An Object-Based Software Package for Interactive Control System Design and Analysis

Yong Zhu
Post Doctor

Bo Chen
Graduate Research Assistant

Harry H. Cheng
Professor, Member ASME[1]
e-mail: hhcheng@ucdavis.edu

Integration Engineering Laboratory, Department of Mechanical and Aeronautical Engineering, University of California, Davis, CA 95616

Ch is an embeddable C/C++ interpreter. It was developed to allow software developers to use one language, anywhere and everywhere, for any programming task. Ch supports C99, a latest C standard ratified in 1999, and contains salient features for two and three dimensional plotting and numerical computing for applications in engineering and science. Developed in Ch, Ch Control System Toolkit provides a control class with member functions for object-based interactive modeling, analysis, and design of linear time-invariant control systems. The software package, available for downloading on the web, has been widely used in industry to solve practical engineering problems and in universities for instructional improvement. The design and implementation of Ch Control System Toolkit are described in this paper. Two application examples of control system design and analysis using Ch Control System Toolkit demonstrate its power and simplicity.
[DOI: 10.1115/1.1630815]

1 Introduction

The modern control engineering design and analysis problems have grown increasingly complex with both technological breakthroughs and theoretical advances over the last few decades. The classical methods which heavily rely on pencil and paper are not capable of solving the problems efficiently. A considerable number of software packages have been developed to bridge the gap between modern control theory and the software/hardware implementation of the algorithms related to the chosen methodology. The existing well-known interpretive software packages, such as MATLAB Control System Toolbox [1], MATRIXx [2], and Mathematica's Control System Professional [3] are commercially available for the purpose of computer-aided control system design (CACSD). In the program developing process using these software packages, a user typically realizes the algorithms in a program written in these special languages. For example, m-files are used in MATLAB. In order to speed up loops that cannot be vectorized and take advantage of previously written code in C and C++, these software packages provide mechanisms to interface with external C/C++ programs. But such interfaces are quite cumbersome. For example, to invoke C functions from MATLAB, a user has to add MATLAB interface code to the original C code and compile them to create MEX file [4].

In addition to commercial software packages, many other CACSD packages, such as SLICOT Control and System Library [5], Scilab [6], Octave Control System Toolbox [7], and Rlab [8] are available under different open source licenses. The SLICOT Control and System Library is written in FORTRAN. Applications using SLICOT library have to be compiled before execution. It cannot be used to perform interactive design and analysis. When the parameters of the system change, the result of this change cannot be observed immediately. The user needs to modify the source code, recompile it, and run the executable code again. The Octave Control System Toolbox (OCST) uses a C struct like data structure to represent systems and allows users to pass a single variable for each dynamic system passed to OCST functions, regardless of the selected internal representation. Octave provides some C-style input and output functions, but it cannot integrate C functions directly.

Recently, Cheng [9,10] has developed Ch which is an embeddable C/C++ interpreter. Ch is a superset of C. It supports all features of the C language standard ratified in 1990. Many new features such as complex numbers, variable length arrays, IEEE floating-point arithmetic and type-generic mathematical functions first implemented in Ch had been adopted in C99, the latest C standard ratified in 1999. As a result, Ch contains more new features in C99 than most existing C compilers. In addition, Ch supports classes in C++ for object-based programming. Furthermore, Ch has built-in support of two and three-dimensional graphical plotting features and computational arrays for matrix computation and linear system analysis with advanced numerical analysis functions based on LAPACK. As a superset of C interpreter, all C programs without any modification can run in Ch interactively without tedious edit/compile/link/debug cycles, which greatly simplifies the programming tasks for many engineering applications.

Taking the advantage of efficient numerical implementation, accuracy, and reliability of Ch, we have developed Ch Control System Toolkit [11] for interactive design and analysis of control systems. Ch Control System Toolkit is object-based. It provides a CACSD environment to perform basic control system modeling, analysis, and design in both time and frequency domain with a user friendly graphical representation. Most functions in the Ch Control System Toolkit utilized advanced numerical methods and control algorithms [12,13]. Unlike other interactive CACSD packages, existing C/C++ source code can be integrated with Ch Control System Toolkit application programs without any modification.

