

Facultad: Ingeniería.

Escuela: Electrónica

Asignatura: Sistemas de Control Automático

Lugar de Ejecución: Instrumentación y Control

Introducción al control automático con MATLAB

Objetivos Específicos

- Conocer uno de los programas para el análisis y diseño de Sistemas de Control Automático más popular.
- Operar de manera básica el programa.

Introducción Teórica

En años recientes, el análisis y diseño de sistemas de control han sido afectados dramáticamente por la proliferación del uso de las computadoras, especialmente de las computadoras personales. Estas se han hecho tan poderosas y avanzadas que pueden resolver problemas de sistemas de control con facilidad. Uno de los programas útiles en el campo del control automático es MATLAB.

MATLAB puede considerarse como un lenguaje de programación, como Fortran o C, aunque sería difícil describirlo en unas cuantas palabras. He aquí algunas de sus características más notables:

- La programación es mucho más sencilla
- Hay continuidad entre valores enteros, reales y complejos
- La amplitud de intervalo y la exactitud de los números son mayores
- Cuenta con una biblioteca matemática amplia
- Abundantes herramientas gráficas, incluidas funciones de interfaz gráfica con el usuario
- Capacidad de vincularse con los lenguajes de programación tradicionales
- Transportabilidad de los programas MATLAB.

- Capacidad de vincularse con los lenguajes de programación tradicionales.

Una característica notable de los números en MATLAB es que no hay distinción entre reales, complejos, enteros, de precisión sencilla y de doble precisión. En MATLAB, todos estos números están conectados continuamente, como debe ser. Esto significa que en MATLAB cualquier variable puede contener números de cualquier tipo sin una declaración especial durante la programación, con lo cual esta última se hace más rápida y productiva. En Fortran se requiere una subrutina distinta para cada variable sencilla o doble, real o compleja, o entera, mientras que en MATLAB no hay necesidad de separarlas.

La biblioteca matemática de MATLAB facilita los análisis matemáticos. Además, el usuario puede crear rutinas matemáticas adicionales con mucha mayor facilidad que en otros lenguajes de programación, gracias a la continuidad entre las variables reales y complejas. Entre las numerosas funciones matemáticas, los solucionadores de álgebra lineal desempeñan un papel crucial; de hecho, todo el sistema MATLAB se basa en estos solucionadores.

IMPORTANCIA DE LAS GRÁFICAS

El análisis visual de los problemas matemáticos ayuda a comprender las matemáticas y a hacerlas más asequibles. Aunque esta ventaja es bien conocida, la presentación de resultados calculados con gráficos de computadora solía requerir un esfuerzo adicional considerable. Con MATLAB, en cambio, bastan unos cuantos comandos para producir presentaciones gráficas del material matemático. Es posible crear objetos gráficos científicos e incluso artísticos en la pantalla mediante expresiones matemáticas.

Es necesario que el estudiante lea el capítulo 1 del libro “Análisis numérico y visualización gráfica con MATLAB” de Shoichiro Nakamura, disponible en la biblioteca de la UDB con la clasificación: Libro 511.7 N163 1998. Otro libro recomendado es: “MATLAB, Fundamentos y Aplicaciones al Cálculo Diferencial e Integral” de la Universidad Don Bosco Departamento de Ciencias Básicas, Unidad de Matemática.

Materiales y equipos

N°	Cantidad	Descripción
1	1	Computadora con sistema operativo Windows 98 o superior con el Programa MATLAB 5.2 o superior

Procedimiento

Parte I. Tutorial.**Funciones de transferencia:**

La sintaxis es: SYS = TF(NUM,DEN)

Se desea crear la siguiente función de transferencia de un sistema SISO en MATLAB:

$$H(s) = \frac{-3.4s + 1.5}{s^2 - 1.6s + 0.8}$$

Para crearla se tiene que escribir lo siguiente:

```
>> H1=tf([0,-3.4,1.5],[1,-1.6,0.8])
```

```
Transfer function:
  -3.4 s + 1.5
-----
s^2 - 1.6 s + 0.8
```

Otra forma de lograr lo mismo es:

```
s=tf('s'); H1 = (-3.4*s+1.5)/(s^2-1.6*s+0.8)
```

```
Transfer function:
  -3.4 s + 1.5
-----
s^2 - 1.6 s + 0.8
```

Para escribir la función de transferencia de sistemas MIMO con NY salidas y NU entradas como el siguiente de dos salidas y una entrada (SIMO):

$$H(s) = \frac{\begin{bmatrix} 3s+2 \\ s^3+2s+5 \end{bmatrix}}{3s^3+5s^2+2s+1}$$

Se puede hacer de la siguiente manera:

```
>> H=tf([3,2];[1,0,2,5]],[3,5,2,1];[3,5,2,1])
```

Transfer function from input to output...

```
#1:      3 s + 2
-----
3 s^3 + 5 s^2 + 2 s + 1
```

```
#2:      s^3 + 2 s + 5
-----
3 s^3 + 5 s^2 + 2 s + 1
```

Ganancia-Polos-Ceros

La sintaxis es: SYS = ZPK(Z,P,K), si no hay ceros se puede poner Z=[].

Se desea crear la siguiente función de transferencia de un sistema SISO en MATLAB:

$$H(s) = \frac{3(s+8)}{(s+4)(s+5)}$$

Para crearla se tiene que escribir lo siguiente:

```
>> Z=[-8];
>> P=[-4 -5];
>> K=3;
>> H2=zpk(Z,P,K)
```

Zero/pole/gain:

```
  3 (s+8)
-----
(s+4) (s+5)
```

Otra forma de realizar lo mismo es:

```
>> s=zpk('s'); H2=3*(s+8)/((s+4)*(s+5))
```

Zero/pole/gain:

```
  3 (s+8)
-----
(s+4) (s+5)
```

También se pueden representar sistemas MIMO con NY salidas y NU entradas.

