CAPÍTULO 4:

MODELO DE FUENTE DE CONTROL DE TIPO PIECEWISE LINEAR (PWL).

4.1. Introducción.

4.2. Modelo de fuente de control tipo PWL.

4.3. Consideraciones sobre las funciones MATLAB a crear.

4.4. Funciones MATLAB trabajando con fuente tipo PWL.

4.5. Ejemplo: resultados con el modelo DR-2.

4.1. Introducción.

En la segunda parte del capítulo anterior se ha mostrado como mediante algoritmos en MATLAB es posible trabajar con el modelo de pendiente simple de una manera eficaz y fiable, independientemente del fichero concreto con el que se trate.

En este capítulo se hace uso de este método para conseguir un modelo que, manteniendo la sencilla estructura del de pendiente simple, sea capaz de generar salidas tan precisas como le sean indicadas. Esto se consigue trabajando con una fuente de control de tipo piecewise linear (PWL)[1].

[1] En español fuente poligonal; se mantendrá la terminología anglosajona.

4.2. Modelo de fuente de control tipo PWL.

La estructura del modelo de fuente de control de tipo piecewise linear (PWL) es idéntica a la utilizada en el modelo de pendiente simple (figura 3.1). La única pero significativa diferencia consiste en que la fuente de control del modelo (Vx),

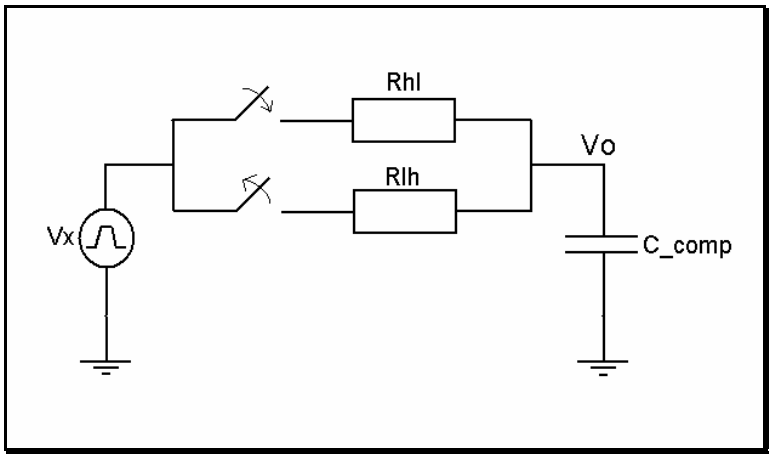

Figura 4.1: Modelo de fuente de control de tipo PWL.

es de tipo piecewise linear (PWL), lo que va a permitir forzar a la salida tantos valores como se desee.

Un ejemplo de la forma que puede tomar esta fuente de control se muestra en la figura 4.2:

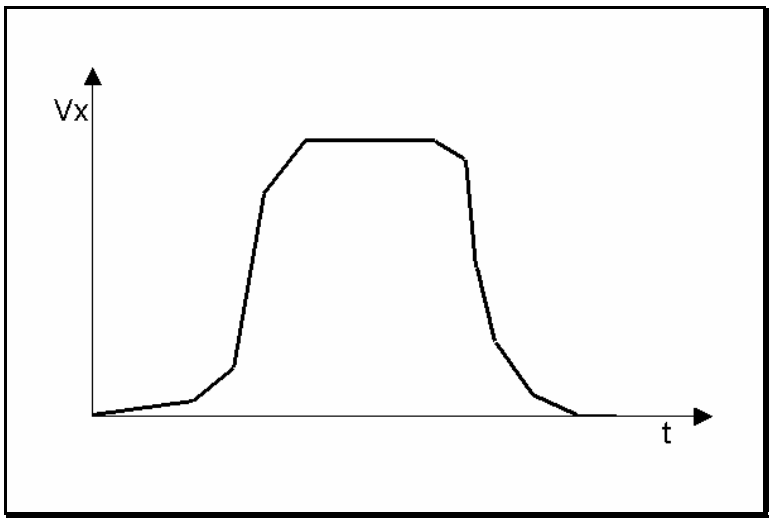

Figura 4.2: Fuente Vx para modelo pwl.