Different from other interactive CACSD packages, Ch Control System Toolkit is the first interactive CACSD environment not only developed in the framework of C/C++, but also can seamlessly interface with the existing C/C++ code. A web-based control system design and analysis system [14] has been developed using the control class described in this paper. The system is readily available to use through the web without any software installation, system configuration and programming. In this paper, design and implementation of Ch Control System Toolkit are described along with its powerful features. The power and simplicity of the software system are illustrated by several practical examples of control system design and analysis in both time domain and frequency domains.

2 Design and Implementation of the Ch Control System Toolkit

2.1 Object-Based Design. A high quality computer-aided control system analysis and design package should be easily extendible to incorporate new control theories and techniques. This can be achieved by object-oriented software design [15]. The fundamentals of object-oriented methods for computer-aided control system design have been described and illustrated in [16]. Ch Control System Toolkit is an object-based software package. It is easy to maintain and extend. A linear time invariant (LTI) system

[1]Address all correspondence to this author.

Contributed by the Embedded/Ubiquitous Committee for publication in the JOURNAL OF COMPUTING AND INFORMATION SCIENCE IN ENGINEERING. Manuscript received January 2003; revised manuscript received October 2003. Associate Editor: P. Wright.

is modeled as an object of a class named CControl. CControl class contains both private data and member functions that operate on the data. The private data members of the class represent the attributes of the system. The functions to perform control system analysis and design are defined as public member functions of the class. The data members of the class are designed to contain all the attributes of three common LTI models in transfer function (TF), zero-pole-gain (ZPK), and state-space (SS). The use of a single class to represent all LTI system models greatly simplifies the programming and maintenance of Ch Control System Toolkit functions, because it is no longer necessary to identify which representation type is being passed in the internal representation. Member functions are encapsulated in the class, which can avoid the conflict of name spaces. The extension of Ch Control System Toolkit can be easily achieved by adding more public functions in the class.

2.2 Data Structure. The data structure is one of the most important factors to the success of a complex CACSD software package. As mentioned above, an LTI system is modeled as a class named CControl in Ch Control System Toolkit. The data structure of the CControl class defined in the header file control.h, with some members omitted for clarity of presentation, is shown as follows.

```
class CControl {
  private:
    /* private data members */
    double *m_Num;      // coefficients of numerator
    double *m_Den;      // coefficients of denominator
    double *m_A;        // matrix A
    int m_isdisc;   // discrete system flag
    double m_ts;   // sample time of discrete-time systems
    ...
    /* private member functions */
    // Adjust default response time
    int m_adjustRespTime(array double a[:], int num1, int num2,
    double tol);
  ...
  public:
    CControl();    // constructor
    ~CControl(); // destructor
    int step(class CPlot *plot, array double &yout, array double
    &tout, array double &xout, ... /* double tf */);
    // step response
      ...
}
```

The private data members of CControl class contain all information of three common LTI models in TF, ZPK, and SS. The coefficients of polynomials of the numerator and denominator of transfer functions; the zeros, poles, and gain of the system; and the values of system matrices in state-space are stored in arrays in the system. The addresses of these arrays are recorded by the private data members of the CControl class. For example, private data *m_Num* is a pointer to an array, which stores the coefficients of a polynomial for the numerator of a transfer function. When a system is created using one of three models, the Ch Control System Toolkit automatically calculates other two types of models of the system internally and keeps all information. The other attributes of the systems such as gain and phase margins, crossover frequencies, discrete or continuous system flag, sampling time of discrete system, input and output delays, etc. are all defined as private data members in CControl class. All functions for control system analysis and design are defined as public member functions in CControl class. They serve as an interface between user's applications and control class. Only those member functions can be used in an application program. For example, to plot a step response of a system, the public member function step() can be called in the application program. The private member functions defined in CControl class are used to support the operations of other member functions. As an example, the private member function m_adjustRespTime() is aimed to find optimal response time. Public member functions, such as step(), impulse(), and initial(), call this private member function to adjust the default response time according to a specified deviation tolerance.

