```
%FIRVSB Vestigial sideband (VSB) filter design from raised-cosine prototype.
%
%  B=FIRVSB(N,alpha,Fs,W) returns a length N+1 FIR complex-baseband VSB
%  filter from a truncated raised-cosine filter with rolloff parameter
%  0 <= alpha <= 1, where N must be even.  The filter supports
%  single-sided message bandwidth W and sampling frequency Fs >= 8*W,
%  where Fs/W must be an integer.  For good results, use N >= 2*Fs/W.
%
%  B=FIRVSB(N,alpha,Fs,W,fc) operates similarly, but returns a linear-phase
%  passband VSB filter for carrier frequency fc such that W <= fc <= Fs/2-3*W


function b=firvsb(Lt,alpha,Fs,W,fc)

plot_on = 0;

if nargin==0
  plot_on = 1;                          % set=1 for plots
  W = 2;                        % single-sided signal BW
  Fs = 32;                              % sampling period
  fc = Fs/8+0.1;                % carrier frequency
  Lt = 20;                              % filter order (even)
  alpha = 1.0;                          % RC parameter
end;

if floor(Fs/W)~=Fs/W,
  error('need Fs/W to be an integer!');
elseif Fs/W<8,
  error('need Fs/W >= 8!');
elseif floor(Lt/2)~=Lt/2,
  error('need even Lt')
end;
```

```
Nt = Lt+1;                                  % filter length
Ts = 1/Fs;                                  % sampling interval
P = 1/(2*Ts*W);                             % effective oversampling ratio
%T = 3;                                     % # symbols
%tb = linspace(-T,T,2*T*P+1);               % baud-normalized times
tb = [-(Nt-1)/2:(Nt-1)/2]/P;                % baud-normalized times
t = tb*P*Ts;                                % times
rc = cos(pi*alpha*tb)./(1-(2*alpha*tb).^2).*sinc(tb);
ii = ~isfinite(rc);
rc(ii) = pi*alpha*sin(pi*alpha*tb(ii))./(8*alpha^2*tb(ii)).*sinc(tb(ii));
vsb_bb = (4*Ts*W)*rc.*exp(j*2*pi*t*W);              % baseband vsb filter
if exist('fc'),
  vsb_pb = 2*real(vsb_bb.*exp(j*2*pi*t*fc));    % passband vsb filter
end;

if nargin==4,
  b = vsb_bb;                               % output a baseband filter
elseif nargin==5,
  b = vsb_pb;                               % output a passband filter
  if (fc<W)||(fc>1/2/Ts-3*W), error('need W <= fc <= Fs/2-3*W!'); end;
elseif nargin>5,
  error('too many inputs!')
end;

if plot_on,
subplot(411)
plot(1/Ts*[-2*Nt:2*Nt-1]/4/Nt,fftshift(abs(fft(rc,4*Nt))))
axe=axis; axis([-1/2/Ts,1/2/Ts,axe(3:4)])
ylabel('raised cosine')
xlabel('frequency')
subplot(412)
plot(1/Ts*[-2*Nt:2*Nt-1]/4/Nt,fftshift(abs(fft(vsb_bb,4*Nt))))
axe=axis; axis([-1/2/Ts,1/2/Ts,axe(3:4)])
```

```
hold on; plot(0*[1,1],axe(3:4),':'); hold off;
ylabel('vsb-bb')
xlabel('frequency')
subplot(413)
plot(1/Ts*[-2*Nt:2*Nt-1]/4/Nt,fftshift(abs(fft(vsb_pb,4*Nt))))
axe=axis; axis([-1/2/Ts,1/2/Ts,axe(3:4)])
hold on; plot(fc*[1,1],axe(3:4),':',-fc*[1,1],axe(3:4),':'); hold off;
ylabel('vsb-pb')
xlabel('frequency')
subplot(414)
vsb_test = vsb_pb.*cos(2*pi*fc*t);
plot(1/Ts*[-2*Nt:2*Nt-1]/4/Nt,fftshift(abs(fft(vsb_test,4*Nt))))
axe=axis; axis([-W,W,axe(3:4)])
hold on; plot([-W,W],[2,2],':'); hold off;
ylabel('test vsb-pb')
xlabel('frequency')
end;
```