Scilab Control System Toolbox

Samiksha S
BSc Data Science
Mount Carmel College,
Bengaluru, Karnataka

GUIDE :
Prof. Prabhu Ramachandran, IIT BOMBAY

MENTOR :
Rashmi Patankar

INTRODUCTION

Overview

This project involves translating control-system functions from GNU Octave into Scilab, producing faithful, working equivalents.

The goal is a Scilab library that mirrors Octave's LTI Syncope and control toolbox behavior. It should be covering state-space and transfer-function models, and control (Riccati-based).

Each function is ported close to line-by-line, with dependencies reconstructed where Scilab lacks a direct equivalent, and validated against Octave's numerical output.

INTRODUCTION

Objective

Main Goal

- Convert Octave control functions to Scilab with near 1:1 translation of the code and structure.
- Reconstruct missing dependencies (SLICOT-style solvers, helper routines) using Scilab's native machinery.

Supporting Objectives

- Verify numerical equivalence between the Octave source and the Scilab port through systematic test cases.
- Build a coherent, interoperable set of functions where ported pieces correctly depend on one another.

INTRODUCTION

What I Learned

Translating between Octave and Scilab required understanding where the two diverge structurally:

- argn(2) for argument counts instead of nargin
- select/case for switch
- explicit for loops in place of cellfun

Octave relies on compiled SLICOT routines (e.g.SB02OD, SG02AD) that don't exist in Scilab. I learned to reconstruct these numerically

For example, computing Riccati solutions via the ordered Schur decomposition of an extended symplectic pencil.

It was quite hard to be able to do this as it led to a FORTRAN routine.
This meant understanding the mathematics behind the function instead of directly finding equivalents in scilab.

It was quite satisfying to find the SLICOT routines that i have implemented in Scilab to be giving the exact outputs as octave.

FUNCTIONS

- DARE
- CARE
- DLQR
- DLQE
- LQR
- LQE
- CTRBF
- OBSVF
- DSS
- VERTCAT @lti
- HORZCAT @lti
- MINUS @lti
- PLUS @lti
- WESTLANDLYNX
- FILTDATA

Riccati Equation

The Riccati equation is a matrix equation that appears in optimal control and estimation problems. It is fundamental to the design of controllers such as the Linear Quadratic Regulator (LQR) and observers such as the Kalman Filter.

For continuous-time systems, the Algebraic Riccati Equation (CARE) is

$$A^\top P + PA - PBR^{-1}B^\top P + Q = 0$$

For discrete-time systems, the Discrete Algebraic Riccati Equation (DARE) is

$$P = A^\top PA - (A^\top PB)(R + B^\top PB)^{-1}(B^\top PA) + Q.$$

- Computes the optimal state-feedback gain.
- Minimizes a quadratic cost function balancing state error and control effort.
- Ensures closed-loop stability for controllable systems.

Applications

- Kalman Filtering & Sensor Fusion : The Riccati equation is solved repeatedly to optimally combine noisy measurements from sensors such as GPS, IMUs, LiDAR, and cameras.
- Robotics : Controls robotic arms, humanoid robots, and drones for smooth and energy-efficient motion.

DARE

Solves the discrete-time algebraic Riccati equation, returning the riccati solution, closed-loop poles, and gain. Ported using a custom __sl_sb02od__ and __sl_sg02ad__ solver built on the ordered Schur decomposition of an extended symplectic pencil.

Scilab

```
"T1 X:"
 1.1327822
"T1 L:"
 0.2344356
"T1 G:"
 0.2655644
"T2 X:"
 4.6841184   0.5784171
 0.5784171   1.4632331
"T2 L:"
 0.8623307 + 0.i
 0.3389636 + 0.i
"T3 X:"
 18.327737   10.87891
 10.87891    18.35069
"T3 G:"
 0.9192097   1.6847034
```

Octave

```
T1 X:
1.1328
T1 L:
0.2344
T1 G:
0.2656
T2 X:
   4.6841   0.5784
   0.5784   1.4632
T2 L:
   0.8623
   0.3390
T3 X:
   18.328   10.879
   10.879   18.351
T3 G:
   0.9192   1.6847
```

Input:
A, B, Q, R[, S[, E]]
Output:
X (Riccati solution),
L (closed-loop poles),
G (optimal state-feedback gain)

Dependencies:
__sl_sb02od__, __sl_sg02ad__

Parameters:

- ***A – State matrix***
- ***B – Input matrix***
- ***Q – State weighting matrix***
- ***R – Input weighting matrix***
- ***S – Cross-weighting matrix (optional)***
- ***E – Descriptor matrix (optional)***

CARE

Solves the continuous-time algebraic Riccati equation. Shares the same __sl_sb02od__/__sl_sg02ad__ Schur-based helpers as dare, differing only in the pencil form and gain formula.
Similarly returns riccati solution, closed-loop poles, and gain.