Por ejemplo se desea representar el siguiente sistema de dos salidas y una entrada:

$$H_1 = \frac{-5}{s-1}, H_2 = \frac{(s-2)(s-3)}{s(s+1)}$$

```
>> H = ZPK([[]; [2 3]], [1; [0 -1]], [-5; 1])
```

```
Zero/pole/gain from input to output...
```

```
      -5
#1:  -----
      (s-1)

      (s-2) (s-3)
#2:  -----
      s (s+1)
```

Fracciones Parciales

Para encontrar la expansión en fracciones parciales o viceversa se puede utilizar el comando `residue`:

Encuentre la expansión en fracciones parciales de la siguiente función de transferencia:

$$H(s) = \frac{-3.4s + 1.5}{s^2 - 1.6s + 0.8}$$

```
>> num = [-3.4 1.5];
>> den = [1 -1.6 0.8];
>> [R, P, K] = residue(num, den)
```

```
R =
-1.7000 + 1.5250i
-1.7000 - 1.5250i
```

```
P =
0.8000 + 0.4000i
0.8000 - 0.4000i
```

```
K =
[]
```

La solución es:

$$H(s) = \frac{r(1)}{s - p(1)} + \frac{r(2)}{s - p(2)} = \frac{-1.7000 + 1.5250j}{s - 0.8000 - 0.4000j} + \frac{-1.7000 - 1.5250j}{s - 0.8000 + 0.4000j}$$

Se vuelve a obtener la función original de la siguiente forma:

```
>> [n, d] = residue(R, P, K)
n =
-3.4000 1.5000
d =
1.0000 -1.6000 0.8000
```

Conversión de modelos

Los comandos para la conversión de modelos son:

residue: Expansión en fracciones parciales.

tf(SYS): Convierte el modelo SYS al formato de función de transferencia.

zpk(SYS): Convierte el modelo SYS al formato de ceros, polos y ganancia.

Funciones de Análisis:

Respuesta en el tiempo.

impulse: Respuesta al impulso unitario.

step: Respuesta al escalón unitario.

lsim: Simula la respuesta en el tiempo de modelos LTI ante entradas arbitrarias.

Encuentre la respuesta en el tiempo ante un impulso de entrada de la siguiente función de transferencia:

$$H(s) = \frac{-3.4s + 1.5}{s^2 - 1.6s + 0.8}$$

En MATLAB se escribiría lo siguiente:

```
>> H1=tf([0,-3.4,1.5],[1,-1.6,0.8]);  
>> impulse(H1)
```

Encuentre la respuesta en el tiempo del sistema anterior ante una entrada impulso unitario en el intervalo de 0 a 15 segundos.

```
>> impulse(H1,15)
```

Encuentre la respuesta del sistema anterior ante una entrada escalón unitario y añádala en la misma gráfica de la respuesta al impulso en el intervalo de 0 a 15 segundos:

```
>> [Yi,T]=impulse(H1,15);  
>> [Ye,T]=step(H1,15);  
>> plot(T,Yi'r',T,Ye,'b')
```

La respuesta al impulso aparecerá de color rojo y la respuesta al escalón de color azul.

Propiedades del modelo:

damp: Factores de amortiguamiento y frecuencias naturales

dcgain: Ganancia de estado estable (D.C.) de sistemas continuos.

Encuentre los factores de amortiguamiento y frecuencias naturales del siguiente sistema ante una entrada escalón unitario (1/S):

$$H(s) = \frac{5}{s^2 + 0.4s + 1000}$$

La solución sería:

```
s=tf('s'); H=5/(s^2+0.4*s+1000)*1/s
```

Transfer function:

$$\frac{5}{s^3 + 0.4 s^2 + 1000 s}$$

```
>> damp(H)
```

Eigenvalue	Damping	Freq. (rad/s)
0.00e+000	-1.00e+000	0.00e+000
-2.00e-001 + 3.16e+001i	6.32e-003	3.16e+001
-2.00e-001 - 3.16e+001i	6.32e-003	3.16e+001

Ya que el amortiguamiento de este sistema es muy pequeño, la respuesta en el tiempo será muy oscilatoria.

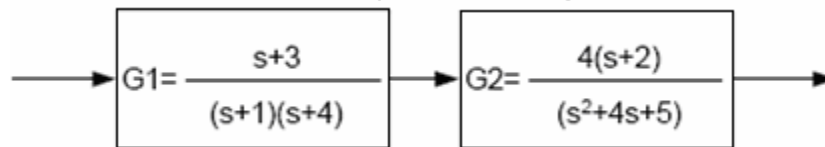
Modelado de sistemas de lazo abierto y de lazo cerrado.

parallel: Conexión en paralelo de sistemas.

series: Conexión en cascada de sistemas.

feedback: Conexión de sistemas en retroalimentación.

Encuentre la función de transferencia simplificada del siguiente sistema usando MATLAB:



Solución:

```
>> s=zpk('s'); G1=(s+3)/((s+1)*(s+4)); G2=4*(s+2)/(s^2+4*s+5);
>> G=series(G1,G2)
```

Zero/pole/gain:

$$\frac{4 (s+2) (s+3)}{(s+4) (s+1) (s^2 + 4s + 5)}$$

```
>> tf(G)
```

Transfer function:

$$\frac{4 s^2 + 20 s + 24}{s^4 + 9 s^3 + 29 s^2 + 41 s + 20}$$

Si la función de respuesta anterior se retroalimenta negativamente con una función $H(s) = 1$. Encuentre la función de transferencia de lazo cerrado G_{CL} .

