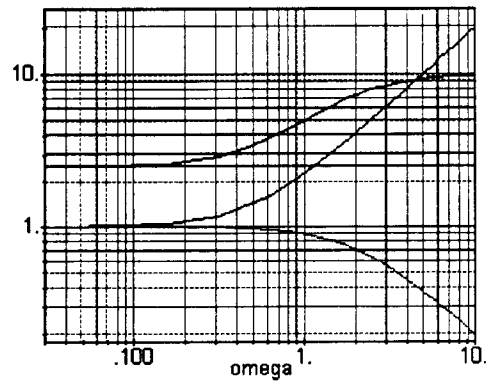
```
> G:=(10*s+5)/(s+2)*exp(-2*s):
> bode1( G, .03..10, `fraction`, true ):
```

$$\text{Bode Plots for } G(s) = \frac{(10s+5)\,e^{(-2s)}}{s+2},$$

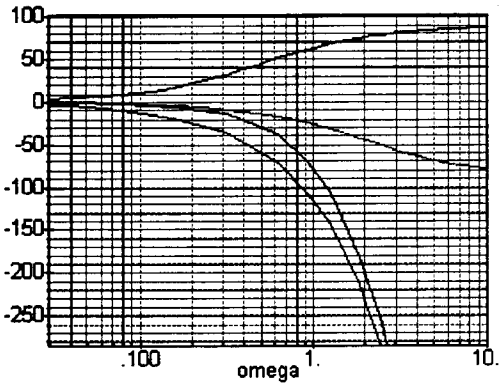
$$G(s) = K\,G_1(s)\,G_2(s)\,G_3(s), \quad \textit{where}$$

$$K = \frac{5}{2},\ G_1(s) = e^{-2s},\ G_2(s) = 2s+1,\ G_3(s) = \frac{1}{\frac{1}{2}s+1}$$

AR: G=black, K=gold, G1=red, G2=blue, G3=green



Phase: G=black, G1=red, G2=blue, G3=green



Plots such as these help the user gain an experiential sense of how individual factors of G(*s*) contribute to overall amplitude and phase behavior.

EXAMPLE 3 Bode plots for a variety of G(*s*)'s

Procedure **bode2( )** plots any combination of up to three G(*s*) formulas and up to three lists of numeric (ω, *AR*) data. All analytic functions and data sets are put in a single list **GL** and plotted together on amplitude ratio and phase lag plots. The syntax for calling **bode2( )** is

```
bode2( GL, omegarng, plotttl )
```

where $\mathbf{GL}_i$ is either a transfer function G(*s*) or a data list of the form [*omegaList*, *ARList*]; **omegarng** is the ω range *a* . . *b*; and **plotttl**, the desired plot title, is a character string in backquotes (` `).

Procedure **bode2( )** was written to expedite process control laboratory experiments. The following call to **bode2( )** is from the documentation for an experiment to study second-order underdamped behavior, with amplitude data obtained empirically over a range of frequencies. The list **GL** contains two analytic G(*s*)'s, *G3theo* and **G3semi**, and one list [*omeg3L*, *AR3L*] of empirically obtained (ω, *AR*) data.

```
> tau :='tau': zeta:='zeta':
> Gs:=1/((tau*s)^2+2*zeta*tau*s+1):
> G3theo:=subs(tau=.21,zeta=.05, Gs):
> G3semi:=subs(tau=.21,zeta=.16, Gs):
> GL:=[G3theo, G3semi, [omeg3L,AR3L]]:
> bode2( GL, .01..100, `Manometer 3` );
```

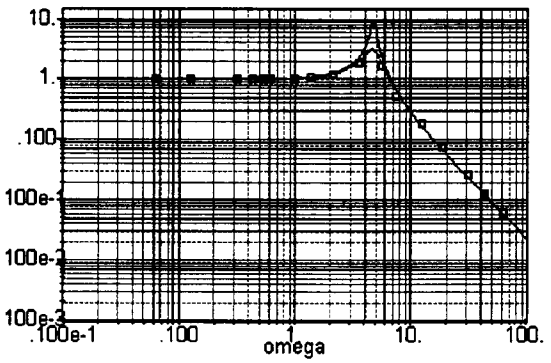
BODE PLOTS for CONTINUOUS G(s)'s and/or DISCRETE ARk LISTS

