```
% HdeconV3.m
% Version 3.0 written by J.R. Andrews, KH6HTV, 7 Sept. 2009
% added Noise Floor Filter option
% changed freq. plots to semilog
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Version 2.0, PICOSECOND PULSE LABS, written by J.R.Andrews, 10/10/04
% modified from ImpDecon.m, Version 1.0, PSPL, J.R.Andrews, 30 nov 2002
% modified graphing & filtering options, 7/6/04
% filter impulse & step response & filtered H(f) plotting, 7/12/04
% output file writing to disc, 7/14/04
% added Nahman-Guillaume Optimal Filter, 7/27/04
% added auto gamma find, 8/16/04, 8/20/04
% enhance graphics & data output 8/20/04
% major modification in treatment of step-like pulse waveforms.
% incorporated Riad's [3] combo, complete FFT of steps using Nicholson's
% ramp subtraction [4] and Gan's square wave [5].   9/7/04
% added selection of data in/out via either A: or C: drives.  9/8/04
% added optional baseline correction  9/9/04
% added Bennia-Nahman optimal, min. phase filter, [10], 9-30 thru 10/10/04
%
%%%%%%%%%%%%%%%%%%%%%%%%%%
% Technical References:
% [1] J.R.Andrews & M.G.Arthur, "SPECTRUM AMPLITUDE --- Definition,
% Generation and Measurement", NBS Tech.Note 699, NBS, Boulder, Colo.USA,
% Oct. 1977, 92 pages.
% [2] W.L.Gans & J.R.Andrews, "Time Domain Automatic Network Analyzer for
% Measurement of RF & Microwave Components", NBS Tech.Note 672, NBS,
% Boulder, Colo.USA, Sept.1975, 165 pages.  See chp. 3 for square wave
% treatment of step-like pulses
% [3] A.Shaarawi & S.Riad, "Computing the Complete FFT of a Step-Like
% Waveform", IEEE Trans. Inst.& Meas., vol IM-35, no.1, pp.91-92,March 1986
% [4] A.M.Nicolson, "Forming the fast Fourier transform of a step response
% in time-domain metrology", Electron.Lett., vol.9,pp.317-318, July 1973
% [5] W.L.Gans & N.S.Nahman, "Continuous and discrete Fourier tranform of
% step-like waveforms", IEEE Trans. Inst. & Meas., vol.IM-31, pp.97-101,
% June 1982
% [6] J.Waldmeyer, "Fast Fourier transform for step-like functions: The
% synthesis of three apparently different methods", IEEE Trans. Inst. &
% Meas., vol, IM-29, pp.36-39, March 1980
% [7] N.S.Nahman, "Software Correction of Measured Pulse Data"
% Fast Electrical & Optical Measurements", Vol.I, pp.351-417, 1986
% NATO ASI Series, Martin Nijhoff Publishers
% [8]  N.S.Nahman & M.Guillaume, "Deconvolution of Time Domain Waveforms in
the
% Presence of Noise", NBS Tech.Note 1047, NBS, Boulder, Colo Oct.1981
% [9]  also see Martin VanPelt's PSPL program OPTFILT.m, 1989
```

```
% [10] A.Bennia & N.S.Nahman, "Deconvolution of Causal Pulse & Transient
% Data",  IEEE Trans. I&M, vo. 39, no.4, Dec. 1990, pp.933-939
% [11]  Alan V. Oppenheim & Ronald W. Schafer, "DIGITAL SIGNAL PROCESSING",
% (book) Prentice-Hall, Englewood Cliffs, NJ, 1975, 585 pages
% [12] J.R.Andrews, "Deconvolution of System Impulse Responses & Time
Domain
% Waveforms", Picosecond Pulse Labs, Boulder, CO, Application Note, AN-18,
% Oct. 2004
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
disp('  ')
disp('HdeconV3.m --- MatLab Program')
disp('Version 3.0, 7 Sept. 2009')
disp('written by J.R. Andrews, KH6HTV')
disp('Picosecond Pulse Labs, Boulder, Colo, USA')
disp('Deconvolution of h(t) impulse response from vin(t) & vout(t)')
disp('Data comes from an oscilloscope as TDT .txt input files')
disp('Incident test signal, vin(t), is either an impulse or step')
disp('Time window should be large enough to include entire transient')
disp('Program calculates h(t) = ifft(fft(vout(t))/fft(vin(t))')
disp('Program includes selectable low-pass filters')
disp('Filters include Rectangular, Gaussian (order 1-5),')
disp('Nahman-Guillaume Optimal, Bennia-Nahman Optimal, Min. Phase')
disp('and Noise Floor')
disp('Program gives option of saving results in .txt format')
disp('  ')
clear
warning off
disp('Please use the keyboard to respond to the following questions.')
