GUISDAP Documentation

M. S. Lehtinen[*] and A. Huuskonen[**]

[*] Sodankylä Geophysical Observatory, Sodankylä, Finland
[**] Finnish Meteorological Institute, Helsinki, Finland

Running the Analysis package

The analysis of incoherent scatter experiment with GUISDAP requires certain hardware and software facilities. Especially, the following are needed:

- A Matlab licence (and computer)
- An initialization file for the experiment.
- The measurements as EISCAT raw data files or binary Matlab files on the disk.
- A startup program for the analysis.

A basic startup program

The first startup program (`CP1K_demo1.m`) shows a simple example how to invoke the GUISDAP package. It includes all the necessary parameters and a basic collection of the optional ones.

The first parameter (`name_expr`) contains the name of the experiment. The names are intrinsic to the GUISDAP package and need not be the same as those used in EISCAT experiments. However, the names are chosen so that they resemble their EISCAT counterparts. The next parameter (`name_site`) shows the measurement site, using the EISCAT convention. The allowed values are `'T'`, `'K'`, `'S'` and `'V'`. The next two parameters give the data and result directories.

The parameters `analysis_start` and `analysis_end` specify the time interval to be analyzed. Each is a vector with six elements, which give the year, month, day, hour, minutes and seconds of the time instant. The next parameters (`analysis_integr` and `analysis_skip`) control the post integration of the data and are given in seconds. These vectors are used ina a cyclical manner. If the `_skip` parameter is not given, the skips are assumed to be equal to zeroes.

The meaning and use of the previous parameters is self–evident. The next parameter brings in new possibilies which are not normally included in the analysis packages. The parameter `analysis_altit` defines a set of altitudes, which are the border lines for the analysis gates. The i^{th} gate in the analysis uses all those crossed products for which the center point of the range ambiguity functions is between altitudes `analysis_altit(i)` and `analysis_altit(i+1)`. So the altitude (or range) dimension is handled by the package just like the time integration. One specifies the interesting event, one dimension in time and the other in altitude, and the program finds all measurements falling between the specified limits. It is possible to use ranges instead of altitudes. If `analysis_range` is given the parameter `analysis_altit` does not have any effect. This possibility could be useful in scanning experiments.

The vector `analysis_control` gives some control on how the fitting is done. It contains four elements (The standard values are given in parenthesis):

1 If the error of electron density, evaluated at the start of iteration, exceeds the limit, no fit is done to the data. (∞).

2 The iteration stops, when the step for all parameters is shorter than the limit (0.01)

3 Gives the maximum number of iterations done (6)

4 Controls the way the variances are calculated. Normally they are calculated experimentally during the integration (1) but it is possible to used theoretical estimates (2).

The last parameter (`display_figures`) controls the graphical output. The following plots are shown, when the corresponding element is set true

1 After the postintegration has been done, and data scaled by the calibration measurements, and background subtracted, the correlator dump (real part) is displayed.

2 Before proceeding to the analysis, the program calculates raw electron density profile from all zero lag measurements, including the Debye corrections and using the *a priori* temperature ratio model.

3 For each altitude range, a figure containing the measurement with the fitted curve is shown for a visual check of the fit quality. The plot also includes a separate display of the *a priori* parameters with the fitted parameters.

4 After all gates have been analyzed, a figure displaying the fit results is shown.

The last command in the startup program (`an_start`) then invokes the analysis itself.

/geo/gmt/askoh/guisdap/m152/CP1K_demo1.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   GUP ver. 1.50      Helsinki March 2, 1994                          %
%   copyright Asko Huuskonen, Markku Lehtinen,                         %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

start_GUP

name_expr='CP1K';  % Experiment name
name_site='T';     % Site
data_path=[path_GUP 'demodata/CP1K/']; % data directory
result_path=path_tmp;        % result directory

analysis_start=[1993   2    16  11  00   0]; % Start time
analysis_end=  [1993   2    16  11  15   0]; % End time

analysis_integr=[180];
analysis_skip  =[ 0 ];

analysis_altit=[90:3:130, 135:5:170, 180:10:240 260:20:600];
% analysis_range=[90:3:130, 135:5:170, 180:10:240 260:20:600];

analysis_control=[.50 .01 6 1];
% analysis_control(1) : if error of Ne exceeds this, no fit done
% analysis_control(2) : iteration stopped, when step below this
% analysis_control(3) : max number of iterations
% analysis_control(4) : variance calculation

display_figures=[1 1 1 1];
% display_figures(1) : data dump shown
% display_figures(2) : raw electron density shown
% display_figures(3) : data and fitted curve for each fit shown
% display_figures(4) : results shown after each dump

an_start
```

Explicit specification of modulations

The parameters included in `CP1K_demo1.m` do not always give a full control on the altitude integration. For instance, the gate size was chosen to 5 kilometers around the 150 km altitude. The gates then include 1–2 multipulse ACF's and possibly one power profile measurement and the resulting altitude weighting is comparable to the gate size. However, this is not the case, when the center point of a long pulse measurement just happens to fall within a gate. The range ambiguity functions of the long pulse data are much wider and idea of integrating over roughly five kilometers does not come true.

To solve that kind problems, the package contains a way of defining explicitely what modulations are used within each of the gates defined by `analysis_altit`. The parameter is called `analysis_code`. If one wants to use only modulation number 1 (alternating code in CP1), one just gives a value of 1 to each gate. In the example file (`CP1K_demo2`) modulations 1 and 2 are used in the first fourteen gates and modulation 3 in the remaining ones. The `_altit` variable also demonstrates that is is possible to make downward steps, which are skipped over in the analysis. The `_code` variable contains a zero in the corresponding place. This makes it possible to analyze the same data with varying altitude integrations during the same run.

The last feature demonstrated here is how `_integr` and `_skip` variables are used.

/geo/gmt/askoh/guisdap/m152/CP1K_demo2.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%   GUP ver. 1.50      Helsinki March 2, 1994                        %
%   copyright Asko Huuskonen, Markku Lehtinen,                       %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

start_GUP

name_expr='CP1K';  % Experiment name
name_site='T';     % Site
data_path=[path_GUP 'demodata/CP1K/']; % data directory
result_path=path_tmp;         % result directory

analysis_start=[1993   2    16  11  00    0]; % Start time
analysis_end=  [1993   2    16  11  15    0]; % End time

analysis_integr=[180 120 120];
analysis_skip  =[ 0   30  30];

analysis_altit=[110:5:180 140:20:600];
analysis_code= [12*ones(1,14),0, 3*ones(1,23)];
% description of code numbers
% 1: alternating code ACF including the zero lag from power profile
% 2: F-region power profile
% 3: long pulse

display_figures=[1 1 1 1];

an_start
```

How to fit ACF's instead of sets of crossed products?

The GUISDAP program is built so that the concept of autocorrelation function is not necessary at all. Instead of ACF's, the fits are done to a set of measurements (crossed products), which may all have different weightings in lag and range (different two-dimensional ambiguity functions) and may originate from different altitudes, provided that the center points fall to the specified altitude range. Of course, it is possible to define sets which correspond to classical autocorrelation functions and to analyze experiments like CP1K gate-by-gate so that data originating from different modulations are not used in the same fit. One parameter (`analysis_classic`) is included in the package to facilitate this kind of analysis. When this parameter has a non–zero value (is true in Matlab) the GUISDAP program automatically produces such control parameters that all ACF's are analyzed between the lowest and highest altitudes specified in `analysis_altit`. This kind of analysis is shown in the third demo file (`CP1K_demo3.m`)

/geo/gmt/askoh/guisdap/m152/CP1K_demo3.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   GUP ver. 1.50      Helsinki March 2, 1994                        %
%   copyright Asko Huuskonen, Markku Lehtinen,                       %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

start_GUP

name_expr='CP1K';  % Experiment name
name_site='T';     % Site
data_path=[path_GUP 'demodata/CP1K/']; % data directory
result_path=path_tmp;         % result directory

analysis_start=[1993   2    16  11  00    0]; % Start time
analysis_end=  [1993   2    16  11  15    0]; % End time

analysis_integr=[180];
analysis_skip  =[ 0 ];

analysis_altit=[90 600];
analysis_classic=1;

display_figures=[1 1 0 1];

an_start
```

Integration definition file for Nigel Wade's integration package

The input files may be either binary Matlab files, which each contain one (integrated) dump possibly with variance estimates, or EISCAT raw data files. GUISDAP checks the data directory and, if raw data files are found, produces automatically an integration definition file and then invokes theintegration facility by a mex call. To use all the option of Nigel Wade's package, it is necessary to make a separate integration definition file and to give the name of the file to GUISDAP, as in the example below. Then there is no need to give the `data_path`, analysis start and end times and integration strategy.

/geo/gmt/askoh/guisdap/exps/CP1K/An_demo4.m
```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   GUP ver. 1.50      Helsinki March 2, 1994                      %
%   copyright Asko Huuskonen, Markku Lehtinen,                     %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

start_GUP

name_expr='CP1K';  % Experiment name
name_site='T';     % Site

integ_deffile=[path_GUP 'exps/CP1K/integdef'];
result_path=path_tmp;        % result directory

analysis_altit=[90 600];
analysis_classic=1;

display_figures=[1 1 0 1];

an_start
```

What parameters are fitted?

The user can control the fitting process by changing the *a priori* model ionosphere. The model ionosphere is supplied by the function **ionomodel.m**. The function receives a set of altitudes as input and must give a value for the following parameters in the units given and also their uncertainties, if required.

1	N_e	Electron density in m^{-3}
2	T_i	Ion temperature in K
3	T_e/T_i	Electron to ion temperature ratio
4	v_i	Ion velocity in ms^{-1}
5	ν_{in}	Ion-neutral collision frequency in Hz
6	$[O^+]/N_e$	Ion composition

The *a priori* uncertainties give for the user a way to control the analysis. The package always fits the first five parameters. For each parameter, the specified uncertainty puts a limit how much the fitted parameter value can deviate from the *a priori* value. When the uncertainty is small, the parameter will not change and it is a classical fixed parameter. On the other hand, when the uncertainty is large, the parameter is free to change and corresponds to a classical fitted parameter. However, the uncertainty can have any value between these extremes and so a certain parameter can be allowed to vary to some extent, but not completely freely. When considering whether an uncertainty is small or large, one has to compare it with the *a posteriori* error of the parameter in a fit where the parameter would be allowed to change freely. If the *a priori* uncertainty is significantly smaller than that, the fit cannot have any effect on the parameter value. In the opposite case, the *a priori* model is not important for the result. When the both are of the same order of magnitude, the fit result is affected both by the *a priori* model and by the data.

The final conclusion from these considerations is that we need not treat the *a priori* values of the plasma parameters and the measured crossed products separately. They all are data, some of which give two–dimensional averages of the plasma autocorrelation function within certain uncertainties and some others give the *a priori* model. The fitting process gives a new value for all those, so that we will get theoretical values of the crossed producs and the *a posteriori* parameter values. If a certain data point has a large *a priori* uncertainty, the *a posteriori* value brings us new information. Simple examples of such data points are first five parameters, but we will also get refined estimates of the crossed products and even estimates of crossed products which cannot be measured by the code.

The included version of **ionomodel.m** gives so small errors for the the temperature ratio below 110 km that it is, in fact, fixed in the analysis. It is possible, and often necessary, for the user to produce his/her own model, e.g. in the cases of electron heating in the E-region.

The composition is an example of a 'difficult' parameters. Its determination is based on the *a posteriori* distribution method and is explained separately.

/geo/gmt/askoh/guisdap/m152/ionomodel.m
```
% GUISDAP v1.50   94-03-10 Copyright Markku Lehtinen, Asko Huuskonen
%
% 'ionomodel' is a user-supplied function which for a given set of heights
% outputs the plasma parameters (Ne, Ti, Te/Ti, coll, [O+]/Ne, velocity)
% with their a priori uncertainties. This simple example is static, a more complete
% model might use the date and time values to modify the model.
%
% function [apriori, apriorierror]=ionomodel(heights);

function [apriori, apriorierror]=ionomodel(heights);

global ionomodel_control

maxCollAltit=107; % Below this collision frequency is given freedom to change
minRatioAltit=107; % Above this temperture ratio is given freedom to change
% Note: This above need not be equal, but setting minRatioAltit less than
% maxCollAltit will produce unconvergent fits, because both parameters will change.
% It is completely OK to have a range where neither is fitted.

len=length(heights);
heights=col(heights);

if nargout==2, error=1; end
if length(ionomodel_control)==0, ionomodel_control=0; end

% The global variable ionomodel_control affects the a priori electron density model.
% This feature is needed in the experiment design, where a model ionosphere is needed
% to calculate a correlator dump.
if ionomodel_control==2; % constant SNR
  ne=1e11*(heights/100).^2;
else
  if ionomodel_control==1; % Strong ionosphere
    Elowerwidth=15; Eupperwidth=50;  Emaxheight=105; Emaxdensity=6e11;
    Flowerwidth=75; Fupperwidth=125; Fmaxheight=300; Fmaxdensity=1e12;
  elseif ionomodel_control==-1; % Weak ionosphere
    Elowerwidth=15; Eupperwidth=60;  Emaxheight=115; Emaxdensity=5e10;
    Flowerwidth=75; Fupperwidth=125; Fmaxheight=300; Fmaxdensity=2e11;
  elseif ionomodel_control==0; % Standard ionosphere
    Elowerwidth=15; Eupperwidth=60;  Emaxheight=115; Emaxdensity=2e11;
    Flowerwidth=75; Fupperwidth=125; Fmaxheight=300; Fmaxdensity=5e11;
  end
  ne=zeros(size(heights));
  ind=find(heights<=Emaxheight);
  ne(ind)=ne(ind)+Emaxdensity*exp(-((heights(ind)-Emaxheight)/Elowerwidth).^2);
  ind=find(heights>Emaxheight);
  ne(ind)=ne(ind)+Emaxdensity*exp(-((heights(ind)-Emaxheight)/Eupperwidth).^2);
  ind=find(heights<=Fmaxheight);
  ne(ind)=ne(ind)+Fmaxdensity*exp(-((heights(ind)-Fmaxheight)/Flowerwidth).^2);
  ind=find(heights>Fmaxheight);
  ne(ind)=ne(ind)+Fmaxdensity*exp(-((heights(ind)-Fmaxheight)/Fupperwidth).^1.3);
end
par=1; % Electron density
   apriori(:,par)=ne;
   if error, apriorierror(:,par)=10*apriori(:,par); end

par=2; % Ion temperature
ExosphericTemp=1000;
CenterHeight=140;
ScaleLength=30;
   ti=ExosphericTemp*(1+(atan((heights-CenterHeight)/ScaleLength))/(pi/2))/2;
   apriori(:,par)=ti;
   if error, apriorierror(:,par)=10*apriori(:,par); end

par=3; % Temperature ratio
MaxRatio=1.6;
CenterHeight=140;
ScaleLength=30;
   ratio=0.5+(MaxRatio-0.5)*(1+(atan((heights-CenterHeight)/ScaleLength))/(pi/2))/2;
   apriori(:,par)=max(1,ratio);
   if error,
    apriorierror(:,par)=10*apriori(:,par);
    ind=find(heights<=minRatioAltit);
    if length(ind)>0,apriorierror(ind,par)=apriori(ind,par)/100;end
  end

par=4; % Ion-neutral collision frequency
   apriori(:,par)=max(3578*(exp(-(heights-100)/5.8)),1);
   if error,
     apriorierror(:,par)=apriori(:,par)/100;
     ind=find(heights<=maxCollAltit);
     if length(ind)>0,apriorierror(ind,par)=apriori(ind,par);end
   end

par=5; % Ion velocity
  apriori(:,par)=zeros(len,1);
  if error, apriorierror(:,par)=1000*ones(len,1); end

par=6; % Ion composition
   xi=[149 150 250 251];
   yi=[0.0 0.0 1.0 1.0];
```

```
        apriori(:,par)=inter3(heights',xi,yi)';
        if error, apriorierror(:,par)=ones(len,1); end
```

The main analysis loop

The main analysis loop is executed in program `an_start`. The following program description explains the main operations done in the program.

The names like `radar_eq` can refer either to functions or variables.

