Control Systems Lab Manual

(Scilab is used here as a free alternative to Matlab)

No.	Title	Grade	Date
1	Introduction to Matrix and Polynomial operations		
2	Introduction to basic plotting and control flow techniques		
3	Introduction to Control System Toolbox		
4	System Design using Control System Toolbox		
5	System Modelling in Simulink		
9	Simulation of second order model of a motor		
10	Using MATLAB and GUI based SISO Tool for root locus		

Lab 6, 7 and 8 were removed as they were not Scilab related.

Lab 1

Title: Introduction to Matrix and Polynomial operations

Objectives:

1) Familiarisation with Matrix operations
2) Familiarisation with Polynomial Operations

Software:

For this course we need a software package that can handle technical computation, such as matrix manipulation, polynomials and Control System tools. Although traditionally Matlab is used in such courses, but this Lab manuals aims to use a free alternative to Matlab, known as Scilab. Scilab is in no way the only free and open source contender to Matlab, Octave is another number one alternative, but it relative ease of use and the provision of built in Simulink[1] like alternative makes it a good choice. It has a nice user interface, although it may not have the perfect aesthetics like that of Matlab.

To do quick overview of what Matlab and Scilab are lets see what does wikipedia has to say about them():

Matlab:

MATLAB (matrix laboratory) is a multi-paradigm numerical computing environment and fourth-generation programming language. Developed by MathWorks, MATLAB allows matrix manipulations, plotting of functions and data, implementation of algorithms, creation of user interfaces, and interfacing with programs written in other languages, including C, C++, Java, Fortran and Python.
Although MATLAB is intended primarily for numerical computing, an optional toolbox uses the MuPAD symbolic engine, allowing access to symbolic computing capabilities. An additional package, Simulink, adds graphical multi-domain simulation and model-based design for dynamic and embedded systems.

Scilab:

Scilab is an open source, cross-platform numerical computational package and a high-level, numerically oriented programming language. It can be used for signal processing, statistical analysis, image enhancement, fluid dynamics simulations, numerical optimization, and modelling, simulation of explicit and implicit dynamical systems and (if the corresponding toolbox is installed) symbolic manipulations.

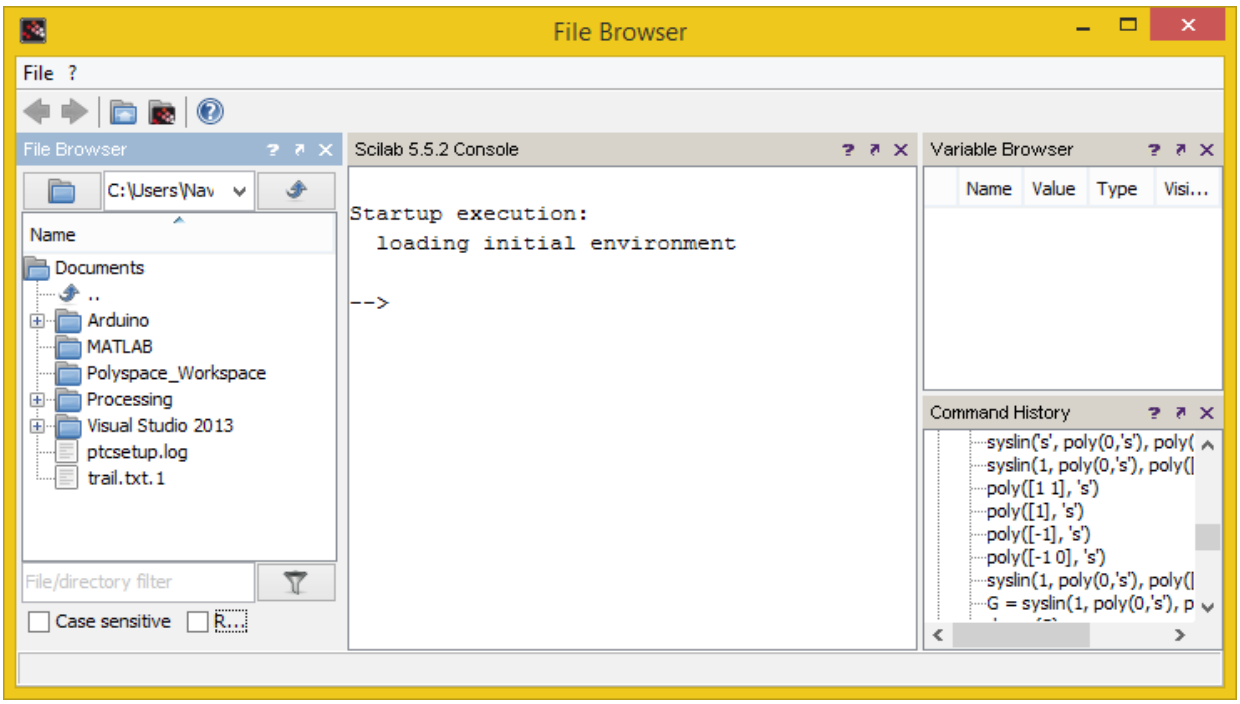
Scilab is one of the two major open-source alternatives to MATLAB, the other one being GNU Octave. Scilab is similar enough to MATLAB that some book authors (who use it) argue that it is easy to transfer skills between the two systems. Scilab however puts less emphasis on (bidirectional) syntactic compatibility with MATLAB than Octave does.

Note: So this just an overview of what these software are, and essentially we are going to be mainly focussing on their capability to handle Matrices, Polynomials and later on we will get into Control System Tool box.

[1] *Simulink is another software package that we will later talk about.*

The commonly used tools to look for are:

- The Console ('Command Window' in Matlab)
- The Command History (same in Matlab)
- The Variable Browser ('The Workspace' in Matlab)
- The Current Directory (same in Matlab), in the File Browser window
- The Help Browser (same in Matlab)
- The Execute Button ('Start' in Matlab)



(note: from now on forward there will be less comparison (between Matlab and Scilab))

Matrix Operations:

Matrices are the basic elements of the Scilab environment. A matrix is a two-dimensional array consisting of m rows and n columns. In this section of lab we will illustrate how to apply different operations on matrices.

Entering a vector:

A vector is a special case of a matrix. The elements of vectors in MATLAB are enclosed by square brackets and are separated by spaces or by commas. For example, to enter a row vector, v, type

-->v = [1 4 7 10 13]

and when you press enter, the following is displayed:

```
 v  =
   1.    4.    7.    10.    13.
```

Column vectors are created in a similar way; however, semicolon (;) must separate the components of a column vector,

-->w = [1; 4; 7; 10; 13]

Press enter and you get:

```
 w  =
   1.
   4.
   7.
   10.
   13.
```

Entering a matrix

A matrix is an array of numbers. To type a matrix into MATLAB

- begin with a square bracket, [
- separate elements in a row with spaces or commas (,)
- use a semicolon (;) to separate rows
- end the matrix with another square bracket,].

Here is a typical example. To enter a matrix A, such as,

A =

```
1 2 3
4 5 6
7 8 9
```

type,

```
-->A = [1 2 3; 4 5 6; 7 8 9]
```

press enter:

```
 A  =
   1.    2.    3.
   4.    5.    6.
   7.    8.    9.
```

Matrix indexing:

The element of row i and column j of the matrix A is denoted by A(i, j). Thus, A(i, j) in MATLAB refers to the element Aij of matrix A. Single elements of a matrix are accessed as A(i, j).

Colon operator in a matrix:

The colon operator can also be used to pick out a certain row or column. To access rows and columns, we use Scilab's colon notation (:). For example, to access second row of A, we write,

```
-->A(2, : )
 ans  =
   4.    5.    6.
```

these are the second row elements of A.

The colon operator can also be used to extract a sub-matrix from an existing matrix. e.g.:

```
-->A(:, 2:3)
 ans  =
   2.    3.
   5.    6.
   8.    9.
```

A(:, 2:3) is a sub-matrix with the last two columns of A.

