

021 517

## Final Report

Octave: a MARSYAS Post-Processor for Computer-Aided Control System Design  
NASA Contract NASA-NAG8-1040  
Auburn contract 4-20616

A. S. Hodel

Mar. 1997

The Octave Control Systems Toolbox is essentially completed. This year the package was almost entirely re-written in order to take advantage of Octave's data structure capabilities. This re-write greatly simplifies function calls to the toolbox package, and makes much more practical and convenient the interface between Octave and Marsyas.

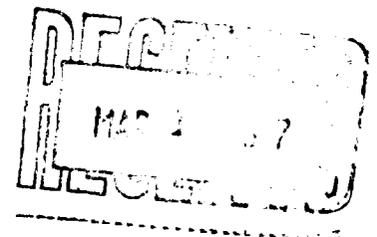
The use of existing and new Octave functions is illustrated in the m-file DEMOcontrol Functions include "block diagram" manipulations (series, parallel, and closed-loop connections of systems), frequency response calculations and plots, state-space analysis, and root locus calculations. All functions are now required to use the revised "packed" system data representation, which automatically includes the signal names of internal states, inputs, and outputs, and makes transparent to the user the representation form of the system being used.

Some mild details listed in the statement of work remain in progress, but will be completed soon. Octave and the Octave Control Systems Toolbox are now installed on a Sun compute server and a Linux PC at MSFC. Revisions of the OCST are available by anonymous ftp at [ftp://ftp.eng.auburn.edu/pub/hodel/OCST-ss\\*](ftp://ftp.eng.auburn.edu/pub/hodel/OCST-ss*). Marsyas/Octave interface functions are in the same directory with file names maroct-ss\*.

A more complete description of the OCST is in the attached paper, [HTCI96].

## References

- [HTCI96] A. S. Hodel, R. B. Tenison, D. A. Clem, and J. E. Ingram. The octave control systems toolbox: A matlab<sup>tm</sup>-like CACSD environment. In *Proceedings of the 1996 IEEE International Symposium on Computer-Aided Control System Design*, pages 386-391, Dearborn, Mich., Sept. 15-18 1996. IEEE Control Systems Society.



# The Octave Control Systems Toolbox: A MATLAB<sup>tm</sup>-like CACSD Environment<sup>1</sup>

A. S. Hodel<sup>2</sup>

R. B. Tenison<sup>3</sup>

D. A. Clem<sup>4</sup>

J. E. Ingram<sup>5</sup>

## Abstract

Octave is a freely distributable MATLAB<sup>tm</sup>-like programming environment. The Octave Control Systems Toolbox (OCST) was developed as a set of script files to add functionality to the Octave environment. The OCST was designed to use an object-oriented data structure to represent linear system parameters and relevant companion data in a single data structure.

## 1. Introduction

The techniques and computational requirements for the analysis and design of automatic control systems has grown increasingly complex with theoretical and technological advancements of the past few decades. Control systems conferences appropriately have entire sessions dedicated to the to numerical issues involved in the computational aspects of control systems engineering. Several commercial packages (e.g., the MathWorks MATLAB<sup>tm</sup>, Boeing's Easy-5<sup>tm</sup>, etc.) are currently available for the purpose of computer-aided control systems design (CACSD). In this paper we describe a CACSD package design for use with Octave, a freely distributable MATLAB<sup>tm</sup>-like programming environment licensed under the terms of the Free Software Foundation's copyright policy. Octave was originally developed for use in control education by Dr. John Eaton, currently at the University of Wisconsin-Madison. The Octave Control Systems Toolbox (OCST) was developed by the authors.

## 2. Software organization

The OCST is divided into 6 sections, each section with its own demonstration program:

<sup>1</sup>This work was supported in part by the NASA Marshall Space Flight Center Summer Faculty Fellowship Program and in part by NASA Contract NASA-NAG8-1040

<sup>2</sup>a.s.hodel@eng.auburn.edu, Dept. Elect. Eng, Auburn University, AL, 36849