```
initialize the control variables (chk_par1 and other calls)

loop through postintegration periods
   integrate datafiles which belong to the integration period (integr_data and integr_NW)
   in case of error (OK is false) go to the next integration period
   when all files have been processed (EOF is true) stop analysis

   transfer the parameter block contents to GUISDAP variables (decodeparblock)
   if the radar controller program has changed
      load in the initialiazation file for the new radar controller program (load_initfile)
      load in the GUP variables, if variances calculated by ambiguity function method(load_GUPvar)
      scale spectral ambiguity function by lpg_ND factors (scale_lpgwom)
      calculate some parameters (ad_) for each signal crossed product (form_adpar)
      load in files needed in the spectral calculations (spektri_init)
      calculate some basic constants (constants)
   end if

   calculate the radar equation factors (radar_eq)
   update control variables which depend on the antenna direction (chk_par2)
   scale the data by the correlator algorithm factors (scale_data)
   scale the data using the background and calibration measurements (also in scale_data)
   display the data dump graphically, if required
   subtract the background(subr_backgr)
   calculate the a priori ionosphere (get_apriori)
   clear the result vectors (clear_results)
   do the half profile analysis (half_prof)
   store results to disk (save_results)
   show results graphically (plot_result)
end loop through postintegration periods
```

/geo/gmt/askoh/guisdap/m152/an_start.m

```
% GUISDAP v1.50   94-03-10 Copyright Markku Lehtinen, Asko Huuskonen
%
% Main program for the data analysis. The most important operations performed are
% routine name:  action:
% chk_par1       1) transforms the user supplied control parameters to internal parameters
%                2) checks the data source (matlab files/EISCAT .dtst files)
% init_graphics  opens a sufficient number of figure windows and defines there sizes etc.
% integr_NW      calls Nigel Wade's integration package, when EISCAT .dtst files are used
% integr_data    integrates data from Matlab files
% decodeparblock transfers the radar parameters (power etc) to internal GUISDAP parameters
% load_initfile  loads the ambiguity functions etc
% scale_lpgwom   scales the spectal amb. function with the correlator algorithm factors (lpg_ND)
% radar_eq       radar equation
% scale_data     scales the data with the correlator algorithm factors (lpg_ND)
% subr_backgr    background subtraction to the data
% get_apriori    the a priori model, electron density obtained from the data
% half_prof      performs the gated analysis to the data
% save_results   results stored to the disk
% plot_results   displays the results
%
% Other routines called: globals nat_const get_ADDRSHIFT  load_GUPvar GUIZARD GUISPERT form_adpar
%                        spektri_init constants chk_par2 simul_dump clear_results

t_init
t_start(1)

globals          % Defines (nearly) all global variables
chk_par1
nat_const
get_ADDRSHIFT
```

```
init_graphics

rcprog_old=-1;
EOF=0;
while ~EOF,

  if a_rawdata,
    [OK,EOF]=integr_NW(a_integdeffile);
  elseif any(a_simul)
    OK=1;EOF=1;
  else
    [OK,EOF]=integr_data;
  end
  if OK, decodeparblock; end

%*************************************************************************************
%
% At this point, an integrated complex data dump is stored in variable d_data
% It is possible to study the contents of the data by graphical and other means, e.g.
% ind=0:length(d_data)-2; plot(ind,real(d_data(ind)));
%
%*************************************************************************************

  if OK,
    if d_rcprog~=rcprog_old
      load_initfile
      if a_control(4)==2, load_GUPvar, end
      if exist('GUIZARD')==2, GUIZARD, end
      scale_lpgwom % scales the spectral ambiguity function with lpg_ND factors
      form_adpar
      spektri_init % Loads in plasma dispersion function table
      constants
      rcprog_old=d_rcprog;
    end
    if exist('GUISPERT')==2, GUISPERT, end
    radar_eq   % calculates the radar constant
    chk_par2

    if any(a_simul),
      simul_dump
    else
      scale_data
    end

%*************************************************************************************
%
% Now the data has been scaled by calibration, so that it appears in units of K
% ind=0:length(d_data)-2; plot(ind,real(d_data(ind))); % This would show the data
%
%*************************************************************************************

    subr_backgr

%*************************************************************************************
%
% At this point one finds background subtracted data in vector d_data
%
%*************************************************************************************

    if di_figures(1),
      figure(di_figures(1)); clf; ind=find(real(d_data>-200 & d_data<2000));
      plot(ind-1,real(d_data(ind))), title(' Correlator dump, real part');
      xlabel('Address'); ylabel('Power [K]'); grid; drawnow
    end

    get_apriori(any(a_simul))

%*************************************************************************************
%
% The get_apriori call calculated the raw electron density profile.
% It is stored in variables
% pp_range   : range to power measurements
% pp_profile : Ne with a priori temperature ratio model
% pp_sigma   : Ne with Te=Ti
%
%*************************************************************************************

    clear_results
%****************************************
         half_prof
%****************************************
    save_results
    if di_figures(4), plot_result(di_figures(4),1); end
  end

```

```
end
t_stop(1)
t_result
```

The postintegration of data

The routine `integrate_data` is used to postintegrate the data dumps. It uses the following criteria in the integration

- The dump time must be greater than the integration start time (`a_interval(1)`)
- The dump time is less or equal to the integration end time (`a_interval(2)`)
- The transmitter must be on, i.e. `d_parbl(95)` is even or zero.

During the integration, certain parameters in `d_parbl` are checked (numbers 1, 5...90, 92, 93, 127 and 128). If any of these changes, a warning message is printed on the screen for evaluation but the integration proceeds. The power parameters (96...99) are averaged and the bits in the status word of the result dump are set if they were set in any of the averaged dumps. The integration time is the difference of the dump time of the last dump and start time of the first one. Finally, the correlator dump is arranged to a complex column vector. The program flow is as follows:

```
read in the list of data files (filelist) in the data directory
find all files within the integration window
if no files found, then return. If no files left, set EOF true
loop through files
  read in files until one good found
    initialize the averaging variables, calculate start time etc.
  read in the remaining good files
    if certain parameters have changed, give a warning message and continue
    update the averaging variables
end loop through files
if at least one good file found
  store back to original variables (d_parbl and d_data)
  store variance estimates to (d_var1 and d_var2)
  display message about successful integration
```

/geo/gmt/askoh/guisdap/m152/integr_data.m

```
% GUISDAP v1.50   94-03-10 Copyright Markku Lehtinen, Asko Huuskonen
%
% Integration program for radar data in Matlab files. The files may either contain
% individual dumps or integrated dumps. The latter may or may not have
% the variance estimates of the data included.
% Each call to this routine performs the integation for one integration period
%
% NOTE: This is a EISCAT specific function
%
% Input parameters (all global):
% a_ind a_indold a_interval a_year a_start a_integr a_skip a_end
%     : specify the integration periods
% d_filelist : contains the names of the data files
% Output parameters:
% OK  : if true, the integration was succesful
% EOF : if true, the end of file was found during integration
% Output parameters (global):
% d_parbl : the parameter block returned by the integration program
% d_data : the integrated data vector
% d_var1 d_var2: data variances
%
% See also: an_start integr_NW integrate
% function  [OK,EOF]=integr_data
function  [OK,EOF]=integr_data

global d_parbl d_data d_var1 d_var2  data_path d_filelist a_control
global a_ind a_indold a_interval a_year a_start a_integr a_skip a_end

OK=0;EOF=0;
if a_ind==0
  a_ind=1;
  a_interval=a_start+[0 ,a_integr(1)];
else
   a_indold=a_ind;a_ind=a_ind+1; if (a_ind>length(a_integr)), a_ind=1;end
```

```
    a_interval=a_interval(2)+a_skip(a_indold)+[0 ,a_integr(a_ind)];
  end
  if a_interval(2)>=a_end; EOF=1; end
  if (d_filelist(length(d_filelist))<a_interval(1)), EOF=1; return, end,

  fixed=[1, 5:90 92:93 127:128];  % parameters which are not allowed to change
  averaged=[96:99];               % parameters which are averaged
  ORed=95;                        % parameter which is OR'ed

  for file=d_filelist(find(d_filelist>a_interval(1) & d_filelist<=a_interval(2)))'
    filename='00000000';str=int2str(file);filename(9-length(str):8)=str;
    load(canon([data_path, filename]))
    % For compatibility with some old integrated files
    if exist('n_preint'), i_averaged=n_preint; end

    [Md,Nd]=size(d_data);
    if Md==1,  % For compatibility with some incorrect data file, data as row vectors
  %    fprintf(' The file contains row vectors instead of columns\n')
      d_parbl=d_parbl(:);  d_data=d_data(:);
    end

    [secs,year]=tosecs(d_parbl(2:4));
    if (secs~=file | year~=a_year),
      disp('Filename conflicts with file contents or years do not match'), keyboard
    end

    if d_parbl(95)~=0, fprintf(' Status word is %g\n',d_parbl(95)), end
    dumpOK= rem(d_parbl(95),2)==0 & d_parbl(95)~=64;
    if ~OK & dumpOK,  % initialize with the first good dump
      first_parbl=d_parbl;          % save the first parameter block
      aver=d_parbl(averaged);       % initialize averaging
      status=d_parbl(ORed);         % save the status word
      starttime=secs-d_parbl(94);   % calculate starttime of first dump
      if (exist('pre_integrated') == 1),
        aver=d_parbl(averaged)*i_averaged;      % initialize averaging
        data=d_data;
        if exist('i_var1') % The integrated file need not have variances in it
          d_var1=i_var1(:);
          d_var2=i_var2(:);
          N_averaged=i_averaged;
        else
          d_var1=d_data.*d_data;
          d_var2=d_data.*conj(d_data);
          N_averaged=1;
        end
      else
        aver=d_parbl(averaged);       % initialize averaging
        lendata=length(d_data);
        d_data(lendata-1:lendata)=abs(d_data(lendata-1:lendata));
        d_data=d_data(1:2:lendata)+sqrt(-1)*d_data(2:2:lendata);
        data=d_data;                  % data is now a complex column vector
        d_var1=d_data.*d_data;        % for the variance calculations
        d_var2=d_data.*conj(d_data);
        N_averaged=1;
      end
      OK=1;
    elseif OK & dumpOK,  % update with the following files
      if any(d_parbl(fixed)~=first_parbl(fixed)),
        disp(' changes in the parameter block, continuing')
        indfixed=find(d_parbl(fixed)~=first_parbl(fixed));
        disp('          #   ,  original   , last dump')
        indfixed=fixed(indfixed);
        disp([indfixed',first_parbl(indfixed),d_parbl(indfixed)])
      end
      status=bitwiseor(status,d_parbl(ORed),16);
      if (exist('pre_integrated') == 1),
        aver=aver+d_parbl(averaged)*i_averaged;
        data=data+d_data;
        if exist('i_var1')
          d_var1=d_var1+i_var1(:);
          d_var2=d_var2+i_var2(:);
          N_averaged=N_averaged+i_averaged;
        else
          d_var1=d_var1+d_data.*d_data;
          d_var2=d_var2+d_data.*conj(d_data);
          N_averaged=N_averaged+1;
        end
      else
        aver=aver+d_parbl(averaged);
        d_data(lendata-1:lendata)=abs(d_data(lendata-1:lendata));
        d_data=d_data(1:2:lendata)+sqrt(-1)*d_data(2:2:lendata);
        data=data+d_data;
        d_var1=d_var1+d_data.*d_data;
        d_var2=d_var2+d_data.*conj(d_data);
        N_averaged=N_averaged+1;
      end
    end
  end

  if OK,  % if at least one good data dump was found
    if a_control(4)==1 & N_averaged<2,
      fprintf(' One file is not enough for variance determination\n')
      fprintf(' Skipping this integration period\n')
      fprintf(' command ''analysi_control(4)=2'' in the startup file will enable the analysis\n')
      OK=0;
      return
    elseif a_control(4)==1 & N_averaged<5,
      fprintf(' *******************    WARNING    **********************************\n')
      fprintf(' %.0f files may not be enough for reliable variance determination\n',N_averaged)
      fprintf(' *******************    WARNING    **********************************\n')
    end

    % update parameter block, accept the last parameter block as starting point
    d_parbl(averaged)=aver/N_averaged;
    d_parbl(ORed)=status;
    d_parbl(94)=secs-starttime;
    d_data=data;
    d_var1=d_var1-data.*data/N_averaged;
    d_var2=d_var2-data.*conj(data)/N_averaged;

  end
```

An alternative method is to integrate data directly from EISCAT raw data files. This is accomplished by a max-routine **integrate.mex**, programmed by Nigel Wade. Function **integr_NW** provides an interface to the mex call

/geo/gmt/askoh/guisdap/m152/integr_NW.m