drive = input('Data in/out via drive A: or C:?  (1=A: 3=C:)   ');
if drive == 1
    disp('Data in/out will be as .txt files via floppy disc in drive A:')
end
if drive == 3
    disp('Data in/out will be as .txt files in current C: drive directory')
end
disp(' ')
type = input('Type of test signal? 1 = step, 2 = impulse:  ');
Tw = input('Enter Time Window in NanoSeconds (ns)?  ');
disp('NOTE: Baseline Correction is strongly encouraged for enhanced
accuracy')
baseline = input('Do you want to do baseline correction?  (1=yes  0=no)
');
disp('Please input oscilloscope data files')
% note: This works with floppy disc data files written by the HP-54750
% oscilloscope.  Verbose data headers need to be first stripped off. The HP
% o'scope data is organized as one voltage data point per line.
```

```
fname1 = input('Enter Input Waveform Data File Name:   ','s');
dname1 = [fname1, '.txt'];
if drive == 1
    dname1 = ['A:', fname1, '.txt'];
end
fname2 = input('Enter Output Waveform Data File Name:   ','s');
dname2 = [fname2, '.txt'];
if drive == 1
    dname2 = ['A:', fname2, '.txt'];
end
if baseline == 1
    fname3 = input('Enter Input Baseline Data File Name:   ','s');
    dname3 = [fname3, '.txt'];
    if drive == 1
        dname3 = ['A:', fname3, '.txt'];
    end
    fname4 = input('Enter Output Baseline Data File Name:   ','s');
    dname4 = [fname4, '.txt'];
    if drive == 1
        dname4 = ['A:', fname4, '.txt'];
    end
end
disp('loading data from drive A:')
vin = load(dname1);
vout = load(dname2);
if baseline == 1
    vblin = load(dname3);
    vblout = load(dname4);
% subtract baselines from  data to enhance accuracy
    vin = vin - vblin;
    vout = vout - vblout;
end
N = length(vin);  % = # of data points
dt=Tw/N;  %(in ns)
f0=1/Tw;  %(in GHz)
fny=0.5/dt;  %(in GHz)
for i=1:N
    t(i)=i*dt;
    f(i) = (i-1)*f0;
end
tin = t;
plot(t,vin,'b',t,vout,'g')
grid
xlabel('time in ns')
ylabel('Volts')
title('Input (blue) & Output (green) Waveforms from Oscilloscope')
```

```
disp('After each graph, press any key to continue')
pause  % pause command allows program to stop running until key is pressed
disp('Number of data points is')
disp(N)
disp('Delta T time spacing in ns is')
disp(dt)
disp('Nyquest folding frequeny, fny, in GHz is')
disp(fny)
% use FFTs to calc freq. domain Fourier transforms from time domain signals
% NOTE:  FFT arrays range from i = 1 to N   FFT(1) is DC value, FFT(2) is
% first harmonic at f0, FFT(3) is second harmonic at 2*f0, etc. up to
% FFT(N/2+1) which is the Nyquist Folding Frequency, fny. Beyond N/2+1 are
% the negative frequency components running from -fny back to -f0 at FFT(N)
%
if baseline == 1
% Determine spectrum of baselines
    Vblin = fft(vblin);
    Vblin = Vblin/N;
    Vblout = fft(vblout);
    Vblout = Vblout/N;
% for plotting purposes, discard DC, Nyquist & neg. freqs (i.e. >N/2)
    for i=1:(N/2-1)
        fpbl(i) = i*f0;
        SAdbVblin(i) = 20*log10((2*Tw*1000)*abs(Vblin(i+1)));
        SAdbVblout(i) = 20*log10((2*Tw*1000)*abs(Vblout(i+1)));
    end
end
%
if type == 2  % i.e. impulse waveforms -- use normal processing for FFTs
    % note -- need to divide fft by N to get proper scaling in volts
    Vin = fft(vin);
    Vin = Vin/N;
    Vout = fft(vout);
    Vout = Vout/N;
    disp('Lowest frequency, fo, and freq. spacing, df, in GHz is')
    disp(f0)
end
if type == 1  % i.e. step-like waveform -- special processing required
%   Create new waveform, vsr, by subtracting Nicolson [4] linear ramp from
%   step-like pulse waveform.