Solución:

```
>> H=1;
>> Gcl=feedback(G,H,-1)

Zero/pole/gain:
      4 (s+3) (s+2)
-----
(s+3.59) (s+1.673) (s^2 + 3.737s + 7.326)

>> tf(Gcl)

Transfer function:
      4 s^2 + 20 s + 24
-----
s^4 + 9 s^3 + 33 s^2 + 61 s + 44
```

Encuentre la función de transferencia Gsum si G1 y G2 se colocan en paralelo.

```
>> Gsum=parallel(G1,G2)

Zero/pole/gain:
 5 (s+3.826) (s+1.834) (s+1.339)
-----
(s+1) (s+4) (s^2 + 4s + 5)

>> tf(Gsum)

Transfer function:
 5 s^3 + 35 s^2 + 73 s + 47
-----
s^4 + 9 s^3 + 29 s^2 + 41 s + 20
```

La transformada y Antittransformada de Laplace

La caja de herramientas de matemática simbólica de MATLAB posee la función laplace e ilaplace para transformar una función en el tiempo al dominio de la frecuencia compleja y viceversa.

Ejemplo: Encontrar la respuesta en el tiempo de la siguiente función de transferencia cuando a la entrada se presenta una señal rampa unitaria.

$$G(s) = \frac{5s^3 + 35s^2 + 73s + 47}{s^4 + 9s^3 + 29s^2 + 41s + 20}$$

Solución:

```
>> syms s t
>> G=(5*s^3+35*s^2+73*s+47)/(s^4+9*s^3+29*s^2+41*s+20);
>> g=ilaplace(G*1/s^2);
>> pretty(g)
```

$$- \frac{467}{400} + \frac{1}{48} \exp(-4 t) + \frac{2}{3} \exp(-t) + \frac{47}{20} t + \frac{4}{25} \exp(-2 t) (3 \cos(t) - 4 \sin(t))$$

```
>> ezplot(g, [0,15])
```

Ejemplo:

Encuentre la transformada de Laplace de la siguiente función:

$$g(t) = (t \cdot \sin(at) + e^{-at})u(t)$$

Donde a es una constante real positiva.

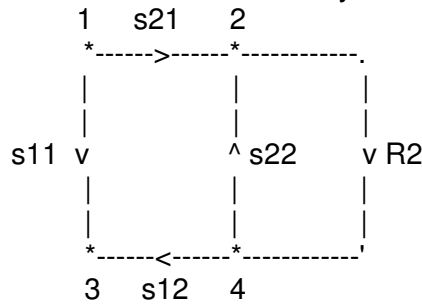
Solución:

```
>> syms a t s
>> g=t*sin(a*t)+exp(-a*t);
>> G=laplace(g);
>> pretty(G)
```

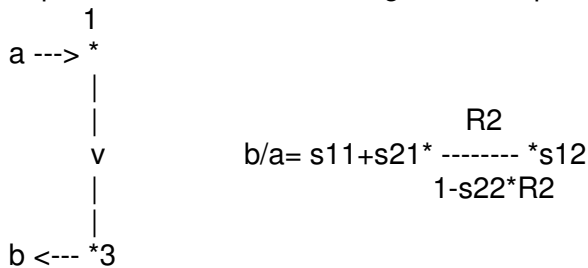
$$\frac{2}{(s^2 + a^2)^2} + \frac{1}{s + a}$$

Programa en MATLAB que resuelve las gráficas de flujo de señales por medio de la regla de Mason.

Este programa resuelve las gráficas de flujo de señales para generar una ecuación simbólica equivalente que relaciona el nodo de salida y el nodo de entrada. Por ejemplo:



Hay cuatro nodos 1,2,3 y 4 y hay cinco coeficientes S11,S21,S12,S22 y R2. Si colocamos el nodo 1 como el nodo de entrada independiente y escogemos el nodo 3 como un nodo dependiente obtenemos la siguiente simplificación:



Si colocamos al nodo 1 como el nodo de entrada independiente y el nodo 2 como el nodo de salida dependiente obtenemos:

$$\begin{array}{ccc}
 & 1 & 2 \\
 a \text{ ---} & \xrightarrow{*} & \xrightarrow{*} & \text{---} & b \\
 & s_{21} & & & \\
 b/a = & \text{-----} & & & \\
 & 1-s_{22} \cdot R_2 & & &
 \end{array}$$

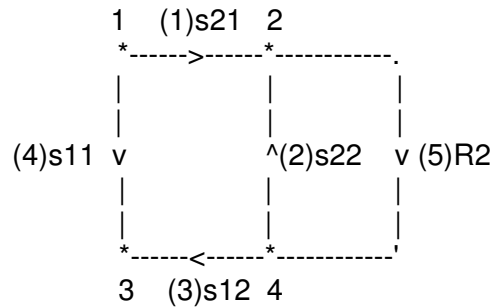
Este programa genera estas ecuaciones.

Especificando la red.

Se usa un archivo descriptor para especificar la topología del diagrama de flujo. A cada nodo se le asigna un número y cada rama se define como un número coeficiente. Cada rama se define como una línea en el archivo, descrito como sigue:

[# Coeficiente] [# Nodo de Inicio] [# Nodo Final] [Nombre del Coeficiente]

Los números de los coeficientes deben estar en orden. Por ejemplo, para describir el siguiente diagrama (donde el número del coeficiente está entre paréntesis):



Puede usar este archivo (donde los espacios en blanco son tabulaciones, pero cualquier espacio en blanco es aceptable):

```

1      1      2      s21
2      4      2      s22
3      4      3      s12
4      1      3      s11
5      2      4      R2
    
```

Este se guarda como 'example.red' en la carpeta donde está el programa.