<sup>3</sup>Reid State Technical College, Evergreen AL

<sup>4</sup>Dept. Elect. Eng. Auburn University

<sup>5</sup>Dept. Elect. Eng. Auburn University

1. State-space analysis (demo program `analdemo`). Functions include numerical solution of the Lyapunov equation, numerical solution of the algebraic Riccati equation, and the computation of Hankel singular values for model reduction.
2. Block diagram manipulation (demo program: `bddemo`). Functions included allow manual connection and modification of linear, time-invariant systems through parallel/series combinations, closed loop connections, "pruning" of inputs and outputs, etc.
3. LQG,  $\mathcal{H}_2$  and  $\mathcal{H}_\infty$ -optimal controller design: (demo program: `dgkfdemo`). Currently implemented functions include standard LQG design, system  $\mathcal{H}_2$  and  $\mathcal{H}_\infty$ -norm computation, and  $\mathcal{H}_2$  and  $\mathcal{H}_\infty$ -optimal controller design as specified in [1]. Checks for controllability and observability make use of Krylov subspace iterations with complete re-orthogonalization [2]. Structured singular value (" $\mu$ ") analysis and design is planned, but not yet implemented.
4. Frequency response functions (demo program: `frdemo`). Implemented functions include Nyquist and Bode plots in both continuous and discrete-time systems. Automated frequency range and step size is provided to provide useful and smooth plot curves. Speed in SISO plot computation is provided by conversion to zero-pole format (see discussion of system representation below).
5. Root locus functions (demo program: `rldemo`). Pole-zero maps and standard root-loci are presented in graphical form. The latest release of the OCST includes automated gain range and step size selection.
6. System representation and conversion functions (demo program: `sysrepdemo`). The initial release of the OCST supported the same system representation formats as MATLAB<sup>tm</sup>: `tf` transfer function form (ratios of polynomials), `zp` zero-pole form (ratios of factored polynomials with a leading gain coefficient), `ss` state-space form, and a "packed" system form similar to that used in

the MATLAB<sup>tm</sup>  $\mu$ -toolbox [3]. The current release utilizes the Octave structured variables feature so that all three representations `tf`, `zp`, and `ss` are contained in a single data structure variable. (This is discussed further in §3.

Care is taken in the conversion from one system representation to another; see, e.g., [4], [5]. Artificial transfer function zeros created by numerical perturbation of zeros at infinity is avoided the use of the Emami-Naeini/Van Dooren `mvzero` algorithm [6].

Documentation is available for all functions in three forms: in “help” commands on-line in Octave, in  $\TeX$  form, and in the Free Software Foundation’s `info` format, providing further on-line documentation.

The remainder of this paper is organized as follows. We first present in §3 our structured variable representation of systems and describe therein related functions and issues for the use of this useful data structure. Following this, in §4, we discuss some of the relevant numerical features of the implementations, especially where they differ significantly from MATLAB<sup>tm</sup> behavior.

### 3. Packed system representation

The OCST packed system representation is a data structure, similar in use to a C language `struct`. The packed system representation was developed in response to the following observations:

1. Due to the wide variety of system representations, a substantial portion of the code is devoted to determining which of four representation formats (`tf`, `zp`, `ss`, or packed system form) is being passed to a given OCST function. Such evaluations are not always reliable, and they leave room for undetected user error when calling such functions
2. It is desirable to assign meaningful *names* to system states, inputs, and outputs. In turn, it is desirable to make use of these names when designing observers, compensators, etc. This is especially critical when Octave is being used as a CACSD tool, generating a code description of a controller for use in closed-loop with a system being simulated and/or analyzed, possibly by another software package.

The OCST packed system representation is a flexible data structure that allows users to pass a single variable for each dynamic system passed to OCST functions, regardless of the selected internal representation

and replaces the packed system format used in an earlier release. This is in contrast to `tf`, `zp`, and `ss` forms which each require 2, 3, and 4 variables to be passed, respectively, for continuous time systems, and an additional variable (the sampling interval) for discrete-time systems. The packed system format allows an object-oriented interface to all software, so that users need not concern themselves with the internal representation of a system once they have put it into a structured form. The internal representation is capable of simultaneously representing a given system in all three forms above, if required to do so; typically only one is used.