4.3. Consideraciones sobre las funciones MATLAB a crear.

El esquema de trabajo para este modelo es idéntico al ya explicado en el apartado 3.8, con la salvedad de que al poder aumentarse en gran medida el número de restricciones para la curva de salida, las funciones en MATLAB son bastante más complejas.

La evolución de las curvas de subida y bajada son calculadas mediantes las mismas expresiones desarrolladas en el apartado 3.9. Como se indicó, eran expresiones generales independientes del tipo de fuente de control

Es necesaria una referencia para el comportamiento deseado de las funciones. Ésta puede ser obtenida de dos maneras:

- A partir de [Ramp] se podrá imponer que las curvas verifiquen el correcto funcionamiento en el 20% y 80%, así como el valor inicial y final.

- Con la información descrita en [Rising Waveform] y [Falling Waveform] se pueden seleccionar pares de valores (tiempo, tensión) para forzar a los programas que sigan la forma descrita por éstas secuencias. Un inconveniente es que estas palabras reservadas no son de inclusión obligatoria en la norma.

En ambos casos, la referencia de comportamiento deseado se introduce en las funciones mediante *Tdes_* y *Vdes_*.

Los programas van a trabajan con secciones. Es posible incorporar el número de puntos en cada sección mediante *interv*. A mayor número de ellos mayor precisión en las curvas generadas pero también mayor tiempo de simulación.

4.4. Funciones MATLAB trabajando con fuente tipo PWL.

La siguiente figura muestra un esquema del modo de operación del programa *pwl_r.m* , utilizado para la generación automática de la fuente de control Vx tipo pulso cuando se caracteriza la curva de subida de un driver. Un esquema similar se sigue para calcular la característica de bajada con *pwl_f.m.*

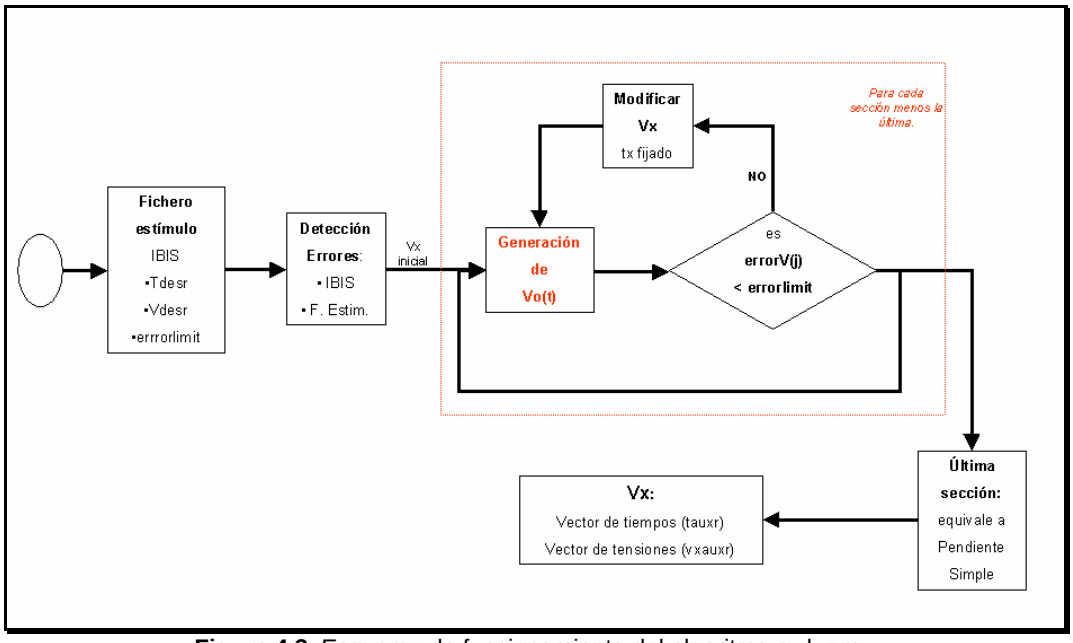

Figura 4.3: Esquema de funcionamiento del algoritmo pwl_r.m.

