```
% VinDeconV31.m
% Version 3.1, Jim Andrews, KH6HTV, PSPL, 6 Sept. 2009
% V3.1 9/5/09 added Noise Floor Filter option
% V3, 8/30/09,jra changed freq.plots to semilog, added dc offset correction
% see PSPL AN-18 [12] for examples of use of this program
% Version 2.0, PICOSECOND PULSE LABS, written by J.R.Andrews, 11/16/04
% modified from VinDeconV2.m (11/16/04,jra)
% modified from HdeconV2.m --- original ver, 1.0, 10/29/04, jra
% ver.1.1, 11/16/04, corrected 2X scale factor error in impulse decon.
% ver.2.0, 11/16/04, eliminated impulse option. I found that the step decon
% method works equally well for impulse type signals also. jra
%%%%%%%%%%%%%%%%%%%%%%%%%
% 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('VinDeconV31.m --- MatLab Program')
disp('Version 3.1, 6 Sept. 2009')
disp('Picosecond Pulse Labs, Boulder, Colo, USA, J.R.Andrews')
disp('Deconvolution of vin(t) from vout(t) & impulse response h(t)')
disp('Data comes from an oscilloscope as voltage only .txt input files')
disp('Time window, Tw, should be large enough to include entire transient')
disp('Impulse Response, h(t), must be for a time window of 2*Tw and')
disp('also have 2N data points')
disp('Program calculates vin(t) = ifft(fft(vout(t))/fft(h(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 vin(t) results to disc 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 signal? 1 = step, 2 = impulse:  ');
type = 1; % eliminated impulse option, v2.0, 11/16/04
Tw = input('Enter Time Window in NanoSeconds (ns)?  ');
% 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.  Data files
% from LeCroy & Tek scopes have also been tested successfully.
fname1 = input('Enter Output Waveform Data File Name:   ','s');
dname1 = [fname1, '.txt'];
if drive == 1
    dname1 = ['A:', fname1, '.txt'];
end
disp('NOTE: Baseline Correction is strongly encouraged for enhanced
accuracy')
baseline = input('Do you want to do baseline correction?  (1=yes  0=no)
');
if baseline == 1
    fname2 = input('Enter Output Baseline Data File Name:   ','s');
```

```
    dname2 = [fname2, '.txt'];
    if drive == 1
        dname2 = ['A:', fname2, '.txt'];
    end
end
disp('NOTE: h(t) time window must = 2 * Tw(vout) & have 2N data points')
fname3 = input('Enter System Impulse Response, h(t) Data File Name:
','s');
dname3 = [fname3, '.txt'];
if drive == 1
    dname3 = ['A:', fname3, '.txt'];
end
disp('loading data')
vout = load(dname1);
if baseline == 1
    vblout = load(dname2);
    vout = vout - vblout;
end
ht = load(dname3);
Hf = fft(ht);
N = length(vout);  % = # 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
tout = t;
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)
disp('After each graph, press any key to continue')
plot(tout,vout,'b')
grid
xlabel('time in ns')
ylabel('Volts')
title('Output Waveform from Oscilloscope')
pause % pause command allows program to stop running until key is pressed
% 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 baseline
    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;
        SAdbVblout(i) = 20*log10((2*Tw*1000)*abs(Vblout(i+1)));
    end
end
% note: type == 2 option, impulse was deleted for this revision
% Special Procession of vout follows to allow inputing of any type waveform
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
        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.
    vsqout = vout;
    for i= 1:N
        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
    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]
    Vout = Vsqout;
    for i=3:2:Nsq-1  % i.e. insert ramp spec. into nulls of sq.spec.
        Vout(i) = Vsrout((i+1)/2);
    end
    N = Nsq;
    t = tsq;
    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;
    SAdbVout(i) = 20*log10((2*Tw*1000)*abs(Vout(i+1)));
end
if baseline == 1
    semilogx(fp,SAdbVout,'b',fpbl,SAdbVblout,'g')
    grid
    xlabel('Frequency in GHz')
    ylabel('Spectrum Amplitude in dBuV/MHz')
    title('ffts Vout(blue), BLout(green) vs. Freq.')
    pause
end
if baseline == 0
    semilogx(fp,SAdbVout,'b')
    grid
    xlabel('Frequency in GHz')
    ylabel('Spectrum Amplitude in dBuV/MHz')
    title('fft of Vout vs. Freq.')