```
% GUISDAP v1.50   94-03-10 Copyright Markku Lehtinen, Asko Huuskonen
%
% Interface to Nigel Wade's integration package (mex call)
% Parameters:
% a_integdeffile : the integration definition file
% OK  : if true, the integration was succesful
% EOF : if true, the end of file was found during integration
% global parameters:
% d_parbl : the parameter block returned by the integration program
% d_data : the integrated data vector
% d_var1 d_var2: data variances
%
% See also: an_start integr_data integrate
%
% function [OK,EOF]=integr_NW(a_integdeffile)
function [OK,EOF]=integr_NW(a_integdeffile)

global d_parbl d_data d_var1 d_var2 a_control

[status,d1,N_averaged,par,d_parbl,d_data,d_var1,d_var2,scan]=...
        integrate(['-id ',a_integdeffile]);

    if status==0,  OK=1; EOF=0;
elseif status>0,   OK=1; EOF=1;
elseif status<0,   OK=0; EOF=0; fprintf(' Error in integration\n'); return, end

% If no files found, the routine returns empty matrices, exit here
if length(N_averaged)==0, OK=0; return; end

d_data=d_data(:);
d_var1=d_var1(:)-d_data.*d_data/N_averaged;
d_var2=d_var2(:)-d_data.*conj(d_data)/N_averaged;

if a_control(4)==1 & N_averaged<2,
  fprintf(' One file is not enough for variance determination\n')
  fprintf(' Skipping this integration period\n')
  fprintf(' command ''analysi_control(4)=2'' in the startup file will enable the analysis\n')
  OK=0;
  return
elseif a_control(4)==1 & N_averaged<5,
  fprintf(' *******************    WARNING    **********************************\n')
  fprintf(' %.0f files may not be enough for reliable variance determination\n',N_averaged)
  fprintf(' *******************    WARNING    **********************************\n')
end
```

After succesfull integration, the EISCAT parameter block is interpreted and relevant information is transferred to corresponding GUISDAP parameters:

The following parameters are read and used later in analysis

`ch_az`	Antenna azimuth from parameter block
`ch_el`	Antenna elevation from parameter block
`ch_range`	Range to common volume for multistatic radars
`ch_Pt`	Transmitter power from parameter block (W)
`d_time`	Start and end times for data
`d_rcprog`	Radar controller program number, needed in loading the initfiles

The rest of the parameters are stored for checking purposes. The filter widths, for example, can be compared to the filter information read from the init file. Note that at the present version the frequencies are not compared to their counterparts in the initialization. This is, as must be confessed, a bug which will be repaired in the future versions. Another point to note is that the routine will not work for VHF without changes.

`ch_f`	channel frequency calculated from parameter block (kHz)
`ch_filt`	Filter width and type as coded in EISCAT
`ch_adc`	A/D conversion sample time

`/geo/gmt/askoh/guisdap/m152/decodeparblock.m`

```
% GUISDAP v1.50   94-03-10 Copyright Markku Lehtinen, Asko Huuskonen
%
% function  decodeparblock
% The relevant information in the parameter block is now transferred to
% GUISDAP variables
% NOTE: This is a EISCAT specific script
%
% See also: an_start, integr_data integr_NW

%global d_parbl d_rcprog d_time p_RECloc
%global ch_az ch_el ch_f ch_filt ch_adc ch_Pt ch_range name_site

if any(a_simul),
  d_rcprog=1;
  d_time(1,:)=toYMDHMS(a_year,a_start);
  d_time(2,:)=toYMDHMS(a_year,a_end);
  ch_Pt=a_simul(3)*ones(1,8);
  ch_range=a_simul(5)*ones(1,8);
  ch_az=a_simul(6)*ones(1,8);
  ch_el=a_simul(7)*ones(1,8);
  return
end

if d_parbl(128)>4 & d_parbl(128)<=10,
  % The site information
  names=['K';'T';'V';'S'];
  if name_site~='V'
    name_site=names(d_parbl(1));
  end

  % Put the remote receiver locations to GUP variables
  % Why? Because the remote sites may have common initialization file
  if name_site=='K'
    p_RECloc=[67.863, 20.44, .412];
  elseif name_site=='S'
    p_RECloc=[67.367, 26.65, .180];
  end


  % The radar controller program number
  d_rcprog=d_parbl(92);
  % endtime of integration
  [time,year]=tosecs(d_parbl(2:4));
  d_time(2,:)=toYMDHMS(year,time);
  % starttime of integration
  time=time-d_parbl(94);
  d_time(1,:)=toYMDHMS(year,time);
  fprintf('%4.0f/%2.0f/%2.0f  %2.0f:%2.0f:%2.0f',d_time(1,1:6))
  fprintf('-%2.0f:%2.0f:%2.0f integrated\n',d_time(2,4:6))

  % Antenna pointing direction
  ch_az=(d_parbl(6)/10)*ones(1,8);
  ch_el=(d_parbl(9)/10)*ones(1,8);

  % transmit frequencies for channels
  if d_parbl(13)>1000,    % upper first local oscillator for UHF
    ch_f=d_parbl(13)/10-d_parbl(19:26)'/100-30;
  elseif d_parbl(13)>800, % lower first local oscillator for UHF
    ch_f=d_parbl(13)/10+d_parbl(19:26)'/100-30;
  else
    error(' VHF experiment?, update decodeparblock')
  end
```

```

    % filter widths
    ch_filt(2,:)=d_parbl(35:42)'/10;
    % filter types are stored in two words, 4 bits reserved for each channel
    for i=1:4,  ch_filt(1,[4+i,i])=rem(floor(d_parbl(87:88)'/16^(i-1)),16);  end

    % analog-to-digital conversion intervals
    ch_adc=[d_parbl(78:79)/20;d_parbl(80:85)/10]';

    % transmitter power
    ch_Pt=ones(1,8)*d_parbl(99)*1000;
    if rem(d_parbl(127),2)==1,
      ch_Pt=ones(1,8)*max(d_parbl([99,101]))*1000;
    end

    % Range to the common volume (for remotes on multistatic radars only)
    range=d_parbl(11);
    if d_parbl(11)==0,
      if (name_site=='K' | name_site=='S')
        fprintf(' Distance to the common volume is not stored at the parameter block\n')
        fprintf(' Skipping this integration period\n')
        OK=0; return
      else
        range=1000;
      end
    end
    ch_range=range*ones(1,8)/10;

  else
    error('Unknown parameter block version, update decodeparblock')
  end
```

The radar constant

The system constant C is defined so that the received power P_R in units of K is obtained from the equation

$$\kappa B_W P_R = C P_T (\frac{R_0}{R})^2 l_{t,t} \frac{N_e/N_0}{(1+(k\lambda)^2)(1+(k\lambda)^2+T_e/T_i)},$$

where κ the Boltzmann constant, B_W the receiver bandwidth, P_T is the transmitted power, R_0 the reference range, R is the range to the gate, $l_{t,t}$ the effective pulse length in seconds, and N_0 the reference electron density. For monostatic case the constant C is then

$$C = \frac{P_0(R_0)}{P_T} A_{eff}(R_0) N_0 \frac{c}{2},$$

where $P_0(R_0)$, the power scattered by a single electron at the beam axis at range R_0, is given by Eq. 2.21 in Lehtinen&Huuskonen (JATP special Issue 1994, submitted) and A_{eff}, the effective beam cross section, is given by Eq. 2.36. Note that the defintion os P_0 by Eq. 2.21 contains the transmitted power, and therefore P_0 is divided by P_T in the equation above. If the reference range is 150 km and the reference electron density 10^{11}m^{-3}, the numeric value of C is $8.443 \cdot 10^{-19}$.

The system constant C_s is often (e.g. Kirkwood et al., 1986, JATP 48, 773-775) defined by the equation

$$N_e = \frac{C_s R^2 \kappa B_W P_R (1+(k\lambda)^2)(1+(k\lambda)^2+T_e/T_i)}{2 P_T l_{t,t}}.$$

The system constant definitions give

$$C_s = \frac{2N_0}{C R_0^2}.$$

The numeric value of C shown above gives a value of $1.0543 \cdot 10^{19}$ for C_s. This can now be compared with values obtained by comparing EISCAT with other instruments. Bjørnå and Kirkwood (1986, Ann. Geophys. 4, 137) obtained $1.18 \cdot 10^{19}$ by comparing ion–line and plasma–line measurements. The present EISCAT values (1992-93) of $0.95 \cdot 10^{19}$ for CP1 and $1.06 \cdot 10^{19}$ for CP3 have been obtained by comparision with ionosonde and dynasonde measurements. It seems evident that there remains an uncertainty of the order of 10% in the electron densities.

Function **radar_eq.m** calculates a coefficient

$$C(addr) = CP_T(\frac{R_0}{R})^2/(\kappa B_W)$$

for each correlator result memory location separately. It also includes a scaling factor which changes the effective pulse length from **p_dtau** units to meters. The procuct of the last two factors in the radar equation ($l_{t,t}$ and $N_e/N_0\ldots$) is obtained when the scaled spectrum is multiplied by the spectral ambiguity functions in **dirthe.m**. Then, to get the received power in units of K, it will only be necessary the multiply the products by the addresswise factor C(addr).

/geo/gmt/askoh/guisdap/m152/radar_eq.m

```
% GUISDAP v1.50   94-03-10 Copyright Markku Lehtinen, Asko Huuskonen
%
% radar equation for monostatic and bistatic cases
% function radar_eq
  function radar_eq

global ch_fradar  ch_gain  ch_Pt ch_az ch_el p_XMITloc p_RECloc
global v_lightspeed v_electronradius v_Boltzmann p_dtau p_R0 p_N0
global ad_coeff ad_range ad_w
global lpg_h lpg_w lpg_code lpg_bcs lp_vc vc_ch
global ADDR_SHIFT name_site
global sc_angle sc_R0 sc_R1 ch_range k_radar k_radar0

scale=(p_dtau*1e-6*v_lightspeed/2); % Scale factor from p_dtau units to meters

Cbeam=0.460; % This factor depends on the beam geometry
eff=0.651;
% Antenna efficiency is the ratio of the true gain and the
% theoretical gain 4*pi*A/lambda^2


% First find the ranges and the scattering angles from the common volume
% For monostatic case quite simply
% Check that distance from transmitter to receiver less than 100 m
if max(abs(gg2gc(p_XMITloc)-gg2gc(p_RECloc)))<.1
  sc_angle=pi;
  sc_R0=p_R0*scale;
  sc_R1=p_R0*scale;
  k_radar=k_radar0;

  % Effective beam cross section
  Aeff=Cbeam*eff*(4*pi*sc_R0^2)./ch_gain;

  % Volume for unit pulse length
  Veff=Aeff*scale;
  % Comment: Pulse lengths are also expressed in units of p_dtau
  %    Multiplication by scale is included here so that
  %    is is not needed in functions dirthe and power_prof
else
  ch=1;
  [gg_sp,angle,ranges]=loc2gg(p_RECloc,[ch_el(ch),ch_az(ch),ch_range(ch)],p_XMITloc);
  sc_angle=angle;
  sc_R0=ranges(2)*1E3;
  sc_R1=ranges(1)*1E3;

% Effective scattering volume in m^3 for the remotes
  P=(32*pi/0.3215)^2/3.63;
  Veff=(pi/P)^1.5*(sc_R0^2*sc_R1^2)/sqrt(sc_R0^2+sc_R1^2)/sin(sc_angle);
  fprintf(' Veff is %.4g\n',Veff)
  % Update range variables for the new geometry
   ind=find(lpg_bcs=='s');
   range=sc_R1/scale;
   lpg_h(ind)=range*ones(size(ind));
   lpg_w(ind)=(range/100)*ones(size(ind));
   ad=ADDR_SHIFT+lpg_addr(ind);
   ad_range(ad)=range*ones(size(ad));
   ad_w(ad)=(range/100)*ones(size(ad));
  % Update now the radar k
  k_radar=k_radar0*sin(sc_angle/2);

% Now testing the Uppsala remote equation
 if 0,
   lpg=find(lpg_bcs=='s'); lpg=lpg(1);
   w=wrlpg(lpg_lp(lpg));
   r=1:length(w);
   pp=round(sum(w'.*r)/sum(w));
   rr=round((sc_R0+sc_R1)/scale/2);
   r=r-pp+rr;
   global lpg_ND
   A=sqrt(3.63/2)*(0.3215/pi/32);
```

```
      TX=[ch_fradar(ch),ch_gain(ch),A,A];
      [Cs,r2,leff,wwr]=radar_MB(TX,TX,sc_R0/1E3,sc_R1/1E3,sc_angle,r*1e-6,w);
      leff=leff/lpg_ND(lpg)*1e6;
      Aeff=Cbeam*eff*(4*pi*sc_R0^2)./ch_gain; % This is for monostatic
      Veff=Aeff(1)*leff*scale;
      fprintf(' Veff is %.4g\n',Veff)
    end
  end

  lambda=v_lightspeed./ch_fradar;

  % Single electron cross section at the beam intersection for unit power
  polfac=1-0.5*sin(sc_angle)^2;
  %fprintf(' Polarization factor is %.2f\n',polfac)
  P0perPt=4*pi*(v_electronradius^2)*polfac*...
    (ch_gain./(4*pi*sc_R0^2)).*(ch_gain./(4*pi*sc_R1^2)).*(lambda.^2/(4*pi));

  % Multiplier for scale electron density and true power
  p_coeff0=P0perPt.*ch_Pt.*Veff*p_N0;
  % Calculate now the factor for virtual channels
  vc=find(vc_ch>0); % These virtual channels in use
  p_coeff0=p_coeff0(vc_ch(vc))./(v_Boltzmann*Ap(vc,0)/(p_dtau*1e-6));

  % For monostatic case calculate for each point the range factor
  for sig=find(lpg_bcs=='s')
    % Must find the virtual channel now
    lp=lpg_lp(sig);
    vc=lp_vc(lp(1));
    addr=ADDR_SHIFT+lpg_addr(sig);
    ad_coeff(addr)=p_coeff0(vc)*(sc_R1./(scale*ad_range(addr))).^2;
  end

  % Finally factor to compensate for all the inaccuracies in the numeric constants
  %Magic_constant=1.11;
  %ad_coeff=ad_coeff/Magic_constant;
```

Scaling and background subtraction

Before proceeding to the half profile analysis, all the data is divided by factors arising from the correlator algorithm (`lpg_ND`). This operation is needed because the correlator algorithm and hence the factors may be different for signal, background and calibration. This scaling is also done to the spectral ambiguity functions.

At the second stage, the data is scaled by the calibration measurements, so that all data is in units of K. The scaling factor is

$$\frac{T_{cal}}{P_{cal} - P_{back}},$$

where T_{cal} is the calibration temperature and the P's are the calibration and background powers for the virtual channel. This operation conveniently gives small numbers for presentation in the figures and an easy way to compare the signal strength to the system temperature.

/geo/gmt/askoh/guisdap/m152/scale_data.m