    for i=1:N
        vrampin(i) = vin(1) + (i-1)*(vin(N)-vin(1))/(N-1);
        vsrin(i) = vin(i) - vrampin(i);
        vrampout(i) = vout(1) + (i-1)*(vout(N)-vout(1))/(N-1);
        vsrout(i) = vout(i) - vrampout(i);
    end
```

```
%  Convert step pulse of N pts. to a Gans' [5] square wave, vsq, of 2N pts.
    vsqin = vin;
    vsqout = vout;
    for i= 1:N
        vsqin(i+N) = (vin(1)+vin(N)) - vin(i);
        vsqout(i+N) = (vout(1)+vout(N)) - vout(i);
    end
    Nsq=2*N;
    Twsq=2*Tw;
    f0sq=1/Twsq; %(in GHz)
    for i=1:Nsq
        tsq(i)=i*dt;
    end
%   calculate the FFTs
% note: for proper scaling of periodic waveforms, need to divide FFT by N
% to get correct V(f) in units of Volts
    Vsrin = fft(vsrin);
    Vsrin = Vsrin/N;
    Vsqin = fft(vsqin);
    Vsqin = Vsqin/Nsq;
    Vsrout = fft(vsrout);
    Vsrout = Vsrout/N;
    Vsqout = fft(vsqout);
    Vsqout = Vsqout/Nsq;
% note: f0 of ramped waveform is 1/Tw.  All harmonics of f0 are present up
% to Nyquist freq, fny = 1/2*dt.  The dc value is not valid.
% For sq.wave waveform the fundamental is 1/2Tw and only the odd harmonics
% are present up to the Nyquist freq, fny.  The even harmonics of 1/2Tw are
% zero due to symmetry.  The dc value is valid.  The frequency lines of the
% ramped waveform and the square wave are interleaved and are all valid.
%
% Mesh Vsr & Vsq arrays into single array V. See Shaarawi & Riad [3]
    Vin = Vsqin;
    Vout = Vsqout;
    for i=3:2:Nsq-1
        Vin(i) = Vsrin((i+1)/2);  % i.e. insert ramp spec. into nulls of
sq.spec.
        Vout(i) = Vsrout((i+1)/2);
    end
    N = Nsq;
    f0 = f0sq;
    disp('Lowest frequency, fo, and freq. spacing, df, in GHz is')
    disp(f0)
end   % end of special processing for step-like pulses
%
% for plotting purposes, discard DC, Nyquist & neg. freqs (i.e. >N/2)
```

```
for i=1:(N/2-1)
    fp(i) = i*f0;
    SAdbVin(i) = 20*log10((2*Tw*1000)*abs(Vin(i+1)));
    SAdbVout(i) = 20*log10((2*Tw*1000)*abs(Vout(i+1)));
end
if baseline == 1
    semilogx
(fp,SAdbVin,'b',fp,SAdbVout,'g',fpbl,SAdbVblin,'r',fpbl,SAdbVblout,'c')
    grid
    xlabel('Frequency in GHz')
    ylabel('Spectrum Amplitude in dBuV/MHz')
    title('ffts Vin(blue), Vout(green), BLin(red) & BLout(cyan) vs. Freq.')
end
if baseline == 0
    semilogx(fp,SAdbVin,'b',fp,SAdbVout,'g')
    grid
    xlabel('Frequency in GHz')
    ylabel('Spectrum Amplitude in dBuV/MHz')
    title('ffts of Vin(blue) & Vout(green) vs. Freq.')
end
pause
% CALC TRANSFER FUNCTION, H(f)  (note: also called S21(f)  )
H = Vout ./ Vin;  % note ./ necessary to force single element division
for i=1:(N/2-1)
Hmagp(i) = 20*log10(abs(H(i+1)));
end
semilogx(fp,Hmagp)
grid
xlabel('Frequency in GHz')
ylabel('dB')
title('raw, unfiltered, transfer function H(f) in dB vs. Freq.')