Scilab

```
"test case 1:"
"X1:"
 1.236068   0.236068
 0.236068   0.236068
"L1:"
-1.        + 0.i
-2.236068  + 0.i
"G1:"
 0.236068   0.236068
"test case 2:"
"X2:"
 11.385165   17.641941
 17.641941   29.958791
"L2:"
-5.3851648 + 0.i
-1.        + 0.i
"G2:"
 11.385165   17.641941
```

Octave

```
test case 1:
X1:
   1.2361   0.2361
   0.2361   0.2361
L1:
  -2.2361
  -1.0000
G1:
   0.2361   0.2361
test case 2:
X2:
   11.385   17.642
   17.642   29.959
L2:
  -1.0000
  -5.3852
G2:
   11.385   17.642
```

Input:
A, B, Q, R[, S[, E]]
Output:
X (Riccati solution)
L (closed-loop poles)
G (optimal state-feedback gain)

Dependencies:
__sl_sb02od__, __sl_sg02ad__

Parameters:

- *A – State matrix*
- *B – Input matrix*
- *Q – State weighting matrix*
- *R – Input weighting matrix*
- *S – Cross-weighting matrix (optional)*
- *E – Descriptor matrix (optional)*

DLQR

Designs the discrete-time linear-quadratic regulator, returning the optimal state-feedback gain.
Depends on dare and care according to if the embedded system is continuous or not, dssdata, and issample.

Scilab

```
"Test Case 1 - Gain G:"
-1.7991399   2.2473141
"Test Case 1 - Riccati X:"
 4.5982798  -4.4946282
-4.4946282   8.9571787
"Test Case 2 - Gain G:"
 0.   0.
"Test Case 2 - Riccati X:"
 1.   0.
 0.   2.
"Test Case 3 - Gain G:"
-1.2739545  -1.5579937
"Test Case 3 - Riccati X:"
 7.3785041   10.120031
 10.120031   21.461065
"Test Case 4 - Gain G:"
-0.9705663  -4.7034262  -5.0759851
"Test Case 4 - Riccati X:"
 1.9705663   4.7034262   5.0759851
 4.7034262   24.982293   25.393066
 5.0759851   25.393066   32.974634
```

Octave

```
Test Case 1 - Gain G:
  -1.7991   2.2473
Test Case 1 - Riccati X:
   4.5983  -4.4946
  -4.4946   8.9572
Test Case 2 - Gain G:
            0   8.2809e-35
Test Case 2 - Riccati X:
   1.0000e+00   1.7390e-34
   1.7390e-34   2.0000e+00
Test Case 3 - Gain G:
  -1.2740  -1.5580
Test Case 3 - Riccati X:
    7.3785   10.1200
   10.1200   21.4611
Test Case 4 - Gain G:
  -0.9706  -4.7034  -5.0760
Test Case 4 - Riccati X:
    1.9706    4.7034    5.0760
    4.7034   24.9823   25.3931
    5.0760   25.3931   32.9746
```

Input:
A, B, Q, R[, S[, E]] or sys, Q, R[, S]
Output:
G (optimal gain),
X (Riccati solution)
L (closed-loop poles)

Dependencies:
dare, care, dssdata, issample

Parameters:

- ***A/sys – State-space system***
- ***B – Input matrix***
- ***Q – State weighting matrix***
- ***R – Input weighting matrix***
- ***S – Cross-weighting matrix (optional)***
- ***E – Descriptor matrix (optional)***

DLQE

Designs the discrete-time linear-quadratic estimator gain and error covariance.
Depends on dare, using the dual formulation on the transposed system.