A continuación son listadas las dos funciones, con gran cantidad de comentarios explicativos:

```
function [vxauxr,tauxr,time] = pwl_r(Vol,Voh,Rlh,Ro,C,tfin,Tdesr,Vdesr,interv,errorlimit)
% MATLAB function used to calculate automatically the Vx's raise slope,
% necessary for the raise characteristic model, when working with a PWL
% source (the PWL source will have, at least, 4 sections). Developed by
% Mario Palma.
% --'Tdesr'and 'Vdesr' are vectors containing information about the desirable output
% times and voltages. This information is obtained from the IBIS file.
% --The first time value is supposed to be zero, and the first voltage
% value is vini=Vdesr(1)=Vol/k; this two values are included in Tdesr and Vdesr.
% If Tdesr or Vdesr does not have increasing values, there are mistakes.
%
% For example:          Tdesr=[0 t2 t3 t4]
%                       Vdesr=[Vol/k V2 V3 V4]
%
% --'tfin' defines the final time of the simulation; must be greater than
% the biggest value of Tdesr.
% --'interv' defines the number of points in each section.
% --'errorlimit' is the maximum error that it is allowed over the desirable points;
% it is not a percentage:
% For instance:   maximum error=5 per cent -----> errorlimit=0.05

    tiempo=clock;               % to control the execution time.
    alfa=1/(Rlh*C);                 % Circuit defined constans.
    k=1+(Rlh/Ro);
    beta=alfa*k;
    leng=length(Tdesr);   % Tdesr and Vdesr length;

    % to correct a usual mistake.
    if (tfin<Tdesr(leng))
        error('tfin must be greater than the biggest value of Tdesr')
    end

    % It is necessary to work between Vol and Voh.
    if(max(Vdesr)>Voh)
        error('It not possible to generate values over Voh.');
    end
    if(min(Vdesr)<Vol)
        error('It not possible to generate values below Vol.');
    end
```

```
% If Tdesr or Vdesr does not have increasing values, there are mistakes.
ayuda=sort(Tdesr);              % ayuda is sorted in ascending order.
if(strcmp(ayuda,Tdesr)==0)  %if Tdesf is not a increasing vector.
        error('Tdesr must be an increasing vector')
end

ayuda=sort(Vdesr);              % ayuda is sorted in ascending order.
if(strcmp(ayuda,Vdesr)==0)  %if Tdesf is not a increasing vector.
        error('Vdesr must be a increasing vector')
end

% time vector creation.
% the PWL source will have, at least, 4 sections.
t=linspace(Tdesr(1),Tdesr(2),interv);
for i=3:(leng-1)
        tauxr=linspace(Tdesr(i-1),Tdesr(i),(interv+1));
        t=[t tauxr(2:interv+1)];
end

vxauxr=Vdesr*2;
vxauxr=[Vol,vxauxr(2:(leng-1)),Voh]; % Initial voltages for Vx voltage.
Vout=Vdesr(1);
errorV=10*ones(1,leng);                 % To enter the loop.

% Now Vx is created to generate the desireable output voltages.
for j=2:(leng-1)
        ground=Vdesr(j);                % Minimun value for vxauxr(j).
        tophelp=0;

        while (abs(errorV(j))>errorlimit)
                if (j==2)       % when working with the first time vector section

                vini=Vdesr(1);
                Vxhelp=linspace(Vol,vxauxr(2),interv);
                t2=linspace(Tdesr(1),Tdesr(2),interv);
                t2=[Tdesr(1) t2];

                for n=1:interv
                        V(n)=((1/beta)*alfa*Vxhelp(n))+
                    exp(-beta*(t2(n+1)-t2(n)))*((-1/beta)*alfa*Vxhelp(n)+vini);
                        vini=V(n);
```

```
        end

        else            % if j>2
        vini=Vout(interv*(j-2));
        Vxhelp=linspace(vxauxr(j-1),vxauxr(j),(interv+1));
        t2=linspace(Tdesr(j-1),Tdesr(j),(interv+1));