El nombre del coeficiente puede ser cualquier expresión simbólica válida. Si la expresión tiene múltiples términos, ellos deben encerrarse entre paréntesis. Por ejemplo:

$$(-D * z^{-1}) \text{ ó } (1+B)$$

Usando el programa

Una vez se ha hecho el archivo de la red, corra 'mason.m' desde MATLAB. Se le preguntará por el nombre del archivo de la red, y los números de los nodos de Inicio y Final. Es importante que las líneas en el archivo de red estén ordenadas de modo que los números de los coeficientes se incrementen a partir de 1. No use 0 como número de coeficiente o nodo.

Escriba en MATLAB el siguiente archivo de función llamado 'mason.m':

```
function [Num,Den] = mason(NetFile,Start,Stop)
% mason.m
% This function takes a netfile describing a signal flow graph
% with symbolic coefficients and generates an equation representing
% the equivalent term between an independent input node, and dependent
% output node. Please see the *readme* file for a full description, and
% an example netfile.
%
% Author : Rob Walton
% Organisation : TRILabs and the University of Calgary, Alberta, Canada
% Date : January 25th 1999
% Revised : January 20th 2000 (Removed bug that caused the odd loop to be
thrown away)
%
% Please email me at <walton@ieee.org> if you find any errors!
%
% USAGE:
% [Numerator,Denominator] = mason(Netfile,StartNode,StopNode)
%
% Netfile - is a string with the name of the netfile to load
% StartNode - is the integer number describing the independent input node
% StopNode - is the integer number describing the dependent output node
% Numerator - is a string containing the equation for the Numerator
% Denominator - is a string containing the equation for the Denominator
%*** Load the the net description file into Net ***
% *** The first column in the file is the coefficient number (These must
be in order starting
% at 1). The second and third column are start node and stop node numbers
respectively. The last
% column is the name of the coefficient. ** See readme file **
% *** Net will be the first 3 columns of the file. Coeff will be the last
%column. ***
fid=fopen(NetFile); % Open the file
if (fid==-1)
fprintf('\n*** File, %s, not found ***\n\n',NetFile)
return
end
Net=[]; % Initialize the Numerical Net matrix
line_number=0; % Line number read from file
Coeff_Names={}; % Initialize cell array of strings
while 1 % Loop Until end of file, read one line at a time
line_number=line_number+1; % Count which line we are on
x=fscanf(fid,'%d',3); % Read the three decimal numbers into x
Coeff=fscanf(fid,'%s\n',1); % Read the one coefficient name into coeff
if isempty(x) % Stop the loop if no data is left in file
break
end
Net(line_number,:)=transpose(x); % Append one row to bottom of numerical
%matrix
Coeff_Names{line_number}= Coeff; % Append the coefficient name to the
%coefficient array
end
fclose(fid); % Remember to close the file!
%*** Determine Number of Coefficients in net file ***
temp=size(Net); % Determine the number of rows in the net matrix
```

```
Number_Coeff=temp(1);
%*** Find all the paths connecting the start and end nodes, along which
***
%*** no node is crossed more than once ***
[PathCoeffList,PathNodeList]=findpaths(Start,Stop,[],[],Net);
% PathCoeffList and PathNodeList are matrixes where each row lists the
number of the coefficients or the nodes
%visited respectively. Each row is padded with zeros to make them the
same length.
fprintf('\n- Path List -\n');
%print_paths(PathCoeffList);
%*** Determine all the first-Order loops ***
LoopCoeffList=[]; % Initialise list of coefficients in for each loop
LoopNodeList=[]; % Initialise list of nodes for each loop found
for index=1:Number_Coeff; % Find list of loops originating from each node
% Get the matrix describing all the loops from at node #index
[FoundLoops,FoundNodes]=findpaths(index,index,[],[],Net);
LoopCoeffList=[LoopCoeffList;FoundLoops]; % Append Coefficients of loops
LoopNodeList=[LoopNodeList;FoundNodes]; % Append nodes visited for each
loop
end
% Remove duplicate loops
[LoopCoeffList,LoopNodeList]=RemoveDuplicateLoops(LoopCoeffList,LoopNodeL
ist);
fprintf('\n\n- 1st Order Loop List -\n');
%print_paths(LoopCoeffList);
%*** Convert to nomenclature used by Pozars RF Book ***
% P{n} represents the nth path found connecting start to stop
% P{n}.Coeff is an array with a list of the coefficients passed through
in order
% P{n}.Nodes is an array listing the nodes passed through in order.
Including the end nodes
% NOTE: A cell array is used because the path lengths can be different
resulting in different sized arrays
% *** Make a cell array of P the different length paths ***
temp=size(PathCoeffList); % Determine number of paths
NumberPaths=temp(1);
if (NumberPaths==0);
fprintf('\n*** There are no paths connecting those nodes ***\n')
return
end
for index=1:NumberPaths % Do each path separately
Coeff=PathCoeffList(index,:); % Read Coefficients for a path