Dimension:
To determine the dimensions of a matrix or vector, use the command size. For example,

```
-->size(A)
 ans  =
    3.    3.
```

which means 3 rows and 3 columns.
Or more explicitly with,

```
-->[m, n] = size(A)
 n  =
    3.
 m  =
    3.
```

Transpose Operator:
The transpose operation is denoted by an apostrophe or a single quote ('). e.g.:

```
-->z = v'
 z  =
    1.
    4.
    7.
    10.
    13.
```

Elementary Matrix generators:
Scilab provides functions that generate elementary matrices. The matrix of zeros, the matrix of ones, and the identity matrix are returned by the functions zeros, ones, and eye, respectively.
eye(n) Returns an n-by-n square identity matrix
zeros(m,n) Returns an m-by-n matrix of zeros
ones(m,n) Returns an m-by-n matrix of ones
rand(m,n) Returns an m-by-n matrix of random numbers

Here are some examples:

```
-->b = ones(3, 1)
 b  =
    1.
    1.
    1.

-->l = eye(3, 3)
 l  =
    1.    0.    0.
    0.    1.    0.
    0.    0.    1.

-->zeros(2, 3)
 ans  =
    0.    0.    0.
0.    0.    0.
```

```
-->rand(2, 2)
 ans  =
   0.3616361    0.5664249
   0.2922267    0.4826472
```

Concatenation of matrices:

In addition, matrices can be constructed in a block form. With C defined as C = [12; 3 4], we may create a matrix D as follows

```
-->D = [C zeros(2,2); ones(2, 2) eye(2, 2)]
 D  =
   1.    2.    0.    0.
   3.    4.    0.    0.
   1.    1.    1.    0.
   1.    1.    0.    1.
```

Matrix inverse:

Let's consider the same matrix A.

A =

```
1 2 3
4 5 6
7 8 0
```

In Scilab, we've a handy command to find inverse of a matrix

Example:

```
-->A = [1 2 3; 4 5 6; 7 8 0];

-->inv(A)
 ans  =
        column 1 to 2

  - 1.7777778    0.8888889
    1.5555556  - 0.7777778
  - 0.1111111    0.2222222

       column 3

  - 0.1111111
    0.2222222
  - 0.1111111
```

We can also find the determinant of A as follows:

```
-->det(A)
 ans  =

   27.
```

Matrix functions:

MATLAB provides many matrix functions for various matrix/vector manipulations, e.g.

detOfA = det(A)	returns the determinant of a matrix
diagOfA = diag(A)	returns the diagonal of a matrix
evals=spec(A)	returns in vector `evals` the eigenvalues.
[R, diagevals] =spec(A)	returns in the diagonal matrix `evals` the eigenvalues and in `R` the right eigenvectors.
inv	Matrix inverse
rank	Number of linearly independent rows or columns

In addition, it is important to remember that the three elementary operations of addition, subtraction, and multiplication apply also to matrices whenever the dimensions are compatible.

Polynomial Operations

Scilab displays polynomials as row vectors containing the coefficients ordered by descending powers. For example, the polynomial $x^4+3x^3+5x^2+7x+2$ can be entered in Scilab by typing

myPoly = [1 3 5 7 2]

MATLAB provides functions for standard polynomial operations, such as polynomial roots, and differentiation.

-->r = [1 2 3];

Polynomial with specified roots:

p = poly(r, 'x') where r is a vector whose elements are specifying the roots of desired polynomial p, and 'x' is the variable of the polynomial that we are specifying; poly(r, 'x') returns a row vector whose elements are the coefficients of the polynomial of variable 'x'. For example,

```
-->r = [1 2 3];
-->p = poly(r, 'x')
 p  =
                    2     3
  - 6  +  11x  - 6x  + x
```

The number 3 over the term x is its power, that means this is essentially x^2 and thus the term -6x under 2 represents $-6x^2$. The whole equation is like so: $x^3-6x^2+11x-6$.

Roots of a polynomial:

r = roots(c) returns a column vector whose elements are the roots of the polynomial c.

Using the polynomial p from the previous example, the roots of this polynomial are returned in a column vector by

```
-->roots(p)
 ans  =
    3.
    2.
    1.
```

Characteristic polynomial of a matrix:
The characteristic equation of the matrix
A =

```
 1 2 3
 4 5 6
 7 8 0
```

is returned in a row vector by poly: p = poly(A)

```
-->A = [1 2 3; 4 5 6; 7 8 0];
-->p = poly(A, 'x')
 p  =
                    2      3
 - 27  -  72x  -  6x  +  x
```

This has returned us the polynomial $x^3-6x^2-72x-27$.

Residue Command (work in progress, need help on this):
V=residu(P,Q1,Q2) returns the matrix V such that V(i,j) is the sum of the residues of the rational fraction P(i,j)/(Q1(i,j)*Q2(i,j)) calculated at the zeros of Q1(i,j).

Q1(i,j) and Q2(i,j) must not have any common root.

Lab Tasks
Task 1: Matrix Operations
1. Add 2, 3x4 matrices

2. Multiply 2 matrices of different dimensions

3. Find Inverse of matrix of order 4

4. Find rank of matrix

5. Find eigen values of matrix

Task 2: Polynomial Operations
1. Find characteristics polynomial of matrix

2. Find roots of characteristic polynomial

3. Find Partial Fraction Expansion for

4. Explore commands (all commands require polynomial as type, use poly method to create them)
– derivat
– k = derivat(p)
– k = derivat(a*b)
– k = derivat(a/b)
– kNumerator = numer(k)
– kDenomenator = denom(k)

Lab 2

Title: Introduction to Basic Plotting and Control flow Techniques

Objectives:

- Familiarisation with basic plotting
- Familiarisation with control flow

1. Basic Plotting

Mathematical functions:

Scilab offers many predefined mathematical functions for technical computing which contains a large set of mathematical functions. There is a long list of mathematical functions that are built into Scilab. These functions are called built-ins, such as sin(x), cos(x), tan(x), exp(x), ln(x) and more.

Defining mathematical functions:

We illustrate here some typical examples which related to the elementary functions defined in Scilab.

For example, the value of the expression $y=e^{a}\sin(x)+10\sqrt{y}$, for a = 5, x = 2 and y = 8 is computed by:

```
-->a=5; x=2; y=8;

-->y= ( exp(-a)*sin(x) ) + 10*sqrt(y)
 y  =
   28.290398
```

To calculate $\sin(\pi/4)$ and e^{10} , we enter the following commands in Scilab,

```
-->sin(%pi/4)
 ans  =
   0.7071068

-->exp(10)
 ans  =
   22026.466
```

Plotting:

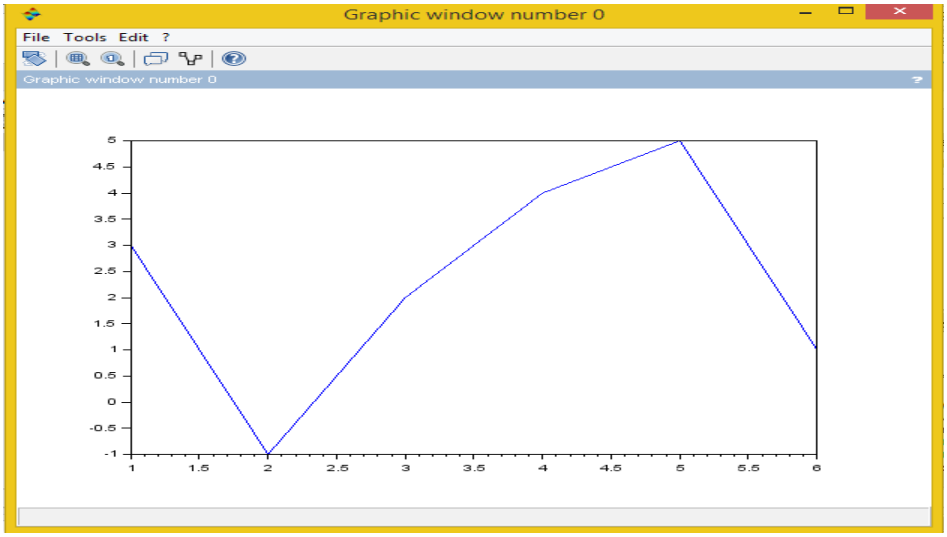
Scilab has an excellent set of graphic tools. Plotting a given data set or the results of computation is possible with very few commands.