        t2=[Tdesr(j-1) t2];

        for n=1:(interv+1)
            V(n)=((1/beta)*alfa*Vxhelp(n))+exp(-beta*(t2(n+1)-t2(n)))*
            ((-1/beta)*alfa*Vxhelp(n)+vini);
            vini=V(n);
        end
        end

% Now the accuracy is controled.
lon=length(V);
errorV(j)=(V(lon)-Vdesr(j))/Vdesr(j);

        if (errorV(j)>errorlimit)               % V(lon) is too big.
            tophelp=1;
            top=vxauxr(j);
            vxauxr(j)=(vxauxr(j)+ground)/2;

        elseif (errorV(j)<-errorlimit & tophelp==0)         % V(lon) is too short.
            ground=vxauxr(j);
            vxauxr(j)=vxauxr(j)*2;

        elseif (errorV(j)<-errorlimit & tophelp==1)         % V(lon) is too short.
            ground=vxauxr(j);
            vxauxr(j)=(vxauxr(j)+top)/2;

        else (abs(errorV(j))<errorlimit)            % Now this section is correct.
            Vout=[Vout V(2:lon)];
        end

% If Vx's slope or the required voltage is too high the problem cannot be solved:
        mist=(vxauxr(j)-vxauxr(j-1))/(Tdesr(j)-Tdesr(j-1));
        if (mist>1e13)
            error('The slope required for Vx is too high; check your IBIS file')
        end
```

```
        end                         % end of the while loop

        if (vxauxr(j)>(Voh+1e-3))
                error('Vx needs a voltage higher than Voh; check your IBIS file')
        end
end                                 % end of the for loop



% From now on the last part of the curves are created.
Vxhelp=[linspace(vxauxr(leng-1),Voh,(interv+1)) ones(1,interv)*Voh];
tam=length(Vxhelp);
tr=Tdesr(leng)-Tdesr(leng-1);       % Initial value for Vxhelp's rise time.
                                    % This is frequently too big.
errorV=-10;                         % Initial values so that the program can enter the
errorT=-10;                         % loops.
aux0=0;
aux1=tfin-Tdesr(leng-1);            % Biggest value for tr.
aux2=0;                             % Smallest value for tr.
errlimitT=0.06;                     % It is better to limit the error so as not to have
                                    % convergency problems with the algoritm.

% The program is in the loop while any error function is bigger than errorlimit.
while (abs(errorV)>errorlimit | abs(errorT)>errlimitT)
        vini=Vout(interv*(leng-2));     % Initial output voltage
        a=linspace((Tdesr(leng-1)+tr),tfin,(interv+1));
        t2=[linspace(Tdesr(leng-1),(Tdesr(leng-1)+tr),(interv+1)) a(2:(interv+1))];
        t3=[Tdesr(leng-1) t2];

% Output voltage creation.
        for n=1:tam
                V(n)=((1/beta)*alfa*Vxhelp(n))+exp(-beta*(t3(n+1)-t3(n)))*
                ((-1/beta)*alfa*Vxhelp(n)+vini);
                vini=V(n);
        end
        Vres=interp1(t2,V,Tdesr(leng));         % Interpolation to get the output voltage
                                                % when the model is in the ideal time.
  errorV=(Vres-Vdesr(leng))/(Vdesr(leng));      % Definition of the voltage error function.

        if (errorV<-errorlimit & aux0==0)       % Rise time is still too high.
```

```
            aux1=tr;
            tr=tr/5;

elseif (errorV<-errorlimit & aux0==1)    % tr is now boundered. Rise time is still
                                         % too high.
            aux1=tr;
            tr=(tr+aux2)/2;

elseif (errorV>errorlimit)               % sometimes tr is bigger than
                                         % Tdesr(leng)-Tdesr(leng-1)
            aux0=1;
            aux2=tr;
            tr=(tr+aux1)/2;