```
% GUISDAP v1.50   94-03-10 Copyright Markku Lehtinen, Asko Huuskonen
%
% Here the data points are divided by the correlator algorithm factors lpg_ND.
% By this procedure different signal/background/calibration algorithms are
% handled automatically. The spectral ambiguity functions are scaled
% accordingly by routine 'scale_lpgwom'
%
% Next the data is transformed to units of K with the calibration data
% This is not an optimal solution, and will be reconsidered in later releases
%
% Warning: NEVER perform this operation twice, as the resuls is returned
%          in the input variables 'd_data', 'd_var1' and 'd_var2'
% See also: an_start, scale_lpgwom

function scale_data

global lpg_ra lpg_ND lpg_cal lpg_bac lpg_T d_data d_var1 d_var2 ADDR_SHIFT

%*************** SCALING BY CORRELATOR ALGORITHM FACTORS *****************
data=d_data;
d_data=zeros(size(data));
for lpg=1:length(lpg_ra)  % go through all lag profile groups
  addr=lpg_addr(lpg);    % result memory addresses for lpg;
  addr=addr+ADDR_SHIFT; % To change from radar to Matlab addressing
  d_data(addr)=data(addr)/lpg_ND(lpg);
  d_var1(addr)=d_var1(addr)/(lpg_ND(lpg)*lpg_ND(lpg));
  d_var2(addr)=d_var2(addr)/(lpg_ND(lpg)*lpg_ND(lpg));
```

```
end

%*********************** AND BY CALIBRATION POWER ***********************
%calculate first scale for all calibration measurements
calibs=diff_val(lpg_cal);      % find all different values
calibs=calibs(find(calibs>0)); % Accept non-zero values
scale=zeros(size(lpg_cal));
for cal=calibs
  bac=lpg_bac(cal);
  bac_power=mean(d_data(lpg_addr(bac)+ADDR_SHIFT));
  cal_power=mean(d_data(lpg_addr(cal)+ADDR_SHIFT));
  scale(cal)=(cal_power-bac_power)/lpg_T(cal);
end

for lpg=1:length(lpg_ra)  % go through all lag profile groups
  cal=lpg_cal(lpg);
  addr=lpg_addr(lpg)+ADDR_SHIFT; % To change from radar to Matlab addressing
  d_data(addr)=d_data(addr)/scale(cal);
  d_var1(addr)=d_var1(addr)/(scale(cal)*scale(cal));
  d_var2(addr)=d_var2(addr)/(scale(cal)*scale(cal));
end
```

After the scaling operation the background can be subtracted from the signal in `subr_backgr`. In case a background estimate is found and subtracted, the data variances are also updated.

`/geo/gmt/askoh/guisdap/m152/subr_backgr.m`

```
% GUISDAP v1.50   94-03-10 Copyright Markku Lehtinen, Asko Huuskonen
%
% function to estimate and subtract the background component in the data
% uses the stored background lag profile group numbers in 'lpg_bac'
% Tries to include the subraction effect also in the variance estimate
%
% See also: an_start
%function subr_backgr
function subr_backgr

global d_data d_var1 d_var2 lpg_bac lpg_nt lpg_background ADDR_SHIFT

data=d_data;
var1=d_var1;
var2=d_var2;

%*********************** SUBTRACTING BACKGROUND ***************************
bacs=lpg_bac(lpg_bac>0);
bacs=diff_val(bacs);  % find all different values

for bac=bacs,
  addr=lpg_addr(bac)+ADDR_SHIFT; % To change from radar to Matlab addressing
  background=mean(data(addr));
   variance1=mean(var1(addr))/lpg_nt(bac);
   variance2=mean(var2(addr))/lpg_nt(bac);
  for lpg=find(lpg_bac==bac)
    addr=lpg_addr(lpg)+ADDR_SHIFT; % To change from radar to Matlab addressing
    lpg_background(lpg)=background;
    d_data(addr)=data(addr)-background;
    d_var1(addr)=var1(addr)+variance1;
    d_var2(addr)=var2(addr)+variance2;
  end
end
```

The initial values for plasma parameters

The initial values for the plasma parameters are defined by function **get_apriori**. It takes the the plasma parameters from function **ionomodel**. The electron density is calculated from the power measurements, if they exist. The calculation uses the temperature values supplied by the **ionomodel**.

/geo/gmt/askoh/guisdap/m152/get_apriori.m

```
% GUISDAP v1.50   94-03-10 Copyright Markku Lehtinen, Asko Huuskonen
%
% This function find the center altitudes of the analysis gates and calls
% ionomodel to get the model ionosphere for these altitudes.
% Next, all zero lag measurements are used to get an initial estimate for
% the electron density
%
% See also: ionomodel, power_prof range_to_height
function get_apriori(simul)

global a_addr a_adstart a_adend a_range a_priori a_priorierror ADDR_SHIFT
global lpg_bcs lpg_lag ch_el p_N0 ad_w  ad_range di_results di_figures
global pp_profile pp_range pp_sigma

if nargin==0, simul=0; end

for gate=1:length(a_adstart)
  range(gate)=mean(ad_range(a_addr(a_adstart(gate):a_adend(gate))+ADDR_SHIFT));
end
height=range_to_height(range,ch_el(1));
% form the a priori model for the analysis
[a_priori,a_priorierror]=ionomodel(height');
% change from physical to scaled variables
a_priori=real_to_scaled(a_priori);
a_priorierror=real_to_scaled(a_priorierror);

% Exit here, if the a_priori model is loaded for simulation purposes,
% or it should not be updated with the measured raw electron denstity
if simul, return, end

% Calculate electron density estimates from zero lags, if available.
ind=find(lpg_bcs=='s' & lpg_lag==0);
if length(ind)>0,
  [pp_profile,pp_range,pp_sigma]=power_prof(lpg_addr(ind)',0);
  pp_height=range_to_height(pp_range,ch_el(1));

  if  di_figures(2),
    figure(di_figures(2));set(gcf,'NextPlot','replace');clf reset
    p=plot(pp_profile*(p_N0/1e11),pp_height,'o');
    set(p,'MarkerSize',2)
    title('Ne with model Te/Ti')
    ylabel('Altitude/km'); xlabel('Raw electron density/1e11'), drawnow
  end

  % Update the a priori electron density
  for gate=1:length(a_adstart)
    ind=find(abs(pp_range-range(gate))<ad_w(ADDR_SHIFT+a_addr(a_adstart(gate))));
    if length(ind)>0,
      a_priori(gate,1)=mean(pp_profile(ind));
      a_priorierror(gate,1)=10*a_priori(gate,1);
    end
  end
end
```

The equation for the received power can be solved for the electron density. It gives the following third order equation:

$$N_e{}^3 - \sigma(1+\frac{T_e}{T_i})N_e{}^2 - \sigma A(2+\frac{T_e}{T_i})N_e - \sigma A^2 = 0,$$

where

$$\sigma = \frac{N_0 \kappa B_W P_r}{C P_T l_{t,t}}(\frac{R}{R_0})^2$$

and

$$A = k^2 \epsilon_0 \kappa_B T_e / e^2.$$

When solving the equation, the temperature values are taken from the a priori model. If the equation has more than one real root, the one closest to $\sigma(1+T_e/T_i)$ is chosen. The latter is the simple solution obtained when the Debye effects are neglected. When speed is important, it is possible to calculate that simple solution only.

/geo/gmt/askoh/guisdap/m152/power_prof.m

```
% GUISDAP v1.50   94-03-10 Copyright Markku Lehtinen, Asko Huuskonen
%
% Function calculates the electron density from a given set of measurements
% using the temperature ratio values in the model ionosphere
% Note that this calculation operates in physical units instead of scaled units
% Input parameters:
% addr: addresses to use
% Debyecorr: Two different options are available
%          The first one is faster but neglects Debye correction (Debyecorr=0)
%          The other one is slower but includes Debye correction (Debyecorr=1)
% output: pp_prof : Raw electron density with the a priori Te/Ti model
%         pp_range: The range to the center of gate
%         pp_sigma: Raw power profile with Te/Ti=1, excl. Debye term
%
% See also: get_apriori
%
% function [pp_prof,pp_range,sigma]=power_prof(addr,Debyecorr)
function [pp_prof,pp_range,sigma]=power_prof(addr,Debyecorr)

global ad_range ch_el d_data lpg_womscaled ad_lpg p_om ad_coeff
global v_epsilon0 v_Boltzmann v_elemcharge k_radar p_N0 ADDR_SHIFT

addr=addr+ADDR_SHIFT; % To change from radar to Matlab addressing
pp_range=ad_range(addr);
pp_height=range_to_height(pp_range,ch_el(1));
apriori=ionomodel(pp_height);

signal_power=real(d_data(addr));
len_eff=max(real(lpg_womscaled(ad_lpg(addr),:))')';
sigma=p_N0*signal_power./(ad_coeff(addr)'.*len_eff);

% Here one can choose of two different solutions
% The first one is faster but neglects Debye correction (Debyecorr=0)
% The other one is slower but includes Debye correction (Debyecorr=1)
% Note that this calculation operates in physical units instead of scaled
if Debyecorr,
  % solve the third order equation for electron density
  ratio=apriori(:,3);
  Te=apriori(:,2).*ratio;
  ch=1;  % hyi hyi
  A=(k_radar(ch))^2*v_epsilon0*v_Boltzmann*Te/(v_elemcharge)^2;
  B=-sigma.*(1+ratio);
  C=-sigma.*A.*(2+ratio);
  D=-sigma.*A.*A;
  for i=1:length(A);
    apu=roots([1,B(i),C(i),D(i)]);
    % choose the root closest to the first order solution equal to -B.
    [hups,ind]=min(abs(apu+B(i)));
    res(i,1)=apu(ind);
  end
else
  % Solve only the first order equation (neglect Debye effect)
  ratio=apriori(:,3);
  res=sigma.*(1+ratio);
end

pp_prof=res/p_N0;
pp_sigma=2*sigma;
```

The half profile analysis

The half profile analysis goes through the vector `a_addr` and performs the analysis assuming that the plasma parameters are constant in the altitude range covered by the range ambiguity functions of the crossed products included in each gate. The program flow is as follows:

```
loop through gates
  divide addresses into zero lag measurements (addr1) and others (addr2)
  select the spectral ambiguity functions (f_womega)
  select the radar equation factors (p_coeff)
  form the data vector for the fit (measurement)
  initialize the fit variables (aa)
  form the variance vector(variance) either
    from empirical variance estimates calculated during integration
    or by calculating those from theory
  update the fit variables (aa)

  call the fit routine
    if mex routines not in use (mrqmn)
    if mex routines available (mrqmndiag)
  store results, print them on console and dipsplay graphically (store_results)
end loop through gates
```

/geo/gmt/askoh/guisdap/m152/half_prof.m

```
% GUISDAP v1.50   94-03-10 Copyright Markku Lehtinen, Asko Huuskonen
%
% Main analysis routine, NOT documented yet, sorry
%function half_prof
function half_prof
t_start(2)

global a_addr a_adstart a_adend a_control ad_lpg ad_coeff ad_range ADDR_SHIFT
global d_data d_var1 d_var2 d_time p_rep p_dtau ch_el dp_comp di_fit dp_fit
global lpg_ND lpg_lag lpg_womscaled lpg_bac lpg_cal lpg_background
global a_priori a_priorierror f_womega p_coeffg p_ND
global r_range r_status r_ind g_ind
global pldfvv p_D0 p_om k_radar k_radar0

r_ind=0;
for g_ind=1:length(a_adstart)
  r_ind=r_ind+1;
  % ADDR_SHIFT is added to result memory addresses
  % in order to transfer from radar indexing to Matlab indexing
  addr=a_addr(a_adstart(g_ind):a_adend(g_ind));
  lpgs=ad_lpg(addr+ADDR_SHIFT); % These are the lag profile groups of the data points

  f_womega=[real(lpg_womscaled(lpgs,:));imag(lpg_womscaled(lpgs,:))];
  p_coeffg=[ad_coeff(addr+ADDR_SHIFT),ad_coeff(addr+ADDR_SHIFT)]';

  signal_acf=[real(d_data(addr+ADDR_SHIFT));imag(d_data(addr+ADDR_SHIFT))];
  measurement=[signal_acf;col(a_priori(g_ind,1:5))];

  % Take starting point from the a priori model, temperature values are taken from
  % the previous gate (if previous fit was successful and gate below the present one).
  aa=a_priori(g_ind,:);
  if r_ind>1, if mean(ad_range(addr+ADDR_SHIFT))>=r_range(r_ind-1) & r_status(r_ind-1)==0,
    aa(2:3)=result(2:3); aa(1)=aa(1)*(1+result(3))/(1+a_priori(r_ind,3));
  end, end

  if a_control(4)==1, % Using variance estimates calculated from data
      signal_var=real([d_var1(addr+ADDR_SHIFT)+d_var2(addr+ADDR_SHIFT);...
                       d_var2(addr+ADDR_SHIFT)-d_var1(addr+ADDR_SHIFT)])/2;
  elseif a_control(4)==2  % calculating variance estimates from ambiguity functions
%       t=clock;
      lpgbac=lpg_bac(diff_val(lpg_cal(lpgs)));
ind=find(lpg_lag(lpgbac)==0);
      Tback=mean(lpg_background(lpgbac(ind)));
      [covRe,covIm]=adgr_var(addr,Tback,aa);
      ND2=lpg_ND(lpgs).^2;
      % The variance scaling is calculated from the integration time here
      % In fact, it should be based on the loop counter value stored to the data dump
      Int_time=(d_time(2,3:6)-d_time(1,3:6))*[24*3600; 3600; 60 ; 1];
      if Int_time<0, Int_time=Int_time+24*3600; end
      Var_scale=(p_rep*p_dtau*1e-6/Int_time)/p_ND;
      signal_var=[covRe./ND2,covIm./ND2]'.*Var_scale;
```