Creating a plot:

The basic Scilab graphing procedure, for example in 2D, is to take a vector of x-coordinates, x = (x1; : : : ; xN), and a vector of y-coordinates, y = (y1; : : : ; yN), locate the points (xi; yi), with i=1;2; : : : ; n and then join them by straight lines. You need to prepare x and y in an identical array form; namely, x and y are both row arrays or column arrays of the same length.

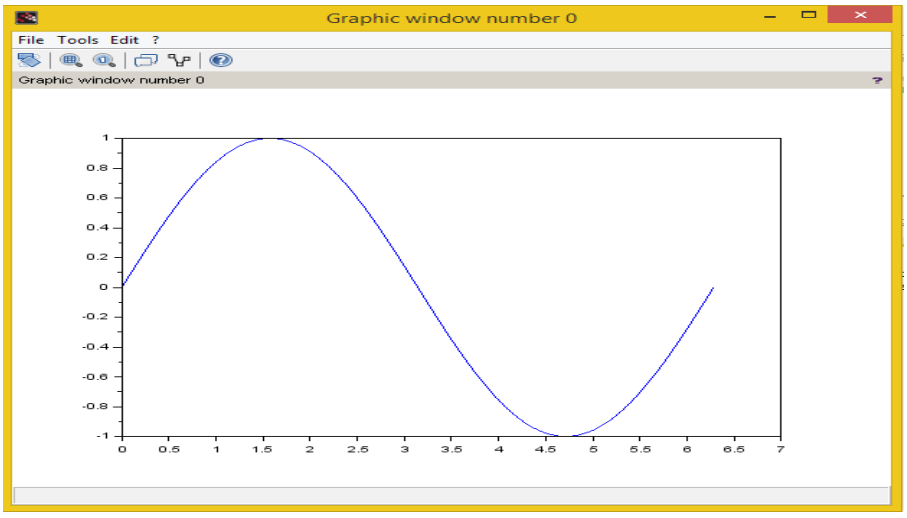
The Scilab command to plot a graph is plot(x, y). The vectors x = (1; 2; 3; 4; 5; 6) and y = (3; -1; 2; 4; 5; 1) procedure the picture shown in Figure.

-->x = [1 2 3 4 5 6]; y = [3 -1 2 4 5 1];
-->plot(x, y)



For example, to plot the function sin (x) on the interval [0; 2π], we first create a vector of x values ranging from 0 to 2π, then compute the sine of these values, and finally plot the result:

-->x = 0:%pi/100:2*%pi; // 0 to 2pi
-->y = sin(x);
-->plot(x, y)



Adding titles, axis labels, and annotations:

Scilab enables you to add axis labels and titles. For example, using the graph from the previous example, add an x-axis and y-axis labels.
Now label the axes and add a title. The character %pi creates the symbol π. An example of 2D plot is shown in Figure.

-->xlabel('x = 0 : 2*pi')
-->ylabel('Sine of x')

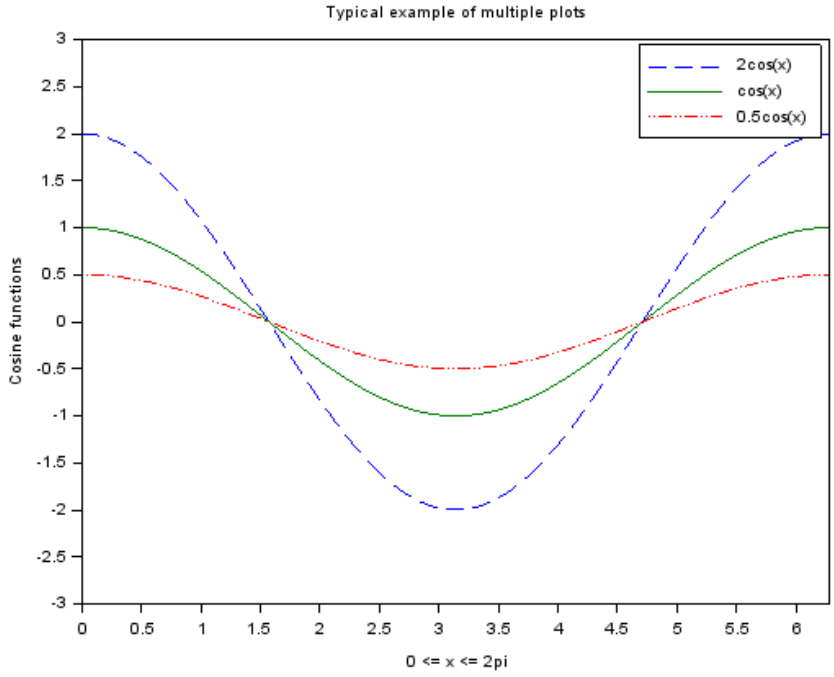
-->title('Plot of the Sine function')

The colour of a single curve is, by default, blue, but other colours are possible. The desired colour is indicated by a third argument. For example, red is selected by plot(x, y, 'r').
(Note the single quotes, ' ', around r)

Multiple data sets in one plot:

Multiple (x; y) pairs arguments create multiple graphs with a single call to plot. For example, these statements plot three related functions of x:
y1 = 2 cos(x), y2 = cos(x), and y3 =0:5 * cos(x), in the interval $0 \le x \le 2\pi$

```
-->x = 0 : %pi/100 : 2*%pi;
-->y1 = 2*cos(x); y2 = cos(x); y3 = 0.5*cos(x);
-->plot(x, y1, '--', x, y2, '-', x, y3, ':')
-->xlabel('0 <= x <= 2pi')
-->ylabel('Cosine functions')
-->legend('2cos(x)','cos(x)','0.5cos(x)')
-->title('Typical example of multiple plots')
```



2. Control Flow:

Scilab is also a programming language. Like other computer programming languages, Scilab has some decision making structures for control of command execution. These decision making or control flow structures include for loops, while loops, and if-else-end constructions.

Relational and logical operators:

A relational operator compares two numbers by determining whether a comparison is true or false. Operators Description:

> Greater than
< Less than
>= Greater than or equal to
<= Less than or equal to
== Equal to
~= Not equal to
& AND operator
| OR operator
~ NOT operator

Control Flow Structures:

Scilab has four control flow structures: the if statement, the for loop, the while loop, and the switch statement.

The ``if...end'' structure:

Scilab supports the variants of 'if' construct:

- if … then ... end
- if … then ... else ... end
- if ... elseif ... else ... end

The simplest form of the if statement is:

if <expression> then
    <statements>
end

Here are some examples based on the familiar quadratic formula.

```
discr = (b^2) - 4*a*c;
if discr<0 then
  disp('Warning: discriminant is negative, roots are imaginary')
end
// or
if discr<0 then
  disp('Warning: discriminant is negative, roots are imaginary')
else
  disp('Roots are real, but may be repeated')
end
// or
if discr<0 then
  disp('Warning: discriminant is negative, roots are imaginary')
elseif discr == 0 then
  disp('Discriminant is zero, roots are repeated')
else
  disp('Roots are real')
end
```

The ``for...end'' loop:

In the for ... end loop, the execution of a command is repeated at a fixed and predetermined number of times. The syntax is:

for <iterator_variable> = <range_expression>

<statements>

end

Usually, expression is a vector of the form i:s:j. A simple example of for loop is

```
for x = 1:5
  disp(x^2)
end
```

Multiple for loops can be nested, in which case *indentation* helps to improve the readability.

