```
% timef() - Returns estimates and plots of mean event-related spectral
%           perturbation (ERSP) and inter-trial coherence (ITC) changes
%           across event-related trials (epochs) of a single input time series.
%
%         * Uses either fixed-window, zero-padded FFTs (fastest), wavelet
%           0-padded DFTs (both Hanning-tapered), OR multitaper spectra
('mtaper').
%         * For the wavelet and FFT methods, output frequency spacing
%           is the lowest frequency ('srate'/'winsize') divided by 'padratio'.
%           NaN input values (such as returned by eventlock()) are ignored.
%         * If 'alpha' is given, then bootstrap statistics are computed
%           (from a distribution of 'naccu' surrogate data trials) and
%           non-significant features of the output plots are zeroed out
%           (i.e., plotted in green).
%         * Given a 'topovec' topo vector and 'elocs' electrode location file,
%           the figure also shows a topoplot() of the specified scalp map.
%         * Note: Left-click on subplots to view and zoom in separate windows.
% Usage:
%        >> [ersp,itc,powbase,times,freqs,erspboot,itcboot] = ...
%                timef(data,frames,tlimits,srate,cycles,...
%                        'key1',value1,'key2',value2, ... );
% NOTE:
%        >> timef details  % scrolls more detailed information about timef
%        some default values are not relevant if this function is called from
%           pop_timef()
%
% Required inputs:
%       data        = Single-channel data vector (1,frames*ntrials) (required)
%       frames      = Frames per trial                               {750}
%       tlimits     = [mintime maxtime] (ms) Epoch time limits    {[-1000 2000]}
%       srate       = data sampling rate (Hz)                        {250}
%       cycles      = If 0 -> Use FFTs (with constant window length) {0}
%                     If >0 -> Number of cycles in each analysis wavelet
%                     If [wavecycles factor] -> wavelet cycles increase with
frequency
%                     beginning at wavecyles (0<factor<1; factor=1 -> no
increase,
%                     standard wavelets; factor=0 -> fixed epoch length, as in
FFT.
%                     Else, 'mtaper' -> multitaper decomposition
%
%    Optional Inter-Irial Coherence (ITC) type:
%       'type'      = ['coher'|'phasecoher'] Compute either linear coherence
%                      ('coher') or phase coherence ('phasecoher') also known
%                      as the phase coupling factor           {'phasecoher'}.
%
%    Optional detrending:
%       'detret'    = ['on'|'off'], Detrend data in time.       {'off'}
%       'detrep'    = ['on'|'off'], Detrend data across trials  {'off'}
%
%    Optional FFT/DFT parameters:
%       'winsize'   = If cycles==0: data subwindow length (fastest, 2^n<frames);
%                     If cycles >0: *longest* window length to use. This
%                      determines the lowest output frequency       {~frames/8}
%       'timesout'  = Number of output times (int<frames-winframes)       {200}
%       'padratio'  = FFT-length/winframes (2^k)                          {2}
%                      Multiplies the number of output frequencies by
```

```
%                       dividing their spacing. When cycles==0, frequency
%                       spacing is (low_freq/padratio).
%       'maxfreq'   = Maximum frequency (Hz) to plot (& to output, if cycles>0)
%                       If cycles==0, all FFT frequencies are output.      {50}
%       'baseline'  = Spectral baseline end-time (in ms).                  {0}
%       'powbase'   = Baseline spectrum to log-subtract. {def|NaN->from data}
%
%    Optional multitaper parameters:
%       'mtaper'    = If [N W], performs multitaper decomposition.
%                       (N is the time resolution and W the frequency resolution;
%                       maximum taper number is 2NW-1). Overwrites 'winsize' and
'padratio'.
%                     If [N W K], forces the use of K Slepian tapers (if
possible).
%                       Phase is calculated using standard methods.
%                       The use of mutitaper with wavelets (cycles>0) is not
%                       recommended (as multiwavelets are not implemented).
%                       Uses Matlab functions DPSS, PMTM.      {no multitaper}
%
%    Optional bootstrap parameters:
%       'alpha'     = If non-0, compute two-tailed bootstrap significance prob.
%                     level. Show non-signif. output values in green       {0}
%       'naccu'     = Number of bootstrap replications to accumulate       {200}
%       'baseboot'  = Bootstrap baseline to subtract (1 -> use 'baseline'(see
above)
%                                                    0 -> use whole trial) {1}
%    Optional scalp map:
%       'topovec'   = Scalp topography (map) to plot
{none}
%       'elocs'     = Electrode location file for scalp map
%                     File should be ascii in format of  >> topoplot example
%                     May also be an EEG.chanlocs struct. {file named in
icadefs.m}
%    Optional plotting parameters:
%       'ploterps'  = ['on'|'off'] Plot power spectral perturbations
{'on'}
%       'plotitc'   = ['on'|'off'] Plot inter trial coherence
{'on'}
%       'plotphase' = ['on'|'off'] Plot sign of the phase in the ITC panel, i.e.
%                     green->red, pos.-phase ITC, green->blue, neg.-phase ITC
{'on'}
%       'erspmax'   = [real dB] set the ERSP max. for the scale (min= -
max){auto}
%       'itcmax'    = [real<=1] set the ITC maximum for the scale
{auto}
%       'title'     = Optional figure title
{none}
%       'marktimes' = Non-0 times to mark with a dotted vertical line (ms)
{none}
%       'linewidth' = Line width for 'marktimes' traces (thick=2, thin=1)  {2}
%       'pboot'     = Bootstrap power limits (e.g., from timef())    {from data}
%       'rboot'     = Bootstrap ITC limits (e.g., from timef())      {from data}
%       'axesfont'  = Axes text font size                                {10}
%       'titlefont' = Title text font size                               {8}
%       'vert'      = [times_vector] -> plot vertical dashed lines at given
times in ms.
```

```
%       'verbose'   = ['on'|'off'] print text
{'on'}
% Outputs:
%            ersp   = Matrix (nfreqs,timesout) of log spectral diffs. from
baseline (dB)
%            itc    = Matrix of inter-trial coherencies (nfreqs,timesout)
(range: [0 1])
%          powbase  = Baseline power spectrum (subtracted from each window to
compute
%                     the ERSP).
%            times  = Vector of output times (sub-window centers) (in ms).
%            freqs  = Vector of frequency bin centers (in Hz).
%         erspboot  = Matrix (2,nfreqs) of [lower;upper] ERSP significance
diffs.
%          itcboot  = Matrix (2,nfreqs) of [lower;upper] ITC thresholds (not
diffs).
%
% Plot description:
%   Assuming both 'plotersp' and 'plotitc' options are 'on' (= default). The
upper panel
%   presents the data ERSP (Event-Related Spectral Perturbation) in dB, with
mean baseline
%   spectral activity (in dB) subtracted. Use "'baseline', NaN" to prevent
timef() from
%   removing the baseline. The lower panel presents the data ITC (Inter-Trial
Coherence).