else                                     % Voltage error is already controlled.

    fin=V(tam);
    q=find(V >=(floor(fin*100)/100));
    k=q(1);
    tres=t2(k);
errorT=(tres-Tdesr(leng))/Tdesr(leng);   % Definition of the time error function.

    if errorT<-errlimitT                 % Rise time is too high.
        aux2=tr;
        tr=(tr+aux1)/2;
    end

    if errorT>errlimitT                  % Rise time is too low.
        aux0=1;
        aux1=tr;
        tr=(tr+aux2)/2;
    end
end

if (tr<1e-20)
    error('Vx´s rise time (tr) is shorter than 1*e-20 seconds; check your IBIS file.')
end

if (((tfin-Tdesr(leng-1))-tr)<5e-10)     % If it is needed a bigger tf and tfin is not big
    errorlimit=10;                       % enough, a warning message is displayed
                                         % and this % tr is considered the right one.
```

```
            disp('Warning: On the last desirable point the curve may have an error bigger
            than `errorlimit´. Increase tfin')
        end
end

t=[t t2(2:tam)];                % Definitive time and output voltage vectors.
Vout=[Vout V(2:tam)];

vxauxr=[vxauxr Voh];                        % Creation of Vx.
leng=length(vxauxr);
Vx=linspace(vxauxr(1),vxauxr(2),interv);
for i=3:leng
        tauxr=linspace(vxauxr(i-1),vxauxr(i),(interv+1));
        Vx=[Vx tauxr(2:interv+1)];
end
tauxr=[Tdesr(1:(leng-2)) (Tdesr(leng-2)+tr) tfin];


plot(t,Vx,'--',t,Vout); grid on; zoom;          % Vx and output voltage are plotted.
xlabel('time'); ylabel('voltage');
time=etime(clock,tiempo);           % It is possible to control the
                                            % execution time.
end;
```

```
function [vxauxf,tauxf,time] = pwl_f(Vol,Voh,Rhl,Ro,C,tfin,Tdesf,Vdesf,interv,errorlimit)
% MATLAB function used to calculate automatically the Vx's falling slope,
% when working with a PWL source (the PWL source will have, at least, 4 sections).
% Developed by Mario Palma.
% --'Tdesf', 'Vdesf' are vectors containing information about the desirable output
% times and voltages. This information is obtained from the IBIS file.
% --The first time value is supposed to be zero, and the first voltage
% value is vini=Vdesf(1)=Voh; this two values are included in Tdesf and Vdesf.
% If Tdesf does not have increasing values or Vdesf does not have
% decreasing values, there are mistakes.
%
% For example:          Tdesf=[0 t2 t3 t4]
%                       Vdesf=[Voh V2 V3 V4]
%
% --'tfin' defines the final time of the simulation; must be greater than
% the biggest value of Tdesf.
% --'interv' defines the number of points in each section.
% --'errorlimit' is the maximum error that it is allowed over the desirable points;
% it is not a percentage:
% For instance:     maximum error=5 per cent -----> errorlimit=0.05.
    tiempo=clock;               % to control the execution time.
    tau=1/(Rhl*C);                  %Circuit defined constans.
    ka=1/(Ro*C);
    omega=ka*(1+Ro/Rhl);
    leng=length(Tdesf);   % Tdesf and Vdesf length;

    % to correct a usual mistake.
    if (tfin<Tdesf(leng))
        error('tfin must be greater than the biggest value of Tdesf')
    end


    % It is necessary to work between Vol and Voh.
    if(max(Vdesf)>Voh)
        error('It not possible to generate values over Voh.');
    end
    if(min(Vdesf)<Vol)
        error('It not possible to generate values below Vol.');
    end
```

```
% If Tdesf does not have increasing values or Vdesf does not have
% decreasing values, there are mistakes.