```
%      fprintf(' Time used in variance calculation %.1f s\n',etime(clock,t))
  end
  % The variance for zero lag imaginary parts is zero. These must be corrected
% to some non-zero value in order to program to work.
  ind=find(signal_var==0); % Assume here diagonal variance
signal_var(ind)=eps*ones(size(ind));
  variance=[signal_var; (a_priorierror(g_ind,1:5)').^2];
  r_range(r_ind)=sum(ad_range([addr,addr]+ADDR_SHIFT)./signal_var')/sum(1 ./signal_var);

  ch=1;  kd2=k_radar(ch)^2*p_D0^2; Fscale=k_radar0(ch)/k_radar(ch); % hyi hyi
  [small_f_womega,small_p_om]=find_om_grid(aa,p_coeffg,f_womega,kd2,p_om,pldfvv);

  errorlim=a_control(1); status=0;
  if errorlim>0 & errorlim<10000 % To prevent unnecessary error estimation
    % Check if the error of Ne larger than given limit when the fit is started
    [error,correl,alpha]=error_estimate(aa,variance,kd2,p_coeffg,...
                                    small_f_womega,Fscale*small_p_om,pldfvv);
    if error(1)/aa(1) > errorlim, result=aa; chi2=inf; status=2 ; end  % No fit done
  end
  if status==0, % Now proceed to the fitting routine
    tol=a_control(2); maxiter=a_control(3);
    t_start(3),
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    [result,chi2,iter,alpha]=mrqmndiag(aa,measurement,variance,tol,maxiter,...
             kd2,p_coeffg,small_f_womega,Fscale*small_p_om,pldfvv);
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    status=(iter>=maxiter);
    t_stop(3),
  end
  store_results(aa,measurement,variance,result(1:5),alpha,chi2,status)
end

t_stop(2)
```

The result is transferred into physical units and the error estimation is done in the function `store_results`. It also prints the results to the console, stores the results into result variables starting with `r_` and produces a plot, which shows the data with the best fit theoretical values and the *a priori* and *a posteriori* values of the plasma parameters

/geo/gmt/askoh/guisdap/m152/store_results.m

```
% GUISDAP v1.50   94-03-10 Copyright Markku Lehtinen, Asko Huuskonen
%
% Internal routine to print and plot analysis results after each gate
%
% See also: half_prof
% function store_results(aa,meas,var,result,alpha,chi2,status)
  function store_results(aa,meas,var,result,alpha,chi2,status)

global a_priori a_priorierror ad_lpg p_RECloc
global ch_el di_fit g_ind
global lpg_lag di_figures
global r_range r_status r_param r_dp r_error r_res r_apriori r_apriorierror r_ind
global k_radar k_radar0 p_D0 p_coeffg f_womega p_om pldfvv

% Scale residual and Xfer results to physical units
comp=aa(6);
aa=aa(1:5);
len=length(meas)-length(aa);
chi2=chi2/len;
err=covm2vec(alpha);
res=scaled_to_real(result);
er=err;er(1:5)=scaled_to_real(err(1:5));
height=range_to_height(r_range(r_ind),ch_el(1));

% Print results to the console
if rem(r_ind,20)==1,
fprintf(' alt   Ne/1e11      Ti     Te/Ti      coll     velocity [O+]/Ne resid status\n')
end
fprintf('%5.1f',height);
fprintf(' %4.2f:%4.2f',res(1)/1e11,er(1)/1e11);
fprintf(' %4.0f:%3.0f',res(2),er(2));
fprintf(' %4.2f:%4.2f',res(3),er(3));
fprintf(' %4.1f:%4.1f',res(4)/1e3,er(4)/1e3);
fprintf(' %6.1f:%4.1f',res(5),er(5));
fprintf(' %5.2f '     ,comp);
fprintf(' %4.2f  ',chi2);
if status==0, str='OK';
elseif status==1 str='Max iter';
elseif status==2 str='No fit done';
end
fprintf(str);
fprintf('\n')

```

```
% Store results to result variables
r_param(r_ind,:)=res;
r_dp(r_ind,:)=comp;
r_error(r_ind,:)=er;
r_res(r_ind,:)=[chi2,sqrt(2/len)];
r_status(r_ind,:)=status;
r_apriori(r_ind,1:6)=scaled_to_real(a_priori(g_ind,:));
r_apriorierror(r_ind,1:6)=scaled_to_real(a_priorierror(g_ind,:));

if di_figures(3),
  ch=1;  kd2=k_radar(ch)^2*p_D0^2;  Fscale=k_radar0(ch)/k_radar(ch); % hyi hyi
  l_om=length(p_om);
  dom=0.5*[p_om(2)-p_om(1); p_om(3:l_om)-p_om(1:l_om-2); p_om(l_om)-p_om(l_om-1)]';
  [M,N]=size(f_womega);
  womega=f_womega.*dom(ones(M,1),:);
  theo=dirthe([result,comp],p_coeffg,womega,kd2,Fscale*p_om,pldfvv);
  res_err=err(1:5)';
  sig_err=sqrt(var);
  indr=1:len/2;
  indi=len/2+1:len;
  indp=len+(1:5);
  sig_r=meas(indr);err_r=sig_err(indr);
  sig_i=meas(indi);err_i=sig_err(indi);
  sig_p=meas(indp);err_p=sig_err(indp);
  fitted_r=theo(indr);
  fitted_i=theo(indi);
  fitted_p=theo(indp);
  indi=indi-length(indi);
  indp=1:5;

  figure(di_figures(3));clf

  set(gca,'Position',[.1 .1 .7 .8],'NextPlot','replace');
  plot(0,-1,'b.',indr,sig_r,'ro',indr,fitted_r,'g-'),hold on
  plot(indi+.2,sig_i,'bo',indi+.2,fitted_i,'g-');
  plot([indr;indr],[sig_r-err_r,sig_r+err_r]','r-')
  plot([indi+.2;indi+.2],[sig_i-err_i,sig_i+err_i]','b-')
  set(get(gca,'Children'),'MarkerSize',4)
  title(' Data  (o) and fit results (solid line)');
  ylabel('Power [K]'); xlabel(' # of data point');

  axes('Position',[.85 .1 .12 .8],'NextPlot','replace');
  plot(indp-0.15,sig_p,'ro',indp+0.15,fitted_p,'go'),hold on
  plot([indp;indp]-0.15,[sig_p-err_p,sig_p+err_p]','r:',...
         [indp;indp]+0.15,[fitted_p-res_err,fitted_p+res_err]','g-')
  set(get(gca,'Children'),'MarkerSize',4)
  axis([0 6 -1 max([5,ceil(fitted_p')])]);
  set(gca,'Xtick',1:5,'XTickLabels',['N';'T';'r';'c';'v']);
  title('parameters')
  drawnow
end
```

Theoretical autocorrelation function

The theoretical autocorrelation function is calculated in `dirthe`. It multiplies the theoretical spectrum with the spectral ambiguity function `f_womega` and then with the scaling factor `p_coeffg`, so that the output is in units of K. The calculation of spectrum is explained in a separate document

/geo/gmt/askoh/guisdap/m152/dirthe.m
```
% GUISDAP v1.50   94-03-10 Copyright Markku Lehtinen, Asko Huuskonen
%
% DIRect THEory for Inocherent Scatter radar measurements
% Available also as a fast mex-version
% Input parameters:
% param: plasma parameters in scaled units
% p_coeffg: radar factor for each measurement
% f_womega: spectral ambiguity function for each measurement
% kd2: (k D)^2, k is radar k-vector and D is Debye length for scale parameters
% p_om: frequency axis for f_womega
% pldfvv: plasma dispersion function interpolation table
% Output parameter:
% theo: theoretical values for the measurements
% See also: spec, transf
function theo=dirthe(param,p_coeffg,f_womega,kd2,p_om,pldfvv);

t_start(4)

param=real(param); % hyi hyi

% The allowed ranges of the scaled parameters are defined here
param(1)=max(0.005,param(1));
param(2)=min(100,max(0.1,param(2)));
param(3)=min(8,max(0.20,param(3)));
param(4)=max(param(4),0);

[nin0,tit0,mim0,psi,vi]=transf(param);
s=spec(nin0,tit0,mim0,psi,vi,kd2,p_om,pldfvv);

theo=[p_coeffg.*(f_womega*s);col(param(1:5))];
t_stop(4)
```

Other routines called by an_start

The Matlab indexing starts from 1, which may be different from the radar indexing. This is handled by `get_ADDRSHIFT.m` which produces a global variable, which is always added to address variables before a reference to Matlab matrices is done.

/geo/gmt/askoh/guisdap/m152/get_ADDRSHIFT.m
```
% GUISDAP v1.50   94-03-10 Copyright Markku Lehtinen, Asko Huuskonen
%
% ADDR_SHIFT is needed as indices to Matlab matrices start from 1 and the first address
% used by the radar can be diffrent.  For instance it is 0 for the EISCAT radars
% GUISDAP uses the true result memory addresses, until a reference to a Matlab-matrix
% is needed. At that point ADDR_SHIFT is added to the address.
% The routine assumes that the base address is zero. If that is not the case the
% value of the base address must be supplied using the global parameter p_baseaddr
% This parameter is meaningful only for data analysis
%
% function get_ADDRSHIFT.m
function get_ADDRSHIFT

global ADDR_SHIFT p_baseaddr

if length(p_baseaddr)==0,
  p_baseaddr=0; % This is the first result memory address used by EISCAT
end
ADDR_SHIFT=max(0,1-p_baseaddr);
```

`Load_filelist` checks whether the data directory contains EISCAT raw data files or binary matlab files. The former ones have an extension `.dtst` whereas the latter ones end with `.mat`. In the latter case, the filelist file is also loaded.

/geo/gmt/askoh/guisdap/m152/load_filelist.m
```
% GUISDAP v1.50   94-03-10 Copyright Markku Lehtinen, Asko Huuskonen
%
% Routine to check whether data comes in EISCAT .dtst format of binary Matalb files
% Output parameters:
% rawdata: true if .dtst files found on the data_path
% filelist: contains the dump times, if matlab files were found
%
```

```
% See also: integr_data integr_NW chk_par1
%
%function  [filelist,rawdata]=load_filelist
function  [filelist,rawdata]=load_filelist

global data_path

OK=0; rawdata=0;
file=canon([data_path 'filelist'],0);
if exist([file,'.mat'])==2,
  OK=1;eval(canon(['load ', file, '.mat -ascii']))
elseif exist([file,'.dat'])==2,
  OK=1;load([file,'.dat'])
else
  files=ls(canon(data_path,0));
  for ind=find(files=='d')
    if files(ind-1:ind+3)=='.dtst'
      rawdata=1; OK=1;break
    end
  end
end

if OK & rawdata
  fprintf('*************************************************************\n')
  fprintf(' Data directory contains EISCAT raw data files\n')
  fprintf(' Data will be integrated by Nigel Wade''s integration front end\n')
  fprintf('*************************************************************\n')
elseif OK & ~rawdata
  fprintf('*************************************************************\n')
  fprintf(' Data directory contains matlab files\n')
  fprintf(' Data will be integrated by GUISDAP integration routine\n')
  fprintf('*************************************************************\n')
  filelist=diff_val(filelist);
else
  fprintf('*************************************************************\n')
  fprintf([' No data found in directory ', data_path, '\n Contents of the directory is:\n'])
  fprintf(ls(canon(data_path,0)))
  fprintf('*************************************************************\n')
  error(' ')
end
```

Loading of initialization file. The routine first checks, if there is a special init file corresponding to `d_rcprog` (radar controller program number) – in this case the file name ends with that number. (In experiments with only one init file, these are not numbered at all).

It is necessary to create many init files in experiments with many radar controller programs, because pulse timing differs from program to program.

/geo/gmt/askoh/guisdap/m152/load_initfile.m

```
% GUISDAP v1.50   94-03-10 Copyright Markku Lehtinen, Asko Huuskonen
%
% A script to load the ambiguity functions etc. into the workspace for analysis
% The script assumed that variables name_expr and name_site exist in the workspace
% The script contains reference to EISCAT remote sites. However, the script works
% without modifications for other radars as long as name_site is different from K and S
%
% See also: path_expr save_toinitfile
%
clear vcg_Aenv

temp=[path_expr name_expr name_site];
if name_site=='K' | name_site=='S';
  temp=[path_expr name_expr 'R'];
end
if exist('d_rcprog')==1, rcp=d_rcprog; else rcp=0; end
if exist(canon([temp '_' int2str(rcp) 'init.mat'],0))==2,
  eval(canon(['load ' temp '_' int2str(rcp) 'init']))
elseif exist(canon([temp 'init.mat'],0))==2,
  eval(canon(['load ' temp 'init']))
else
  fprintf(['\n\n\n File    ', canon([temp 'init.mat'],0),'    not found \n\n\n'])
  error(' ')
end
if GUP_iniver<1.52,
  fprintf('*\n*\n* Files produced by GUP version %.2f not usable\n', GUP_iniver)
  fprintf('* Please, reinitialize the experiment with GUP 1.52 or later\n*\n*\n')
  error(' ')
end

if exist('vcg_Aenv')
  vc_Aenv=vcg_Aenv(:,vc_group);
  vc_Ap=vcg_Ap(:,vc_group);
  vc_Apenv=vcg_Apenv(:,vc_group);
```

```
    vc_penv=vcg_penv(:,vc_group);
    vc_penvabs=vcg_penvabs(:,vc_group);
    clear vcg_Aenv vcg_Ap vcg_Apenv vcg_penv vcg_penvabs
  end
  clear rcp temp
```

The routine **form_adpar.m** calculates certain parameters for all those correlator address memory locations, which contain signal crossed products. The parameters are:

ad_range	Range to gate
ad_code	Modulation type
ad_coeff	Factor in the radar equation, to be calculated in radar_eq.m
ad_w	Width of the range ambiguity function
ad_lpg	Lag profile group number

/geo/gmt/askoh/guisdap/m152/form_adpar.m

```
  % GUISDAP v1.50   94-03-10 Copyright Markku Lehtinen, Asko Huuskonen
  %
  % This internal routine calculates certain parameters for each memory location in advance
  % so that they can be obtained rapidly later
  % ad_range : Range to the center of ambiguity function
  % ad_w     : Width of the ambiguity functions
  % ad_code  : Code number
  % ad_lpg   : Lag profile group number
  % ad_coeff : The radar factor (to be calculated in radar_eq)
  %
  function form_adpar

  global lpg_ra lpg_nt lpg_ri lpg_bcs lpg_h lpg_dt lpg_w lpg_code
  global ad_range ad_w ad_code ad_lpg ad_coeff ADDR_SHIFT

  len=max(lpg_ra+(lpg_nt-1).*lpg_ri)+1;
  ad_range=zeros(1,len); ad_w=zeros(1,len);
  ad_code=zeros(1,len);  ad_lpg=zeros(1,len);
  ad_coeff=zeros(1,len);  % used in radar_eq
  for sig=find(lpg_bcs=='s')
    addr=lpg_addr(sig)+ADDR_SHIFT; % To change from radar to Matlab addressing
    len=length(addr);
    ad_range(addr)=lpg_h(sig)+(0:lpg_nt(sig)-1)'*lpg_dt(sig);
    ad_w(addr)=lpg_w(sig)*ones(len,1);
    ad_code(addr)=lpg_code(sig)*ones(len,1);
    ad_lpg(addr)=sig*ones(len,1);
  end
```

The main fit routine uses the Levenberg–Marquardt algorithm. The implemented version accepts either a full covariance matrix or a vector containing the diagonal.

/geo/gmt/askoh/guisdap/m152/mrqmn.m