#### 3.1. Internal structure

The structured format representation uses the following internal variables:

- a, b, c, d** State-space representation matrices
- num, den** Transfer-function representation vectors (SISO systems only)
- k, zer, pol** Zero-pole representation variables (SISO systems only)
- stname, inname, outname** String matrices containing names of states, inputs, and outputs, respectively.
- sys** an integer vector of four variables, indicating which of the three representations above was used to initialize this system representation, and which of the remaining representations are “up to date.” The vector format is

$$\begin{bmatrix} \text{primary type} \\ \text{tf up to date} \\ \text{zp up to date} \\ \text{ss up to date} \end{bmatrix}$$

The primary system type is a variable with value 0-2, representing (in order) a primary system type of `tf`, `zp`, or `ss`. The up to date variables are true/false flags, with zero indicating that a given form is not up to date (not consistent with the primary system type), and non-zero indicating that the given form is up to date. For example, `sys=[2 1 0 1]` indicates that a system was initialized with a state-space format (using function `ss2sys`) and that it also has a transfer function format representation available, but not a zero-pole format representation.

- n, nz, tsam** Number of continuous time (differential equation) states, number of discrete-time (difference equation) states, and discrete-time sampling interval, respectively.

If a system has either `tf` or `zp` representation internally, then exactly one of `n` and `nz` is non-zero.

State space systems may have both continuous and discrete states; it is assumed that the first  $n$  states are continuous and the last  $n_z$  states are discrete. This convention is respected by the block diagram manipulation functions.

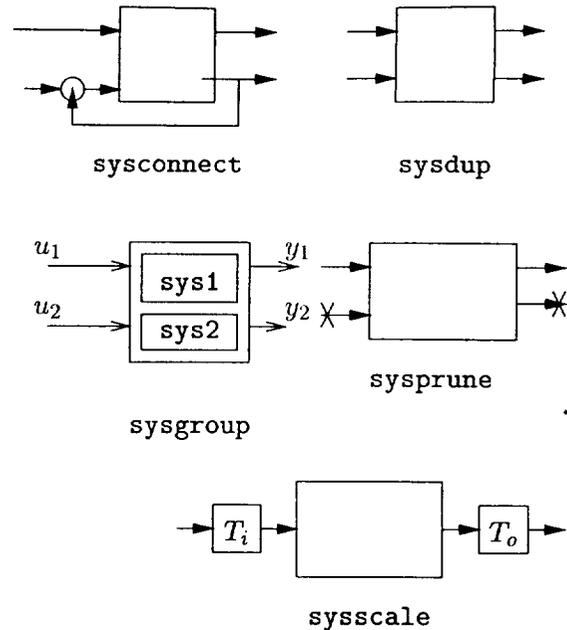
**Example 1** In order to represent a double-integrator  $1/s^2$  plant in state-space form as a packed system `sys`, the following OCST commands may be used:

```
sys.a = [ 0 1 ; 0 0];
sys.b = [0;1];
sys.c = [1 0];
sys.d = 0;
sys.n = 2;
sys.nz = 0;
sys.sys = [2 0 0 1];
sys.stname = ["pos";"vel"];
sys.inname = ["f"];
sys.outname = ["pos"];
```

C programmers will recognize the notation for struct variables; other than this the notation is identical to MATLAB<sup>™</sup>.

**Remark 1** The OCST uses packed system form in an object-oriented fashion; that is, the user should access structure members only using the provided structure manipulation functions e.g., `ss2sys` and `sys2ss` to convert from state space to packed system and vice versa. The use of a single structure to represent all linear time-invariant systems greatly simplifies the writing and maintenance of OCST functions, since it is no longer necessary to identify which representation type is being passed, only to reference the structure variable `sys` and update the appropriate internal variables via the OCST `sysupdate` function. Further, since input, output, and state names are stored internally in the structure, it is straightforward for users to write automatic code generation software (done by the authors for Marshall Space Flight Center) whose output is much more easily integrated into the target package (an internal simulation package, in our case). Should the names of inputs, outputs, and states be required, the OCST command `sysout` may be used to obtain output of the form:

```
2 continuous states:
    1: (BALL\DYNAMICS) X'
    2: (BALL\DYNAMICS) X
0 discrete states:
    None
1 inputs:
    1: I\MAG
```



**Figure 1:** Block diagram manipulations with packed system format. Function names are shown below the diagram indicating the corresponding operation.

2 outputs:

- 1: X
- 2: XDOT

### 3.2. Block diagram manipulations

The packed system format provides a convenient vehicle for performing block diagram manipulations. The available manipulations and their corresponding function names are depicted in Figure 1, and are as follows.

**sysconnect** connect selected system outputs to selected system inputs. This option is used to “close the loop” around a system/controller pair.