    pause
end
% plot transfer function,  both h(t) & H(f)
plot(t,ht,'b')
grid
xlabel('time in ns')
ylabel('Volts')
title('Transfer Function, Impulse Response, h(t)')
pause
% for plotting purposes, discard DC, Nyquist & neg. freqs (i.e. >N/2)
for i=1:(N/2-1)
    Hfdb(i) = 20*log10(abs(Hf(i+1)));
end
semilogx(fp,Hfdb,'b')
grid
```

```
xlabel('Frequency in GHz')
ylabel('dB')
title('Transfer Function, H(f) vs. Freq.')
pause
%
% CALCULATE INPUT SIGNAL, Vin(f)
Vin = Vout ./ Hf;  % note ./ necessary to force single element division
for i=1:(N/2-1)
VinSAdB(i) = 20*log10((2*Tw*1000)*abs(Vin(i+1)));
end
semilogx(fp,VinSAdB)
grid
xlabel('Frequency in GHz')
ylabel('dB')
title('raw, unfiltered, Vin(f) Spectrum Amplitude in dBuV/MHz vs. Freq.')
pause
% FILTER SECTION
disp('Now apply a low-pass filter to noisy Vin(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
if filtype == 0 % i.e. no filter
    F(1:N) = 1;
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 H(f)
% first reaches its respective noise floor.  Beyond there we retain
% Vout(f)'s noise floor
% analyze noise floor of Vout & H 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 H 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('H(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;
    Hfnf(k) = Hfdb(i);
end
Hfnfdb = mean(Hfnf);
Hfnfdbmax = max(Hfnf);
% now find freq where H(f) first drops to noise floor max value
fpnh = 1;
while Hfdb(1:fpnh) > Hfnfdbmax , fpnh = fpnh+1; end
fhnf = fpnh*f0;
disp('Mean H(f) Noise Floor in dB is')
disp(Hfnfdb)
disp('H(f) reaches Noise Floor max at frequency in GHz')
disp(fhnf)
nfcutfN = fpn';
if fpnh < fpn, nfcutfN = fpnh; end
end
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
% Nahman-Guillaume Optimal Filter -- see references [7-9]
if filtype == 3  % i.e. Nahman-Guillaume Optimal Filter
    c2 = 16*(sin(pi*(0:N-1)/N)).^4;
    c2 = c2';
    X2 = (abs(Hf)).^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 Vin(f)
            VinNF = Vin .* F;
            % compute error in imag. part of vin(t)
            vinnf = ifft(VinNF);
            % note: in the ideal situation there would be nothing present
            % in the imag. part of 'vinnf', the time domain solution.  Thus
            % anything present in the imag(vinnf) 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(vinnf). Minimizing either rms or std.dev.
gives
            % same results.
            agErr(i) = std(imag(vinnf));
```

```
        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
% Bennia/Nahman min. phase Filter -- see technical references [7-11]
if filtype == 4  % i.e. Bennia/Nahman min. phase Filter
    c2 = 16*(sin(pi*(0:N-1)/N)).^4;
    c2 = c2';
    X2 = (abs(Hf)).^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 Vin(w)
            for k=1:N
                VinNF(k) = Vin(k)*F(k);
            end
            % compute error in imag. part of h(t)
            vinnf = ifft(VinNF);
            % note: in the ideal situation there would be nothing present
            % in the imag. part of 'vinnf', the time domain solution.  Thus
            % anything present in the imag(vinnf) 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(vinnf). Minimizing either rms or std.dev.
gives
            % same results.