%   Click on any plot axes to pop up a new window (using 'axcopy()')
%   -- Upper left marginal panel presents the mean spectrum during the baseline
period
%      (blue), and when significance is set, the significance threshold at each
frequency
%      (dotted green-black trace).
%   -- The marginal panel under the ERSP image shows the maximum (green) and
minimum
%      (blue) ERSP values relative to baseline power at each frequency.
%   -- The lower left marginal panel shows mean ITC across the imaged time range
(blue),
%      and when significance is set, the significance threshold (dotted green-
black).
%   -- The marginal panel under the ITC image shows the ERP (which is produced
by ITC
%      across the data spectral pass band).
%
% Author: Sigurd Enghoff, Arnaud Delorme & Scott Makeig
%          CNL / Salk Institute 1998- | SCCN/INC, UCSD 2002-
%
% Known problems:
%   Significance masking currently fails for linear coherence.
%
% See also: crossf()

%123456789012345678901234567890123456789012345678901234567890123456789012

% Copyright (C) 1998 Sigurd Enghoff, Scott Makeig, Arnaud Delorme,
% CNL / Salk Institute 8/1/98-8/28/01
%
% This program is free software; you can redistribute it and/or modify
```

```
% it under the terms of the GNU General Public License as published by
% the Free Software Foundation; either version 2 of the License, or
% (at your option) any later version.
%
% This program is distributed in the hope that it will be useful,
% but WITHOUT ANY WARRANTY; without even the implied warranty of
% MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
% GNU General Public License for more details.
%
% You should have received a copy of the GNU General Public License
% along with this program; if not, write to the Free Software
% Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA

% $Log: timef.m,v $
% Revision 1.74  2004/08/31 00:28:24  arno
% help msg
%
% Revision 1.73  2004/03/25 15:12:08  scott
% help message re EEG.chanlocs struct possibility
%
% Revision 1.72  2004/03/09 17:35:33  arno
% msg
%
% Revision 1.71  2004/03/01 16:17:49  arno
% default baseboot 1
%
% Revision 1.70  2004/03/01 02:25:06  arno
% change help message
%
% Revision 1.69  2004/02/09 18:01:27  arno
% debug multitaper
%
% Revision 1.68  2004/01/23 02:55:15  scott
% documenting maxersp
%
% Revision 1.67  2004/01/23 02:23:35  scott
% add erspmax
%
% Revision 1.66  2004/01/06 17:00:37  arno
% header typo
%
% Revision 1.65  2003/12/03 02:32:41  arno
% verbose option
%
% Revision 1.64  2003/10/22 17:37:03  arno
% slight imprecision on frequency for FFT when padratio > 1
%
% Revision 1.63  2003/08/27 21:49:11  arno
% header typo
%
% Revision 1.62  2003/08/25 21:58:50  arno
% header typo
%
% Revision 1.61  2003/08/05 22:01:52  scott
% header msg edits
%
% Revision 1.60  2003/08/04 16:38:36  arno
```

```
% plot description and curve color
%
% Revision 1.59  2003/08/04 14:40:51  arno
% description of plot
%
% Revision 1.58  2003/08/02 23:07:10  arno
% debug h(9)
%
% Revision 1.57  2003/07/23 00:28:22  scott
% help msg
%
% Revision 1.56  2003/05/12 23:41:54  arno
% text typo
%
% Revision 1.55  2003/05/05 16:27:11  arno
% debug FFT scale
%
% Revision 1.54  2003/03/14 01:04:12  arno
% typo header
%
% Revision 1.53  2003/03/12 20:09:28  scott
% header edits -sm
%
% Revision 1.52  2003/01/08 23:37:52  arno
% add discalimer about linear coherence bug
%
% Revision 1.51  2002/11/15 03:04:07  arno
% header for web
%
% Revision 1.50  2002/10/18 16:49:53  arno
% win' -> transpose(win)
%
% Revision 1.49  2002/08/14 21:07:05  arno
% hanning debug
%
% Revision 1.48  2002/08/14 21:02:19  arno
% implementing hanning funciton
%
% Revision 1.47  2002/08/12 01:47:29  arno
% color
%
% Revision 1.46  2002/08/11 22:30:20  arno
% color
%
% Revision 1.45  2002/08/09 22:29:44  arno
% implementing wavelet factor
%
% Revision 1.44  2002/07/22 14:28:56  arno
% debugging input baseline spectrum
%
% Revision 1.43  2002/07/11 18:19:30  arno
% header typo
%
% Revision 1.42  2002/07/11 15:27:28  arno
% debuging linear coherence
%
% Revision 1.41  2002/07/11 15:18:40  arno
```

```
% programing phase coherence 2
%
% Revision 1.40  2002/07/11 00:04:57  arno
% same
%
% Revision 1.39  2002/07/11 00:03:09  arno
% debugging itcmax
%
% Revision 1.38  2002/07/10 23:59:57  arno
% implement itcmax
%
% Revision 1.37  2002/07/10 21:46:03  arno
% adding phase argument
%
% Revision 1.36  2002/05/03 03:17:45  arno
% diffdlt 0.5 -> 1
%
% Revision 1.35  2002/05/01 22:25:21  arno
% no modif
%
% Revision 1.34  2002/05/01 21:23:24  arno
% no changes
%
% Revision 1.33  2002/04/30 04:27:54  arno
% trying to debug phsamp, still unsucessfull
%
% Revision 1.32  2002/04/29 15:13:50  scott
% debugging PA -sm
%
% Revision 1.31  2002/04/29 15:06:51  scott
% same -sm
%
% Revision 1.30  2002/04/29 15:03:31  scott
% same -sm
%
% Revision 1.29  2002/04/29 15:01:58  scott
% debugging -sm
%
% Revision 1.28  2002/04/29 14:58:56  scott
% debugging -sm
%
% Revision 1.27  2002/04/29 14:32:56  scott
% removing debugging statements -sm
%
% Revision 1.26  2002/04/29 14:32:31  scott
% adding ; -sm
%
% Revision 1.25  2002/04/29 14:29:53  scott
% debugging cumulX/PA -sm
%
% Revision 1.24  2002/04/29 14:24:13  scott
% debugging -sm
%
% Revision 1.23  2002/04/29 14:21:23  scott
% same -sm
%
% Revision 1.22  2002/04/29 14:15:58  scott
```

```
% insured cumulX sized -sm
%
% Revision 1.21  2002/04/29 14:12:33  scott
% used switches to test g.phsamp -sm
%
% Revision 1.20  2002/04/29 14:07:08  scott
% fixed typo -sm
%
% Revision 1.19  2002/04/29 14:02:34  scott