Scilab

```
"test case 1:"
"m1:"
 0.6061423
 0.1178593
"p1:"
 1.538988    0.2992435
 0.2992435   2.715078
"z1:"
 0.6061423   0.1178593
 0.1178593   2.6798093
"e1:"
 0.3643278 + 0.i
 0.7783582 + 0.i
"test case 2:"
"m2:"
 0.6061423
 0.1178593
"p2:"
 1.538988    0.2992435
 0.2992435   2.715078
```

Octave

```
test case 1:
m1:
   0.6061
   0.1179
p1:
   1.5390   0.2992
   0.2992   2.7151
z1:
   0.6061   0.1179
   0.1179   2.6798
e1:
   0.3643
   0.7784
test case 2:
m2:
   0.6061
   0.1179
p2:
   1.5390   0.2992
   0.2992   2.7151
```

Input:
A, G, C, Q, R[, S] or sys, Q, R[, S]
Output:
M (Kalman gain)
P (error covariance),
Z (Riccati solution)
E (estimator poles)

Dependencies:
dare
Parameters:

- ***A – State matrix***
- ***G – Process noise matrix***
- ***C – Output matrix***
- ***Q – Process noise covariance***
- ***R – Measurement noise covariance***
- ***S – Cross-covariance matrix (optional)***

LQR

Continuous-time counterpart of dlqr; computes the optimal LQ regulator gain.
Depends on dare and care according to if the embedded system is continuous or not, dssdata, and issample.

Scilab

```
"test case 1:"
"m1:"
 0.6061423
 0.1178593
"p1:"
 1.538988    0.2992435
 0.2992435   2.715078
"z1:"
 0.6061423   0.1178593
 0.1178593   2.6798093
"e1:"
 0.3643278 + 0.i
 0.7783582 + 0.i
"test case 2:"
"m2:"
 0.6061423
 0.1178593
"p2:"
 1.538988    0.2992435
 0.2992435   2.715078
```

Octave

```
test case 1:
m1:
   0.6061
   0.1179
p1:
   1.5390   0.2992
   0.2992   2.7151
z1:
   0.6061   0.1179
   0.1179   2.6798
e1:
   0.3643
   0.7784
test case 2:
m2:
   0.6061
   0.1179
p2:
   1.5390   0.2992
   0.2992   2.7151
```

Input:
A, B, Q, R[, S[, E]] or sys, Q, R[, S]
Output:
G (optimal gain)
X (Riccati solution)
L (closed-loop poles)

Dependencies:
care, dare, dssdata, issample

Parameters:

- ***A/sys – State-space system***
- ***B – Input matrix***
- ***Q – State weighting matrix***
- ***R – Input weighting matrix***
- ***S – Cross-weighting matrix (optional)***
- ***E – Descriptor matrix (optional)***

LQE

Continuous-time counterpart of dlqr; computes the optimal LQ regulator gain.
Depends on dare and care according to if the embedded system is continuous or not, dssdata, and issample.

Scilab

```
"test case 1:"
"m1:"
 0.6061423
 0.1178593
"p1:"
 1.538988    0.2992435
 0.2992435   2.715078
"z1:"
 0.6061423   0.1178593
 0.1178593   2.6798093
"e1:"
 0.3643278 + 0.i
 0.7783582 + 0.i
"test case 2:"
"m2:"
 0.6061423
 0.1178593
"p2:"
 1.538988    0.2992435
 0.2992435   2.715078
```

Octave

```
test case 1:
m1:
   0.6061
   0.1179
p1:
   1.5390   0.2992
   0.2992   2.7151
z1:
   0.6061   0.1179
   0.1179   2.6798
e1:
   0.3643
   0.7784
test case 2:
m2:
   0.6061
   0.1179
p2:
   1.5390   0.2992
   0.2992   2.7151
```

Input:
A, G, C, Q, R[, S] or sys, Q, R[, S]
Output:
L (Kalman gain)
P (error covariance)
E (estimator poles)

Dependencies:
lqr, care, dare

Parameters:

- ***A – State matrix***
- ***G – Process noise matrix***
- ***C – Output matrix***
- ***Q – Process noise covariance***
- ***R – Measurement noise covariance***
- ***S – Cross-covariance matrix (optional)***

CTRBF

Computes the controllability staircase (Kalman controllable) decomposition of a state-space system, separating controllable and uncontrollable state subspaces using an orthogonal similarity transformation. Returns the transformed state-space matrices Ac, Bc, Cc, the orthogonal transformation matrix Z, and the number of controllable states ncont. Depends on the custom __sl_tb01ud__ routine.