```
  % GUISDAP v1.50   94-03-10 Copyright Markku Lehtinen, Asko Huuskonen
  %
  % See also: mrqmndiag
    function [aa,chi2,its,alpha,OK]=...
            mrqmn(a,ym,variance,ftol,itmax,kd2,p_coeffg,f_womega,p_om,pldfvv,errorlim)

    comp=a(6);a=a(1:5);
    if nargin==10, errorlim=inf; end
    OK=1;
    if min(size(variance))==1, diagonal=1;end
    if diagonal,
      invsigma=1 ./variance;
      apu=invsigma*ones(1,length(a));
    else
      invsigma=inv(variance);
    end
    lambda=0.001;
    aa=a; validder=0;
    ya=dirthe([aa,comp],p_coeffg,f_womega,kd2,p_om,pldfvv);
    dyda=zeros(length(ya),length(aa));
    if diagonal,
      chi2=(ym-ya)'*(invsigma.*(ym-ya));
    else
      chi2=(ym-ya)'*invsigma*(ym-ya);
    end
    its=0;
    while its<itmax
      if ~validder
        its=its+1;
        for i=(1:length(aa))
          aa2=aa; aa2(i)=aa(i)+0.0001;
          dyda(:,i)=(ya-dirthe([aa2,comp],p_coeffg,f_womega,kd2,p_om,pldfvv))/0.0001; %calculate deriva-
tives;
```

```
        end
        if diagonal,
          alpha=dyda'*(apu.*dyda);
          beta=dyda'*(invsigma.*(ya-ym));
        else
          alpha=dyda'*invsigma*dyda;
          beta=dyda'*invsigma*(ya-ym);
        end
       if its==1 error=sqrt(diag(inv(alpha)));
         if error(1)>errorlim, OK=2; return,  end
       end
      end
      da=((alpha+lambda*eye(size(alpha)))\beta)';
  %    disp(da);
      chi2old=chi2; yaold=ya; aaold=aa;
      aa=aa+da;
      ya=dirthe([aa,comp],p_coeffg,f_womega,kd2,p_om,pldfvv);
      if diagonal,
        chi2=(ym-ya)'*(invsigma.*(ym-ya));
      else
        chi2=(ym-ya)'*invsigma*(ym-ya);
      end
      if chi2>=chi2old
        chi2=chi2old; aa=aaold; ya=yaold; lambda=lambda*10; validder=1;
        if max(abs(da))<ftol, OK=0; break, end
      else
        lambda=lambda/10; validder=0;
        if max(abs(da))<ftol, OK=0; break, end
      end
    end
  end
  aa=[aa,comp];
  alpha=inv(alpha);
```

script calculating the radar k vector (`k_radar`) and the Debye length (`p_D0`) for the reference temperature (`p_T0`) and density (`p_N0`).

/geo/gmt/askoh/guisdap/m152/constants.m

```
% GUISDAP v1.50   94-03-10 Copyright Markku Lehtinen, Asko Huuskonen
%
% script defining useful radar constants
%
% See also: nat_const

k_radar0=2*pi*2*ch_fradar/v_lightspeed;
p_D0=sqrt(v_epsilon0*v_Boltzmann*p_T0/(p_N0*v_elemcharge^2));
p_om0=k_radar0*sqrt(2*v_Boltzmann*p_T0/(p_m0(1)*v_amu));
```

The following script defines all global parameters

/geo/gmt/askoh/guisdap/m152/globals.m

```
% GUISDAP v1.50   94-03-10 Copyright Markku Lehtinen, Asko Huuskonen
%
% This script defines global variables needed in the data analysis
%
% See also: glob_EISCAT, glob_GUP, start_GUP

% Define first the global variables used in the initialization
glob_GUP

% These globals are needed only in the analysis

global a_priori a_priorierror p_ND

global ad_range ad_w ad_code ad_lpg ad_coeff ADDR_SHIFT

global ch_az ch_el ch_f ch_filt ch_adc ch_Pt ch_scangle ch_range

global d_data d_parbl  d_rcprog d_time d_filelist
global d_var1 d_var2 % sig_var1 sig_var2

global a_addr a_adstart a_adend a_control a_ind
global a_year a_start a_integr a_skip a_end
global di_figures figure_fit
global dp_comp dp_comppost dp_fit

global lpg_womscaled  k_radar

% global p_coeff0 p_coeffg f_womega

 global r_range r_param  r_dp r_error r_res r_status
 global r_apriori r_apriorierror r_h r_time r_XMITloc r_RECloc r_SCangle

```

```
% Global definitions for the spectrum calculations
 global pldfvv  pldfv % tabc2powc  powc

% Global variables for full profile analysis grids
%global ug_r ug_ip ug_p eg_r

```

The two routines `chk_par1` and `chk_par2` form an interface between the internal GUISDAP parameters ane the parameters supplied by the user in the startup files.

/geo/gmt/askoh/guisdap/m152/chk_par1.m

```
% GUISDAP v1.50   94-03-10 Copyright Markku Lehtinen, Asko Huuskonen
%
% chk_par1 is the main interface between the user supplied control parameters and
% the internal GUISDAP control parameters. There is mostly a one-to-one correspondence
% between the parameters e.g.
% User parameter   GUISDAP parameter
% analysis_altit    a_altit
% analysis_integr   a_integr
% display_figures   di_figures
% integ_deffile     a_integdeffile
% The user parameters are local to the workspace whereas the GUISDAP parameters are global
% If any of the user parameters is not specified, or empty, the corresponding GUISDAP
% parameter will get the values specified in the routine.
%
% See also: an_start globals chk_par2 load_filelist

% The first if-block tries to locate the data source and produces variables
% necessary for integration:
a_simul=[]; % If this variable present, the data is produced from theory (So clear it)
% The first 'if' looks for the integration definition file for Nigel Wade's integration
% package. If this variable is present and the file exists, the data comes through that package.

if exist('integ_deffile')
  if exist(canon(integ_deffile,0))
    % The first branch looks for the integration definition file for Nigel Wade's integration
    % package. If this variable is present and the file exists, the data comes through that package.
    a_integdeffile=canon(integ_deffile);
    a_rawdata=1; % This flags that NW's package is to be used
    fprintf('*************************************************************\n')
    fprintf(' An integration definition file found\n')
    fprintf(' Data will be integrated by Nigel Wade''s integration front end\n')
    fprintf('*************************************************************\n')
  else
    fprintf('*************************************************************\n')
    fprintf([' File ', integ_deffile,' not found\n'])
    fprintf(' Stopping execution\n')
    fprintf('*************************************************************\n')
    error(' ')
  end
elseif exist('analysis_simul')
  % The second branch is for simulated data to be calculated from the theory
  % The presence of the variable a_simul is the flag, which is used later. Contents
  % a_simul(1): integration time
  % a_simul(2): controls the start time in steps of 7200 s, i.e. 2 hours.
  % a_simul(3): The transmitter power
  % a_simul(4): The background temperature
  % a_simul(5:6): Antenna azimuth and elevation
  a_simul=[0 1 1.2e6 100 300 180 90 ];
  a_simul(1:length(analysis_simul))=analysis_simul;
  a_year=2222;
  a_start=7200*a_simul(2);
  a_end=7200*a_simul(2)+a_simul(1);
else
  % The final branch is the normal case. Call to load_filelist checks whether EISCAT  .dtst
  % files are present at the data_path (a_rawdata=1) or whether matlab files are there
  [d_filelist,a_rawdata]=load_filelist;
  a_start=tosecs(analysis_start);  % the programs uses internally only seconds,
  a_end  =tosecs(analysis_end);    % counted from the beginning of year
  a_year=analysis_start(1);
  a_ind=0;

  if exist('analysis_integr'),
      a_integr=analysis_integr;
  else
    % If analysis_integr not present, produce integration times from the dump times
    % Then no (further) integration is performed by GUISDAP
    if a_rawdata
      a_integr=2;  % This should make sure that all dumps are analyzed separately
    else % This is for Matlab files. Note that the files may contain integrated data
      a_integr=diff([a_start d_filelist(find(d_filelist>a_start & d_filelist<=a_end))']);
    end
    a_skip=zeros(size(a_integr));
  end
```

```
    if exist('analysis_skip'),
      a_skip=analysis_skip;
    else                            % if skips were not defined, assume
      a_skip=zeros(size(a_integr));  % them to be zero
    end

    if a_rawdata
      % If EISCAT .dtst files were found, the integration definition file is produced here
      % The code transfers the contents of the GUISDAP control parameters to the control
      % parameters of Nigel's analysis package. The standard header lines are read first:
      fid=fopen(canon([path_GUP,'matfiles/integdefs']),'r');
      aa=fread(fid,inf,'char');
      fclose(fid);
      a_integdeffile=canon([path_tmp,'integdef'],0);
      fid=fopen(a_integdeffile,'w');
      fwrite(fid,aa,'char');
      if name_site=='T'; fprintf(fid,['data-source tromso\n']);
      elseif name_site=='K'; fprintf(fid,['data-source kiruna\n']);
      elseif name_site=='S'; fprintf(fid,['data-source sodankyla\n']);
      elseif name_site=='V'; fprintf(fid,['data-source vhf\n']);
      end
      fprintf(fid,['input-names ', canon(data_path),'\n']);
      fprintf(fid,'start-time %.0f:%0.f:%.0f',analysis_start(4:6));
      fprintf(fid,' %0.f/%.0f/%0.f\n', analysis_start(1:3));
      fprintf(fid,'end-time %.0f:%0.f:%.0f',analysis_end(4:6));
      fprintf(fid,' %0.f/%.0f/%0.f\n', analysis_end(1:3));
      fprintf(fid,'cycle-time %0.f\n', sum(a_integr)+sum(a_skip));
      starts=[0,cumsum(a_integr+a_skip)];
      for i=1:length(a_integr)
        fprintf(fid,'scan-pos timed %0.f  %0.f\n',starts(i), a_integr(i));
      end
      fclose(fid);
    end
  end

  a_control=[100000, 0.01, 6 1];
  % a_control(1)  No fit is tried, if the error of Ne is larger than (1) at the start
  % a_control(2)  Fitting is stopped when step for all parameters is less than (2)
  % a_control(3)  Maximum number of iterations
  % a_control(4)  Variance calculation
  %               = 1 when variance estimated from data
  %               = 2 when variance estimated using ambiguity functions
  if exist('analysis_control')
    ind=find(analysis_control>0);
    a_control(ind)=analysis_control(ind);
  end
  if any(a_simul)
    a_control(4)=2;
  end

  %dp_comp=[0:.1:1];
  %dp_fit=0;
  %if exist('discretepar_fit')==1,
  %  dp_fit=discretepar_fit;
  %end

  %if length(dp_comp)>36,
  %  fprintf('Too many discrete parameter values\n')
  %  error('Contact your local GUispert')
  %end

  di_figures=[1 1 1 1];
  if exist('display_figures'),
    di_figures(1:length(display_figures))=display_figures;
  end
  if exist('display_fit'),  % For compatibility with GUISDAP 1.44 and earlier
    di_figures(3)=display_fit;
  end
  if exist('display_results'), % For compatibility with GUISDAP 1.44 and earlier
    di_figures(4)=display_results;
  end

  % check that the various path names have directory separators at the end
  len=length(data_path); char=data_path(len);
  if char~=':' & char~='/' & char~='\', data_path(len+1)='/';end

  len=length(result_path); char=result_path(len);
  if char~=':' & char~='/' & char~='\', result_path(len+1)='/';end

  len=length(path_GUP); char=path_GUP(len);
  if char~=':' & char~='/' & char~='\', path_GUP(len+1)='/';end

  len=length(path_exps); char=path_exps(len);
  if char~=':' & char~='/' & char~='\', path_exps(len+1)='/';end

```

```
len=length(path_tmp); char=path_tmp(len);
if char~=':' & char~='/' & char~='\', path_tmp(len+1)='/';end


```

/geo/gmt/askoh/guisdap/m152/chk_par2.m

```
% GUISDAP v1.50   94-03-10 Copyright Markku Lehtinen, Asko Huuskonen
%

if exist('analysis_classic')==1,
  a_classic=analysis_classic;
else
  a_classic=0;
end

fac=1000/(v_lightspeed/2)/(p_dtau*1e-6);
if exist('analysis_range')==1,
  a_range=analysis_range*fac;
elseif exist('analysis_altit')==1,
  a_range=height_to_range(analysis_altit,ch_el(1));
else
  a_range=[0 100000];
end

if exist('analysis_addr')==1,
  a_addr=analysis_addr;
  a_adstart=analysis_adstart;
  a_adend=[a_adstart(2:length(a_adstart))-1,length(a_addr)];
  return
elseif a_classic
  minrange=min(a_range);
  maxrange=max(a_range);
  a_range=[];
  for code=diff_val(lpg_code),
    lags=lpg_lag(find(lpg_code==code & lpg_bcs=='s'));
    if max(lags)>0,  % ACF data, analyze these
      lpg=find(lpg_code==code & lpg_bcs=='s' & lpg_lag==max(lags));
      ranges=lpg_h(lpg)+(0:lpg_nt(lpg)-1)*lpg_dt(lpg);
      ranges=ranges(find(ranges>=minrange & ranges<=maxrange));
      lenr=length(ranges);
      if lenr>0,
        len=length(a_range);
        a_range=[a_range ranges-lpg_dt(lpg)/2 max(ranges)+lpg_dt(lpg)/2 0];
        a_code(code,len+(1:lenr))=ones(1,lenr);
        a_code(code,len+lenr+1)=0;
      end
    end
    a_minwidth=zeros(1,length(a_range));
    a_maxwidth=1000000000*ones(1,length(a_range));
  end
else

  lenr=length(a_range);
  a_minwidth=zeros(1,lenr-1);
  a_maxwidth=inf*ones(1,lenr-1);

  if exist('analysis_minwidth')==1,
    len=length(analysis_minwidth);
    a_minwidth(1:len)=analysis_minwidth*fac;
  end

  if exist('analysis_maxwidth')==1,
    len=length(analysis_maxwidth);
    a_maxwidth(1:len)=analysis_maxwidth*fac;
  end

  a_code=[];
  gates=[];
  codes=[];
  if exist('analysis_code')==1,
    if length(analysis_code)<length(a_range)-1;
      fprintf(' Analysis_code variable is too short and neglected\n')
    else
      temp=analysis_code;
      while any(temp>0);
        code=rem(temp,10);
        temp=floor(temp/10);
        ind=find(code>0);
        gates=[gates,ind];
        codes=[codes,code(ind)];
      end
      for i=1:max(codes)
        ind=find(codes==i);
        len=length(ind);
        if len>0; a_code(i,gates(ind))=ones(1,len); end
      end
      clear code temp ind gates codes i
    end
  end
end