pause
% FILTER SECTION
disp('Now apply a low-pass filter to noisy H(f)')
disp('0 = no filter, 1 = Rect, 2 = Gauss')
disp('3 = Nahman/Guillaume Optimal, 4 = Bennia/Nahman min.phase')
disp('5 = Noise Floor Filter')
filtype = input('Enter type of filter?   ');
F(1:N) = 1; % basic no filter
end
if filtype == 5 % Noise Floor Filter, jra, 9/4/09
% extracted from JitterDeconV1.m program
% no filtering is applied up to limit where either Vout(f) or Vin(f)
% first reaches its respective noise floor.  Beyond there we retain
% Vout(f)'s noise floor
% analyze noise floor of Vout & Vin for upper 60% of spectrum
```

```
% note: necessary for the upper part of the spectrum to be in the noise
% floor
disp('Noise Floor filter stops applying decon at lowest frequency')
disp('at which either Vout or Vin reaches its noise floor max')
disp('Do you want to manually select the noise floor')
nfopt = input('cut-off frequency ?  (1=yes, 0=no)');
if nfopt == 1
    nfcutf = input('enter cut-off freq. in GHz =  ');
    nfcutfN = round(1 + nfcutf/f0);
end
if nfopt == 0
fnfa = (round(N/5) + 1)*f0;
disp('Vout(f) Noise Floor analyzed from f(nyquist) down to GHz ')
disp(fnfa)
for i= round(N/5) : round(N/2)-1
    k = i -round(N/5) +1;
SAnf(k) = SAdbVout(i);
end
voutnfdb = mean(SAnf);
voutnfdbmax = max(SAnf);
% now find freq where Vout(f) first drops to noise floor max value
fpn = 1;
while SAdbVout(1:fpn) > voutnfdbmax , fpn = fpn+1; end
fvoutnf = fpn*f0;
disp('Mean Vout(f) Noise Floor in dB is')
disp(voutnfdb)
disp('Vout(f) reaches Noise Floor max at frequency in GHz')
disp(fvoutnf)
%
disp('Vin(f) Noise Floor analyzed from f(nyquist) down to GHz ')
disp(fnfa)
for i= round(N/5) : round(N/2)-1
    k = i -round(N/5) +1;
    Vinnf(k) = SAdbVin(i);
end
Vinnfdb = mean(Vinnf);
Vinnfdbmax = max(Vinnf);
% now find freq where H(f) first drops to noise floor max value
fpnVin = 1;
while SAdbVin(1:fpnVin) > Vinnfdbmax , fpnVin = fpnVin+1; end
fVinnf = fpnVin*f0;
disp('Mean Vin(f) Noise Floor in dB is')
disp(Vinnfdb)
disp('Vin(f) reaches Noise Floor max at frequency in GHz')
disp(fVinnf)
nfcutfN = fpn';
```

```
if fpnVin < fpn, nfcutfN = fpnVin; end
end
end
if filtype ~= 5 % i.e. proceed with all filters but Noise Floor
if filtype == 0 % i.e. no filter
    F(1:N) = 1;
end
% Rectangular Filter
if filtype == 1 % Rectangular Filter
    BW = input('Enter Filter -3dB Bandwidth in GHz ?  ');
    Nco = round(BW/f0);
    F(1:N) = 0;
    for i=1:Nco+1  % sets rect. low-pass filter
    F(i) = 1;
    end
end
if filtype == 2  % Gaussian Filter
    disp('Filter = exp(-0.5*[f/fc]^m )')
    disp('Normal Gaussian filter is order m = 2')
    m = input('Enter order (1-5) of Gaussian filter?  ');
    m = round(m);
    BW = input('Enter Filter -3dB Bandwidth in GHz ?  ');
    if m == 1, fc=1.442695041*BW;, end
    if m == 2, fc=1.201122409*BW;, end
    if m == 3, fc=1.129947276*BW;, end
    if m == 4, fc=1.095957302*BW;, end
    if m == 5, fc=1.076056085*BW;, end
    for i=1:N/2+1
        F(i) = exp(-0.5*((((i-1)*f0)/fc)^m));
    end
end
if filtype == 3  % i.e. Nahman-Guillaume Optimal Filter
    c2 = 16*(sin(pi*(0:N-1)/N)).^4;
    c2 = c2';
    X2 = (abs(Vin)).^2;
    autog = input('Do you want program to search for best Gamma? (1=yes,
0=no)');
    if autog == 0
        gamadb = input('Enter Nahman-Guillaume Filter Gamma in dB?   ');
    end
    if autog == 1  % i.e. do best gamma search
        dbhi = input('enter hi search limit in db  ');