Scilab

```
"test case 1:"
"Ac1:"
-3.  -2.
 1.   0.
"Bc1:"
-1.
 0.
"Cc1:"
 0.  -1.
"Z1:"
 0.  -1.
-1.   0.
"Ncont1:"
 2.
"test case 2:"
"Ac2:"
 1.   0.
 0.   2.
"Bc2:"
 1.
 0.
```

Octave

```
test case 1:
Ac1:
  -3  -2
   1   0
Bc1:
  -1
   0
Cc1:
   0  -1
Z1:
   0  -1
  -1   0
Ncont1:
2
test case 2:
Ac2:
   1   0
   0   2
Bc2:
   1
   0
```

Input:
A, B, C[, tol] or sys[, tol]
Output:
Ac, Bc, Cc (transformed system)
Z (orthogonal transformation)
ncont (number of controllable states)

Dependencies:
__sl_tb01ud__, ssdata

Parameters:

- ***A – State matrix***
- ***B – Input matrix***
- ***C – Output matrix***
- ***sys – State-space system***
- ***tol – Rank tolerance (optional)***

OBSVF

Computes the observability staircase decomposition (Kalman observable) of a state-space system, separating observable and unobservable state subspaces using an orthogonal similarity transformation.
Returns the transformed state-space matrices, the orthogonal transformation matrix Z, and the number of observable states nobs. Depends on the ctrbf function.

Scilab

```
"Test 1:"
"Nobs1:"
 2.
"Ao1:"
 0.    1.
-2.   -3.
"Test 2:"
"Nobs2:"
 1.
"Ao2:"
 1.    0.
 0.    2.
"Test 3:"
"Nobs3:"
 0.
"Ao3:"
 1.    2.
 3.    4.
```

Octave

```
Test 1:
Nobs1:
2
Ao1:
   0   1
  -2  -3
Test 2:
Nobs2:
1
Ao2:
   1   0
   0   2
Test 3:
Nobs3:
0
Ao3:
   1   2
   3   4
```

Input:
A, B, C[, tol] or sys[, tol]
Output:
Ao, Bo, Co (transformed system)
Z (orthogonal transformation)
nobs (number of observable states)

Dependencies:
ctrbf, ssdata

Parameters:

- ***A – State matrix***
- ***B – Input matrix***
- ***C – Output matrix***
- ***sys – State-space system***
- ***tol – Rank tolerance (optional)***

DSS

dss creates a descriptor (generalized state-space) model of a linear time-invariant (LTI) system. It represents systems of the form $E\dot{x}=Ax+Bu$, $y=Cx+Du$ allowing the descriptor matrix E to be specified explicitly. It supports both continuous-time and discrete-time descriptor systems.

Scilab

```
"test case 2: identity e (equivalent to standard ss)"
[state-space]
A (matrix) = [0,1;-2,-3]
B (matrix) = [0;1]
C (matrix) = [1,0]
D (matrix) = 0
X0 (initial state) = [0;0]
dt (time domain) = "c"
"test case 3: discrete time descriptor system with tsam = 0.01"
[state-space]
A (matrix) = [0.5,0.1;0,0.9]
B (matrix) = [1;1]
C (matrix) = [1,0]
```

Octave

```
test case 2: identity E (equivalent to standard ss)
  <class ss>
test case 3: discrete time descriptor system with Ts = 0.01
  <class ss>
```

Input:

A, B, C, D, E, tsam (optional)

Output:

- ***sys – Descriptor state-space (dss) LTI system.***

Dependencies:

__sys_data__

Parameters:

- ***A – State matrix.***
- ***B – Input matrix.***
- ***C – Output matrix.***
- ***D – Feedthrough matrix.***
- ***E – Descriptor matrix.***
- ***tsam – Sampling time.***
- ***sys – Descriptor state-space model.***

Linear Time-Invariant (LTI) Systems and State-Space Representation

Linear Time-Invariant (LTI) Systems:
An LTI system is a mathematical model whose parameters remain constant over time and whose behavior satisfies the principles of linearity and time invariance.