% form the addresses for each gate
a_addr=[];
a_adstart=[];
a_adend=[];
diffran=diff(a_range);
for gate=find(diffran>0),
  addr=find(ad_range>a_range(gate) & ad_range<=a_range(gate+1));
  % Select suitable codes
  if length(a_code)>0,
    ind1=[];
    for code=find(a_code(:,gate)==1)';
      ind1=[ind1,find(ad_code(addr)==code)];
    end
    addr=addr(ind1);
  end
  % Remove too narrow or too broad responses
  ind1=find(ad_w(addr)>a_minwidth(gate) & ad_w(addr)<a_maxwidth(gate) );
  addr=addr(ind1);

  if length(addr)>1, % store addresses if at least two points found
    len=length(a_addr);
    a_adstart=[a_adstart,len+1];
    a_addr=[a_addr,addr-ADDR_SHIFT]; % From matlab indexing to radar indexing
    a_adend=[a_adend,len+length(addr)];
  end
end

if length(a_addr)==0,
  fprintf(' No correlator result memory locations have been selected\n')
  fprintf(' Check the data selection parameters\n')
  fprintf(' Execution will be stopped\n')
  error(' ')
end

%clear fac minrange maxrange code codes lags lpg ranges
%clear lenr len diffran gate gates addr ind1 ind i temp
```

Geometry routines

Conversion routines to calculate the altitude of a gate from range and vice versa. The calculation assumes spherical earth and so only antenna elevation is needed. Ought to be upgraded.

/geo/gmt/askoh/guisdap/m152/range_to_height.m

```
% GUISDAP v1.50   94-03-10 Copyright Markku Lehtinen, Asko Huuskonen
%
% range_to_height.m
% function that calculates the height of a volume (in km)
% when the range (in us) and antenna direction (in deg) is known.
%
% See also: loc2gg height_to_range
%
% function height=range_to_height(range,el)
  function height=range_to_height(range,el)

  global p_dtau v_lightspeed

  Earth_radius=6372;
  ran=range*(p_dtau*1e-6*v_lightspeed/2)/1000/Earth_radius;
  sin_el=sin(pi*el/180);
  height=Earth_radius*(sqrt(1+2*ran*sin_el+ran.^2)-1);

% This code for elliptical earth
%  scale=(p_dtau*1e-6*v_lightspeed/2);
%  gg_site=[69,25,0];
%  gg_sp=loc2gg(gg_site,[el,180,range*scale/1000])
%  [height,gg_sp(3)]
```

/geo/gmt/askoh/guisdap/m152/height_to_range.m

```
% GUISDAP v1.50   94-03-10 Copyright Markku Lehtinen, Asko Huuskonen
%
% height_to_range.m
% function that calculates the range of a volume (in us)
% when the height (in km) and antenna direction (in deg) is known.
% assumes spherical earth for the moment
%
```

```
% See also: range_to_height
%
% function range=height_to_range(height,el)
  function range=height_to_range(height,el)

  global p_dtau v_lightspeed

  Earth_radius=6372;
  hei=height/Earth_radius;
  sin_el=sin(pi*el/180);
  range=1000*Earth_radius*(sqrt(sin_el^2+2*hei+hei.^2)-sin_el);
  range=range/(p_dtau*1e-6*v_lightspeed/2);
```

The transformation from the local coordinate system, which is specified by giving the range to the common volume, antenna azimuth and elevation, to geographic coordinates is done by `loc2gg`. In a bistatic case, the routine is used to give the ranges and the scattering angle.

/geo/gmt/askoh/guisdap/m152/loc2gg.m

```
% GUISDAP v1.50   94-03-10
%
% This function transforms the scattering point location given in local coordinates
% loc   [elevation, azimuth, range] at location
% site1 [latitude, longitude, height]  to geographic coordinates
% Thanks to J. Murgin (KGI Reprot NO 80:2), EISCAT analysis package, and P.Pollari
%
% In bistatic case, where another reference site (site2) is given
% the routine returns the scattering angle and ranges.
%
% See also: gg2gc, gc2gg, radar_eq
%
%function [gg_sp,angle,ranges]=loc2gg(site1,loc,site2)
function [gg_sp,angle,ranges]=loc2gg(site1,loc,site2)

factor=pi/180;        % conversion factor from degrees to radians
% earth radius (km) and flatness factor
r_earth=6378.135;
g=1.00673944;

% first calculate the  transformation matrices
sinlat=sin(site1(1)*factor);
coslat=cos(site1(1)*factor);
tanlat=tan(site1(1)*factor);
sinlon=sin(site1(2)*factor);
coslon=cos(site1(2)*factor);

rlocgc=  [ sinlat*coslon -sinlon coslat*coslon;
           sinlat*sinlon  coslon coslat*sinlon;
          -coslat           0      sinlat    ];

s1=loc(1);s2=loc(2);s3=loc(3);
loc=s3*[-cos(factor*s2)*cos(factor*s1),...
         sin(factor*s2)*cos(factor*s1),...
         sin(factor*s1)];

gc_site1=gg2gc(site1);          % Site1 to geogentric
gc_sp=gc_site1+(rlocgc*loc')';  % Add scattering distance in geocentric
gg_sp=gc2gg(gc_sp);             % Transform back to geographic

if nargin==3 % bistatic case
  gc_site2=gg2gc(site2);     % Site2 to geogentric
  gc_site1=gc_site1-gc_sp;   % Move origin to scattering point
  gc_site2=gc_site2-gc_sp;   % And same for site2
  ranges=[sqrt(sum(gc_site1.^2)), sqrt(sum(gc_site2.^2))];
  angle=pi-acos(sum(gc_site1.*gc_site2)./ranges(1)/ranges(2));
end
```

/geo/gmt/askoh/guisdap/m152/gc2gg.m

```
% GUISDAP v1.50   94-03-10
%
% This function transforms coordinates from geocentric to geographic (lat, lon, h)
% Thanks to J. Murgin (KGI Reprot NO 80:2), EISCAT analysis package, and P.Pollari
%
% See also: gg2gc, loc2gg
%
% function gg=gc2gg(gc)
function gg=gc2gg(gc)

factor=pi/180;        % conversion factor from degrees to radians
% earth radius (km) and flatness factor
r_earth=6378.135;
g=1.00673944;

if gc(1)==0 & gc(2)==0,
  fprintf('Beware of the spinning earth axis!\n');
  gg=[90, 0, gc(3)-r_earth/g];
else
  gg(2)=atan2(gc(2),gc(1))/factor;
  r0=sqrt(sum(gc(1:2).*gc(1:2)));
  xi0=gc(3)/(r0*sqrt(g));
  xi_iter=r_earth*(g-1)/(g*r0);
  tanxi=xi0;
  tanxi=xi0+xi_iter*tanxi/sqrt(1+tanxi^2);
  tanxi=xi0+xi_iter*tanxi/sqrt(1+tanxi^2);
  gg(1)=atan(sqrt(g)*tanxi)/factor;
  gg(3)=sqrt(1+g*tanxi^2)*(r0-r_earth/sqrt(1+tanxi^2));
end;

```

/geo/gmt/askoh/guisdap/m152/gg2gc.m

```
% GUISDAP v1.50   94-03-10
%
% This function transforms coordinates from geographic (lat, lon, h) to geocentric
% Thanks to J. Murgin (KGI Reprot NO 80:2), EISCAT analysis package, and P.Pollari
%
% See also: gc2gg, loc2gg
%
% function gc=gg2gc(gg)
function gc=gg2gc(gg)

factor=pi/180;        % conversion factor from degrees to radians
% earth radius (km) and flatness factor
r_earth=6378.135;
g=1.00673944;

lat=gg(1)*factor;
lon=gg(2)*factor;
h=gg(3);

hor=(r_earth/sqrt(1+tan(lat)^2/g)+h*cos(lat));
gc=[hor*[cos(lon), sin(lon)], r_earth/sqrt((g+g^2/tan(lat)^2))+h*sin(lat)];
```

Variance calculation using ambiguity functions

There are two methods to calculate the variance estimates for the measured crossed products within GUISDAP. The first one is based on calculating the estimates emprically during the data integration. This methods fails, however, when the number of dumps integrated is small, or when the data actually does not exist at all. The latter happens, when the package is used to design new experiments.

In the other method, the variance estimate is calculated directly from theory, using expressions which give the covariance in terms of the expected values of the crossed products. The method is documented in Huuskonen and Lehtinen (submitted for publication in the JATP Special EISCAT Issue, 1994).

The calculation requires that an estimate of the crossed product values is available for all transmitted pulses and for all delays. This is calculated by `calc_vcsignal.m` with the help of `ACF.m`, by giving the noise temperature and the plasma parameter values. The simplification used is that the signal strength is independent of range. Therefore only one reference address is given `addr` and the result is expressible as a two–dimensional matrix.

/geo/gmt/askoh/guisdap/m152/calc_vcsignal.m

```
% GUISDAP v1.50   94-03-10 Copyright Markku Lehtinen, Asko Huuskonen
%
% A function to calculate the signal strength for all virtual channels and for all
% lag values up to the length of p*env
% Input parameters:
% addr: A reference address to give the radar factor
% T_noise: Background temperature in K
% param: plasma parameters in scaled units
% Output parameter: (global)
% vc_signal: signal strength for all virtual channels and lag values
% See also: ACF, real_to_scaled, adgr_covar, adgr_var, addr_covar
%function calc_vcsignal(addr,T_noise,param)
function calc_vcsignal(addr,T_noise,param)

global vc_signal vc_group vc_Apenv vc_Ap ad_coeff ADDR_SHIFT

% Faster calculation possible if results calculated for virtual channel groups
% One group contains all virtual channels with identical transmission and filters
[a,ind]=wind(diff_val(vc_group),vc_group);
vcg_Apenv=vc_Apenv(:,ind);
vcg_Ap=vc_Ap(:,ind);
```

```
[M,N]=size(vcg_Apenv);
len_vcsig=M;

% Calculate the plasma autocorrelation function for all lags up to length of p*env
ac=col(real(ACF(param,[0:M-1])));
% Multiply by effective pulse length (vcg_Apenv) and radar factor
K=ad_coeff(addr+ADDR_SHIFT)*(vcg_Apenv.*ac(:,ones(1,N)));

% Background power for all lags
B=zeros(size(vcg_Apenv));
[M2,N2]=size(vcg_Ap);B(1:M2,1:N2)=vcg_Ap;
apu=T_noise./B(1,:);
B=B.*apu(ones(M,1),:);

vcg_signal=K+B; % Result is sum of plasma and background contributions
vc_signal=vcg_signal(:,vc_group); % Store to all virtual channels for fast reference
```

/geo/gmt/askoh/guisdap/m152/ACF.m

```
% GUISDAP v1.50   94-03-10 Copyright Markku Lehtinen, Asko Huuskonen
%
% this function returns the theoretical autocorrelation function without
% any pulse form or receiver effects. It is implemented by calling dirthe,
% and defining f_womega so that multiplying by it makes a Fourier transform
% Input parameters
% param: plasma parameters in scaled units
% lags: lag values (in p_dtau units)
% pldfvv: Plasma dispersion function in interpolation table (global)
% Output parameters:
% plasma_acf: plasma ACF for the specified parameters and lags
% See also: dirthe, real_to_scaled
%function plasma_acf=ACF(param,lags)
function plasma_acf=ACF(param,lags)

global p_om p_om0 p_dtau p_D0 k_radar pldfvv

lags=col(lags);
% Create complex exponential to make the Fourier transform
paino=exp( lags*p_om'*((p_dtau*1e-6)*p_om0(1)*sqrt(-1)) ) ;
len=length(p_om);
dom=[0; 0.5*(p_om(3:len)-p_om(1:len-2)); 0]';
[M,N]=size(paino);
% Take the frequency bin widths into account
f_womega=paino.*(dom(ones(M,1),:));

p_coeffg=ones(size(lags));
ch=1;  kd2=k_radar(ch)^2*p_D0^2;

plasma_acf=dirthe(param,p_coeffg,f_womega,kd2,p_om,pldfvv);
plasma_acf=plasma_acf(1:length(lags));
```

The covariance between any two addresses in the result memory is given by

/geo/gmt/askoh/guisdap/m152/addr_covar.m

```
% GUISDAP v1.50   94-03-10 Copyright Markku Lehtinen, Asko Huuskonen
%
% A low level routine to calculate the covariance of two result memory addresses.
% This matlab version is slow. The routine is available also as a mex-version.
% This routine is called by adgr_covar and adgr_var, through which the use
% of addr_covar is recommended.
% Input parameters:
% addr1, addr2 : result memory addresses
% vc_signal    : signal strength as calculated by function calc_vcsignal
% lp_XXX       : lag profile parameters
% Output parameters
% covarRe : Covariance between the real parts of addr1 and addr2
% covarIm : Covariance between the imaginary parts of addr1 and addr2
% See also adgr_covar, adgr_var, calc_vcsignal
%function [covarRe,covarIm]=addr_covar(addr1,addr2,vc_signal,lp_vc,lp_dt,lp_ra,...
%          lp_ri,lp_nt,lp_t1,lp_t2,lp_dec,lp_nfir,lp_fir)
function [covarRe,covarIm]=addr_covar(addr1,addr2,vc_signal,lp_vc,lp_dt,lp_ra,...
         lp_ri,lp_nt,lp_t1,lp_t2,lp_dec,lp_nfir,lp_fir)

[len,Nvc]=size(vc_signal);

% Find the lag profiles which contribute to address addr1
lps1=find( lp_ra<=addr1 & addr1<=lp_ra+((lp_nt-1).*lp_ri) ...
           & round((addr1-lp_ra)./lp_ri)==(addr1-lp_ra)./lp_ri );