The ``while...end'' loop:

This loop is used when the number of *passes* is not specified. The looping continues until a stated condition is satisfied. The while loop has the form:

while <expression>

<statements >

end

```
x = 1;
while x <= 10
  x = 3*x;
  disp(x)
end
```

It is important to note that if the condition inside the looping is not well defined, the looping will continue *indefinitely*. If this happens, we can stop the execution by pressing Ctrl-C.

Lab Tasks:

Task 1: Linearly independent columns of a matrix

Steps:

- Define A (4x4 Matrix)
- Define B (4x3 Matrix)
- Define W = con_mat(A,B)
- Find dimension of W
- Find linearly independent columns of W and store them in matrix M.

Hints:

- Use for loop to access each column of W
- Use rank command to check independence of columns

Algorithm:

For loop

Take first/next column of W

Check the rank

If (rank increases)

Store in M

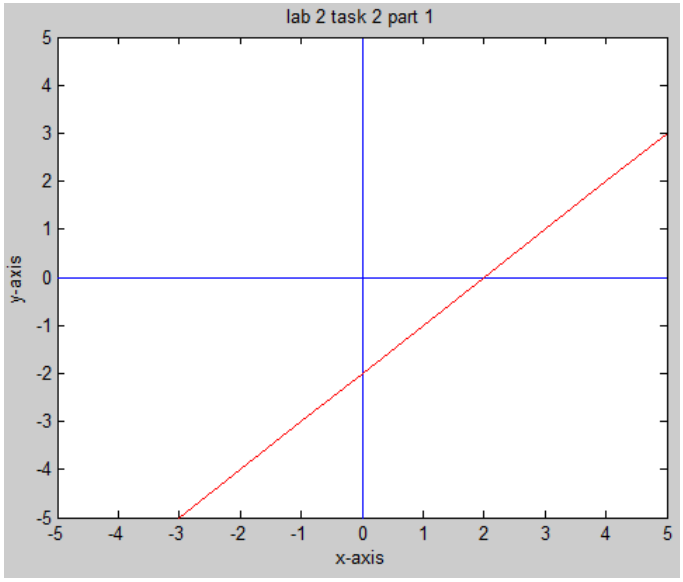
Else

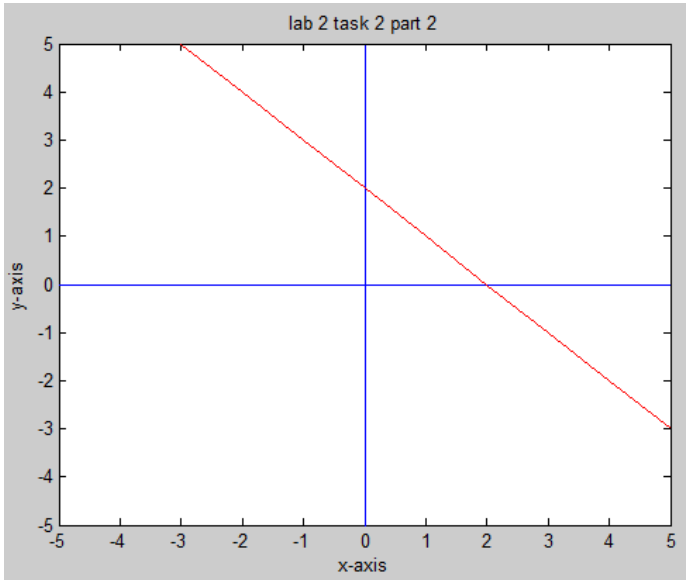
Discard it

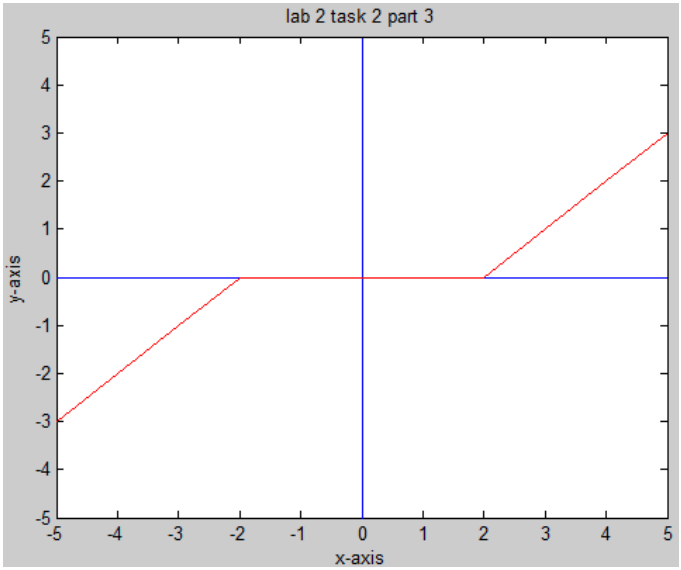
Check on for loop iterations:

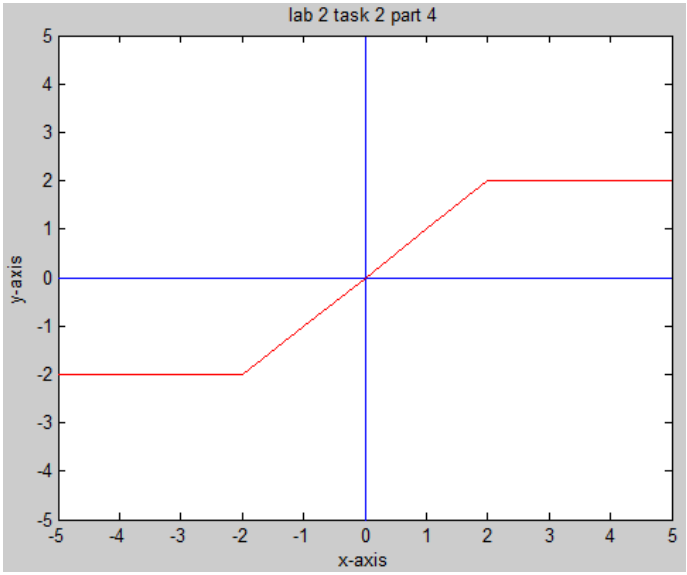
Number of iterations= length (W)

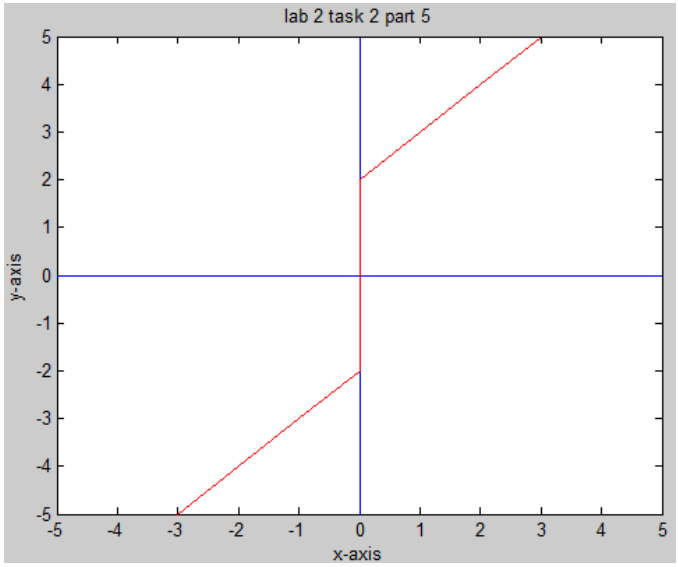
Task 2: Reproduce the following curves in Scilab:

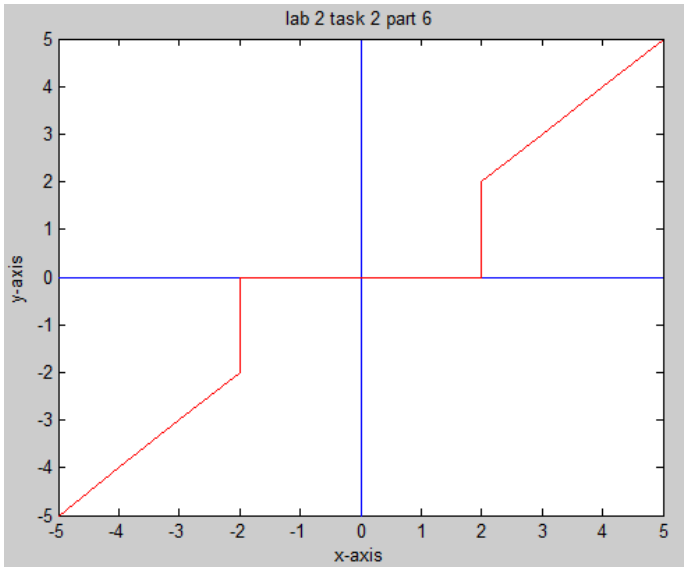












Lab 3

Title: Introduction to Control System Toolbox.