        dblo = input('enter lo search limit in db  ');
        ddb = (dbhi - dblo)/500;
        for i = 1:501
            gamadb(i) = dblo + (i-1)*ddb;
```

```
            gama = 10^(gamadb(i)/20);
            F = X2 ./ (X2 + (gama*c2));  % This is the Nahman-Guillaume
Filter
            % apply the filter to H(w)
            HNF = H .* F;
            % compute error in imag. part of h(t)
            hnf = ifft(HNF);
            % note: in the ideal situation there would be nothing present
            % in the imag. part of 'hnf', the time domain solution.  Thus
            % anything present in the imag(hnf) is to be considered an
            % error.  The optimum solution for the Nahman-Guillaume filter
is to
            % select a gamma which minimizes the Std.Dev. of the residuals
            % left in the imag(hnf). Minimizing either rms or std.dev.
gives
            % same results.
            agErr(i) = std(imag(hnf));
        end
        [ErrMin,Imin] = min(agErr);
        minErrDB = dblo + (Imin-1)*ddb;
        disp('Nahman-Guillaume Filter')
        disp('Best Gamma (in dB) for min. error is   ')
        disp(minErrDB)
        plot(gamadb,agErr)
        grid
        xlabel('Gamma in dB')
        ylabel('Std.Dev')
        title('Std.Dev. of imag(h(t) vs. Gamma, Nahman-Guillaume Filter')
        pause
        clear gamadb
        gamadb = minErrDB;
    end
    gama = 10^(gamadb/20);
    F = X2 ./ (X2 + (gama*c2));  % This is the Nahman-Guillaume Filter
end
% end of Nahman-Guillaume Filter section
%
% Bennia/Nahman min. phase Filter
if filtype == 4  % i.e. Bennia/Nahman min. phase Filter
    c2 = 16*(sin(pi*(0:N-1)/N)).^4;
    c2 = c2';
    X2 = (abs(Vin)).^2;
    autog = input('Do you want program to search for best Gamma? (1=yes,
0=no)');
    if autog == 0
        gamadb = input('Enter Bennia/Nahman min.phase Filter Gamma in dB?
```

```
');
    end
    if autog == 1  % i.e. do best gamma search
        dbhi = input('enter hi search limit in db  ');
        dblo = input('enter lo search limit in db  ');
        ddb = (dbhi - dblo)/500;
        for i = 1:501
            gamadb(i) = dblo + (i-1)*ddb;
            gama = 10^(gamadb(i)/20);
            F = X2 ./ (X2 + (gama*c2)); % this is Nahman/Guillaume filter
            Rk = F;
            Rkmag = abs(Rk);
            alphak = log(Rkmag);
            cn = ifft(alphak);
            cnre = real(cn);
            upn(1:N) = 0;
            upn(1) = 1;
            upn((N/2)+1) = 1;
            upn(2:N/2) = 2;
            for k=1:N
                ccn(k) = upn(k) * cnre(k);
            end
            alphajphase = fft(ccn);
            Rmin = exp(alphajphase); % this is Bennia/Nahman filter
            F = Rmin;
            % apply the filter to H(w)
            for k=1:N
                HNF(k) = H(k)*F(k);
            end
            % compute error in imag. part of h(t)
            hnf = ifft(HNF);
            % note: in the ideal situation there would be nothing present
            % in the imag. part of 'hnf', the time domain solution.  Thus
            % anything present in the imag(hnf) is to be considered an
            % error.  The optimum solution for the Nahman-Guillaume filter
is to
            % select a gamma which minimizes the Std.Dev. of the residuals
            % left in the imag(hnf). Minimizing either rms or std.dev.
gives
            % same results.