2.3 Member Functions

2.3.1 Constructor and Destructor. The LTI systems can be specified by linear time invariant equations in TF, ZPK, and SS models. The CControl class contains all information of above three types of models. The constructor of class CControl creates an empty object of this class. The constructor is a special member function of a class, which is invoked automatically each time an object of that class is instantiated. CControl class constructor initializes private data members of the class. The destructor is called when an object is destroyed. In Ch Control System Toolkit, the memory allocated for the class is released in the destructor.

2.3.2 Public Member Functions. The program below creates an LTI system model from its transfer function $G(s)=3/(s^2+2s+2)$, and plot the step response of the system.

```
#include <control.h>

int main() {
    double num[1]={3};
    double den[3]={1, 2, 2};
    class CPlot plot;
    class CControl sys;

    sys.model("tf," num, den);
    sys.step(&plot, NULL, NULL, NULL);
    return 0;
}
```

In this program, the first two lines inside function main() declare two arrays with double data type to store the coefficients of the numerator polynomial and denominator polynomial s^2+2s+2. The third line instantiates an object of CPlot class. CPlot is a Ch class for generating 2D/3D plots. The details of using this plotting class for customized plotting will be addressed in the subsequent section. The next line instantiated an object of CControl class with paramenters of transfer function. Once the system model is constructed, public member functions can be called to perform analysis and design of the control system. In this program, we use public member function CControl::step() to plot the step response of the system. The output of step response could be a separate plot or a data set stored in the computational arrays specified by the arguments of the member function CControl::step().

2.3.3 Private Member Functions. The private member functions serve as utility functions for the other member functions of the class. The private member functions cannot be called in user application programs. For example, the private member function CControl::m_adjustRespTime() defined in Ch Control System Toolkit is for adjusting the default response time to an optimal value. It is obvious that the specified default response time is not an optimal value for different systems. Ch Control System Toolkit uses this utility function to find the optimal response time period for different systems. The function removes the response values with a small deviation over time in order to increase time interval with a significant transient response for plotting. All the time response public member functions call this private member function to adjust the default response time.

2.3.4 Polymorphism. Ch supports polymorphism—the ability that the same function responses differently in different calls when the number of arguments and the argument data types are different. This feature provides flexibility and simplicity in the software implementation. Many member functions in CControl class are polymorphic member functions. For example, the mem-

ber function CControl::model() which will be described later is a polymorphic member function. The number of arguments and argument data type are different when it is used to create different type of models.

2.4 User Friendly Graphical Presentations. One of the salient features of Ch Control System Toolkit is its user-friendly graphical presentations. The toolkit can represent analysis and design results graphically using the plotting class. CPlot is a class for high-level creation and manipulation of two and three dimensional plotting. It allows output visually displayed or exported as external files with a variety of different file formats including postscript file, PNG, LaTeX, etc. In Ch Control System Toolkit, all member functions with plotting features internally invoke member functions of class CPlot to generate 2D/3D plots. To allow the users to customize a plot, an object of class CPlot needs to be instantiated. Once a plot object is instantiated, properties such as title and axis labels can be set by calling member functions of class CPlot. The address (a pointer) of this object needs to be passed as an argument to the member functions with plotting features of Ch Control System Toolkit.

If the user wants to create an image file for output rather than displaying it on the screen, the member function CPlot::outputType() can be used to specify the output format and filename.