Objectives:

- Familiarisation with system representation forms in Scilab.
- Creating Transfer Function model of system

System representation:

A system is interconnection of elements and devices for a desired purpose. Mathematical models of systems can be created to have quantitative and qualitative insight of them. The Control System tools provide algorithms that implements common control system design, analysis and modelling techniques. Convenient graphical user interfaces (GUI's) simplify typical control engineering tasks.

Control systems can be modelled as transfer functions, in zero-pole gain, or state-space form. The functions syslin(dom, H) and syslin(dom, A, B, C, D) create transfer function models and state-space models respectively. These functions take the model data as input and produce respective system response. To model zero-pole-gain control systems the approach is a little meticulous; poles and zeros have to be defined as polynomials and then a product with the gain is taken, though the final result will be a simplified transfer function, any system can be viewed as zero-pole-gain molel when needed using the trfmod() function. This lab introduces transfer function based models of system.

Transfer Function models:

Transfer functions are created using the function syslin(dom, H) or syslin(dom, N, D).(*dom* can be 'c' for continuous or 'd' for discrete systems)

Purpose:

Creates or converts to transfer function model.

Syntax:

sys_tf= syslin('c', num, den) or syslin('c', H)

sys_tf= ss2tf(sys_ss)

where *num*, *den* and *H* are polynomials, and *sys_ss* is control system in state-space model form.

Description:

Returns transfer function models or converts a state-space model to transfer function form. It creates a continuous-time transfer function with numerator and denominator specified by *num* and *den*.

Example:

```
-->s = poly(0, 's');
-->num = (s^2) + (2*s);
-->den = (2*s^2) + (2*s) + 1;
-->sys_tf = syslin('c', num, den)
sys_tf  =
              2
     2s + s
   ----------
                2
   1 + 2s + 2s
```

In the example above first we created the variable s as a polynomial with root 0; essentially we wrote s = 's', this helps in evaluation of polynomials when needed. As s does not has a value thus an unknown key 's' is assigned instead; had we used a scalar value in place of 's' then the polynomial would have been evaluated immediately.

Step input function:

Step input function is used to generate step response of LTI systems.

Syntax: csim('step', t, sys_tf)

example:

```
t = linspace(0, 25, 500);
step_resp = csim('step', t, sys_tf);
plot(t, step_res);                          // use plot() to plot the response
xgrid();                                    // xgrid() turns the grid on
xtitle('step response', 'time', 'response'); // this adds the title, and axis labels
```

Impulse input function:

Impulse input function is used to generate impulse response of LTI systems.

Syntax: csim('imp', t, sys_tf)

example:

```
t = linspace(0, 25, 500);
imp_resp = csim('impulse', t, sys_tf);
plot(t, imp_res);
xgrid();
xtitle('Impulse response', 'time', 'response');
```

Ramp input function:

Ramp input function is used to generate ramp response of LTI systems.

Syntax: csim(t, t, sys_tf)

example:

```
t = linspace(0, 25, 500);
ramp_resp = csim(t, t, sys_tf);
plot(t, ramp_resp);
xgrid();
xtitle('Ramp response', 'time', 'response');
```

Parallel connection of systems:

Parallel connection is basically the addition of two LTI models and is achieved in Scilab as seen below:

```
sys = sys1 + sys2
```

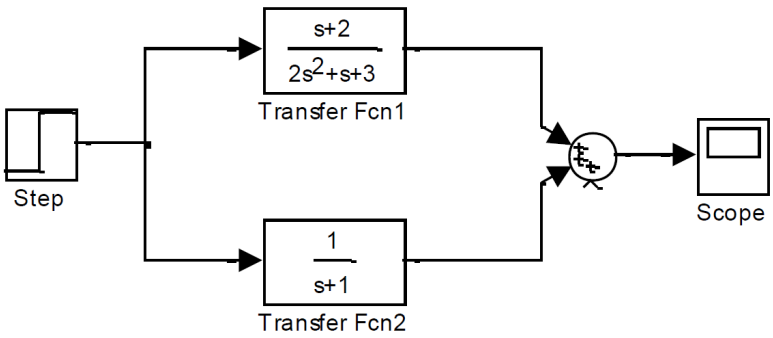
Series connection of systems:

Series connection of two LTI models is basically the product of those two systems and is achieved in Scilab as seen below:

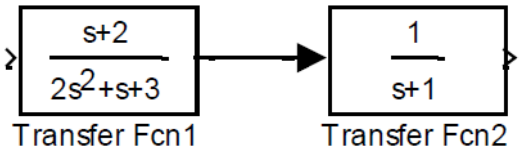
```
sys = sys1*sys2
```

Lab Tasks:

1) Plot the output of the parallel system:



2) Plot the impulse response of the series systems:



Lab 4

Title: Control System Toolbox

Objectives:

1) Creating zero-pole-gain(zpk) model of system
2) Creating state-space model of system
3) Creating hybrid model of system

Zero-pole-gain models:

Scilab, at the moment, does not have a zpk type; rather transfer functions (and state-space) models can be created using the zeros, poles and gains of a system. An existing system can also be viewed as a Zero-pole-gain model.

To create a transfer function using zeros, poles and gains see the example below:

```
z = poly([1 2 3], 's');
p = poly([-2 -1 -4], 's');
k = 5;
sys_zpk = (z*k)/p; // creates a TF using the systems zeros, poles and gain
trfmod(sys_zpk)    // only displays a system in zero-pole-gain form
```

State-space model:

State-space models are created using:

```
sys_ss = syslin(dom, A, B, C, D);
// where dom can be 'c' for continuous or 'd' for discrete. another method is to convert
// a transfer function to state-space form

sys_ss = tf2ss(sys_tf);
// there are more functions for converting other forms to state-space form too.

// A, B, C and D matrices can be extracted from a state-space model as seen below:
[A, B, C, D] = abcd(sys_ss);
```

Description of syslin(dom, A, B, C, D):

Returns state-space models

$$\dot{x} = Ax + Bu$$
$$y = Cx + Du$$

For a model with n states, p outputs, and m inputs, the matrices A, B, C, D have to be:

- A = n×n
- B = n×m
- C = p×n
- D = p×m

Example:

```
A=[1 2;5 6]; B=[1;2]; C=[1 2]; D=[0];
sys_ss = syslin('c', A, B, C, D);
[Aa, Bb, Cc, Dd] = abcd(sys_ss)
```

Output:

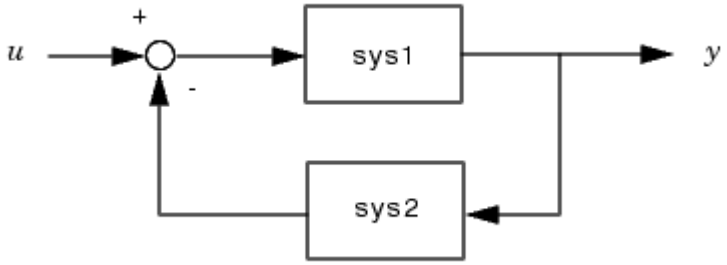
```
 Dd  =
   0.

 Cc  =
   1.    2.

 Bb  =
   1.
   2.

 Aa  =
   1.    2.
 5.    6.
```

Feedback:



Negative feedback, in Scilab, can be achieved using the following syntax:

```
sys = sys1 /. sys2
```

the /. is the operator for feedback. It gives the same effect as:

```
sys = sys1 / (1 + sys1*sys2)
```

Note: Use the /. operator instead of calculating it manually with the feedback equation, this is because Scilab has functions that do calculations efficiently and more precisely. Developing your own functions is a noble cause and a good learning experience, but to do it properly require knowledge of how the computer works. Although mathematically your function would be correct but it may produce results that are unexpected, this can happen if the calculations asked by the your code exceed the computers ability. Good code takes into account how does the computer behaves to the given commands, thus efficiency and precision can increased.