ayuda=sort(Tdesf);          % ayuda is sorted in ascending order.
if(strcmp(ayuda,Tdesf)==0)  %if Tdesf is not a increasing vector.
        error('Tdesf must be an increasing vector')
end

ayuda=sort(Vdesf);
ayuda=fliplr(ayuda);        % ayuda is sorted in descending order.
if(strcmp(ayuda,Vdesf)==0)  %if Tdesf is not a increasing vector.
        error('Vdesf must be a decreasing vector')
end

% time vector creation.
% the PWL source will have, at least, 4 sections.
t=linspace(Tdesf(1),Tdesf(2),interv);
for i=3:(leng-1)
        tauxf=linspace(Tdesf(i-1),Tdesf(i),(interv+1));
        t=[t tauxf(2:interv+1)];
end

vxauxf=Vdesf/2;
vxauxf=[Voh,vxauxf(2:(leng-1)),Vol]; % Initial voltages for Vx voltage.
Vout=Vdesf(1);
errorV=10*ones(1,leng);             % To enter the loop.

% Now Vx is created to generate the desireable output voltages.
for j=2:(leng-1)
        top=Vdesf(j);       % Maximum value for vxauxf(j).
        floorhelp=0;

        while (abs(errorV(j))>errorlimit)


                if (j==2)       % when working with the first time vector section
                vini=Vdesf(1);
                Vxhelp=linspace(Voh,vxauxf(2),interv);
                t2=linspace(Tdesf(1),Tdesf(2),interv);
                t2=[Tdesf(1) t2];
```

```
        for n=1:interv
V(n)=1/omega*ka*Voh+1/omega*tau*Vxhelp(n)+exp(-omega*(t2(n+1)-t2(n)))*
(-1/omega*ka*Voh-1/omega*tau*Vxhelp(n)+vini);
                vini=V(n);
        end

        else            % if j>2
        vini=Vout(interv*(j-2));
        Vxhelp=linspace(vxauxf(j-1),vxauxf(j),(interv+1));
        t2=linspace(Tdesf(j-1),Tdesf(j),(interv+1));
        t2=[Tdesf(j-1) t2];

        for n=1:(interv+1)
V(n)=1/omega*ka*Voh+1/omega*tau*Vxhelp(n)+exp(-omega*(t2(n+1)-t2(n)))*
(-1/omega*ka*Voh-1/omega*tau*Vxhelp(n)+vini);
                vini=V(n);
        end
        end

% Now the accuracy is controled.
        lon=length(V);
        errorV(j)=(V(lon)-Vdesf(j))/Vdesf(j);

        if (errorV(j)<-errorlimit)              % V(lon) is too big.
                floorhelp=1;
                ground=vxauxf(j);
                vxauxf(j)=(vxauxf(j)+top)/2;

        elseif (errorV(j)>errorlimit & floorhelp==0)        % V(lon) is too short.
                top=vxauxf(j);
                vxauxf(j)=vxauxf(j)/2;

        elseif (errorV(j)>errorlimit & floorhelp==1)        % V(lon) is too short.
                top=vxauxf(j);
                vxauxf(j)=(vxauxf(j)+ground)/2;

        else (abs(errorV(j))<errorlimit)            % Now this section is correct.
                Vout=[Vout V(2:lon)];
        end

% If Vx's slope or the required voltage is too low the problem cannot be solved:
```

```
        mist=abs((vxauxf(j)-vxauxf(j-1))/(Tdesf(j)-Tdesf(j-1)));
        if (mist>1e13)
            error('The slope required for Vx is too low; check your IBIS file')
        end

    % When some vxauxf(j) (if j is not leng) takes a value very near to zero, it is needed
    % a value lower than zero volts. It is considered an mistake (The lowest Vol value
    % is supposed to be zero):
        if(vxauxf(j)<1e-3)
            error('Vx needs a voltage lower than zero; check your IBIS file')
        end
    end                         % end of the while loop


        if (vxauxf(j)<(Vol-1e-3))
            error('Vx needs a voltage lower than Voh; check your IBIS file')
        end

end                             % end of the for loop


% From now on the last part of the curves are created.

Vxhelp=[linspace(vxauxf(leng-1),Vol,(interv+1)) ones(1,interv)*Vol];
tam=length(Vxhelp);