% made sure cumulX is computed -sm
%
% Revision 1.18  2002/04/29 13:57:49  scott
% modified PC->PA, 'phasecouple'->'phsamp', made output format (phs,amp,time) -
sm
%
% Revision 1.17  2002/04/27 21:26:24  scott
% debugging PC -sm
%
% Revision 1.16  2002/04/27 21:19:02  scott
% debugging PC -sm
%
% Revision 1.15  2002/04/27 21:17:27  scott
% debugging PC -sm
%
% Revision 1.14  2002/04/27 21:13:57  scott
% added undocumented arg 'phasecouple',{'on'|'off'} -sm
%
% Revision 1.13  2002/04/25 02:56:03  arno
% redebugging topovec
%
% Revision 1.12  2002/04/25 02:54:33  arno
% improved topovec check
%
% Revision 1.11  2002/04/23 18:34:29  arno
% modified baseline way of computation
%
% Revision 1.10  2002/04/23 18:28:02  arno
% correcting coher computation
%
% Revision 1.9  2002/04/11 19:56:12  arno
% debuging baseboot -ad & lf
%
% Revision 1.8  2002/04/11 02:10:27  arno
% correcting typo
%
% Revision 1.7  2002/04/09 00:24:55  arno
% editing latest modifications
%
% Revision 1.6  2002/04/09 00:21:17  arno
% adding vertical bars
%
% Revision 1.5  2002/04/08 19:57:19  arno
% Editing, maxfreq-> Niquist
%
% Revision 1.4  2002/04/07 02:49:35  scott
% clarified hlpe message, changed default srate and winsize -sm
%
```

```
% Revision 1.3  2002/04/06 03:48:07  arno
% changing input for 1 channel for topoplot
%
% Revision 1.2  2002/04/06 03:40:58  arno
% modifying location file check
%
% Revision 1.1  2002/04/05 17:36:45  jorn
% Initial revision
%

% 10-19-98 avoided division by zero (using MIN_ABS) -sm
% 10-19-98 improved usage message and commandline info printing -sm
% 10-19-98 made valid [] values for tvec and g.elocs -sm
% 04-01-99 added missing freq in freqs and plots, fixed log scaling bug -se & -
tpj
% 06-29-99 fixed frequency indexing for constant-Q -se
% 08-24-99 reworked to handle NaN input values -sm
% 12-07-99 adjusted ERPtimes to plot ERP under ITC -sm
% 12-22-99 debugged ERPtimes, added BASE_BOOT -sm
% 01-10-00 debugged BASE_BOOT=0 -sm
% 02-28-00 added NOTE on formula derivation below -sm
% 03-16-00 added axcopy() feature -sm & tpj
% 04-16-00 added multiple marktimes loop -sm
% 04-20-00 fixed ITC cbar limits when spcified in input -sm
% 07-29-00 changed frequencies displayed msg -sm
% 10-12-00 fixed bug in freqs when cycles>0 -sm
% 02-07-01 fixed inconsistency in BASE_BOOT use -sm
% 08-28-01 matlab 'key' value arguments -ad
% 08-28-01 multitaper decomposition -ad
% 01-25-02 reformated help & license -ad
% 03-08-02 debug & compare to old timef function -ad
% 03-16-02 timeout automatically adjusted if too high -ad
% 04-02-02 added 'coher' option -ad

function [P_1,R,mbase,times,freqs,Pboot,Rboot,PA] = timef( X, frame, tlimits,
Fs, varwin, varargin);

% Note: PA is output of 'phsamp','on'

%varwin,winsize,g.timesout,g.padratio,g.maxfreq,g.topovec,g.elocs,g.alpha,g.mark
times,g.powbase,g.pboot,g.rboot)

% ITC:   Normally, R = |Sum(Pxy)| / (Sum(|Pxx|)*Sum(|Pyy|)) is coherence.
%        But here, we consider    Phase(Pyy) = 0 and |Pyy| = 1 -> Pxy = Pxx
%        Giving, R = |Sum(Pxx)|/Sum(|Pxx|), the inter-trial coherence (ITC)
%        Also called 'phase-locking factor' by Tallon-Baudry et al. (1996)

% Constants set here:
ERSP_CAXIS_LIMIT = 0;           % 0 -> use data limits; else positive value
% giving symmetric +/- caxis limits.
ITC_CAXIS_LIMIT  = 0;           % 0 -> use data limits; else positive value
% giving symmetric +/- caxis limits.
MIN_ABS          = 1e-8;        % avoid division by ~zero

% Commandline arg defaults:
DEFAULT_EPOCH     = 750;                % Frames per trial
DEFAULT_TIMLIM = [-1000 2000];      % Time range of g.frames (ms)
```

```
DEFAULT_FS  = 250;                  % Sampling frequency (Hz)
DEFAULT_NWIN      = 200;                % Number of windows = horizontal
resolution
DEFAULT_VARWIN    = 0;              % Fixed window length or fixed number of
cycles.
% =0: fix window length to that determined by nwin
% >0: set window length equal to varwin cycles
%     Bounded above by winsize, which determines
%     the min. freq. to be computed.
DEFAULT_OVERSMP   = 2;              % Number of times to oversample frequencies
DEFAULT_MAXFREQ = 50;               % Maximum frequency to display (Hz)
DEFAULT_TITLE     = '';             % Figure title
DEFAULT_ELOC    = 'chan.locs';      % Channel location file
DEFAULT_ALPHA   = NaN;              % Percentile of bins to keep
DEFAULT_MARKTIME= NaN;

% Font sizes:
AXES_FONT       = 10;           % axes text FontSize
TITLE_FONT      = 8;

if (nargin < 1)
    help timef
    return
end

if isstr(X) & strcmp(X,'details')
    more on
    help timefdetails
    more off
    return
end

if (min(size(X))~=1 | length(X)<2)
    error('Data must be a row or column vector.');
end

if (nargin < 2)
    frame = DEFAULT_EPOCH;
elseif (~isnumeric(frame) | length(frame)~=1 | frame~=round(frame))
    error('Value of frames must be an integer.');
elseif (frame <= 0)
    error('Value of frames must be positive.');
elseif (rem(length(X),frame) ~= 0)
    error('Length of data vector must be divisible by frames.');
end

if (nargin < 3)
    tlimits = DEFAULT_TIMLIM;
elseif (~isnumeric(tlimits) | sum(size(tlimits))~=3)
    error('Value of tlimits must be a vector containing two numbers.');
elseif (tlimits(1) >= tlimits(2))
    error('tlimits interval must be ascending.');
end

if (nargin < 4)
    Fs = DEFAULT_FS;
elseif (~isnumeric(Fs) | length(Fs)~=1)
```

```
    error('Value of srate must be a number.');
elseif (Fs <= 0)
    error('Value of srate must be positive.');
end

if (nargin < 5)
    varwin = DEFAULT_VARWIN;
elseif (~isnumeric(varwin) | length(varwin)>2)