3 Functionalities of Ch Control System Toolkit

3.1 System Model Construction. For analysis and design of a control system using Ch Control System Toolkit, the model of the system needs to be constructed first. As mentioned before, one important feature of Ch Control System Toolkit is using a single class to represent all LTI system models. The different type of system models can be created by a polymorphic member function CControl::model(). The first argument indicates the type of the model. The remaining arguments contain the information related to the model. For example, the statement

```
sys.model("tf," num, den);
```

constructs a continuous-time TF model for the system. The first argument "tf" of a string type indicates that the model to be created is a transfer function model. The remaining two arguments *num* and *den* are two one-dimensional computational arrays containing the coefficients of the polynomials which represent the numerator and denominator of the transfer function, respectively. Based on these information, the TF model can be created readily. Furthermore, the member function CControl::model() calculates the values of system matrices in state-space, zeros, poles, and gain of the system and initializes proper private data members of the class.

A zero-pole-gain model can be instantiated as below.

```
sys.model("zpk," z, p, k);
```

where the first argument "zpk" indicates that the model to be created is a zero-pole-gain model. The other arguments z, p and k contain the zeros, poles and gain of the system, respectively.

In Ch Control System Toolkit, both single-input/single-output (SISO) systems and multiple-input/multiple-output (MIMO) systems can be modeled in the state-space model. Typically, the state-space model of a system consists of two first-order differential vector equations in the form of

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u} \tag{1}$$

$$\mathbf{y} = \mathbf{C}\mathbf{x} + \mathbf{D}\mathbf{u} \tag{2}$$

where vectors **x**, **u**, and **y** are the state, input, and output vectors of the system, respectively. A state-space model can be created as follows,

```
sys.model("ss," A, B, C, D);
```

where the first argument "ss" indicates that a state-space model will be created. The subsequent four arguments are system matrix, input matrix, output matrix and direct transmission matrix.

A discrete-time system model in TF, ZPK, and SS type can be created by passing an extra argument representing a sample time to the member function CControl::model() as illustrated below

```
sys.model("tf," num, den, 0.5);
```

In this case, the member function CControl::model() instantiated a discrete-time system model with a sample time of 0.5 second.

3.2 Retriving System Information. Ch Control System Toolkit supports multiple model representations such as SS models, TF models, and ZPK models. The internal data structure is capable of simultaneously representing a given system in all these three forms. When a system model is created using one of three models, the Ch Control System Toolkit automatically calculates other two types of models of the system internally and keeps all information. The model information and system characteristics can be retrived by member functions. For example, the member function CControl::size() can obtain dimensions of output/input/system matrix for SS models, orders of numerator and denominator for transfer function models, or numbers of zeros and poles for ZPK models, according to different values of the argument passed in.

3.3 System Conversions and Model Interconnections. Ch Control System Toolkit provides the capability of converting from continuous to discrete, discrete to continuous, and discrete to discrete systems. It also provides the state-space transformations. A fairly complete set of interconnecting member functions is also available to perform model interconnection in series, parallel, and feedback. It also contains member functions for system reduction such as minimal realization of the system and pole-zero cancellation.

3.4 Design and Analysis in Time and Frequency Domains. Ch Control System Toolkit provides a group of member functions to compute the time responses of step input, impulse input, even arbitrary inputs, and initial conditions for both SISO and MIMO systems. The output responses of these functions can be displayed by either a plot or a data set passed through arguments. In the frequency domain, Ch Control System Toolkit can also generate commonly used frequency response in Bode, Nyquist and Nichols plots; calculate the bandwidth, DC gain, gain and phase margins of an SISO system for system dynamics and stability analysis.

3.5 Root-Locus Design. The locations of the roots of the characteristic equation, which are the closed-loop poles, in the s-plane will affect the transient-response features of the system. It also determines the system stability. The root locus is a technique which shows how changes in one of a system's parameters will change the pole positions and thus change the system's dynamic response. Although the parameter could be any parameter which enters the equation linearly, the root locus is most commonly used to study the effect of loop gain variations. Ch Control System Toolkit offers a member function rlocus() to calculate and plot the root locus of an SISO system when the loop gain k varies from 0 to ∞.