The state-space representation used in Scilab is

$$\begin{aligned} X' &= Ax + Bu \\ Y &= Cx + Du \end{aligned}$$

where the matrices define the system dynamics.
Example: For the transfer function

$$G(s) = \frac{1}{s+2} \qquad \begin{aligned} \dot{x} &= -2x + u \\ y &= x \end{aligned}$$

The state-space realization is : A=[−2],B=[1],C=[1],D=[0]

HORZCAT @lti

horzcat horizontally concatenates two or more LTI systems, combining their inputs while preserving the same outputs. All systems must have the same number of outputs and compatible sampling times.
Depends on __sys_group__

Scilab

```
"T1 sys:"
[1x2 state-space]
A (matrix) = [1,0;0,2]
B (matrix) = [1,0;0,1]
C (matrix) = [1,1]
D (matrix) = [0,0]
X0 (initial state) = [0;0]
dt (time domain) = "c"
"T2 sys:"
[1x2 state-space]
A (matrix) = [0.9,0;0,0.8]
B (matrix) = [1,0;0,1]
C (matrix) = [1,1]
D (matrix) = [0,0]
X0 (initial state) = [0;0]
dt (time domain) = "d"
```

Octave

```
test case 1:

sys1.a =
       x1  x2
   x1   1   0
   x2   0   2

sys1.b =
       u1  u2
   x1   1   0
   x2   0   1

sys1.c =
       x1  x2
   y1   1   1

sys1.d =
       u1  u2
   y1   0   0

Continuous-time model.
```

```
test case 2:

sys2.a =
         x1    x2
   x1   0.9     0
   x2     0   0.8

sys2.b =
       u1  u2
   x1   1   0
   x2   0   1

sys2.c =
       x1  x2
   y1   1   1

sys2.d =
       u1  u2
   y1   0   0

Sampling time: 1 s
Discrete-time model.
```

Input:
sys1, sys2, ...

Output:
sys

Dependencies:
__sys_group__

Parameters:

- ***sys1, sys2, ... – LTI systems to concatenate.***
- ***sys – Horizontally concatenated LTI system.***

VERTCAT @lti

vertcat vertically concatenates two or more LTI systems, combining their outputs while preserving the same inputs. All systems must have the same number of inputs and compatible sampling times.
Depends on __sys_group__

Scilab

```
"T4 sys:"
[3x1 state-space]
A (matrix) = [-1,0,0;0,-1,0;0,0,-1]
B (matrix) = [1;1;1]
C (matrix) = [1,0,0;0,2,0;0,0,3]
D (matrix) = [0;0;0]
X0 (initial state) = [0;0;0]
dt (time domain) = "c"
```

Octave

```
sys4.a =
       x1  x2  x3
   x1  -1   0   0
   x2   0  -1   0
   x3   0   0  -1

sys4.b =
       u1
   x1   1
   x2   1
   x3   1

sys4.c =
       x1  x2  x3
   y1   1   0   0
   y2   0   2   0
   y3   0   0   3

sys4.d =
       u1
   y1   0
   y2   0
   y3   0

Continuous-time model.
```

Input:
sys1, sys2, ...

Output:
sys

Dependencies:
__sys_group__

Parameters:

- ***sys1, sys2, ... – LTI systems to concatenate.***
- ***sys – Vertically concatenated LTI system.***

MINUS @lti

Subtracts one compatible LTI system from another and returns the resulting system. Both systems must have identical input-output dimensions and compatible sampling times. The operation is implemented by grouping the systems and applying appropriate input/output scaling matrices.

Scilab

```
"T1 sys:"
[state-space]
A (matrix) = [-1,0;0,-2]
B (matrix) = [1;1]
C (matrix) = [1,-1]
D (matrix) = 0
X0 (initial state) = [0;0]
dt (time domain) = "c"
"T2 sys:"
[state-space]
A (matrix) = [0.9,0;0,0.8]
B (matrix) = [1;1]
C (matrix) = [1,-1]
D (matrix) = 0
X0 (initial state) = [0;0]
dt (time domain) = "d"
```