    error('Value of cycles must be a number.');
elseif (varwin < 0)
    error('Value of cycles must be zero or positive.');
end

% consider structure for these arguments
% --------------------------------------
if ~isempty(varargin)
    try, g = struct(varargin{:});
    catch, error('Argument error in the {''param'', value} sequence'); end;
end;
g.tlimits = tlimits;
g.frame   = frame;
g.srate   = Fs;
g.cycles  = varwin(1);
if length(varwin)>1
    g.cyclesfact = varwin(2);
else
    g.cyclesfact = 1;
end;

try, g.title;      catch, g.title = DEFAULT_TITLE; end;
try, g.winsize;    catch, g.winsize = max(pow2(nextpow2(g.frame)-3),4); end;
try, g.pad;        catch, g.pad = max(pow2(nextpow2(g.winsize)),4); end;
try, g.timesout;   catch, g.timesout = DEFAULT_NWIN; end;
try, g.padratio;   catch, g.padratio = DEFAULT_OVERSMP; end;
try, g.maxfreq;    catch, g.maxfreq = DEFAULT_MAXFREQ; end;
try, g.topovec;    catch, g.topovec = []; end;
try, g.elocs;      catch, g.elocs = DEFAULT_ELOC; end;
try, g.alpha;      catch, g.alpha = DEFAULT_ALPHA; end;
try, g.marktimes;  catch, g.marktimes = DEFAULT_MARKTIME; end;
try, g.powbase;    catch, g.powbase = NaN; end;
try, g.pboot;      catch, g.pboot = NaN; end;
try, g.rboot;      catch, g.rboot = NaN; end;
try, g.plotersp;   catch, g.plotersp = 'off'; end;
try, g.plotitc;    catch, g.plotitc  = 'off'; end;
try, g.detrep;     catch, g.detrep = 'off'; end;
try, g.detret;     catch, g.detret = 'off'; end;
try, g.baseline;   catch, g.baseline = 0; end;
try, g.baseboot;   catch, g.baseboot = 1; end;
try, g.linewidth;  catch, g.linewidth = 2; end;
try, g.naccu;      catch, g.naccu = 200; end;
try, g.mtaper;     catch, g.mtaper = []; end;
try, g.vert;       catch, g.vert = []; end;
try, g.type;       catch, g.type = 'phasecoher'; end;
try, g.phsamp;     catch, g.phsamp = 'off'; end;
try, g.plotphase;  catch, g.plotphase = 'on'; end;
try, g.itcmax;     catch, g.itcmax = []; end;
try, g.erspmax;    catch, g.erspmax = []; end;
```

```
try, g.verbose;    catch, g.verbose = 'on'; end;

% testing arguments consistency
% ------------------------------
switch lower(g.verbose)
    case { 'on', 'off' }, ;
    otherwise error('verbose must be either on or off');
end;
if (~ischar(g.title))
    error('Title must be a string.');
end

if (~isnumeric(g.winsize) | length(g.winsize)~=1 | g.winsize~=round(g.winsize))
    error('Value of winsize must be an integer number.');
elseif (g.winsize <= 0)
    error('Value of winsize must be positive.');
elseif (g.cycles == 0 & pow2(nextpow2(g.winsize)) ~= g.winsize)
    error('Value of winsize must be an integer power of two [1,2,4,8,16,...]');
elseif (g.winsize > g.frame)
    error('Value of winsize must be less than frame length.');
end

if (~isnumeric(g.timesout) | length(g.timesout)~=1 |
g.timesout~=round(g.timesout))
    error('Value of timesout must be an integer number.');
elseif (g.timesout <= 0)
    error('Value of timesout must be positive.');
end
if (g.timesout > g.frame-g.winsize)
    g.timesout = g.frame-g.winsize;
    disp(['Value of timesout must be <= frame-winsize, timeout adjusted to '
int2str(g.timesout) ]);
end

if (~isnumeric(g.padratio) | length(g.padratio)~=1 |
g.padratio~=round(g.padratio))
    error('Value of padratio must be an integer.');
elseif (g.padratio <= 0)
    error('Value of padratio must be positive.');
elseif (pow2(nextpow2(g.padratio)) ~= g.padratio)
    error('Value of padratio must be an integer power of two [1,2,4,8,16,...]');
end

if (~isnumeric(g.maxfreq) | length(g.maxfreq)~=1)
    error('Value of maxfreq must be a number.');
elseif (g.maxfreq <= 0)
    error('Value of maxfreq must be positive.');
elseif (g.maxfreq > Fs/2)
    myprintf(g.verbose,['Warning: value of maxfreq reduced to Nyquist rate' ...
            ' (%3.2f)\n\n'], Fs/2);
    g.maxfreq = Fs/2;
end

if isempty(g.topovec)
    g.topovec = [];
    if isempty(g.elocs)
        error('Channel location file must be specified.');
```

```
    end;
end
if isempty(g.elocs)
    g.elocs = DEFAULT_ELOC;
elseif (~ischar(g.elocs)) & ~isstruct(g.elocs)
    error('Channel location file must be a valid text file.');
end

if (~isnumeric(g.alpha) | length(g.alpha)~=1)
    error('timef(): Value of g.alpha must be a number.\n');
elseif (round(g.naccu*g.alpha) < 2)
    myprintf(g.verbose,'Value of g.alpha is out of the normal range
[%g,0.5]\n',2/g.naccu);
    g.naccu = round(2/g.alpha);
    myprintf(g.verbose,'  Increasing the number of bootstrap iterations to
%d\n',g.naccu);
end
if g.alpha>0.5 | g.alpha<=0
    error('Value of g.alpha is out of the allowed range (0.00,0.5).');
end
if ~isnan(g.alpha)
    if g.baseboot > 0
        myprintf(g.verbose,'Bootstrap analysis will use data in baseline (pre-0)
subwindows only.\n')
    else
        myprintf(g.verbose,'Bootstrap analysis will use data in all
subwindows.\n')
    end
end
if ~isnumeric(g.vert)
    error('vertical line(s) option must be a vector');
else
    if min(g.vert) < g.tlimits(1) | max(g.vert) > g.tlimits(2)
        error('vertical line(s) time out-of-bound');
    end;
end;

if ~isnan (g.rboot)
    if size(g.rboot) == [1,1]
        if g.cycles == 0
            g.rboot = g.rboot*ones(g.winsize*g.padratio/2);
        end
    end
end;

if ~isempty(g.mtaper) % mutitaper, inspired from Bijan Pesaran matlab function
    if length(g.mtaper) < 3
        %error('mtaper arguement must be [N W] or [N W K]');

        if g.mtaper(1) * g.mtaper(2) < 1
            error('mtaper 2 first arguments'' product must be higher than 1');
        end;
        if length(g.mtaper) == 2