            agErr(i) = std(imag(hnf));
        end
        [ErrMin,Imin] = min(agErr);
        minErrDB = dblo + (Imin-1)*ddb;
        disp('Nahman-Guillaume-Bennia Filter')
        disp('Best Gamma (in dB) for min. error is   ')
```

```
        disp(minErrDB)
        plot(gamadb,agErr)
        grid
        xlabel('Gamma in dB')
        ylabel('Std.Dev')
        title('Std.Dev. of imag(h(t) vs. Gamma, Nahman-Guillaume-Bennia
Filter')
        pause
        clear gamadb
        gamadb = minErrDB;
    end
    gama = 10^(gamadb/20);
    F = X2 ./ (X2 + (gama*c2));  % This is the Nahman-Guillaume Filter
    %
    % The following code implements the Bennia & Nahman operation to turn a
    % magnitude filter into a min. phase filter.  The operations consist of
    % taking the mag. of F(f), then the nat. log, then the invFFT to get
    % the cepstrum.  Next 1/2 of the cestrum is set to zero.  Then the
    % reverse operations are performed of the FFT followed by the
    % exponential.  For notation used, see Bennia & Nahman ref.[10] fig. 1
    Rk = F;
    Rkmag = abs(Rk);
    alphak = log(Rkmag);
    cn = ifft(alphak); % cn is the cepstrum. see Oppenheim ref. [11] p.
504.
    cnre = real(cn);  % We only need to use the real part
    % now operate on cepstrum with u+(n) causal sequence operator.
    % u+(n) = 0 for n < 0, = 1 for n=1, & = 2 for n > 0
    % see Oppemheim ref. [11], page 354.  Note: both in Oppenheim & Bennia,
    % they run counter from i=0 to N-1, whereas here in MatLab, we run from
    % i = 1 to N.
    upn(1:N) = 0;
    upn(1) = 1;
    upn((N/2)+1) = 1;
    upn(2:N/2) = 2;
    for i=1:N
        ccn(i) = upn(i) * cnre(i);
    end
    % now Bennia&Nahman ref [10] take DFT & exp. to return min. phase
filter
    alphajphase = fft(ccn);
    Rmin = exp(alphajphase);
    F = Rmin;
end
% End of Bennia & Nahman Filter section
end
```

```
%
if filtype ~= 5 % i.e. don't use following section for Noise Floor Filter
%
% set Filter neg. freqs. to be conjugate of pos.freqs.
for i=2:N/2
    F(N+2-i) = conj(F(i));
end
% Determine Filter's freq. response in dB
for i=1:(N/2-1)
    Fmag(i) = abs(F(i+1));
    Fdb(i) = 20*log10(abs(F(i+1)));
end
if filtype ~= 0
% find -3dB freq. of filter
for i=1:(N/2-1)
    if (Fdb(i)+3) > 0
        IBW = (i);
    end
    FBW = (IBW+((-3-Fdb(IBW))/(Fdb(IBW+1)-Fdb(IBW))))*f0;
end
disp(' ')
disp('-3 dB Bandwidth of Filter in GHz is')
disp(FBW)
plot(fp,Fmag,'b',FBW,0.707,'+r')
grid
xlabel('Frequency in GHz')
title('Filter S21(f) vs. Freq.')
pause
semilogx(fp,Fdb,'b',FBW,-3,'+r')
grid
xlabel('Frequency in GHz')
ylabel('dB')
title('Filter S21(f) vs. Freq.')