Octave

```
sys1.a =
        x1  x2
   x1   -1   0
   x2    0  -2

sys1.b =
        u1
   x1    1
   x2    1

sys1.c =
        x1  x2
   y1    1  -1

sys1.d =
        u1
   y1    0

Continuous-time mode
```

```
sys2.a =
         x1   x2
   x1   0.9    0
   x2     0  0.8

sys2.b =
        u1
   x1    1
   x2    1

sys2.c =
        x1  x2
   y1    1  -1

sys2.d =
        u1
   y1    0

Sampling time: 1 s
Discrete-time model.
```

Input:
sys1, sys2

Output:
sys

Dependencies:
__sys_group__

Parameters:

- ***sys1 - first LTI system***
- ***sys2 - second LTI system***
- ***sys - resulting LTI system after addition.***

PLUS @lti

Adds two compatible LTI systems and returns the resulting system. Both systems must have identical input-output dimensions and compatible sampling times. The operation is implemented by grouping the systems and applying appropriate input/output scaling matrices.

Scilab

```
"plus_lti Test 4:"
[state-space]
A (matrix) = [-4,0;0,-4]
B (matrix) = [1.7320508;1.7320508]
C (matrix) = [1.7320508,1.7320508]
D (matrix) = 0
X0 (initial state) = [0;0]
dt (time domain) = "c"
"plus_lti Test 5:"
[state-space]
A (matrix) = [-5,0;0,-6]
B (matrix) = [1;2.236068]
C (matrix) = [1,2.236068]
D (matrix) = 0
X0 (initial state) = [0;0]
dt (time domain) = "c"
```

Octave

```
Test Case 4:
[state-space]
A (matrix) =
  -4   0
   0  -4
B (matrix) =
   1.7321
   1.7321
C (matrix) =
   1.7321   1.7321
D (matrix) =
0
X0 (initial state) =
   0
   0
dt (time domain) = "c"
```

```
Test Case 5:
[state-space]
A (matrix) =
  -5   0
   0  -6
B (matrix) =
   1.0000
   2.2361
C (matrix) =
   1.0000   2.2361
D (matrix) =
0
X0 (initial state) =
   0
   0
dt (time domain) = "c"
```

Input:
sys1, sys2

Output:
sys

Dependencies:
__sys_group__

Parameters:

- ***sys1 - first LTI system***
- ***sys2 - second LTI system***
- ***sys - resulting LTI system after addition.***

FILTDATA

Extracts the numerator and denominator coefficients and sample time from a discrete-time LTI system.

Pads the num/den polynomials to equal lengths so they can be used directly as digital filter coefficients.

Scilab

```
"Test Case 1:"
 0.    0.5    1.
 0.1   -0.2    1.
 0.1
"Test Case 2:"
 5.
 1.
 0.2
"Test Case 3:"
 0.1    0.2    0.1
 0.36   -1.2    1.
 1.
```

Octave

```
Test Case 1:
{
  [1,1] =

           0   1.0000   0.5000

}
{
  [1,1] =

     1.0000  -0.2000   0.1000

}
0.1000
Test Case 2:
{
  [1,1] = 5
}
{
  [1,1] = 1
}
0.2000
Test Case 3:
{
  [1,1] =

     0.1000   0.2000   0.1000

}
{
  [1,1] =

     1.0000  -1.2000   0.3600

}
1
```

Input:
sys, rtype (optional)
Output:
num, den, tsam

Dependencies:
tf, tfdata, isdt, issiso, prepad

Parameters:

- ***sys – Discrete-time transfer function.***
- ***rtype – Output format ("cell" or "vector" for SISO systems).***
- ***num – Numerator coefficient vector(s).***
- ***den – Denominator coefficient vector(s).***
- ***tsam – Sampling time.***

WESTLANDLYNX

westlandlynx returns the linearized state-space model of the Westland Lynx helicopter, a well-known benchmark aircraft used in control systems research. The model is commonly used for studying multivariable control design, state estimation, system identification, and robustness analysis of helicopter dynamics. It provides the helicopter dynamics as an LTI state-space system suitable for simulation and controller development.