tf=Tdesf(leng)-Tdesf(leng-1);       % Initial value for Vxhelp's fall time.
                            % This is frecuently too big.
errorV=-10;                 % Initial values so that the program can enter the
errorT=-10;                 % loops.
aux0=0;
aux1=tfin-Tdesf(leng-1);    % This is the biggest value for tf.
aux2=0;                         % This is the smallest value for tf.
errlimitT=0.06;                                                     .

% The program is in the loop while any error function is bigger than its error limit.
while (abs(errorV)>errorlimit | abs(errorT)>errlimitT)

    vini=Vout(interv*(leng-2));     % Initial output voltage
    a=linspace((Tdesf(leng-1)+tf),tfin,(interv+1));
    t2=[linspace(Tdesf(leng-1),(Tdesf(leng-1)+tf),(interv+1)) a(2:(interv+1))];
    t3=[Tdesf(leng-1) t2];
```

```
% Output voltage creation.
    for n=1:tam
        V(n)=1/omega*ka*Voh+1/omega*tau*Vxhelp(n)+exp(-omega*(t3(n+1)-t3(n)))*
        (-1/omega*ka*Voh-1/omega*tau*Vxhelp(n)+vini);
        vini=V(n);
    end

    Vres=interp1(t2,V,Tdesf(leng));          % Interpolation to get the output voltage
                                             % when the model is in the ideal time.
    errorV=(Vres-Vdesf(leng))/(Vdesf(leng)); % Definition of the voltage error function.

    if (errorV>errorlimit & aux0==0)         % Fall time is still too high.
            aux1=tf;
            tf=tf/5;

    elseif (errorV>errorlimit & aux0==1)% tf is now boundered. Fall time is still too high.
            aux1=tf;
            tf=(tf+aux2)/2;

    elseif (errorV<-errorlimit)              % sometimes tf is bigger than
                                             % Tdesf(leng)-Tdesf(leng-1)
            aux0=1;
            aux2=tf;
            tf=(tf+aux1)/2;

    else
            fin=V(tam);
            q=find(V <=(ceil(fin*100)/100));
            k=q(1);
            tres=t2(k);
            errorT=(tres-Tdesf(leng))/Tdesf(leng);   % Definition of the time
                                                     %error function.

        if errorT<-errlimitT                 % Fall time is too high.
            aux0=1;
            aux2=tf;
            tf=(tf+aux1)/2;
        end

        if errorT>errlimitT                  % Fall time is too low.
```

```
                aux1=tf;
                tf=(tf+aux2)/2;
            end
        end

        if (tf<1e-20)
            error('Vx´s fall time (tf) is shorter than 1*e-20 seconds; check your IBIS file.')
        end

        if (((tfin-Tdesf(leng-1))-tf)<5e-10)    % If it is needed a bigger tf and tfin is not big
            errorlimit=10;          % enough, a warning message is displayed and this
                                    % tf is considered the right one.

disp('Warning: On the last desirable point the curve may have an error bigger than `errorlimit´.
Increase tfin')
        end
    end

    t=[t t2(2:tam)];                % Definitive time and output voltage vectors.
    Vout=[Vout V(2:tam)];

    vxauxf=[vxauxf Vol];                    % Creation of Vx.
    leng=length(vxauxf);
    Vx=linspace(vxauxf(1),vxauxf(2),interv);
    for i=3:leng
        tauxf=linspace(vxauxf(i-1),vxauxf(i),(interv+1));
        Vx=[Vx tauxf(2:interv+1)];
    end
    tauxf=[Tdesf(1:(leng-2)) (Tdesf(leng-2)+tf) tfin];

    plot(t,Vx,'--',t,Vout); grid on; zoom;      % Vx and output voltage are plotted.
    xlabel('time'); ylabel('voltage');

        time=etime(clock,tiempo);       % It is possible to control the
                                        % execution time.
    end
```