3.6 Design and Analysis in State-Space. State-space method is a modern control system design and analysis method. The controllability and observability are important structural properties of a control system. If there exists an input array $U(t)$ that will take the states of the system from any initial state X_0 to any desired final state X_f in a finite time interval, the system is controllable. Normally, the controllability matrix can be easily used to determine the controllability of the system. In Ch Control System Toolkit, the member function CControl::ctrb() can construct the controllability matrix of the system. The member function CControl::obsv() can be used to calculate the observability

matrix of the system. The controllability and observability gramians can be calculated by the member functions CControl::gram().

The positions of the closed-loop poles will affect the system's dynamic response. For a system presented by the SS model with the linear state feedback regulator control law defined as $\mathbf{u} = -\mathbf{kx}$, the closed loop poles are eigenvalues of matrix

$$\mathbf{A}_1 = \mathbf{A} - \mathbf{BK} \tag{3}$$

To make the system more stable and have the desired time response characteristics, the user can first select desired pole locations, and then find the gain vector **k** which moves these poles to the desired locations. This technique is known as pole placement. The member function CControl::acker() and CControl::place() can be used for pole placement. It is recommended that using member function CControl::acker() for SISO systems and CControl::place() for MIMO systems.

3.7 Optimal Control System Design and Equation Solvers

Ch Control System Toolkit provides two functions, CControl::lqr() and CControl::dlqr(), for designing linear-quadratic (LQ) state-feedback regulator for continuous-time and discrete-time plants, respectively. The member function CControl::lyap() can be used to solve continuous-time Lyapunov equations in both general and special forms.

4 Salient Features of Ch Control System Toolkit

C/C++ Compatible Ch Control System Toolkit is the first interactive CACSD environment developed in the framework of C/C++. We believe that it is the simplest possible solution for control system design and analysis in the sprit of C/C++. It can seamlessly interface existing C/C++ code in either source code or binary static/dynamical libraries without re-compilation. This allows users to take advantage of a large body of existing C/C++ programs and speeds up the design and implementation process of control systems. For example, graphical user interface in X11/Motif, Windows, GTK+ and 3D graphics in OpenGL, computer vision in OpenCV, data acquisition in NI-DAQ and motion control in NI-Motion are readily available for application development. Ch Control System Toolkit enables users to do modeling, analysis, design, and real-time implementation of control systems all in one language.

Object-Oriented Ch Control System Toolkit is an object-oriented control toolkit. Class CControl encapsulates the information of three common LTI models and the interface of Ch Control System Toolkit. Using a single class to represent different types of LTI models greatly simplifies the application programs and the maintenance of control toolkit. The functionarities of Ch Control System Toolkit can be easily extended by defining more public functions of CControl class.

Embeddable Taking the advantage of embeddable feature of Ch, Ch Control System Toolkit can be embedded in a C/C++ application program within the same process to perform robust control system design and analysis.

Portable Ch Control System Toolkit is platform independent. A program, developed using Ch Control System Toolkit in one platform, can be executed in other platforms without any modification.

5 Application Examples

Ch Control System Toolkit provides a set of member functions for control system analysis and design. Two examples in this section illustrate how to use Ch Control System Toolkit to design and analyze control systems both in time and frequency domains.

Example 1. Consider the system expressed in state–space Eqs. (1) and (2) with the values of the matrices given below.

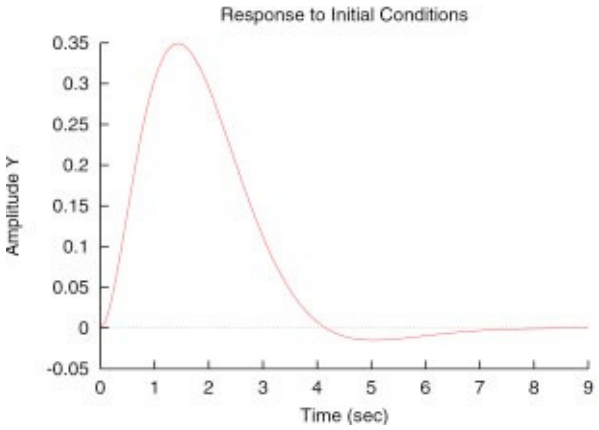

Fig. 1 Initial response of the system.