Code snippet

```
function outsys = WestlandLynx()

    if argn(2) ~= 0 then
        error("Usage: outsys = WestlandLynx()");
    end

    a01 = [                0                 0                 0   0.99857378005981;
                           0                 0   1.00000000000000  -0.00318221934140;
                           0                 0 -11.57049560546880  -2.54463768005371;
                           0                 0   0.43935656547546  -1.99818229675293;
                           0                 0  -2.04089546203613  -0.45899915695190;
         -32.10360717773440                  0  -0.50335502624512   2.29785919189453;
           0.10216116905212  32.05783081054690  -2.34721755981445  -0.50361156463623;
          -1.91097259521484   1.71382904052734  -0.00400543212891  -0.05741119384766];
    a02 = [  0.05338427424431                 0                 0                 0;
             0.05952465534210                 0                 0                 0;
            -0.06360262632370   0.10678052902222  -0.09491866827011   0.00710757449269;
                           0   0.01665188372135   0.01846204698086  -0.00118747074157;
            -0.73502779006958   0.01925575733185  -0.00459562242031   0.00212036073208;
                           0  -0.02121581137180  -0.02116791903973   0.01581159234047;
             0.83494758605957   0.02122657001019  -0.03787973523140   0.00035400385968;
                           0   0.01398963481188  -0.00090675335377  -0.29051351547241];
    A = [a01 a02];
```

```
[6x4 state-space]
A (matrix): [8x8 double]
B (matrix): [8x4 double]
C (matrix): [6x8 double]
D (matrix) = [0,0,0,0;0,0,0,0
X0 (initial state) = [0;0;0;0
dt (time domain) = "c"
```

Conclusion

- I was able to build a set of control system functions in native Scilab that didn't exist in the toolbox before, using Octave's neat documentation.

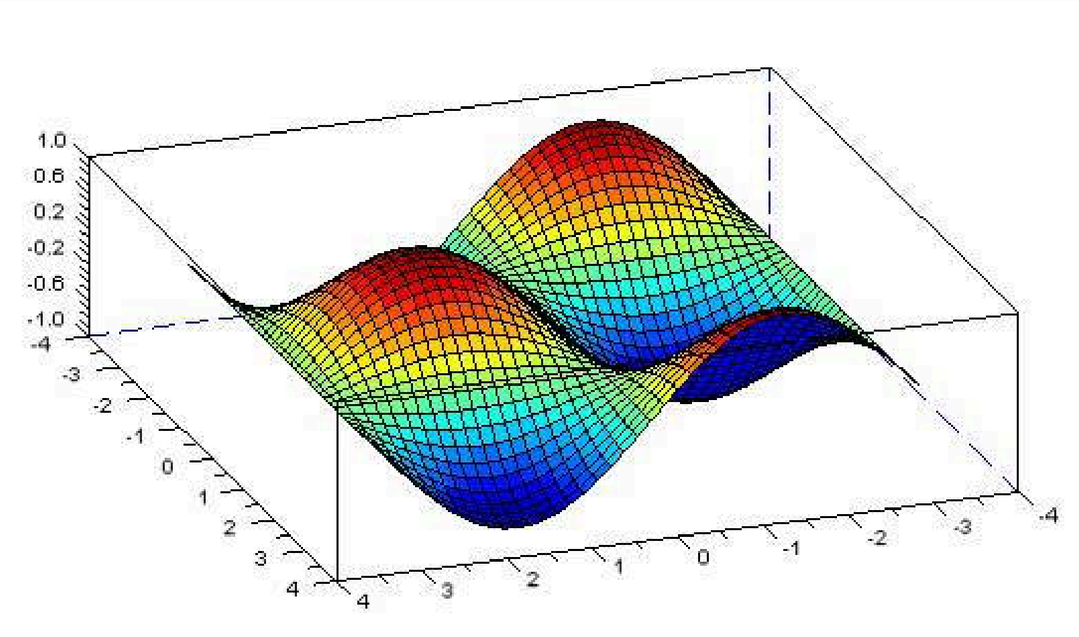
- The hardest part for me was reconstructing SLICOT-dependent logic using Scilab's own tools, testing constantly against Octave's output to make sure I'd actually gotten it right.

- AI tools helped me debug and learn faster along the way. AI could help expand this toolkit to people who feel intimidated by control systems.

Future Scope

- Kalman & Filtering: the next step: with state-space and controllability/observability work already done, extending into Kalman filtering and related estimation tools is the logical next milestone for the toolbox.

- tf & bode: these two are frequently used functions that need to be revamped in Scilab



Thank You