(This note is by no means discouraging your own foray into the world of programming, but rather all it is but a heads-up to those who want to explore how programmes are made and it is good practice to reuse and understand others code (in our case the built-in Scilab functions); and when writing your own code make sure you understand what variables and their types are, and how each variable type has a critical relation to size (in terms of computer memory). By all means learn, make, teach and share)

Lab Task 1:

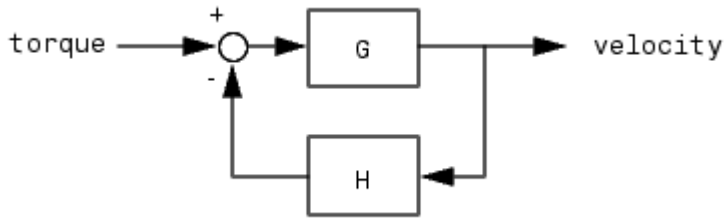
Examine system model and produce the following forms of the system:

1) Transfer function
2) State-space
3) Zero-pole-gain

Perform stability analysis of the system

To connect the plant

$$G(s) = \frac{2s^2 + 5s + 1}{s^2 + 2s + 3}$$

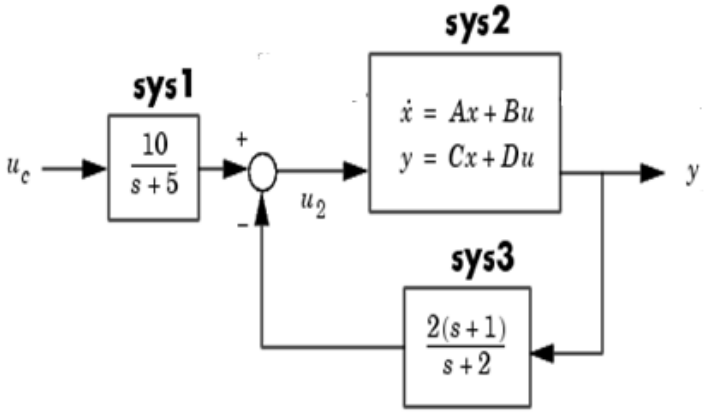


with the controller

$$H(s) = \frac{5(s+2)}{s+10}$$

Lab Task 2:

- Define A, B, C, D matrices of appropriate orders, connect all sub-systems as shown in the figure. Examine system model:
  - Tf, ss and zpk
- Plot impulse response of the system.
- Perform stability analysis of the system.



Lab 5

Title: System Modelling in Xcos

Objectives:

1) Modelling f transfer function system
2) Modelling of zero-pole-gain system
3) Modelling of state-space system

Xcos:

The Scilab website describes Xcos as:

"Xcos is a graphical editor to design hybrid dynamical systems models. Models can be designed, loaded, saved, compiled and simulated."

And the wikipedia article on Scilab says:

"Scilab also includes a free package called Xcos (based on Scicos) for modelling and simulation of explicit and implicit dynamical systems, including both continuous and discrete sub-systems. Xcos is the open source equivalent to Simulink from the MathWorks."

I think we should know what Simulink is too:

Simulink provides user-friendly environment for modelling, simulating and analysing dynamic systems. For modelling, Simulink provides a graphical user interface (GUI) for building models as block diagrams, using click-and-drag mouse operations.

Simulink includes a comprehensive block library of sinks, source, linear and non-linear components, and connectors. Using scopes and other display blocks, you can see the simulation results while simulation is running. In addition, you can change parameters and immediately see what happens. MATLAB and Simulink are integrated, you can simulate, analyse and revise your models in either environment at any point.

Although, Xcos is not that polished as Simulink is, and that Simulink has superior documentation and community support; given the fact that Xcos is free and open source, this makes the effort put into learning Xcos yourself worth while. You can use all of the Scilab/Xcos tools for free, and then freely share your own work or reuse code from the Scilab/Xcos community.

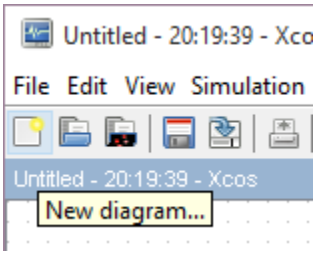
Now that we are through with the definitions, lets see what Xcos can do; but first we need open up Xcos. To do that, just type xcos in the console, and press enter. Two windows should appear; one being the Pallet browser (essentially the model blocks library), and an Untitled windows, actually called the diagram window, where the modelling and simulation happens; user can create more of diagram windows and save them.

Creating model in Xcos:

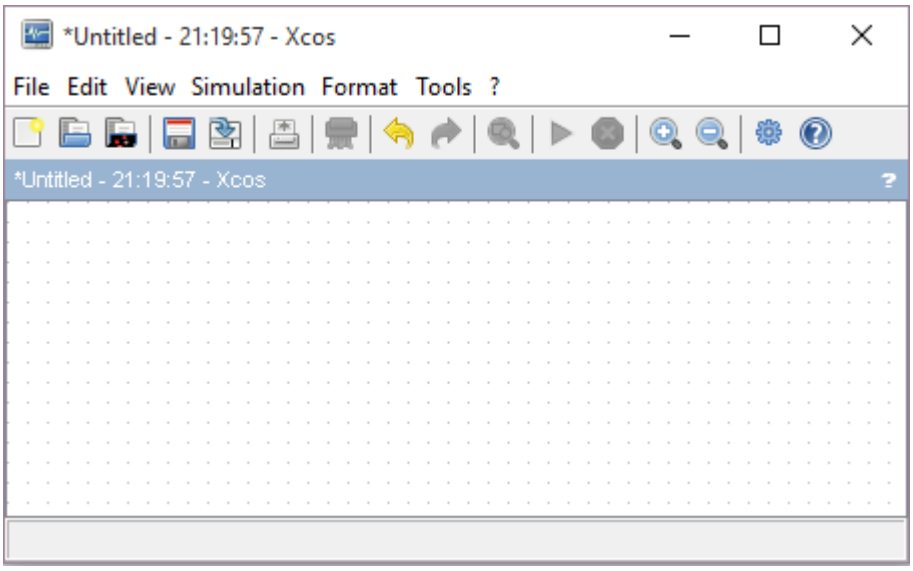
Example:

The model integrates a sine wave and displays the result along with the sine wave.

1) Though already open, if the need be; to create a new diagram window, click the New diagram button on the, already open, diagram window.



2) This opens a new diagram window.