**sysdup** duplicate selected system inputs/outputs. This option is used when, for example, an SISO control input requires a difference  $r(t) - y(t)$ , where  $r(t)$  is a reference signal and  $y(t)$  is the output of a plant. (See the example below.)

**sysgroup** combine two separate systems into a single system

**sysprune** prune selected inputs/outputs. `sysprune`'s input parameters list the inputs and outputs that are to be *kept*, not deleted.

**sysyscale** scale inputs and outputs by specified constant transformation matrices  $T_i$  and  $T_o$ .

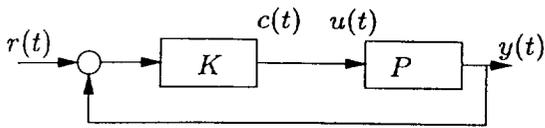
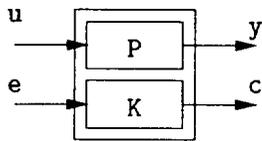


Figure 2: Example closed loop system

**Example 2** Suppose that an SISO plant and controller are stored in respective packed structures P and K, and that it is desired to obtain a closed loop transfer function of the closed loop system shown in Figure 2. For clarity below, we denote the input to the plant P as u, the output of the plant as y, the input of the controller K as e, and the output of the controller as c, as shown in the figure. Thus, we wish to combine together P and K, and to set  $u = c$ , and  $e = r - y$ . The OCST commands to obtain this system are as follows:

```
PK = sysgroup(P,K)
```

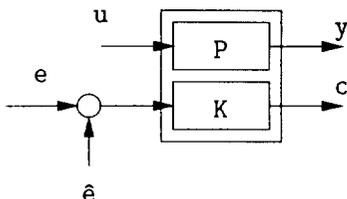
Group the plant and the controller into a single system with inputs  $[u \ e]'$  and outputs  $[y \ c]'$ .



Since the new system is MIMO, `sysgroup` automatically changes the internal representation of the system to a state-space form; this conversion is transparent to the user.

```
PK = sysdup(PK, [], 2)
```

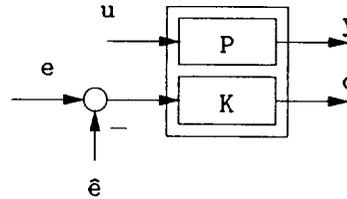
Since the input to the controller is a combination of both a reference signal  $r$  and the output  $y$ , we duplicate the input  $e$  so that now the overall system has inputs  $[u \ e \ \hat{e}]'$  where the new input  $\hat{e}$  enters the system in the same way that  $e$  does; that is, the input to the controller K is now  $e + \hat{e}$ .



We now proceed to set  $\hat{e} = -y$  and  $u = c$ .

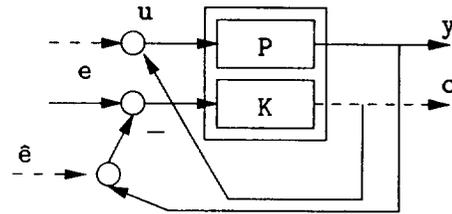
```
PK=sysyscale(PK, [], diag(1,1,-1))
sysyscale adds the scaling matrices (when
```

present) as shown in Figure 1; the system is modified to reflect that the input vector is multiplied by the input matrix  $T_i = \text{diag}(1,1,-1)$  before it is passed to the original system. Thus,  $T_i$  is an input scaling matrix that changes the sign of the input  $\hat{e}$  to  $KP$ , and so the input to the controller K is now  $e - \hat{e}$ .



```
PK = sysconnect(PK, [1 2], [3 1])
```

This command connects output 1 ( $y$ )  $\rightarrow$  3 ( $\hat{e}$ ) and output 2 ( $c$ )  $\rightarrow$  input 1 ( $u$ ). (If the user is unsure of the order in which the inputs and outputs are listed, they may use the command `sysout` to read the contents of the input, output and state names.



(Dashed lines above are removed below with `sysprune`.)

```
PK = sysprune(PK, 1, 2)
```

This command omits all outputs except for output 1 ( $y$ ) and input 2 ( $e$ ), which may easily be seen from the above to correspond to the reference input  $r$ .