pause
end
reply = input('do you want to see filters impulse & step responses? (1=yes,
0=no)  ');
if reply == 1
% calc filter's impulse response
hf = ifft(F);
hfre = real(hf);
%
tin = t;
if type == 1 % i.e. step, then new Tw is twice input window
    t = tsq;
end
```

```
plot(t,hfre)
grid
xlabel('time in ns')
ylabel('Volts')
title('Filter Impulse Response')
pause
shift1 = input('Does Filter impulse response need time shifting? (1=yes,
0=no)  ');
if shift1 == 1
    shift2 = input('shift to the right (1) or left (0)?   ');
    Tshift = input('Enter reqd. shift in ns ?   ');
    Nshift = round(Tshift/dt);
    Tshift = Nshift * dt;
    if shift2 == 1  % i.e. shift to right
        for i=1+Nshift:N
            hfres(i) = hfre(i-Nshift);
        end
        for i=1:Nshift
            hfres(i) = hfre(N-Nshift+i);
        end
    end
    if shift2 == 0  % i.e. shift to left
        for i=1:N-Nshift
            hfres(i) = hfre(i+Nshift);
        end
        for i=N-Nshift+1:N
            hfres(i) = hfre(i-N+Nshift);
        end
    end
plot(t,hfre,t,hfres)
grid
xlabel('time in ns')
ylabel('Volts')
title('Filter impulse response -- before & after time shift')
pause
hfre = hfres;
clear hfres
plot(t,hfre)
grid
xlabel('time in ns')
ylabel('Volts')
title('Filter impulse response -- after time shift')
pause
end
% integrate the filter's impulse response to get step response
fstep(1) = 0;
```

```
for i=2:N
    fstep(i) = fstep(i-1) + hfre(i);
end
%
plot(t,hfre,t,fstep)
grid
xlabel('time in ns')
ylabel('Volts')
title('Filter Impulse Response & Step Response')
pause
plot(t,fstep)
grid
xlabel('time in ns')
ylabel('Volts')
title('Filter Step Response')
pause
end
end
% i.e. end of filter step/impulse response display section
% CALC Filtered TRANSFER FUNCTION, H(f)  (note: also called S21(f)  )
disp('Calculation of Filtered TRANSFER FUNCTION, H(f)')
for i=1:N
    H(i) = H(i) .* F(i);
end
end  % i.e. end of filter section for all but Noise Floor type
if filtype == 5 % Noise Floor type
clear H;
% first approximation for h(t) is vout(t)
H = Vout;
% now apply Vin(f), only up to noise floor limit
% H = Vout ./ Vin;  % note ./ necessary to force single element division
for i=1: (nfcutfN+1)
    H(i) = Vout(i) ./ Vin(i);
end
% now do it also for conjugate freqs
for i=(N-nfcutfN+1):N
    H(i) = Vout(i) ./ Vin(i);
end
for i=1:(N/2-1)
Hdb(i) = 20*log10((2*Tw*1000)*abs(H(i+1)));
end
if nfopt == 0
    for i=1:(N/2-1), NFline(i) = voutnfdb; , end
semilogx(fp,Hdb,'r',fp,SAdbVin,'k',fp,SAdbVout,'b',fp,NFline,'g')
grid
xlabel('Frequency in GHz')
```

```
ylabel('Spectrum Amplitude in dBuV/MHz')
title('ffts Vout(blue), Vin(black), decon H(red) & noise floor(green) vs.
Freq.')
pause
end
semilogx(fp,Hdb,'r',fp,SAdbVin,'g',fp,SAdbVout,'b')
grid
xlabel('Frequency in GHz')
ylabel('Spectrum Amplitude in dBuV/MHz')
title('ffts Vout(blue), Vin(green), & decon H(red) vs. Freq.')
pause
end % end of filtype == 5 Noise Floor filter section
for i=1:(N/2-1)
Hmagp(i) = 20*log10(abs(H(i+1)));
end
%
% find Insertion Loss/Gain & -3dB freq. of H(f)
ILdc = 20*log10(abs(H(1)));
ILlf = 20*log10(abs(H(2)));
IL3db = ILlf - 3 ;
disp('Transfer Function DC Insertion Loss in dB is')
disp(ILdc)
disp('Transfer Function Low Freq Insertion Loss in dB is')
disp(ILlf)
for i=1:(N/2-1)
    if (Hmagp(i)-IL3db) > 0
    HIBW = (i);
    end
    HBW = (HIBW+((IL3db-Hmagp(HIBW))/(Hmagp(HIBW+1)-Hmagp(HIBW))))*f0;
end
disp('-3 dB Bandwidth of Transfer Function, H(f), in GHz is')
disp(HBW)
% note: to emphasize the discrete nature of the calculation, H(f) is only
% plotted at the actual frequencies of calculation with 'dot' plotting.  If
% a continuous line plot of H(f) is desired, then change the '.b' term in
% the plot command to '.b' instead.
semilogx(fp,Hmagp,'.b',HBW,IL3db,'+r')
grid
xlabel('Frequency in GHz')
ylabel('dB')
title('S21(f), Deconvolved Transfer Function H(f) in dB vs. Freq.')