3) Copy blocks into the model from the following Palettes:

Sources (the GENSIN_f and the CLOCK_c blocks)

Sinks (the CSCOPE block)

Continuous time systems (the INTEGRAL_f block)

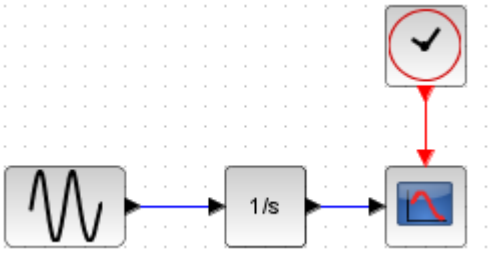
4) To copy the GENSIN_f block from the Palette Browser, click the GENSIN_f node to select the GENSIN_f block and drag the it from the Sources window to your diagram window.

5) Copy the rest of the blocks in a similar manner from their respective palettes into the diagram window.

6) If you examine the blocks, you see an arrow on the right of the GENSIN_f block and two on both sides of INTEGRAL_f. The > symbol pointing out of a block is an output port; if the arrow points to a block, it is an input port. A signal travels out of an output port and into an input port of another block through a

connecting line.

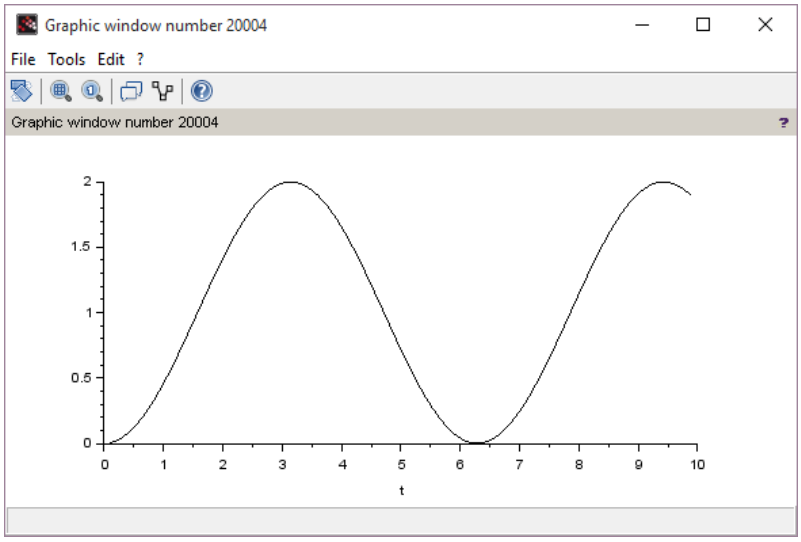
7) Connect the GENSIN_f block to the top input port of integrator. Position the pointer over the output port on the right side of the GENSIN_f block. Notice that the port near the mouse is highlighted by a light-green square. Hold down the mouse button and move the cursor to the input port of the INTEGRAL_f block.
8) Finish making rest of the block connections. This is how the diagram should look:



9) Save the model.

Simulation:

1. Set up Xcos to run the simulation for 10 seconds. First, open the Setup dialogue box by choosing Setup from the Simulation menu. On the dialogue box that appears, notice that the Final integration time is set to 1.0E05 (its default value); change this to 10.
2. Run simulation, by clicking on play button.
3. A graph should appear that is the plot of the CSCOPE. As can be seen, time is on the independent x-axis (inferred from the fact the plot of the values start from 0 and end at 10, which is the time domain we set earlier), and then of course the y-axis has the sine wave generator values; y-axis limits can be set in the CSCOPE settings, accessed when CSCOPE is double clicked, or for more control, when the plot is generated go to the Edit Menu and select the Axes properties to open the a dialogue box will appear, where Data Bounds for x-axis and y-axis can be found, in addition to other properties.



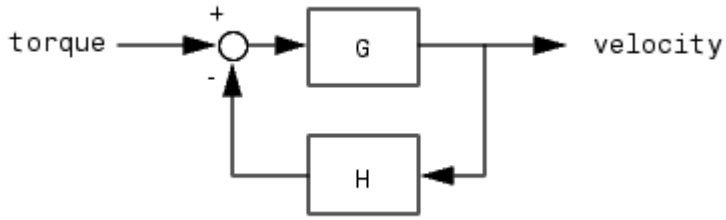
Lab Tasks

Lab Task 1

Model the given system in Simulink and plot step response of the system.

To connect the plant

$$G(s) = \frac{2s^2 + 5s + 1}{s^2 + 2s + 3}$$



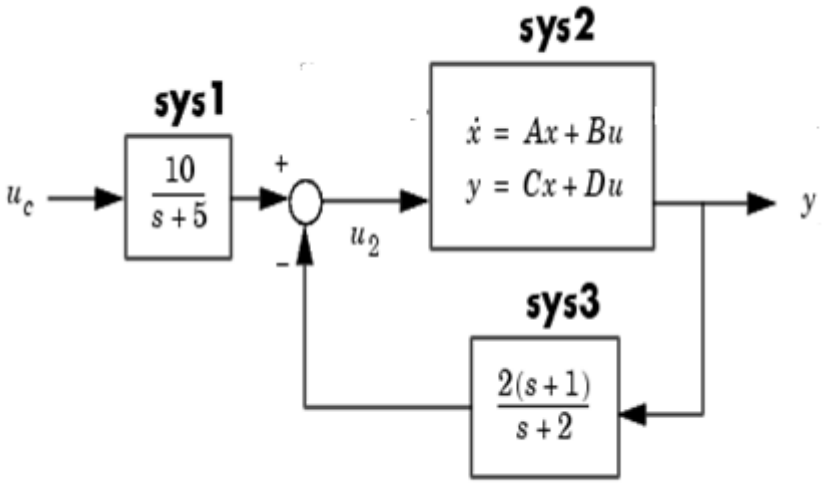
with the controller

$$H(s) = \frac{5(s+2)}{s+10}$$

Lab Task 2

- Define A, B, C, D matrices of appropriate orders, connect all sub-systems as shown in the figure. Model system in Simulink

- Plot step/ramp response of the system.



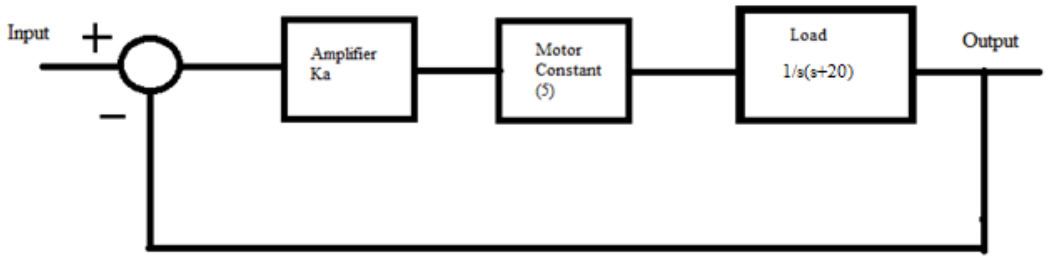
Lab 9

Title: Simulation of second-order model of motor

Objectives:

1. Develop a second-order model of a motor and arm.
2. Study closed-loop response of system to (impulse, step, ramp) inputs.
3. Study the effect of changing system gain.