            g.mtaper(3) = floor( 2*g.mtaper(2)*g.mtaper(1) - 1);
        end
        if length(g.mtaper) == 3
            if g.mtaper(3) > 2 * g.mtaper(1) * g.mtaper(2) -1
```

```
                error('mtaper number too high (maximum (2*N*W-1))');
            end;
        end
        disp(['Using ' num2str(g.mtaper(3)) ' tapers.']);
        NW = g.mtaper(1)*g.mtaper(2);   % product NW
        N  = g.mtaper(1)*g.srate;
        [e,v] = dpss(N, NW, 'calc');
        e=e(:,1:g.mtaper(3));
        g.alltapers = e;
    else
        g.alltapers = g.mtaper;
        disp('mtaper argument not [N W] or [N W K]; considering raw taper
matrix');
    end;
    g.winsize = size(g.alltapers, 1);
    g.pad = max(pow2(nextpow2(g.winsize)),256); % pad*nextpow
    %nfk = floor([0 g.maxfreq]./g.srate.*g.pad); % not used any more
    %g.padratio = 2*nfk(2)/g.winsize;
    g.padratio = g.pad/g.winsize;

    %compute number of frequencies
    %nf = max(256, g.pad*2^nextpow2(g.winsize+1));
    %nfk = floor([0 g.maxfreq]./g.srate.*nf);

    %freqs = linspace( 0, g.maxfreq, diff(nfk)); % this also work in the case of
a FFT

end;

switch lower(g.plotphase)
    case { 'on', 'off' }, ;
    otherwise error('plotphase must be either on or off');
end;
switch lower(g.plotersp)
    case { 'on', 'off' }, ;
    otherwise error('plotersp must be either on or off');
end;
switch lower(g.plotitc)
    case { 'on', 'off' }, ;
    otherwise error('plotitc must be either on or off');
end;
switch lower(g.detrep)
    case { 'on', 'off' }, ;
    otherwise error('detrep must be either on or off');
end;
switch lower(g.detret)
    case { 'on', 'off' }, ;
    otherwise error('detret must be either on or off');
end;
switch lower(g.phsamp)
    case { 'on', 'off' }, ;
    otherwise error('phsamp must be either on or off');
end;
if ~isnumeric(g.linewidth)
    error('linewidth must be numeric');
end;
if ~isnumeric(g.naccu)
```

```
    error('naccu must be numeric');
end;
if ~isnumeric(g.baseline)
    error('baseline must be numeric');
end;
switch g.baseboot
    case {0,1}, ;
    otherwise, error('baseboot must be 0 or 1');
end;
switch g.type
    case { 'coher', 'phasecoher', 'phasecoher2' },;
    otherwise error('Type must be either ''coher'' or ''phasecoher'''');
end;

if (g.cycles == 0) %%%%%%%%%%%%%% constant window-length FFTs %%%%%%%%%%%%%%%%
    %freqs = g.srate/g.winsize*[1:2/g.padratio:g.winsize]/2 % incorect for
padratio > 2
    freqs = linspace(0, g.srate/2, g.padratio*g.winsize/2+1);
    freqs = freqs(2:end);
    win = hanning(g.winsize);

    P  = zeros(g.padratio*g.winsize/2,g.timesout); % summed power
    PP = zeros(g.padratio*g.winsize/2,g.timesout); % power
    R  = zeros(g.padratio*g.winsize/2,g.timesout); % mean coherence
    RR = zeros(g.padratio*g.winsize/2,g.timesout); % (coherence)
    Pboot = zeros(g.padratio*g.winsize/2,g.naccu); % summed bootstrap power
    Rboot = zeros(g.padratio*g.winsize/2,g.naccu); % summed bootstrap coher
    Rn = zeros(1,g.timesout);
    Rbn = 0;
    switch g.type
        case { 'coher' 'phasecoher2' },
            cumulX = zeros(g.padratio*g.winsize/2,g.timesout);
            cumulXboot = zeros(g.padratio*g.winsize/2,g.naccu);
        case 'phasecoher'
            switch g.phsamp
                case 'on'
                    cumulX = zeros(g.padratio*g.winsize/2,g.timesout);
            end
    end;

else % %%%%%%%%%%%%%%%%%% cycles>0, Constant-Q (wavelet) DFTs
%%%%%%%%%%%%%%%%%%%

    freqs = g.srate*g.cycles/g.winsize*[2:2/g.padratio:g.winsize]/2;
    dispf = find(freqs <= g.maxfreq);
    freqs = freqs(dispf);

    %win = dftfilt(g.winsize,g.maxfreq/g.srate,g.cycles,g.padratio,0.5);
    win = dftfilt(g.winsize,g.maxfreq/g.srate,g.cycles,g.padratio,g.cyclesfact);
    P = zeros(size(win,2),g.timesout);       % summed power
    R = zeros(size(win,2),g.timesout);       % mean coherence
    PP = repmat(NaN,size(win,2),g.timesout); % initialize with NaN
    RR = repmat(NaN,size(win,2),g.timesout); % initialize with NaN
    Pboot = zeros(size(win,2),g.naccu);  % summed bootstrap power
    Rboot = zeros(size(win,2),g.naccu);  % summed bootstrap coher
    Rn = zeros(1,g.timesout);
    Rbn = 0;
```

```
    switch g.type
        case { 'coher' 'phasecoher2' },
            cumulX = zeros(size(win,2),g.timesout);
            cumulXboot = zeros(size(win,2),g.naccu);
        case 'phasecoher'
            switch g.phsamp
                case 'on'
                    cumulX = zeros(size(win,2),g.timesout);
            end
    end;
end

switch g.phsamp
    case 'on'
        PA = zeros(size(P,1),size(P,1),g.timesout); % NB: (freqs,freqs,times)
end                                                 %       phs   amp

wintime = 1000/g.srate*(g.winsize/2); % (1000/g.srate)*(g.winsize/2);
times = [g.tlimits(1)+wintime:(g.tlimits(2)-g.tlimits(1)-2*wintime)/(g.timesout-
1):g.tlimits(2)-wintime];
ERPtimes = [g.tlimits(1):(g.tlimits(2)-g.tlimits(1))/(g.frame-
1):g.tlimits(2)+0.000001];
ERPindices = [];
for ti=times
    [tmp indx] = min(abs(ERPtimes-ti));
    ERPindices  = [ERPindices indx];
end
ERPtimes = ERPtimes(ERPindices); % subset of ERP frames on t/f window centers

if ~isempty(find(times < g.baseline))
    baseln = find(times < g.baseline); % subtract means of pre-0 (centered)
windows
else
    baseln = 1:length(times); % use all times as baseline
end
if ~isnan(g.alpha) & length(baseln)==0
    myprintf(g.verbose,'timef(): no window centers in baseline (times<%g) -
shorten (max) window length.\n', g.baseline)
    return
elseif ~isnan(g.alpha) & g.baseboot
    myprintf(g.verbose,'   %d bootstrap windows in baseline (times<%g).\n',...