$$\mathbf{A} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ -0.5 & 0 & 0 & 0 \end{bmatrix},\ \mathbf{B} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ -1 \end{bmatrix},\ \mathbf{C} = \begin{bmatrix} 0 & 0 & 1 & 0 \end{bmatrix},\ \mathbf{D} = 0$$

1. Find the feedback gain K that places the closed-loop poles at $s = -1, -1, -1+j, -1-j$.
2. Find the response of the closed-loop system to an initial condition of $x_1 = 0.175$.

The program below can be used to solve this problem. At the beginning of the program, we declare four computational arrays A, B, C, and D with double data type to store the values of open-loop system matrices. The array k is used to store the feedback gains of the control law. Array p is the desired poles of the closed-loop system. Array $A1$ is the system matrix $\mathbf{A}_1$ of the closed-loop system in Eq. (3). Array $x0$ is the initial values of the state variables. After array declarations, the object of CPlot class is instantiated for initial response. The line

```
sys0.model("ss," A, B, C, D);
```

creates a system model based on the open-loop system matrices. After calling the acker() member function, a feedback gain vector is found and it can be used to calculate the closed-loop system matrix $A1$. Then, the closed-loop system model is created by matrices $A1$, B, C, and D. Once the closed-loop system is instantiated, the initial response of the closed-loop system can be plotted by the member functions initial() shown in Fig. 1. Since only the plot is needed, the arguments $yout$, $tout$, and $xout$ in the member function initial() for the output of initial response, time vector, and state trajectories, respectively, are set to $NULL$.

```
#include ⟨control.h⟩

#define NUMX 4          // order of space-state
#define NUMU 1          // number of input
#define NUMY 1          // number of output

int main() {

    array double A[NUMX][NUMX]= { 0, 1, 0, 0,
                                  1, 0, 0, 0,
                                  0, 0, 0, 1,
                                 -0.5, 0, 0, 0};
    array double B[NUMX][NUMU]={0, 1, 0, -1};
    array double C[NUMY][NUMX]={0, 0, 1, 0};
    array double D[1][1]={0};
    array double K[1][NUMX];
```

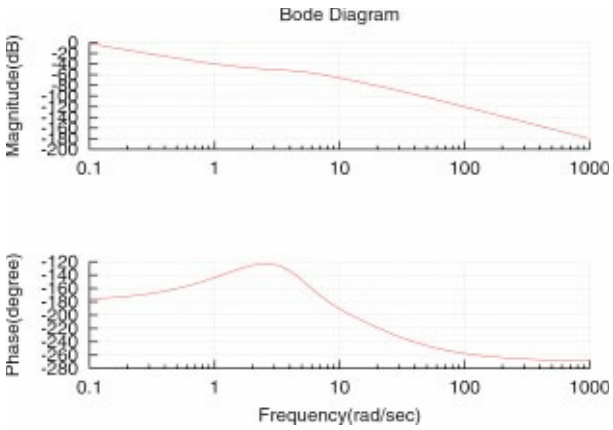

Fig. 2 Bode plot of the system.