pause
end
% calc h(t) impulse response by invFFT of H(f)
h = ifft(H);
hre = real(h);
```

```
him = imag(h);
if type == 1
    clear t
    for i = 1:N
        t(i) = i*dt;
    end
end
%
% compute error
SDerr = std(him);
disp('Std.Dev. of imag(h(t)) is')
disp(SDerr)
RMSerr = 0;
for i = 1:N
RMSerr = RMSerr + (him(i))^2 ;
end
RMSerr = sqrt(RMSerr)/N;
disp('RMS error in imag(h(t)) is')
disp(RMSerr)
plot(t,hre)
grid
xlabel('time in ns')
ylabel('Volts')
title('filtered impulse response h(t)')
shift1 = input('Does h(t) need time shifting? (1=yes, 0=no)  ');
hres = hre;
if shift1 == 1
    shift2 = input('shift to the right (1) or left (0)?   ');
    Tshift = input('Enter reqd. shift in ns ?   ');
    Nshift = round(Tshift/dt);
    Tshift = Nshift * dt;
    if shift2 == 1  % i.e. shift to right
        for i=1+Nshift:N
            hres(i) = hre(i-Nshift);
        end
        for i=1:Nshift
            hres(i) = hre(N-Nshift+i);
        end
    end
    if shift2 == 0  % i.e. shift to left
        for i=1:N-Nshift
            hres(i) = hre(i+Nshift);
        end
        for i=N-Nshift+1:N
            hres(i) = hre(i-N+Nshift);
        end
```

```
    end
plot(t,hre,t,hres)
grid
xlabel('time in ns')
ylabel('Volts')
title('impulse response -- before & after time shift')
pause
plot(t,hres)
grid
xlabel('time in ns')
ylabel('Volts')
title('Deconvolved Impulse Response')
pause
end
% integrate the impulse response to get step response
step(1) = 0;
for i=2:N
    step(i) = step(i-1) + hres(i);
end
%
plot(t,hres,t,step)
grid
xlabel('time in ns')
ylabel('Volts')
title('Impulse Response & Step Response')
pause
plot(t,step)
grid
xlabel('time in ns')
ylabel('Volts')
title('Step Response')
pause
plot(tin,vin,tin,vout,t,hres)
grid
xlabel('time in ns')
ylabel('Volts')
title('Vin, Vout, & Impulse Response')
pause
reply = input('do you want to save results to disc? (1=yes, 0=no)  ');
if reply == 1
    fname = input('Enter Output Data File Prefix Name:   ','s');
% '\r' or '\n' is delimiter for carriage return (newline), i.e. 'enter' key
% this writes output file with one data point per line
% note: this doesn't appear correct in NotePad,
% but is correct in WordPad or Word
    disp('writing Impulse Response, Imp.txt')
```

```
    dname = [fname, 'Imp.txt'];
    if drive == 1
        dname = ['A:', fname, 'Imp.txt'];
    end
    dlmwrite(dname,hre,'\r');
    disp('writing shifted Impulse Response, Imps.txt')
    dname = [fname, 'Imps.txt'];
    if drive == 1
        dname = ['A:', fname, 'Imps.txt'];
    end
    dlmwrite(dname,hres,'\r');
    disp('writing Step Response, Step.txt')
    dname = [fname, 'Step.txt'];
    if drive == 1
        dname = ['A:', fname, 'Step.txt'];
    end
    dlmwrite(dname,step,'\r');
    disp('writing Frequency Response (complex) H(f).txt')
    dname = [fname, 'H(f).txt'];
    if drive == 1
        dname = ['A:', fname, 'H(f).txt'];
    end
    dlmwrite(dname,H, '\r');
    disp('writing Frequency Response HdB.txt (starts at f0)')
    dname = [fname, 'HdB.txt'];
    if drive == 1
        dname = ['A:', fname, 'HdB.txt'];
    end
    dlmwrite(dname, Hmagp, '\r');
    disp('writing Filter Step Response, FilterStep.txt')
    dname = [fname, 'FilterStep.txt'];
    if drive == 1
        dname = ['A:', fname, 'FilterStep.txt'];
    end
    dlmwrite(dname, fstep, '\r');
    disp('Output Data written to disc A:')
end
disp('end of HdeconV3.m program')
warning on;
end
```