**Remark 2** The two steps

```
PK = sysdup(PK, [], 2)
```

```
PK=sysyscale(PK, [], diag(1,1,-1))
```

could have been executed with a single `sysyscale` command,

```
PK=sysyscale(PK, [], [1 0 0; 0 1 -1])
```

However, this requires `sysyscale` to attempt to identify a meaningful input name for newly created inputs. If there is ambiguity, the new names are merely selected as  $u_i$  with  $i$  set to the index number of the input. A similar action is taken for outputs created via `sysyscale`.

**Remark 3** The current implementation of `sysconnect` requires that no inputs or outputs be listed twice in the above vector lists; `sysdup` must be used in order to obtain such connections. This restriction will be removed in a later release of the OCST.

**Remark 4** Notice (see Figure 1) that `sysconnect` does not change the number of inputs and outputs; in effect, `sysconnect` merely inserts a summing junction at the selected input(s) and connects the corresponding output(s) to the summing junction. As such, the above `sysconnect` command effectively reassigns  $\hat{e} := \hat{e} + y$  so that the input to the controller is now  $e - (\hat{e} + y)$ .

**Remark 5** `sysconnect` is able to correct connect inputs and outputs in the presence of algebraic loops. If there is a direct feed through ( $D$ -matrix) term from any of the selected inputs to any of the selected outputs, `sysconnect` correctly closes the loop around these direct feed-through terms and only provides a warning message to the user that there is a possible algebraic loop. An error occurs only when (1) an algebraic loop does in fact exist and (2) the corresponding block  $\hat{D}$  of the system  $D$  matrix has an eigenvalue at 1 ( $I - D$  is singular).

#### 4. Numerical Features

The numerical solution of control problems requires care in the development of solution software. Whenever possible, it is desirable to employ orthogonal matrices in order to reduce the impact of numerical roundoff [7]. The OCST solution methods for Lyapunov equations  $AX + XA' + B = 0$  and algebraic Riccati equations  $A'X + XA - XBX + C = 0$  both employ the Schur decomposition as implemented in LAPACK [8]; the reader may consult [9] and [10] for details.

The design of  $\mathcal{H}_2$  and  $\mathcal{H}_\infty$  optimal controllers requires several checks on system properties, including stabilizability and detectability. These tests are performed in the OCST as follows:

1. construct an orthogonal basis  $V_1$  of the controllable/observable subspaces in an Arnoldi iteration with complete re-orthogonalization [2]
2. construct an orthogonal basis  $V_2$  of the span of the right/left eigenvectors corresponding to eigenvalues of system that are not exponentially stable.
3. The system is stabilizable/detectable if and only if  $V_1'V_2$  has full column rank.

These conditions (among others) are checked with a single OCST function `is_dgkf`, named for the now famous paper [1].

#### 5. Example program

An example of the use of the OCST to design an  $\mathcal{H}_\infty$ -optimal controller is as follows (taken from `dgkfdemo`):

```

help hinfsyn          % get on line description
                    % of command

A = [0 1; 0 0];
B1 = [0 0; 1 0]; B2 = [0; 1];
C1 = [1 0; 0 0]; C2 = [1 0];
D11 = zeros(2); D12 = [0; 1];
D21 = [0 1]; D22 = 0;
D = [D11 D12; D21 D22];

%default state,input,output names
%are selected by ss2sys
Asys = ss2sys(A,[B1 B2], [C1;C2] , D);

% open loop bode plot for MIMO system
bode(Asys);
gmax = 1000;
gmin = 0.1;
gtol = 0.01;

% design controller
[K,gain] = hinfsyn(Asys,1,1,gmax,gmin,gtol);

% close the loop and compute the Hinf norm
K_loop = sysgroup(Asys,K);

% print out system input/output names to
%show connection numbers (listed below)
sysout(K_loop);

Kc1 = sysconnect(K_loop,[3,4],[4,3]);

% delete "u" and "y" inputs
Kc1 = sysprune(Kc1,[1,2],[1,2]);
gain_Kc1 = hinfnorm(Kc1);

% check actual gain vs predicted
gain_err = gain_Kc1 - gain;

%Check: multivariable bode plot
bode(Kc1);

Notice that, due to the flexibility of our system format, there is no need to pass extra parameters to indicate whether a system is continuous or discrete time, etc. Also notice that, by using function sysout, a user can easily check that he/she has the correct indices of the inputs/outputs are passed to sysconnect.
```