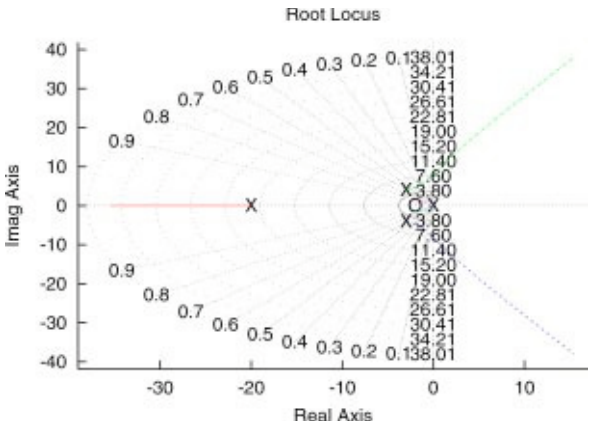

Fig. 3 Root locus of the system.

```
        array double complex p[NUMX]
                ={−1, -1, complex(−1,1), complex(−1,−1)};
        array double A1[NUMX][NUMX];
        array double x0[NUMX]={0.175, 0, 0, 0};
        class CPlot plotinitial;
        class CControl sys0;
        class CControl sys;
        sys0.model("ss," A, B, C, D);
        sys0.acker(K, p);
        printf(''K=%f\n'', K);
        A1=A−B*K;
        sys.model("ss", A1, B, C, D);
        sys.initial(&plotinitial, NULL, NULL, NULL, x0);
        return 0;
}
```

The feedback gain K as output of the above program is
K=12.000000 16.000000 4.000000 12.000000

Example 2. For the system with following open-loop transfer function $(s+2)^2/s^2(s+20)(s^2+6s+25)$

1. Sketch the Bode plot magnitude and phase.
2. Sketch the root locus of the system.

The system is given by a transfer function. The numerator of transfer function is the multiplication of two first order polynomials. We declare two arrays to contain the coefficients of these polynomials. The coefficients of the numerator are stored in array *num*. Similarly, the denominator of transfer function consists of two polynomials of s^3+20s^2 and $s^2+6s+25$. The two arrays of *den*1 and *den*2 are declared to store the coefficients of these two polynomials. The array *den* is used to store the coefficients of the denominator polynomial of the transfer function. The two objects *plotbode* and *plotrlocus* of CPlot class are used for Bode and root locus plots. Function call

```
conv(c, x, y);
```

calculates the convolution *c* of two arrays *x* and *y*. If *x* and *y* are two vectors of polynomial coefficients, the convolution of *x* and *y* is equivalent to the multiplication of these two polynomials. The next line

```
sys.model("tf", num, den);
```

constructs a transfer function model of the system. The member function CControl::grid() adds or removes grid lines for regular plots or Nyquist plot. It has only one argument as shown below.

```
int grid(int flag);
```

If *flag* is set to 1, the grid lines are turned on. Otherwise, the grid lines are turned off. The member function CControl::sgrid() is similar to CControl::grid(). It adds and removes grid lines in s-plane. The syntax of member function CControl::bode() for a Bode plot is shown below.

```
sys.bode(plot, mag, phase, wout, wmin, wmax);
```

The first argument *plot* is an object of Ch **CPlot** class. Arrays *mag* and *phase* contain the returned magnitude (in decibels) and phase (in degrees) of the frequency response corresponding to the frequencies in *wout* (in rad/sec). The last two arguments, *wmin* and *wmax*, explicitly specify the frequency range to be used for the plot. In this example, the arguments *wmin* and *wmax* are absent. As a result, the default frequency range is used for the Bode plot. Since only Bode plot is required, the arguments *mag*, *phase*, and *wout* are set to *NULL*. The syntax of member function CControl::rlocus() is as follows.

```
sys.rlocus(plot, r, kout);
```

The first argument is the same as the member function CControl::bode(). The argument *kout* is array of reference containing selected gains by users or by default. The argument *r* is array of reference containing the complex roots corresponding to gains in argument *kout*. The program for this example is shown below with output in Figs. 2 and 3.

```
#include ⟨control.h⟩

#define M1 2
#define M2 2
#define N1 4
#define N2 3

int main() {
    double num1[M1]={1, 2};
    double num2[M2]={1, 2};
    double den1[N1]={1, 20, 0, 0};//s ^2(s+20)=s ^3+20s ^2
    double den2[N2]={1, 6, 25};
    double num[M1+M2−1];
    double den[N1+N2−1];
    class CPlot plotbode, plotrlocus;
    class CControl sys;

    conv(num, num1, num2);
    conv(den, den1, den2);
    sys.model("tf", num, den);
    sys.grid(1);
    sys.sgrid(1);
    sys.bode(&plotbode, NULL, NULL, NULL);
    sys.rlocus(&plotrlocus, NULL, NULL);
    return 0;
}
```