4.5. Ejemplo: resultados con el modelo DR-2.

Las siguientes curvas han sido generadas trabajando con 10 puntos deseados a la salida, seleccionados de [Rising Waveform] y [Falling Waveform] (**apéndice 1**); se puede comprobar la calidad de las salidas mostradas comparando con los datos del modelo real.

El tiempo de ejecución para estos ejemplos concretos es de 1.37 segundos para la característica de subida y de 1.81 segundos para la de bajada, trabajando con 50 puntos en cada sección. El fichero de tecnología para este modelo que se emplea en MATLAB se lista en el **apéndice 2.**

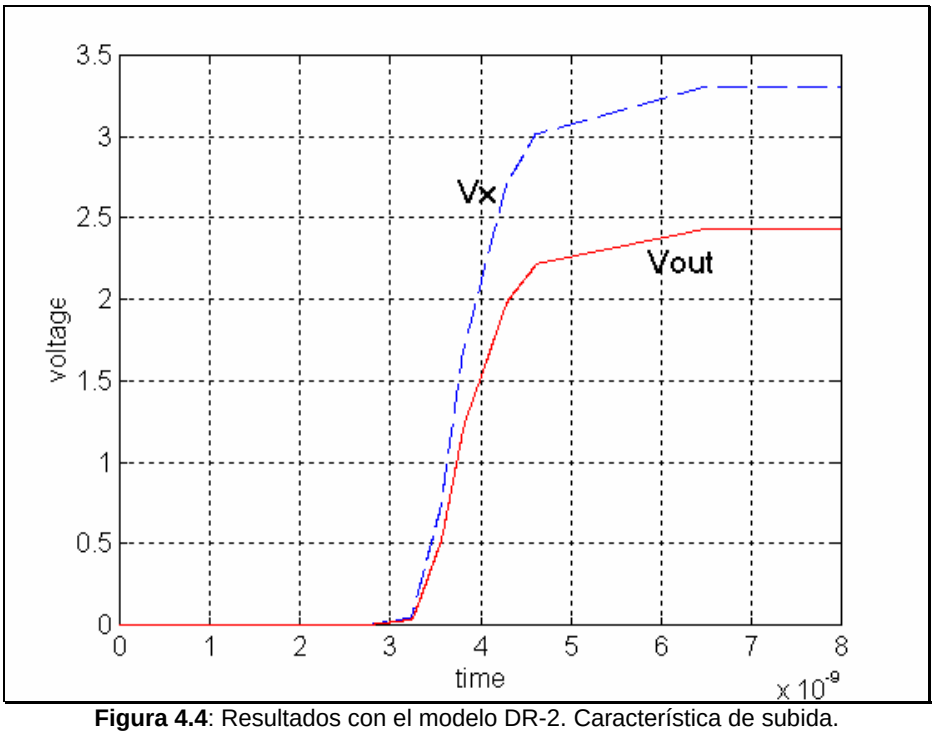

Figura 4.4: Resultados con el modelo DR-2. Característica de subida.

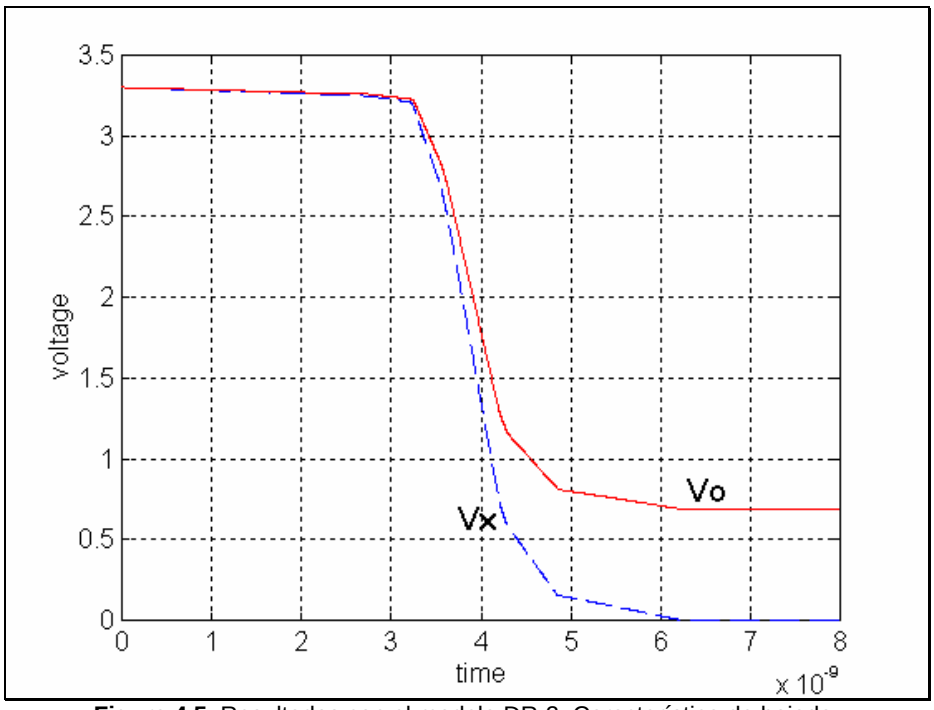

Figura 4.5: Resultados con el modelo DR-2. Característica de bajada.