        length(baseln), g.baseline)
end
dispf = find(freqs <= g.maxfreq);
stp = (g.frame-g.winsize)/(g.timesout-1);

myprintf(g.verbose,'Computing Event-Related Spectral Perturbation (ERSP)
and\n');
switch g.type
    case 'phasecoher',  myprintf(g.verbose,'  Inter-Trial Phase Coherence (ITC)
images based on %d trials\n',length(X)/g.frame);
    case 'phasecoher2', myprintf(g.verbose,'  Inter-Trial Phase Coherence 2
(ITC) images based on %d trials\n',length(X)/g.frame);
    case 'coher',       myprintf(g.verbose,'  Linear Inter-Trial Coherence (ITC)
images based on %d trials\n',length(X)/g.frame);
end;
myprintf(g.verbose,'  of %d frames sampled at %g Hz.\n',g.frame,g.srate);
```

```
myprintf(g.verbose,'Each trial contains samples from %d ms before
to\n',g.tlimits(1));
myprintf(g.verbose,'  %d ms after the timelocking event.\n',g.tlimits(2));
myprintf(g.verbose,'The window size used is %d samples (%g ms)
wide.\n',g.winsize,2*wintime);
myprintf(g.verbose,'The window is applied %d times at an average
step\n',g.timesout);
myprintf(g.verbose,'  size of %g samples (%gms).\n',stp,1000*stp/g.srate);
myprintf(g.verbose,'Results are oversampled %d times; the %d
frequencies\n',g.padratio,length(dispf));
myprintf(g.verbose,'  displayed are from %2.1f Hz to %3.1f
Hz.\n',freqs(dispf(1)),freqs(dispf(end)));
if ~isnan(g.alpha)
    myprintf(g.verbose,'Only significant values (bootstrap p<%g) will be
colored;\n',g.alpha)
    myprintf(g.verbose,'  non-significant values will be plotted in green\n');
end

trials = length(X)/g.frame;
baselength = length(baseln);
myprintf(g.verbose,'\nOf %d trials total, processing trial:',trials);

% detrend over epochs (trials) if requested
% ----------------------------------------
switch g.detrep
    case 'on'
        X = reshape(X, g.frame, length(X)/g.frame);
        X = X - mean(X,2)*ones(1, length(X(:))/g.frame);
        X = X(:)';
end;

for i=1:trials
    if (rem(i,100)==0)
        myprintf(g.verbose,'\n');
    end
    if (rem(i,10) == 0)
        myprintf(g.verbose,'%d',i);
    elseif (rem(i,2) == 0)
        myprintf(g.verbose,'.');
    end

    ERP = blockave(X,g.frame); % compute the ERP trial average

    Wn = zeros(1,g.timesout);
    for j=1:g.timesout,
        tmpX = X([1:g.winsize]+floor((j-1)*stp)+(i-1)*g.frame);
        % pull out data g.frame
        tmpX = tmpX - mean(tmpX); % remove the mean for that window
        switch g.detret, case 'on', tmpX = detrend(tmpX); end;
        if ~any(isnan(tmpX))
            if (g.cycles == 0) % FFT
                if ~isempty(g.mtaper)   % apply multitaper (no hanning window)
                    tmpXMT = fft(g.alltapers .* ...
                        (tmpX(:) * ones(1,size(g.alltapers,2))), g.pad);
                    %tmpXMT = tmpXMT(nfk(1)+1:nfk(2),:);
                    tmpXMT = tmpXMT(2:g.padratio*g.winsize/2+1,:);
                    PP(:,j) = mean(abs(tmpXMT).^2, 2);
```

```
                    P_1(i,j,:) = mean(abs(tmpXMT).^2, 2);
                    % power; can also ponderate multitaper by their eigenvalues
v
                    tmpX = win .* tmpX(:);
                    tmpX = fft(tmpX, g.pad);
                    tmpX = tmpX(2:g.padratio*g.winsize/2+1);
                else
                    tmpX = win .* tmpX(:);
                    tmpX = fft(tmpX,g.padratio*g.winsize);
                    tmpX = tmpX(2:g.padratio*g.winsize/2+1);
                    PP(:,j) = abs(tmpX).^2; % power
                    P_1(i,j,:) = abs(tmpX).^2;
                end;
            else % wavelet
                if ~isempty(g.mtaper)  % apply multitaper
                    tmpXMT = g.alltapers .* (tmpX(:) *
ones(1,size(g.alltapers,2)));
                    tmpXMT = transpose(win) * tmpXMT;
                    PP(:,j) = mean(abs(tmpXMT).^2, 2); % power
                    P_1(i,j,:) = mean(abs(tmpXMT).^2, 2);
                    tmpX = transpose(win) * tmpX(:);
                else
                    tmpX = transpose(win) * tmpX(:);
                    PP(:,j) = abs(tmpX).^2; % power
                    P_1(i,j,:) = abs(tmpX).^2;
                end
            end

            if abs(tmpX) < MIN_ABS
                RR(:,j) = zeros(size(RR(:,j)));
            else
                switch g.type
                    case { 'coher' },
                        RR(:,j) = tmpX;
                        cumulX(:,j) = cumulX(:,j)+abs(tmpX).^2;
                    case { 'phasecoher2' },
                        RR(:,j) = tmpX;
                        cumulX(:,j) = cumulX(:,j)+abs(tmpX);
                    case 'phasecoher',
                        RR(:,j) = tmpX ./ abs(tmpX); % normalized cross-spectral
vector
                        switch g.phsamp
                            case 'on'
                                cumulX(:,j) = cumulX(:,j)+abs(tmpX); %
accumulate for PA
                        end
                end;
            end
            Wn(j) = 1;
        end

        switch g.phsamp
            case 'on' %PA (freq x freq x time)
                PA(:,:,j) = PA(:,:,j)  + (tmpX ./ abs(tmpX)) * ((PP(:,j)))';
                % x-product: unit phase column
                % times amplitude row
        end
```

```
end % window

%figure; imagesc(times, freqs, angle(RR));
%figure; imagesc(times, freqs, log(PP));

if ~isnan(g.alpha) % save surrogate data for bootstrap analysis
    j = 1;
    goodbasewins = find(Wn==1);
    if g.baseboot % use baseline windows only
        goodbasewins = find(goodbasewins<=baselength);
    end
    ngdbasewins = length(goodbasewins);
    if ngdbasewins>1
        while j <= g.naccu
            i=ceil(rand*ngdbasewins);
            i=goodbasewins(i);
            Pboot(:,j) = Pboot(:,j) + PP(:,i);
            Rboot(:,j) = Rboot(:,j) + RR(:,i);
            switch g.type
                case 'coher',       cumulXboot(:,j) =
cumulXboot(:,j)+abs(tmpX).^2;
                case 'phasecoher2', cumulXboot(:,j) = cumulXboot(:,j)+abs(tmpX);
            end;
            j = j+1;
        end
        Rbn = Rbn + 1;
    end
end % bootstrap

Wn = find(Wn>0);
if length(Wn)>0
    P(:,Wn) = P(:,Wn) + PP(:,Wn); % add non-NaN windows
    R(:,Wn) = R(:,Wn) + RR(:,Wn);
    Rn(Wn) = Rn(Wn) + ones(1,length(Wn)); % count number of addends
end
end % trial

% if coherence, perform the division
% ----------------------------------
switch g.type
    case 'coher',
        R = R ./ ( sqrt( trials*cumulX ) );
        if ~isnan(g.alpha)
            Rboot = Rboot ./ ( sqrt( trials*cumulXboot ) );
        end;
    case 'phasecoher2',
        R = R ./ ( cumulX );
        if ~isnan(g.alpha)
            Rboot = Rboot ./ cumulXboot;
        end;
    case 'phasecoher',
        R = R ./ (ones(size(R,1),1)*Rn);
end;

switch g.phsamp
    case 'on'
        tmpcx(1,:,:) = cumulX; % allow ./ below
```

```
        for j=1:g.timesout
            PA(:,:,j) = PA(:,:,j) ./ repmat(PP(:,j)', [size(PP,1) 1]);
        end
end

if min(Rn) < 1
    myprintf(g.verbose,'timef(): No valid timef estimates for windows %s of
%d.\n',...