P{index}.Coeff=Coeff(1:sum(Coeff>0)); % Strip trailing zeros and put in
%struct
Node=PathNodeList(index,:); % Read node list for a path
P{index}.Node=[Node(1:sum(Coeff>0)),Stop]; % Append trailing zeros and
%put in struct.
% Append the Stop node onto the end of the node list
end
% *** Make a cell array of the first order paths, each row a different
%order ***
% *** The first column contains the number of paths of that order
% L{LoopOrder}.NumberLoops = number of loops of this order
% L{LoopOrder}.Coeff{n} = Coefficients of nth loop
% L{LoopOrder}.Node{n} = Nodes of nth loop
temp=size(LoopCoeffList);
```

```
NumberLoops=temp(1); % Determine number of first order paths
L{1}.NumberLoops=NumberLoops; % Set number of loops in the L{1} struct
for index=1:NumberLoops % Do each loop separately
Coeff=LoopCoeffList(index,:); % Read coefficients for that loop
L{1}.Coeff{index}=Coeff(1:sum(Coeff>0)); % Strip Trailing zeros and put
%in struct
Node=LoopNodeList(index,:); % Read Node list for loop
L{1}.Node{index}=[Node(1:sum(Coeff>0)),Node(1)]; % Strip trailing zeros
%and put in struct
% Append Stop node (same as first node in list
end
%*** Determine nth order loops ***
n=1; % n is the order of loops we are finding
while 1 % Loop until an order of loop is reached that is empty
n=n+1; % Count which order we are on
L{n}.NumberLoops=0; % Assume no loops of this order
% compare each first order loop with each n-1th loop. If non touching add
%to the two combined to the nth loop.
for first=1:L{1}.NumberLoops % Take each first order loop
for second=1:L{n-1}.NumberLoops % Compare with each n-1th loop
if not (AreTouchingLoops(L{1}.Node{first},L{n-1}.Node{second})) % Non
%Touching loops found
% Determine if loop is a duplicate
Duplicate=0; % A flag to indicate loop found is duplicate(0=not dup)
for index=1:L{n}.NumberLoops % Add this loop if it is not a duplicate
%entry
if IsSameLoop([L{1}.Coeff{first}, L{n-
1}.Coeff{second}],L{n}.Coeff{index}) %Duplicate found
Duplicate=1; % Set the duplicate flag
end
end
if (Duplicate==0) % Add the loop if not a duplicate
L{n}.NumberLoops=L{n}.NumberLoops+1; % Increment the number of loops of
%that order
% For Node and Coeff structs. Append a new array describing the loop of
%order n found
L{n}.Coeff{(L{n}.NumberLoops)}=[L{1}.Coeff{first}, L{n-1}.Coeff{second}];
L{n}.Node{(L{n}.NumberLoops)}=[L{1}.Node{first}, L{n-1}.Node{second}];
end
end
end
end
if (L{n}.NumberLoops==0) % If no loops of order n where found, then break
break % There will be no loops of order n+1
end
end
% ***** Display File info *****
fprintf('\n-- Network Info --\n')
fprintf('Net File : ');fprintf(NetFile);fprintf('\n');
fprintf('Start Node : %d\n',Start);
fprintf('Stop Node : %d\n',Stop);
% ***** Display the paths found *****
fprintf('\n---- Paths ----\n')
for pathn=1:length(P) % Look at each Path and display it's Coeff numbers
fprintf('P%d : ',pathn); % on a different line
fprintf('%d ',P{pathn}.Coeff);
fprintf('\n');
```

```

end
% ***** Display all the loops found *****
for loop_order=1:length(L)-1 % Look at each loop order (last order has no
%loops
fprintf('\n- Order %d Loops -\n',loop_order) % Print header describing
%loop order
for loop_number=1:L{loop_order}.NumberLoops % Look at each loop of that
%order
fprintf('L%d%d : ',loop_order,loop_number) % Display coefficients on a
%different line
fprintf('%d ',L{loop_order}.Coeff{loop_number})
fprintf('\n')
end
end
% *****
% ***** Generate the final equation *****
% *****
% For now the equations are written in terms of the coefficient number :
%c#
% the coefficient strings will be substituted later
% Determine Numerator
Num=''; % Assume Numerator is empty to start
for pathn=1:length(P) % Add Each path and related
Num=sprintf('%s*(1', Num, CoeffToString(P{pathn}.Coeff)); % Pn*(1 ...
for order=1:length(L)-1 % Append each order of sums of non-touching loops
% if order is odd order append a minus, otherwise a plus
if (rem(order,2)==1)
Num=sprintf('%s-',Num);
else
Num=sprintf('%s+',Num);
end
% Append the sum of all the nontouching loops that don't touch the
%current path
Num=[Num,PrintSumsNotTouching(L,order,P{pathn}.Node)];
end
Num=sprintf('%s+',Num); % Close the bracket around paths list of sums
end
Num=Num(1:length(Num)-1); % Remove the extra plus sign on the end.NOTE
%using /b screws up the symbolic
%math later
% Determine Denominator
Den='1'; % Denominator always start with a zero
for order=1:length(L)-1 % Add order subtract the sum of each orders loops
% if order is odd order append a minus, otherwise a plus,
if (rem(order,2)==1)
Den=sprintf('%s-',Den);
else
Den=sprintf('%s+',Den);
end
%Add or subtract all the loops
% KLUUDGE: all the loops not touching the path with nodes 99999999 are
%added
% That definetly should be all of them!
Den=[Den,PrintSumsNotTouching(L,order,[9999999 999999])]; %Sums of all
%the loops of order order
end
fprintf('\nThe variables returned are strings describing\n')