$$G(s)\text{'s, (black, ___)} = \frac{1}{.0441\,s^2 + .0210\,s + 1},$$

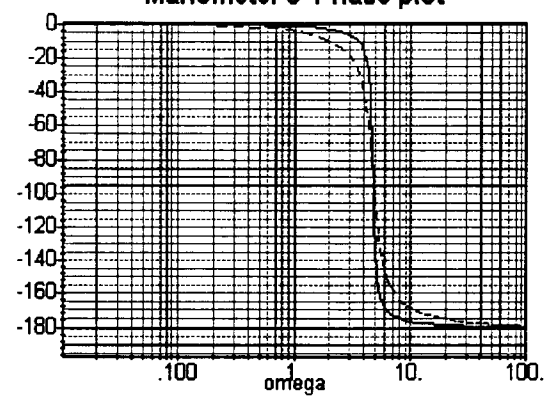
$$\text{(red, ---)} = \frac{1}{.0441\,s^2 + .0672\,s + 1}$$

Ark's, Ark_1 = black box

Manometer 3 Amplitude plot



Manometer 3 Phase plot



On the output, the subscripts on the G(*s*)s and *ARk*'s refer to the order in which they occur in the **GL** list (so *G1* is *G3theo* and G2 is *G3semi*). The discrete data were simulated using slight perturbations of theoretical G(*s*) values. Actual empirical data are unlikely to be nearly as accurate!

Summary and Work in Progress

This paper described a Maple package used for undergraduate and graduate process control instruction at the University of Louisville. Three examples were presented. One showed how "built-in" CAS commands can effectively demonstrate the effects of parameter variation in a second order overdamped system with parallel structure. The other two showed how procedures that generate enhanced Bode plots can effectively display important features of analytic and/or or empirically-obtained transfer functions. The symbolic capability of the package goes beyond most existing process control software. It is currently being used with considerable student and teacher satisfaction.

Work in progress includes providing a graphical interface to formulate simulations, extending the multiple-input multiple-output capabilities to *n* by *n* systems, including model predictive strategies, and incorporating the ability to model the effects of noise to enable the simulation of stochastic systems.

References

1. Marlin, T. E., *Process Control – Designing Processes and Control Systems for Dynamic Performance*, McGraw Hill, New York, 1995.

2. Munro, N. and Tsapekis P., *Some Recent Results Using Symbolic Algebra*, Proceedings of the IEEE/IFAC Joint Symposium on Computer-Aided Control System Design, Tucson, Arizona, March, 1994, pp.109-116.

3. Ogunye, A. B., *Process Control and Symbolic Computation: An Overview with Maple V*, Maple Technical Journal, vol. 3, no. 1, 1996a, pp. 94-103.

4. Ogunye, A. B., *Advanced State Space Analysis Using Computer Algebra*, Proceedings of the American Control Conference, Seattle, Washington, vol. 1, June 1995, pp. 559-563.

5. Ogunye, A. B. and Penlidis A., *State Space Computations Using Maple V*, IEEE Control Systems, 16(1), 1996b.

6. Ohtani, T., Fukuzawa, M., and Masubuchi, M., *A CAD System for Nonlinear Control System Design Using Mathematica*, Proceedings of the IEEE/IFAC Joint Symposium on Computer-Aided Control System Design, Tucson, Arizona, March, 1994, pp.197-204.

7. Ralston, P. and Maron, M., *Computer Algebra System Applications in Process Control*, Proceedings of the 12th IEEE International Symposium on Intelligent Control, Istanbul, Turkey, July 1997, pp. 221-225.

8. Session on *MATLAB uses in Control*, ASEE/IEEE Frontiers in Education Conference, Salt Lake City, Utah, November, 1996.

PATRICIA A. RALSTON

Pat Ralston holds a joint appointment in the Chemical Engineering department, where she teaches process control courses, and the Engineering Mathematics and Computer Science department, which is responsible for teaching the required and elective mathematics courses for all engineering programs at the University of Louisville. Her current research interests are in process modeling, design, and control, particularly in the application of fuzzy set theory to controller design.

MELVIN J. MARON

Mel Maron teaches in the Engineering Mathematics and Computer Science department. His background is in electrical engineering and mathematics. He worked at Bell Telephone Laboratories, the Polytechnic Institute of Brooklyn, and the University of Glasgow before coming to the University of Louisville in 1971. His current research interests are numerical analysis and computer-assisted instruction, with particular emphasis on symbolic computation.