        int2str(find(Rn==0)),length(Rn));
    Rn(find(Rn<1))==1;
    return
end
P = P ./ (ones(size(P,1),1) * Rn);

if isnan(g.powbase)
    myprintf(g.verbose,'\nComputing the mean baseline spectrum\n');
    mbase = mean(P(:,baseln),2)';
else
    myprintf(g.verbose,'Using the input baseline spectrum\n');
    mbase = g.powbase;
end
if ~isnan( g.baseline ) & ~isnan( mbase )
    P = 10 * (log10(P) - repmat(log10(mbase(1:size(P,1)))',[1 g.timesout])); %
convert to (10log10) dB
    P_2 = repmat(log10(mbase(1:size(P,1)))',[1 g.timesout]);
else
    P = 10 * log10(P);
end;

Rsign = sign(imag(R));
R = abs(R); % convert coherence vector to magnitude

if ~isnan(g.alpha) % if bootstrap analysis included . . .
    if Rbn>0
        i = round(g.naccu*g.alpha);
        if isnan(g.pboot)
            Pboot = Pboot / Rbn; % normalize
            if ~isnan( g.baseline )
                Pboot = 10 * (log10(Pboot) - repmat(log10(mbase)',[1 g.naccu]));
            else
                Pboot = 10 * log10(Pboot);
            end;
            Pboot = sort(Pboot');
            Pboot = [mean(Pboot(1:i,:)) ; mean(Pboot(g.naccu-i+1:g.naccu,:))];
        else
            Pboot = g.pboot;
        end

        if isnan(g.rboot)
            Rboot = abs(Rboot) / Rbn;
            Rboot = sort(Rboot');
            Rboot = mean(Rboot(g.naccu-i+1:g.naccu,:));
        else
            Rboot = g.rboot;
        end
    else
        myprintf(g.verbose,'No valid bootstrap trials...!\n');
```

```
    end
end

switch lower(g.plotitc)
    case 'on',
        switch lower(g.plotersp),
            case 'on', ordinate1 = 0.67; ordinate2 = 0.1; height = 0.33; g.plot
= 1;
            case 'off', ordinate2 = 0.1; height = 0.9; g.plot = 1;
        end;
    case 'off', ordinate1 = 0.1; height = 0.9;
        switch lower(g.plotersp),
            case 'on', ordinate1 = 0.1; height = 0.9;  g.plot = 1;
            case 'off', g.plot = 0;
        end;
end;

if g.plot
    myprintf(g.verbose,'\nNow plotting...\n');
    set(gcf,'DefaultAxesFontSize',AXES_FONT)
    colormap(jet(256));
    pos = get(gca,'position');
    q = [pos(1) pos(2) 0 0];
    s = [pos(3) pos(4) pos(3) pos(4)];
end;

switch lower(g.plotersp)
    case 'on'
        %
        %%%%%%% image the ERSP %%%%%%%%%%%%%%%%%%%%%%%%%%
        %

        h(1) = subplot('Position',[.1 ordinate1 .9 height].*s+q);

        PP = P;
        if ~isnan(g.alpha) % zero out nonsignif. power differences
            PP(find((PP > repmat(Pboot(1,:)',[1 g.timesout])) ...
                & (PP < repmat(Pboot(2,:)',[1 g.timesout])))) = 0;
        end

        if ERSP_CAXIS_LIMIT == 0
            ersp_caxis = [-1 1]*1.1*max(max(abs(P(dispf,:))));
        else
            ersp_caxis = ERSP_CAXIS_LIMIT*[-1 1];
        end

        if ~isnan( g.baseline )
            imagesc(times,freqs(dispf),PP(dispf,:),ersp_caxis);
        else
            imagesc(times,freqs(dispf),PP(dispf,:));
        end;
        if ~isempty(g.erspmax)
            %caxis([-g.erspmax g.erspmax]);
            %caxis([30 70]);% for spectrograms
            caxis([-5 5]);

        end;
```

```
        hold on
        plot([0 0],[0 freqs(max(dispf))],'--m','LineWidth',g.linewidth); % plot
time 0
        if ~isnan(g.marktimes) % plot marked time
            for mt = g.marktimes(:)'
                plot([mt mt],[0 freqs(max(dispf))],'--
k','LineWidth',g.linewidth);
            end
        end
        hold off
        set(h(1),'YTickLabel',[],'YTick',[])
        set(h(1),'XTickLabel',[],'XTick',[])
        if ~isempty(g.vert)
            for index = 1:length(g.vert)
                line([g.vert(index), g.vert(index)], [min(freqs(dispf))
max(freqs(dispf))], 'linewidth', 1, 'color', 'm');
            end;
        end;

        h(2) = gca;
        h(3) = cbar('vert'); % ERSP colorbar axes
        set(h(2),'Position',[.1 ordinate1 .8 height].*s+q)
        set(h(3),'Position',[.95 ordinate1 .05 height].*s+q)
        title('ERSP (dB)')

        E = [min(P(dispf,:));max(P(dispf,:))];
        h(4) = subplot('Position',[.1 ordinate1-0.1 .8 .1].*s+q); % plot
marginal ERSP means
        % below the ERSP image
        plot(times,E,[0 0],...
            [min(E(1,:))-max(max(abs(E)))/3 max(E(2,:))+max(max(abs(E)))/3], ...
