Sistemas y Señales I

Experimentos de Laboratorio – Trabajo Práctico 1

Problema 1:

a. Por LKT:

$$u(t) = u_R(t) + u_C(t)$$

$$u(t) = R.i_C(t) + u_C(t)$$

$$u(t) = R.(C.du_C(t)/dt) + u_C(t)$$

$$\boxed{du_C(t)/dt + (1/RC).\, u_C(t) = (1/RC).\, u(t)}$$

b. Utilizando la aproximación de Euler: $dx(t)/dt \approx [x(n+1) - x(n)] / T$

$$[\, u_C(n+1) - u_C(n)] / T + (1/RC).\, u_C(n) = (1/RC).\, u(n)$$

$$u_C(n+1) - u_C(n) + (T/RC).\, u_C(n) = (T/RC).u(n)$$

$$\boxed{u_C(n+1) = [\, 1 - (T/RC)\,].u_C(n) + (T/RC).u(n)}$$

Esta es una fórmula recursiva que puede implementarse con alguna estructura **`for`** o **`while`** a partir de conocer los valores de R, C, un valor inicial de u_C y todos los valores de la entrada.

c. Script `circuito_RC.m`

```
R = 1e6;
C = 1e-6;
Tau = R*C;
% Definimos el valor inicial de la tension en el cap.
% Como se trata de un vector el primer valor se indexa como 1
uc(1) = 0;
% Definimos el periodo de muestreo en funcion de la
% dinamica del sistema (Tau)
T = Tau/20;
% Como queremos ver toda la evolucion uc(n) definimos la entrada
% de manera tal que tengamos la cant. de valores igual a 5.Tau
A = 10;
u = A*ones(1,round(5*Tau/T));
for n = 1:1:round(5*Tau/T)-1
    uc(n+1) = ( 1 - T/Tau)*uc(n) + (T/Tau)*u(n);
end
td = [0:1:round(5*Tau/T)-1];
t = td*T;
plot(t,u,'r',t,uc,'b');
xlabel('tiempo [seg]'); ylabel('Tension [V]');
title('Evoluciones de u(t) y uc(t) en circuito RC');
```

d. Function `circuitoRC.m`

```
function [ t , uc ] = circuitoRC( uc0 , u )
% Esta funcion calcula la tension sobre un capacitor en un
% circuito serie RC alimentado con una tension u(t)
R = 1e6;
```

```
C = 1e-6;
Tau = R*C;
% El valor inicial de la tension en el cap. ahora se ingresa
% cuando se invoca a la funcion
uc(1) = uc0;
% Definimos el periodo de muestreo en funcion de la
% dinamica del sistema (Tau)
T = Tau/20;
% Como la entrada se ingresa cuando se invoca a la funcion
% no la tenemos que definir y el for debe iterar mientras existan
% valores de la entrada
for n = 1:1:length(u)-1
    uc(n+1) = ( 1 - (T/Tau) )*uc(n) + (T/Tau)*u(n);
end
uc=uc';                    %  vector columna
td = [0:1:length(u)-1]'; %  vector columna
t = td*T;
plot(t,u,'r',t,uc,'b')
xlabel('tiempo [seg]')
ylabel('Tension [V]')
title('Evoluciones de u(t) y uc(t) en circuito RC')
end
```

f. En este circuito, considerando una $u(t)=A$ constante, es fácil hallar (utilizando por ejemplo el método de inspección) la evolución exacta de $u_C(t)$. Esto es:

$$u_C(t) = u_C(t\to\infty) + [\, u_C(t=0) - u_C(t\to\infty)\,].e^{-t/\tau} \qquad \tau = RC$$

si para este circuito consideramos $u_C(t=0) = 0$ V :

$$u_C(t) = A\,[\, 1 - e^{-t/\mathbf{RC}}\,] \qquad \forall\, t \geq 0$$

Ésta expresión, que es la solución exacta, podemos discretizarla tomando muestras equiespacias el intervalo T:

$$u_C(n) = A\,[\, 1 - e^{-n/\mathbf{RC}}\,] \qquad \forall\, n = 0\,,\, T,\, 2T,\, 3T,\, \ldots.$$

Así, podemos ver si existen diferencias con esta expresión y el resultado de aproximar la ecuación diferencial por Euler usando el siguiente script: `circuito_RC_exacta.m`

```
% Solucion exacta
R = 1e6;
C = 1e-6;
Tau = R*C;
T = Tau/20;
A = 10;
% Como queremos ver toda la evolucion uc(n) definimos td
% de manera tal que tengamos un tiempo total de simulacion
% igual a 5 Tau
td = [0:1:round(5*Tau/T)-1]'*T;
uc_exacta = A*( 1 - exp(-td/Tau) );
% Usando la funcion anterior calculamos ahora
u =A*ones(round( 5*Tau/T), 1 );
[ t , uc ] = circuitoRC( 0 , u );
% Podemos ahora comparar ambos resultados
figure
plot( td , uc_exacta , t , uc )
legend('uc exacta', 'uc aprox')
xlabel('tiempo [seg]')
ylabel('Tension [V]');
title('Evoluciones de uc(t) exacta y uc(t) aproximada')
```