```

```

fprintf('the numerator and Denominator of the transfer equation.\n')
fprintf('If you have the symbolic toolbox, use
Denominator=sym(Denominator)\n');
fprintf('and Numerator=sym(Numerator) to make these symbolic
equations.\n')
fprintf('You can now use simple(Numerator/Denominator) to boil the
whole\n')
fprintf('thing down. You could also use simple(Numerator) to simplify
the\n')
fprintf('Numerator on it'' own.\n\n')
% ***** Convert to Symbolic and do substitutions *****
for coeff_num=length(Coeff_Names):-1:1; %Cycle through Coefficient array,
%substituting each one
orig=sprintf('c%d',Net(coeff_num,1)); % for each line generate c[Coeff
%Number] to replace
Den=strrep(Den,orig,Coeff_Names{coeff_num}); % Replace all the c#s with
%the strings from net file
Num=strrep(Num,orig,Coeff_Names{coeff_num});
end % This loop had to count down so there was no risk of C12 being
%replace by C1
%*****
*****
function Touching=AreTouchingLoops(Nodes1,Nodes2)
%*****
*****
% This function takes two arrays describing two sets of nodes
%visited(each padded with zeros).
% Return 1 if they are they are touching loops.
% Determine length of loop arrays with zeros removed
Loop1Length=sum(Nodes1>0);
Loop2Length=sum(Nodes2>0);
for first=1:Loop1Length
for second=1:Loop2Length
if (Nodes1(first)==Nodes2(second))
Touching=1;
return;
end
end
end
Touching=0;
%*****
*****
function StrMult=CoeffToString(Coefficients)
%*****
*****
% StrMult=CoeffToString(Coefficients)
% Coefficients is an array with coefficients c1,c2..cN
N=length(Coefficients); % Get length of string
StrMult=sprintf('c%d',Coefficients(1)); % Start with first coefficient
for n=2:N % Append each coefficient in list with * before it
StrMult=[StrMult, sprintf('*c'),sprintf('%d',Coefficients(n))];
end
%*****
*****
function [PathUp,NodesUp]=findpaths(StartNode,StopNode,Path,Nodes,Net)
%*****
%*****

```

```

%[PathUp,NodesUp]=findpaths(StartNode,StopNode,Path,Nodes,Net)
%
%Iterative function to find path between StartNode and StopNode. Net is
%the array with the network
%list in it. Path is the single path to date for a given route through
%the tree. PathUp is a
%list of all paths terminated below that node that are sucesfull.
%Nodes is the list of nodes tvaersed so far on the way down
%Determine number of coefficients in net
temp=size(Net);
NumberCoeff=temp(1,1);
PathUp=[];
NodesUp=[];
% Terminate branch and return nothing if the Nodes to date contains
%repetitions.
for index=1:NumberCoeff
if not(isempty(Nodes)) % Only compare if the the Path has data in it
if (sum(Nodes==index)>1)
PathUp=[];
% fprintf('Repeated Node : ');
% fprintf('%d ',Nodes);
% fprintf('\n');
return
end
end
end
% Terminate branch and return path if start and stop nodes are the same
if ((StartNode==StopNode) & (length(Path>1)))
PathUp=Path;
NodesUp=Nodes;
%fprintf('Sucessfull Path : ');
%fprintf('%d ',Path);
%fprintf('\n');
return
end
% Check for all branches leaving StartNode, and call another iteration
%for them
for index=1:NumberCoeff
if (StartNode==Net(index,2))
% Iterate with appended coeff to path and new startnode
[FoundPath,FoundNodes]=findpaths(Net(index,3),StopNode,[Path,Net(index,1)
],[Nodes,StartNode],Net);
if not(isempty(FoundPath)) %Only append if not empty
PathUp=[PathUp;[FoundPath,zeros(1,NumberCoeff+1-length(FoundPath))]];
NodesUp=[NodesUp;[FoundNodes,zeros(1,NumberCoeff+1-length(FoundPath))]];
end
end
end
%*****
%*****
function Same=IsSameLoop(Loop1,Loop2)
%*****
%*****
% This function takes two arrays describing two loops(Can be padded with
%zeros if desired).
% Return 1 if they are they describe the same circular loop.
% Determine length of loop arrays with zeros removed

```

```

Loop1Length=sum(Loop1>0); % Count all the non zero terms
Loop2Length=sum(Loop2>0);
%Return 0 if different length since the loops can't be the same!
if (Loop1Length~=Loop2Length)
Same=0;
return
end
%They are the same length so see if they contain the same nodes, but in
%any order.
% sort the nodes and subtract the two vectors. The resulting vector
%components will all be zero, only if the
%lists contains the same values.
if (sum(abs(sort(Loop1)-sort(Loop2)))==0)
Same=1; % Loops are the same
else
Same=0; % Loops are different
end
%*****
%*****
function Str=PrintSumsNotTouching(L,order,Pnodes)
%*****
%*****
% Str=PrintSumsNotTouching(L,path)
% L is the array of structs containing all the first and higher order
%loops.
% Pnodes is the array of nodes that deciding a path
%
% The function returns a string with the sum off all the loops of order
order
% that do not touch the path. The sum is contained within a set of
%brackets
No_NonTouching=1; % Flag set so indicate no nontouching loops found
Str=('('); % Open the first bracket
for n=1:L{order}.NumberLoops % Look at each loop of that order
if not (AreTouchingLoops(Pnodes,L{order}.Node{n})) %The loop doesn't touch
%the path
Str=sprintf('%s%s+',Str,CoeffToString(L{order}.Coeff{n}));% So add its
%coefficients
No_NonTouching=0; % Set flag to indicate a nontouching loop was found
end
end
Str=Str(1:(length(Str)-1));% Remove the extra plus sign (or open bracket
% if not loops were found
Str=sprintf('%s)',Str); % Append the closed bracket
%If the sum is zero return zero instead
if No_NonTouching==1 % If no loops found then return '0' instead
Str='0';
end
%*****
%*****
function [LoopList,NodeList]=RemoveDuplicateLoops(LoopList,NodeList);
%*****
%*****
% [LoopList,NodeList]=RemoveDuplicateLoops(LoopList,NodeList)
% Returns a subset of the LoopList matrix, where each duplicate row has
%been removed