System Model:



Scilab Script:

Input type:

1. Impulse input
2. Step input
3. Ramp input

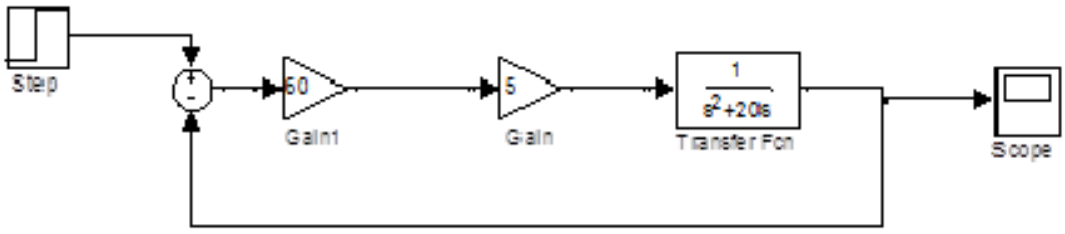
Gain:

1. Ka=30;
2. Ka=60;

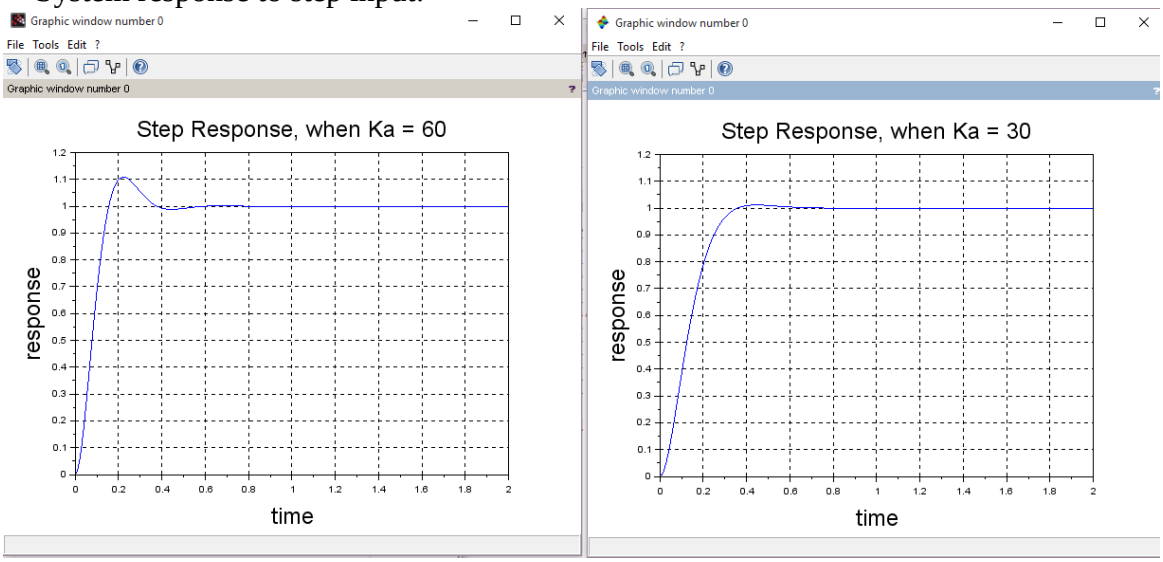
Plot system output for time $0 \leq t \leq 1$ sec

Xcos:

Simulate the motor model for above mentioned specifications in Simulink



System response to step input:



Observations:

Effect on system response by varying gain

Effect on system response by changing input types

System Response Parameters:

Rising time

Settling time

Steady state error

Lab 10

Title: Using Scilab for Root Locus Analysis

Objectives:

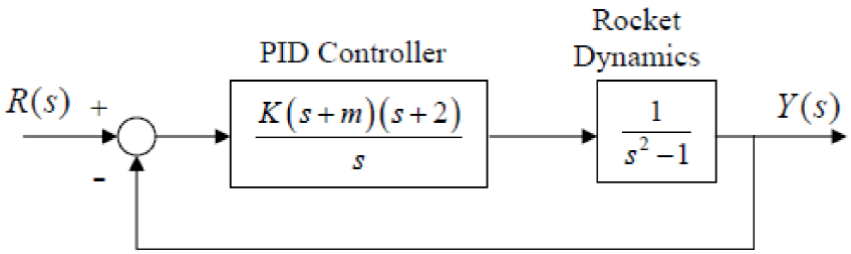
1. Use Scilab to draw the root locus of given system for parameter K.
2. Determination of parameter values associated with poles in the target region.

Root Locus:

The root locus is a curve of the location of the poles of a transfer function as some parameter (generally gain K) is varied. This is a very powerful graphical technique for investigating the effects of variation of a system parameter on the locations of closed-loop poles. The root loci are completed to select the best parameter value for stability. A normal interpretation of improving stability is when the real part of a pole is further left to the imaginary axis.

Using Scilab for Root Locus Analysis:

To demonstrate how to use Scilab to perform a root locus analysis, consider a design problem. The block diagram of the closed-loop system is shown below. The goal is to use Scilab to draw a root locus diagram for parameter K, given the parameter m= 4,



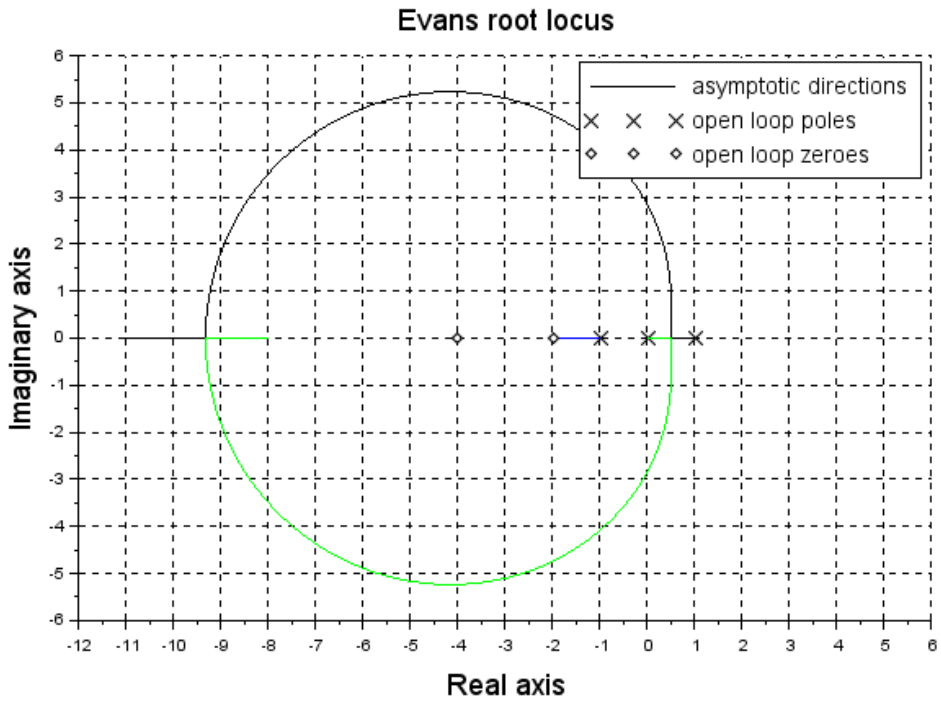
The characteristic equation of the closed-loop system is 1+ GH(s) = 0, Substituting the transfer function from the block diagram gives:

$$1+K\left[\frac{(s+4)(s+2)}{s(s^2-1)}\right]=1+K\left[\frac{s^2+6s+8}{s^3-s}\right]=0$$

The Scilab commands that produce the root locus diagram are:

```
s = poly(0, 's');
sys = syslin('c', ((s^2) + (6*s) + 8), (s^3 - s));
evans(sys, 21);
xgrid();
```

Scilab does not show the direction of the movement of the poles. It is understood that the movement is from poles to zeros.



Parameter values associated with Poles in the Target Region:

To find parameter values associated with poles within the target region. Use the "locate('no_of_points')" command in Scilab. After executing the command, click on a desirable pole location on one of the branches of root locus in the root locus plot window.

The Scilab commands are:

```
>> polePoint = [1,%i] * locate(1);          // Select a pole, this will convert the will
                                            // combine the imaginary part and real part into
                                            // a complex number
>> sys_evaluated = horner(sys , polePoint); // the horner function will evaluate the sys
                                            //transfer function using the polePoint that we
                                            // input into it.
>> k= -1 / real(sys_evaluated)              // finally take the negative reciprocal of the
                                            // evaluated function and you have the value of
                                            // 'k' for the selected pole.
>> k =
        - 0.0148269
```

Lab Task:

Repeat this lab using GUI based SISO tool available in Scilab, it goes by the package name 'rltool'. It can be found in the Module manager – ATOMS, under the Modelling and Control tools option.