            '--m','LineWidth',g.linewidth)
        axis([min(times) max(times) ...
                min(E(1,:))-max(max(abs(E)))/3 max(E(2,:))+max(max(abs(E)))/3])

        tick = get(h(4),'YTick');
        set(h(4),'YTick',[tick(1) ; tick(end)])
        set(h(4),'YAxisLocation','right')
        set(h(4),'TickLength',[0.020 0.025]);
        xlabel('Time (ms)')
        ylabel('dB')

        E = 10 * log10(mbase(dispf));
        h(5) = subplot('Position',[0 ordinate1 .1 height].*s+q); % plot mean
spectrum
        % to left of ERSP image
        plot(freqs(dispf),E,'LineWidth',g.linewidth)
        if ~isnan(g.alpha)
            hold on;
            plot(freqs(dispf),Pboot(:,dispf)+[E;E],'g',
'LineWidth',g.linewidth);
            plot(freqs(dispf),Pboot(:,dispf)+[E;E],'k:','LineWidth',g.linewidth)
        end

        axis([freqs(1) freqs(max(dispf)) min(E)-max(abs(E))/3
max(E)+max(abs(E))/3])
```

```
        tick = get(h(5),'YTick');
        if (length(tick)>1)
            set(h(5),'YTick',[tick(1) ; tick(end-1)])
        end
        set(h(5),'TickLength',[0.020 0.025]);
        set(h(5),'View',[90 90])
        xlabel('Frequency (Hz)')
        ylabel('dB')
end;

switch lower(g.plotitc)
    case 'on'
        %
        %%%%%%%%%%%% Image the ITC %%%%%%%%%%%%%%%%%
        %
        h(6) = subplot('Position',[.1 ordinate2 .9 height].*s+q); % ITC image

        RR = R;
        if ~isnan(g.alpha)
            RR(find(RR < repmat(Rboot(1,:)',[1 g.timesout]))) = 0;
        end

        if ITC_CAXIS_LIMIT == 0
            coh_caxis = min(max(max(R(dispf,:))),1)*[-1 1]; % 1 WAS 0.4 !
        else
            coh_caxis = ITC_CAXIS_LIMIT*[-1 1];
        end

        if exist('Rsign') & strcmp(g.plotphase, 'on')
            imagesc(times,freqs(dispf),Rsign(dispf,:).*RR(dispf,:),coh_caxis); %
<---
        else
            imagesc(times,freqs(dispf),RR(dispf,:),coh_caxis); % <---
        end
        if ~isempty(g.itcmax)
            caxis([-g.itcmax g.itcmax]);
        end;
        tmpcaxis = caxis;

        hold on
        plot([0 0],[0 freqs(max(dispf))],'--m','LineWidth',g.linewidth);
        if ~isnan(g.marktimes)
            for mt = g.marktimes(:)'
                plot([mt mt],[0 freqs(max(dispf))],'--
k','LineWidth',g.linewidth);
            end
        end
        hold off
        set(h(6),'YTickLabel',[],'YTick',[])
        set(h(6),'XTickLabel',[],'XTick',[])
        if ~isempty(g.vert)
            for index = 1:length(g.vert)
                line([g.vert(index), g.vert(index)], [min(freqs(dispf))
max(freqs(dispf))], 'linewidth', 1, 'color', 'm');
            end;
        end;
```

```
        h(7) = gca;
        h(8) = cbar('vert');
        %h(9) = get(h(8),'Children');
        set(h(7),'Position',[.1 ordinate2 .8 height].*s+q)
        set(h(8),'Position',[.95 ordinate2 .05 height].*s+q)
        set(h(8),'YLim',[0 tmpcaxis(2)]);
        title('ITC')

        %
        %%%%% plot the ERP below the ITC image
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        %
        % E = mean(R(dispf,:));

        ERPmax = max(ERP);
        ERPmin = min(ERP);
        ERPmax = ERPmax + 0.1*(ERPmax-ERPmin);
        ERPmin = ERPmin - 0.1*(ERPmax-ERPmin);
        h(10) = subplot('Position',[.1 ordinate2-0.1 .8 .1].*s+q); % ERP

        plot(ERPtimes,ERP(ERPindices),...
            [0 0],[ERPmin ERPmax],'--m','LineWidth',g.linewidth);
        hold on; plot([times(1) times(length(times))],[0 0], 'k');
        axis([min(ERPtimes) max(ERPtimes) ERPmin ERPmax]);

        tick = get(h(10),'YTick');
        set(h(10),'YTick',[tick(1) ; tick(end)])
        set(h(10),'TickLength',[0.02 0.025]);
        set(h(10),'YAxisLocation','right')
        xlabel('Time (ms)')
        ylabel('uV')

        E = mean(R(dispf,:)');
        h(11) = subplot('Position',[0 ordinate2 .1 height].*s+q); % plot the
marginal mean
        % ITC left of the ITC image
        if ~isnan(g.alpha)
            plot(freqs(dispf),E,'LineWidth',g.linewidth); hold on;
            plot(freqs(dispf),Rboot(dispf),'g', 'LineWidth',g.linewidth);
            plot(freqs(dispf),Rboot(dispf),'k:','LineWidth',g.linewidth);
            axis([freqs(1) freqs(max(dispf)) 0 max([E Rboot(dispf)])+max(E)/3])
        else
            plot(freqs(dispf),E,'LineWidth',g.linewidth)
            axis([freqs(1) freqs(max(dispf)) min(E)-max(E)/3 max(E)+max(E)/3])
        end

        tick = get(h(11),'YTick');
        set(h(11),'YTick',[tick(1) ; tick(length(tick))])
        set(h(11),'View',[90 90])
        set(h(11),'TickLength',[0.020 0.025]);
        xlabel('Frequency (Hz)')
        ylabel('ERP')
        %
        %%%%%%%%%%%%%%%% plot a topoplot() %%%%%%%%%%%%%%%%%%%%%%%%
        %
        if (~isempty(g.topovec))
            h(12) = subplot('Position',[-.1 .43 .2 .14].*s+q);
```

```
            if length(g.topovec) == 1
                topoplot(g.topovec,g.elocs,'electrodes','off', ...
                    'style', 'blank', 'emarkersize1chan', 10);
            else
                topoplot(g.topovec,g.elocs,'electrodes','off');
            end;
            axis('square')
        end
end; %switch

if g.plot
    try, icadefs; set(gcf, 'color', BACKCOLOR); catch, end;
    if (length(g.title) > 0)
        axes('Position',pos,'Visible','Off');
        h(13) = text(-.05,1.01,g.title);
        set(h(13),'VerticalAlignment','bottom')
        set(h(13),'HorizontalAlignment','left')
        set(h(13),'FontSize',TITLE_FONT);
    end

    axcopy(gcf);

end;
freq1 = freqs;
time1 = times;
save gvp1 P_1
save gvp2 P_2
save gvf freq1
save gvt time1
gwin1 = g.winsize;
save gwin gwin1

% syemtric hanning function
function w = hanning(n)
if ~rem(n,2)
    w = .5*(1 - cos(2*pi*(1:n/2)'/(n+1)));
    w = [w; w(end:-1:1)];
else
    w = .5*(1 - cos(2*pi*(1:(n+1)/2)'/(n+1)));
    w = [w; w(end-1:-1:1)];
end

function myprintf(verbose, varargin)
if strcmpi(verbose, 'on')
    fprintf(varargin{:});
end;
```