6 Conclusions

An object-based interactive control system toolkit has been developed in Ch—a C/C++ interpreter. The Ch Control System Toolkit supports most classical and modern control techniques and provides a fairly complete set of member functions for control system design and analysis. It is able to represent three commonly used LTI models of TF, ZPK, and SS in a single object of CControl class. The member functions of Ch Control System Toolkit allow users to interconnect models, perform system transformations and conversions, time/frequency domain analysis and design, root-locus design, state-space analysis and design, and optimal control system design. The object-based design method makes Ch Control System Toolkit easy to extend and maintain. A Ch program using Ch Control System Toolkit can integrate seamlessly with existing C/C++ programs or embedded in other C/C++ application programs to speed up the design and analysis of control systems. The software is available for downloading on the web [14].

References

[1] The MathWorks, Inc., 1998, *Control System Toolbox User's Guide*, The MathWorks, Inc., Natick, MA.

[2] Integrated Systems, Inc., 1990, *MATRIXx Control Design*, Integrated Systems, Inc., Santa Clara, CA.

[3] Wolfram Research, Inc., 2003, "Mathematica's Control System Professional," http://www.wolfram.com/products/applications/control.

[4] Hanselman, D., and Littlefield, B., 2001, *Mastering Matlab 6—A Comprehensive Tutorial and Reference*, Prentice Hall, Upper Sadle River, New Jersey.

[5] van den Boom, A., and Van Huffel, S., 1996, "Developments Around the Freeware Standard Control Library Slicot," *Proc. the 1996 IEEE International Symposium on Computer-Aided Control System Design*, IEEE, New York, NY, pp. 473–476.

[6] Delebecque, F., 2000, "A Slicot Based Control Library for Scilab," *Proc. the 2000 IEEE International Symposium on Computer-Aided Control System Design*, IEEE, Piscataway, NJ, pp. 147–149.

[7] Hodel, A. S., Tenison, R. B., Clem, D. A., and Ingram, J. E., 1996, "The Octave Control Systems Toolbox: A MATLAB-like CACSD Environment," *Proc. the 1996 IEEE International Symposium on Computer-Aided Control System Design*, IEEE, New York, NY, pp. 386–391.

[8] Rlab, 2001, http://rlab.sourceforge.net.

[9] Cheng, H. H., 1993, "Scientific Computing in the Ch Programming Language," Scientific Programming, **2**(3), pp. 49–75.

[10] Cheng, H. H., 2003, "Ch Language Environment User's Guide," SoftIntegration, Inc., http://www.softintegration.com.

[11] SoftIntegration, Inc., 2003, "Ch Control System Toolkit User's Guide," http://www.softintegration.com/products/toolkit/control.

[12] Patel, R. V., Laub, A. J., and van Dooren, P. M., 1994, *Numerical Linear Algebra Techniques for Systems and Control*, IEEE Press, New York.

[13] Chen, C. T., 1999, *Linear System Theory and Design*, Oxford University Press, New York.

[14] SoftIntegration, Inc., 2003, "Web-Based Control Design and Analysis System," http://www.softintegration.com/webservices/control.

[15] Bell, W. E., Lamont, G. B., and Trevino, F. L., 1994, "Using Object-Orientation in Developing an Extendible CACSD Package," *Proc. the 1994 IEEE/IFAC Joint Symposium on Computer-Aided Control System Design*, IEEE, New York, NY, pp. 255–260.

[16] Barker, H. A., 1994, "Open Environment and Object-Oriented Methods: The Way Forward in Computer-Aided Control System Design," *Proc. the 1994 IEEE/IFAC Joint Symposium on Computer-Aided Control System Design*, IEEE, New York, NY, pp. 3–12.