```

```

% This function works on the initial loops description where each loop is
% a row in the
% the matrix padded with zeros to make them all the same length
temp=size(LoopList); % Get number of loops
NumberLoops=temp(1);
% Compare each loop with all the loops after it. And remove its
% duplicates from after it.
first=1; % The first loop
while (first<=NumberLoops) % Loop until all the loops have been used as
% first loops
second=first+1; % Start the second loop to compare as being one ahead of
% first loop
while (second<=NumberLoops) % choose the second loop to compare as all
% the ones after first
% Remove the extra loop found at the second loop index.
% NOTE: A for loop was not used since the number of loops is not constant
if (IsSameLoop(LoopList(first,:),LoopList(second,:))=1) %Remove row at
% second loop
LoopList=[LoopList(1:second-1,:);LoopList(second+1:NumberLoops,:)];
NodeList=[NodeList(1:second-1,:);NodeList(second+1:NumberLoops,:)];
NumberLoops=NumberLoops-1; % Decrement the number o loops
else
second=second+1; % Only increment second if no loop was removed
% Otherwise a new loop is move to second
end
end
first=first+1; % increment the first loop pointer
end

```

Una vez creado y guardado el archivo se utilizará para resolver el siguiente problema:

Encuentre la relación entre y_7 / y_2 en la figura 1.1, que representa la dependencia de y_7 sobre y_2 donde este último no es la entrada.

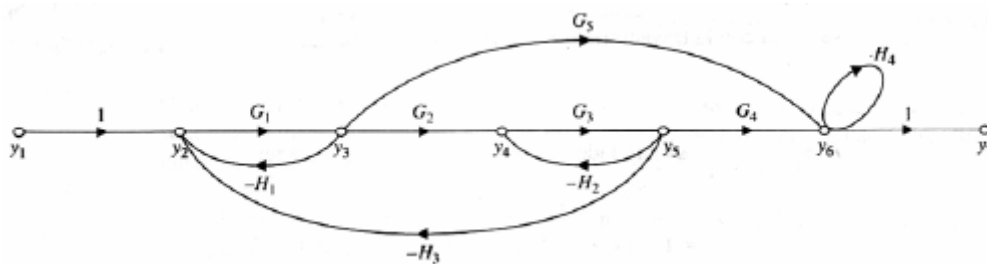


Figura 1.1. Grafica de flujo para el ejemplo del método de Mason.

Solución:

Primero creamos el archivo que define la configuración de la red y lo guardamos con un nombre, por ejemplo, "ejemplo.red".

```

1      1      2      1
2      2      3      G1
3      3      2      -H1
4      5      2      -H3
5      3      6      G5
6      3      4      G2
7      4      5      G3
8      5      4      -H2
9      5      6      G4
10     6      6      -H4
11     6      7      1

```

Notamos que como debemos encontrar y_7 / y_2 , primero hallaremos y_7 / y_1 , luego y_2 / y_1 para finalmente hallar $y_7 / y_2 = \frac{y_7 / y_1}{y_2 / y_1}$

$$\frac{y_7}{y_2} = \frac{y_7 / y_1}{y_2 / y_1}$$

Primero corremos en MATLAB la función de la siguiente manera para crear y_7 / y_1 :

```
>> [y7,y1]=mason('ejemplo.red',1,7)
```

```
-- Network Info --
Net File : ejemplo.red
Start Node : 1
Stop Node : 7
```

```
----- Paths -----
P1 : 1 2 5 11
P2 : 1 2 6 7 9 11
```

```
- Order 1 Loops -
L11 : 2 3
L12 : 2 6 7 4
L13 : 7 8
L14 : 10
```

```
- Order 2 Loops -
L21 : 2 3 7 8
L22 : 2 3 10
L23 : 2 6 7 4 10
L24 : 7 8 10
```

```
- Order 3 Loops -
L31 : 2 3 7 8 10
```

The variables returned are strings describing the numerator and Denominator of the transfer equation.

If you have the symbolic toolbox, use `Denominator=sym(Denominator)` and `Numerator=sym(Numerator)` to make these symbolic equations. You can now use `simple(Numerator/Denominator)` to boil the whole thing down. You could also use `simple(Numerator)` to simplify the Numerator on it' own.

`y7 =`

```
1*G1*G5*1*(1-(G3*-H2)+0-0)+1*G1*G2*G3*G4*1*(1-0+0-0)
```

`y1 =`

```
1-(G1*-H1+G1*G2*G3*-H3+G3*-H2+-H4)+(G1*-H1*G3*-H2+G1*-H1*-H4+G1*G2*G3*-H3*-H4+G3*-H2*-H4)-(G1*-H1*G3*-H2*-H4)
```

Luego obtenemos $y2/y1$:

```
>> [y2,y1]=mason('ejemplo.red',1,2)
```

```
-- Network Info --
```

```
Net File : ejemplo.red
```

```
Start Node : 1
```

```
Stop Node : 2
```

```
----- Paths -----
```

```
P1 : 1
```

```
- Order 1 Loops -
```

```
L11 : 2 3
```

```
L12 : 2 6 7 4
```

```
L13 : 7 8
```

```
L14 : 10
```

```
- Order 2 Loops -
```

```
L21 : 2 3 7 8
```

```
L22 : 2 3 10
```

```
L23 : 2 6 7 4 10
```

```
L24 : 7 8 10
```

```
- Order 3 Loops -
```

```
L31 : 2 3 7 8 10
```

The variables returned are strings describing the numerator and Denominator of the transfer equation.