Problema 2: Function `problema2.m`

```
function [ n , x ] = problema2( a , n_inicial , L )
% Esta funcion calcula permita generar una señal exponencial
% en tiempo discreto de la forma:  x(n) = a^n
% con  n_inicial <= n <= n_inicial + L - 1
% Sintaxis:
%          [ n , x ] = problema2( a , n_inicial , L )
n = [ n_inicial : 1 : n_inicial + L - 1 ]';
% Utilizamos el operador  .  de Matlab para “distribuir” la
% operacion  ^  en todos los elementos del vector  n  a la base a
x = a.^n ;
% Chequeamos si  a  es real o complejo para saber cuantas graficas
% debemos realizar
if isreal(a) == 1
    stem( n , x );
else
subplot(211); stem( n , real(x) );
ylabel('real(x)')
subplot(212); stem( n , imag(x) );
ylabel('imag(x)')
xlabel('Indice n')
end
end
```

Problema 3: Function `problema3.m`

```
function [ y ] = problema3( u , h )
% Esta funcion calcula la convolución de u(n) y  h(n)
% Si considermos a u(n) como la entrada a un sistema L.E.
% cuya respuesta al impulso es h(n). Luego, la salida que genera esa
% entrada aplicada al sistema será y(n)
% Sintaxis:
%          [ y ] = problema3( u , h )
% Primero me aseguro que ambos vectores sean vectores fila.
% Usando la funcion size veo cuantas columnas tienen u y h
if size(u,2) == 1
    u = u';
end
if size(h,2) == 1
    h = h';
end
% Como ya vimos que la convolucion para el caso de señales
% discretas y causales puede resolverse calculando una matriz
% Toeplitz calculamos dicha matriz que se define a partir
% de su primer fila y primer columna
primer_fila = [ h(1) zeros( 1 , length(u) - 1 )];
primer_col = [ h  zeros( 1 , length(u) - 1 ) ]';
H = toeplitz( primer_col , primer_fila);
y = H * u';
% Si se omite el argumento de salida, la función sólo debe mostrar
% las gráficas de u(n), h(n) e y(n) entonces chequeamos si la funcion
% se invoca sin argumentos de salida
if nargout == 0
    n_u = [ 0 : 1 : length(u) - 1 ];
    subplot(311);stem( n_u , u );
    xlabel('n')
    ylabel('u(n)');
    n_h = [ 0 : 1 : length(h) - 1 ];
    subplot(312)
    stem( n_h , h )
    xlabel('n')
    ylabel('h(n)')
    n_y = [ 0 : 1 : length(y) - 1 ];
    subplot(313)
    stem( n_y , y )
    xlabel('n')
    ylabel('y(n)')
end
end
```

b. Empleamos la función desarrollada para calcular la respuesta a una entrada escalón unitario de un sistema con una respuesta al impulso $h(n) = (1/4)^n\ \mu(n)$

```
A = 1;
u = A*ones(1,100);
% Aprovechamos la funcion del Prob 2 para generar la señal h(n)
[ n , h ] = problema2( 0.25 , 0 , 20 )
% Invocamos a la funcion sin argumentos de salida de manera tal
% que nos muestre todas las graficas
problema3( u , h )
```

Problema 4: Function `xcorrelacc.m`

a.

```
function [ rxy ] = xcorrelacc( x , y )
% Esta funcion calcula la correlacion cruzada de x(n) y  y(n)
% Sintaxis:
%           [ rxy ] = xcorrelacc( x , y )
% Me aseguro que el vector y sea una fila para usar la funcion
% fliplr para reordenar los valores del vector y
if size(y,2) == 1
    y = y';
y_inv = fliplr(y);
else
y_inv = fliplr(y);
end
% Invocamos a la funcion con argumento de salida de manera tal
% que solo calcule rxy(n) = x(n)*y(-n) pero no muestre las graficas
[ rxy ] = problema3( x , y_inv );
% Como la correlacion cruzada no es causal debemos definir un vector
% acorde para poder visualizar la misma correctamente
n = [ -length(y) + 1 : 1 : length(x) - 1];
plot( n , rxy )
ylabel('rxy(n)')
xlabel('Indice n')
end
```

b. Ahora utilizamos la funcion anterior para calcular una autocorrelacion, es decir, las señales x(n) e y(n) son la misma

```
[ n , x ] = problema2( 0.5 , 0 , 100 );
[ rxx ] = xcorrelacc( x , x );
```

podemos ver en la gráfica obtenida como la autocorrelación es máxima en el origen

d. Para este ejemplo, tenemos una señal inmersa en ruido y(n) = x(n) + w(n) donde la señal x(n) (señal periódica con período desconocido N), y w(n) es ruido aditivo, entonces,

```
load 'tp1_1.mat'
[ ryy ] = xcorrelacc( y , y );
```

de la gráfica de $r_{yy}(n)$ podemos medir el periodo de la autocorrelacion que para este caso coincidirá con el periodo de x(n)