            agErr(i) = std(imag(vinnf));
        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(vin(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
%
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)
semilogx(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
% CALCULATE Filtered Input Signal, Vin(f)
disp('Calculation of Filtered Input Signal, Vin(f)')
for i=1:N
    Vin(i) = Vin(i) .* F(i);
end
```

```
end % i.e. end of the filter section for all but Noise Floor type
if filtype == 5 % Noise Floor type
clear Vin;
% first approximation for vin(t) is vout(t)
Vin = Vout;
% now apply Hf, only up to noise floor limit
% Vin = Vout ./ Hf;  % note ./ necessary to force single element division
for i=1: (nfcutfN+1)
    Vin(i) = Vout(i) ./ Hf(i);
end
% now do it also for conjugate freqs
for i=(N-nfcutfN+1):N
    Vin(i) = Vout(i) ./ Hf(i);
end
for i=1:(N/2-1)
VinSAdB(i) = 20*log10((2*Tw*1000)*abs(Vin(i+1)));
end
if nfopt == 0
    for i=1:(N/2-1), NFline(i) = voutnfdb; , end
semilogx(fp,VinSAdB,'r',fp,Hfdb,'k',fp,SAdbVout,'b',fp,NFline,'g')
grid
xlabel('Frequency in GHz')
ylabel('Spectrum Amplitude in dBuV/MHz')
title('ffts Vout(blue), H(black), deconVin(red) & noise floor(green) vs.
Freq.')
pause
end
semilogx(fp,VinSAdB,'r',fp,Hfdb,'g',fp,SAdbVout,'b')
grid
xlabel('Frequency in GHz')
ylabel('Spectrum Amplitude in dBuV/MHz')
title('ffts Vout(blue), H(green), & deconVin(red) vs. Freq.')
pause
dbdiff = VinSAdB - SAdbVout;
semilogx(fp,dbdiff)
grid
xlabel('Frequency in GHz')
ylabel('dB')
title('Vin(f) - Vout(f) Spectrum Amplitude in dBuV/MHz vs. Freq.')
pause
% end of filtype == 5 Noise Floor filter section
for i=1:(N/2-1)
VinSAdB(i) = 20*log10((2*Tw*1000)*abs(Vin(i+1)));
end
%
semilogx(fp,VinSAdB,'b')
```

```
grid
xlabel('Frequency in GHz')
ylabel('dBuV/MHz')
title('Vin(f) Spectrum Amplitude in dBuV/MHz vs. Freq.')
pause
end
% calc vin(t)  by invFFT of Vin(f)
vin = ifft(Vin);
vin = N*vin;
vinre = real(vin);
vinim = imag(vin);
%
% compute error
SDerr = std(vinim);
disp('Std.Dev. of imag(vin(t)) is')
disp(SDerr)
RMSerr = 0;
for i = 1:N
RMSerr = RMSerr + (vinim(i))^2 ;
end
RMSerr = sqrt(RMSerr)/N;
disp('RMS error in imag(vin(t)) is')
disp(RMSerr)
%
if type == 1 % note: type now is always == 1, impulse option deleted
    % need to correct vinre.  At this point it is off by a scale factor
    % of 2X and it also has a ramp in it which needs to be removed.
    vinc = vinre/2;
    gi = input('Gain Correction Factor (1=H(dc) or 2=H(f0))?   ');
    Hdc = Hf(gi);
    vinint = vout(1)/Hdc;
    vinfin = vout(N/2);
    voff = vinc(1) - vinint;
    vinc = vinc - voff;
    for i=1:N
        vrampin(i) = vinint + (i-1)*(vinfin-vinint)/(N-1);
    end
    for i=1:N
        vinc(i) = vinc(i) + vrampin(i);
    end
    clear vin
    for i=1:N/2
        vin(i) = vinc(i);
    end
end
plot(tout,vin,'g',tout,vout,'b')
```

```
grid
xlabel('time in ns')
ylabel('Volts')
title('Deconvolved vin(t)(green) & vout(t) (blue)')
pause
reply = input('Is there a DC offset error? (1=yes, 0=no)  ');
% typically found problems with dc offset.  Use simple subtraction
% of vout(t=0) value.
if reply == 1
    dcoff = vin(1) - vout(1)
    vin = vin - dcoff;
end
plot(tout,vin,'b')
grid
xlabel('time in ns')
ylabel('Volts')
title('Deconvolved input signal vin(t)')
pause
plot(tout,vin,'r',tout,vout,'b')
grid
xlabel('time in ns')
ylabel('Volts')
title('Deconvolved vin(t)(red) & vout(t) (blue)')
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 Input Signal, vin(t).txt')
    dname = [fname, 'vin.txt'];
    if drive == 1
        dname = ['A:', fname, 'vin(t).txt'];
    end
    dlmwrite(dname,vin,'\r');
    disp('Data written to disc')
end
disp('end of VinDeconV31.m program')
warning on;
```