If you have the symbolic toolbox, use `Denominator=sym(Denominator)` and `Numerator=sym(Numerator)` to make these symbolic equations.

You can now use `simple(Numerator/Denominator)` to boil the whole thing down. You could also use `simple(Numerator)` to simplify the Numerator on it' own.

$$y2 =$$

$$1 * (1 - (G3 * -H2 + -H4) + (G3 * -H2 * -H4) - 0)$$

$$y1 =$$

$$1 - (G1 * -H1 + G1 * G2 * G3 * -H3 + G3 * -H2 + -H4) + (G1 * -H1 * G3 * -H2 + G1 * -H1 * -H4 + G1 * G2 * G3 * -H3 * -H4 + G3 * -H2 * -H4) - (G1 * -H1 * G3 * -H2 * -H4)$$

Para obtener una mejor forma en la respuesta se puede realizar lo siguiente:

```
>> syms G1 G2 G3 G4 G5 H1 H2 H3 H4
>> y7=sym(1*G1*G5*1*(1-(G3*-H2)+0-0)+1*G1*G2*G3*G4*1*(1-0+0-0))
```

$$y7 =$$

$$G1 * G5 * (1 + G3 * H2) + G1 * G2 * G3 * G4$$

```
>> y2=sym(1*(1-(G3*-H2+-H4)+(G3*-H2*-H4)-0))
```

$$y2 =$$

$$1 + G3 * H2 + H4 + G3 * H2 * H4$$

Finalmente la respuesta es:

```
>> R=y7/y2;
>> pretty(R)
```

$$\frac{G1 \ G5 \ (1 + G3 \ H2) + G1 \ G2 \ G3 \ G4}{1 + G3 \ H2 + H4 + G3 \ H2 \ H4}$$

Análisis de resultados

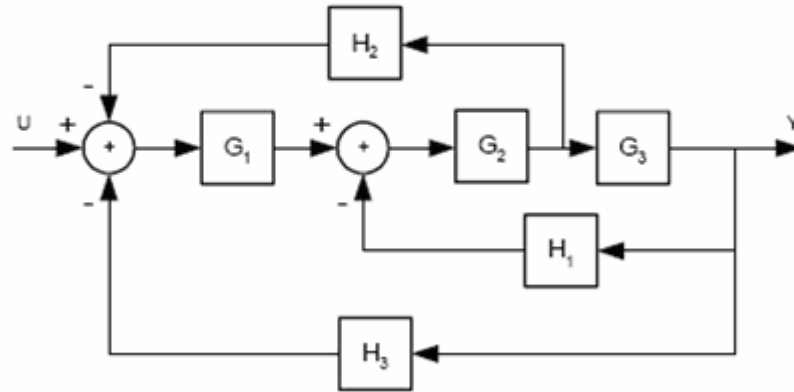
Al resolver los problemas que siguen, siempre prepare sus respuestas con MATLAB.

1. Encuentre la transformada de Laplace de las siguientes funciones:
 - a) $g(t) = \sin(2t)\cos(2t)u(t)$
 - b) $g(t) = (t \cdot \sin 2t + e^{-2t})u(t)$
2. Encuentre la antitransformada de Laplace de las siguientes funciones:

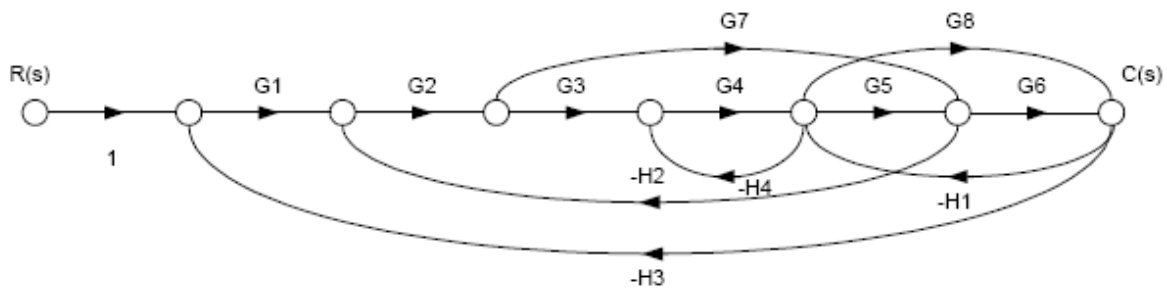
$$a) G(s) = \frac{10(s+2)}{s(s^2+2s+2)} \quad b) G(s) = \frac{e^{-2t}}{10s(s+1)(s+2)}$$

3. Para la planta que aparece en la siguiente figura, muestre que $Y(s)/U(s)$ es:

$$\frac{Y(s)}{U(s)} = \frac{G_1 G_2 G_3}{H_1 G_2 G_3 + H_3 G_1 G_2 G_3 + G_1 G_2 H_2 + 1}$$



4. Encuentre $T(s) = \frac{C(s)}{R(s)}$ para el sistema representado por el gráfico de flujo siguiente:



Bibliografía

- 📖 INGENIERÍA DE CONTROL MODERNA. Tercera Edición. Prentice Hall. Katsuhiko Ogata, Biblioteca UDB, Clasificación: Libro interno 629.8 O34 1998
- 📖 ANÁLISIS NUMÉRICO Y VISUALIZACIÓN GRÁFICA CON MATLAB. Séptima Edición. Prentice Hall. Shoichiro Nakamura, Biblioteca UDB, Clasificación: Libro 511.7 N163 1998