% Find the lag profiles which contribute to address addr2
lps2=find( lp_ra<=addr2 & addr2<=lp_ra+((lp_nt-1).*lp_ri) ...
           & round((addr2-lp_ra)./lp_ri)==(addr2-lp_ra)./lp_ri );

cov1=0; cov2=0;
for lp1=lps1  % This is a loop over all the lag profiles in lps1
```

```
    dt1=lp_dt(lp1); % integer
    for lp2=lps2  % This is a loop over all the lag profiles in lps2
      dt2=lp_dt(lp2); % integer
      if lp_vc(lp1)==lp_vc(lp2) % Products correlate only if virtual channels are equal
        vc=lp_vc(lp1);

        apu=lp_dec(lp1)*dt1*(addr1-lp_ra(lp1))./lp_ri(lp1);
        time1=lp_t1(lp1)+apu; % sampling time for the first product in lp1
        time2=lp_t2(lp1)+apu; % sampling time for the second product in lp1
        apu=lp_dec(lp2)*dt2*(addr2-lp_ra(lp2))./lp_ri(lp2);
        tau1=lp_t1(lp2)+apu; % sampling time for the first product in lp2
        tau2=lp_t2(lp2)+apu; % sampling time for the second product in lp2

        for itime=0:lp_nfir(lp1)-1 % All the filter coeffs in lp1 are treated here
          for itau=0:lp_nfir(lp2)-1 % All the filter coeffs in lp2 are treated here
            ero1=abs(tau1+itau*dt2-(time1+itime*dt1)); %
         ero2=abs(time2+itime*dt1-(tau2+itau*dt2));
         if ero1<len & ero2<len
              cov1=cov1+lp_fir(itime+1,lp1)*lp_fir(itau+1,lp2)*...
                        vc_signal(ero1+1,vc)*vc_signal(ero2+1,vc);
         end

            ero3=abs(tau2+itau*dt2-(time1+itime*dt1));
         ero4=abs(time2+itime*dt1-(tau1+itau*dt2));
         if ero3<len & ero4<len
              cov2=cov2+lp_fir(itime+1,lp1)*lp_fir(itau+1,lp2)*...
                        vc_signal(ero3+1,vc)*vc_signal(ero4+1,vc);
         end
         end
        end

      end % End-if of virtual channels
    end  % End-loop over lps2
  end  % End-loop over lps2

  covarRe=(cov1+cov2)/2;
  covarIm=(cov1-cov2)/2;
```

Covariance matrices or variance vectores are produced by the following two functions. They call `addr_covar` and also they initialize the `vc_signal` matrix. Their use is recommended.

`/geo/gmt/askoh/guisdap/m152/adgr_covar.m`

```
% GUISDAP v1.50   94-03-10 Copyright Markku Lehtinen, Asko Huuskonen
%
% A function to calculate the error covariance matrix for two sets of result
% memory addresses. If the two sets are identical, only diagonal and upper
% (or lower) triangle is calculated. The covariance calculations are performed
% by the routine addr_covar
% Input parameters:
% addr1: first set of result memory addresses
% addr2: second set of result memory addresses
% T_noise: background temperature in K
% param: plasma parameters in scaled units
% vc_signal: signal strength for all virtual channels and for all lags (global)
% Output parameters
% covarRe: covariance between the real parts of the two sets
% covarIm: covariance between the imaginary parts of the two sets
% See also: addr_covar, adgr_var, calc_vcsignal, real_to_scaled
%function [covarRe,covarIm]=adgr_covar(addr1,addr2,T_noise,param)
function [covarRe,covarIm]=adgr_covar(addr1,addr2,T_noise,param)

global vc_signal lp_vc lp_dt lp_ra lp_ri lp_nt lp_t1 lp_t2 lp_dec lp_nfir lp_fir
calc_vcsignal(addr1(1),T_noise,param)

covarRe=zeros(length(addr1),length(addr2));
covarIm=zeros(length(addr1),length(addr2));

% Execution time saved if both sets are equal as Cov(A,B)=Cov(B,A)
% Then only diagonal and upper (or lower) triangle calculated.
if length(addr1)==length(addr2),
  if all(addr1==addr2), identical=1; else, identical=0; end
end

for i=1:length(addr1)
  if identical, first=i; else, first=1; end
fprintf(' %.0f',i);
  for j=first:length(addr2)
      [covarRe(i,j),covarIm(i,j)]=...
        addr_covar(addr1(i),addr2(j),vc_signal,lp_vc,lp_dt,lp_ra,...
         lp_ri,lp_nt,lp_t1,lp_t2,lp_dec,lp_nfir,lp_fir);
      if identical,
        covarRe(j,i)=covarRe(i,j);
        covarIm(j,i)=covarIm(i,j);
      end
  end
end
fprintf(' \n')
```

/geo/gmt/askoh/guisdap/m152/adgr_var.m

```
% GUISDAP v1.50   94-03-10 Copyright Markku Lehtinen, Asko Huuskonen
%
% A function to calculate the variance for a set of result memory addresses.
% The covariance calculations are performed by the routine addr_covar
% Input parameters:
% addr: set of result memory addresses
% T_noise: background temperature in K
% param: plasma parameters in scaled units
% vc_signal: signal strength for all virtual channels and for all lags (global)
% Output parameters
% varRe: covariance between the real parts of the two sets
% varIm: covariance between the imaginary parts of the two sets
% See also: addr_covar, adgr_covar, calc_vcsignal, real_to_scaled
%function [varRe,varIm]=adgr_var(addr,T_noise,param)
function [varRe,varIm]=adgr_var(addr,T_noise,param)

global vc_signal lp_vc lp_dt lp_ra lp_ri lp_nt lp_t1 lp_t2 lp_dec lp_nfir lp_fir
calc_vcsignal(addr(1),T_noise,param)

varRe=zeros(1,length(addr));
varIm=zeros(1,length(addr));

for i=1:length(addr)
  [varRe(i),varIm(i)]=addr_covar(addr(i),addr(i),vc_signal,lp_vc,lp_dt,lp_ra,...
        lp_ri,lp_nt,lp_t1,lp_t2,lp_dec,lp_nfir,lp_fir);

end
```

Scaling of parameters

Only two of the six plasma parameters, the temperature ratio and the ion composition, are used as such in the package and the rest are used as scaled. The following two routines will change from one set of parameters to the other.

/geo/gmt/askoh/guisdap/m152/real_to_scaled.m

```
% GUISDAP v1.50   94-03-10 Copyright Markku Lehtinen, Asko Huuskonen
%
% Function to calculate the scaled variable values from physical ones
% Parameters:
% physical: plasma parameters in physical units
% scaled: plasma parameters in scaled units
% See also: scaled_to_real
%function scaled=real_to_scaled(physical)
function scaled=real_to_scaled(physical)

global p_N0 p_m0 p_T0 p_om0 k_radar0

scaled=physical; % affects element 3 and also 6 (if specified on input)
scaled(:,1)=physical(:,1)/p_N0;
scaled(:,2)=physical(:,2)/p_T0;
ch=1;   % hyi hyi
scaled(:,4)=physical(:,4)/(p_om0(ch));
scaled(:,5)=physical(:,5)/(p_om0(ch)/k_radar0(ch));
```

/geo/gmt/askoh/guisdap/m152/scaled_to_real.m

```
% GUISDAP v1.50   94-03-10 Copyright Markku Lehtinen, Asko Huuskonen
%
% Function to calculate the plasma parameters in physical units
% when input is in scaled values
% Parameters:
% physical: plasma parameters in physical units
% scaled: plasma parameters in scaled units
% See also: real_to_scaled
%function physical=scaled_to_real(scaled)
function physical=scaled_to_real(scaled)

global p_N0 p_m0 p_T0 p_om0 k_radar0

physical=scaled; % affects element 3 and also 6 (if specified on input)
physical(:,1)=scaled(:,1)*p_N0;
physical(:,2)=scaled(:,2).*p_T0;
ch=1;  % hyi hyi
physical(:,4)=scaled(:,4)*(p_om0(ch));
physical(:,5)=scaled(:,5)*(p_om0(ch)/k_radar0(ch));
```

Routines for processing results

The following routine initializes result parameters before analysis of a new integration dump is started.

/geo/gmt/askoh/guisdap/m152/clear_results.m

```
% GUISDAP v1.50   94-03-10 Copyright Markku Lehtinen, Asko Huuskonen
%
% defines the result variable as empty matrices
% function clear_results
  function clear_results

global r_ind r_range r_param r_error r_res r_status r_dp
global r_apriori r_apriorierror

r_ind=0;
r_range=[];r_param=[];r_error=[];r_res=[];r_status=[];r_dp=[];
r_apriori=[];r_apriorierror=[];
```

The following routine writes the result to a binary `.mat` file for each dump.

/geo/gmt/askoh/guisdap/m152/save_results.m

```
% GUISDAP v1.50   94-03-10 Copyright Markku Lehtinen, Asko Huuskonen
%
% function to store whole profile of results into a file
%
% function save_results
  function save_results

global result_path name_expr name_site
global p_XMITloc p_RECloc sc_angle
global d_time GUP_ver ch_az ch_el ch_Pt p_m0 p_dtau v_lightspeed
global r_ver r_time r_az r_el r_Pt r_m0 r_h r_SCangle
global r_ind r_range r_param r_error r_res r_status r_dp
global r_apriori r_apriorierror
global pp_range pp_sigma

filename='00000000';
if length(d_time)>0,
  file=int2str(tosecs(d_time(2,:)));len=length(file);
  filename(9-len:8)=file;
end
file=canon([result_path, filename]);

r_ver=GUP_ver;
r_time=d_time;
r_az=ch_az(1);
r_el=ch_el(1);
r_Pt=ch_Pt(1);
r_m0=p_m0;
r_XMITloc=p_XMITloc;
r_RECloc=p_RECloc;
r_SCangle=sc_angle/2;

r_pp=pp_sigma;
r_pprange=col(pp_range)*(p_dtau*1e-6*v_lightspeed/2/1000);

r_h=col(range_to_height(r_range,ch_el(1)));
r_range=col(r_range)*(p_dtau*1e-6*v_lightspeed/2/1000);
variables=['r_ver name_expr name_site r_time r_az r_el r_Pt r_m0'];
variables= [variables ' r_range r_h r_param r_error r_res r_status r_dp'];
variables= [variables ' r_apriori r_apriorierror r_pp r_pprange '];
variables= [variables ' r_XMITloc r_RECloc r_SCangle'];

eval(['save ', file, '.mat ' variables])
% save file name to "filelist.dat"
fprintf(canon([result_path 'filelist.dat'],0),[filename,'\n']);
```

The following initialized a sufficient number of graphic windows. The window positions are given in pixels.

/geo/gmt/askoh/guisdap/m152/init_graphics.m

```
% GUISDAP v1.50   94-03-10 Copyright Markku Lehtinen, Asko Huuskonen
%
% Init_graphics opens a sufficient number of figure windows.
% The sizes and locations are defined here.
% Parameter:
% di_figures (global): requested windows (on input) and figure handles (on output)

Positions=...
[  50, 50,550,300;  % Correlator dump
   50, 50,200,400;  % Raw electron density
  600,300,600,400;  % Fit results
  580,280,600,400]; % Results

```

```
Names=...
['Correlator dump, real part';
 '   Raw electron density   ';
 '  Fit results and quality ';
 '       Results            '];

undefined=di_figures;

unused=[];
for F=get(0,'Children')'
  if strcmp(get(F,'type'),'figure')==1
    UserData=get(F,'UserData');
    if length(UserData)==1
      if UserData>=1 & UserData<=4,
        undefined(UserData)=0;
        di_figures(UserData)=F;
      else
        unused=[unused, F];
      end
    else
      unused=[unused, F];
    end
  end
end

for fig=find(undefined);
  if length(unused),
    F=unused(1);
    unused(1)=[];
  else
    F=figure;
  end
  figure(F), clf
  di_figures(fig)=F;
  set(F,'Position',Positions(fig,:),'NumberTitle','off','Name',Names(fig,:),'UserData',fig);
  set(F,'DefaultAxesFontName','helvetica')
  set(F,'DefaultTextFontName','helvetica')
  set(F,'DefaultAxesFontWeight','bold')
  set(F,'DefaultTextFontWeight','bold')
  set(F,'DefaultAxesFontSize',14)
  set(F,'DefaultTextFontSize',14)
end

clear undefined  Positions  Names  UserData  unused  fig  F
```

The following prints fitted plasma parameters for each gate (and plots them if `display_figures(4)=1`).

`/geo/gmt/askoh/guisdap/m152/plot_result.m`

```
% GUISDAP v1.50   94-03-10 Copyright Markku Lehtinen, Asko Huuskonen
%
% routine to produce a simple plot of the analysis results
%
% function plot_result(Handle,good_only)
function plot_result(Handle,good_only)

global r_param r_error r_dp r_status r_h r_el r_time

if nargin==2
  ind=find(r_status==0);
  if isempty(ind)
    fprintf(' No succesfull fits in the profile, plotting it anyway\n')
    ind=1:length(r_status);
  end
else
  ind=1:length(r_status);
end

r_Ne=r_param(ind,1)/1e11;
r_Ti=r_param(ind,2);
r_ratio=r_param(ind,3);
r_coll=r_param(ind,4);
r_vel=r_param(ind,5);
r_dNe=r_error(ind,1)/1e11;
r_dTi=r_error(ind,2);
r_dratio=r_error(ind,3);
r_dcoll=r_error(ind,4);
r_dvel=r_error(ind,5);
r_comp=r_dp(ind);
h=r_h(ind);
minheight=10*floor(min(h)/10);
maxheight=10*ceil(max(h)/10);

if nargin>=1,figure(Handle); else figure;end
set(gcf,'NextPlot','replace');clf reset
set(gcf,'PaperOrientation','landscape','PaperUnits','centim','PaperPos',[0 2 25 18])

axes('Position',[.1 .1 .19 .8],'NextPlot','replace')
p=plot(r_Ne,h,'go',[r_Ne-r_dNe,r_Ne+r_dNe]',[h,h]','g-');
set(p,'MarkerSize',4)
set(gca,'FontSize',12)
xlabel('Ne [1e11 m-3]')
ylabel('Altitude [km]')
set(get(gca,'Xlabel'),'FontSize',12)
set(get(gca,'Ylabel'),'FontWeight','bold')
ax=axis;ax(1)=0;ax(3:4)=[minheight maxheight];axis(ax);

axes('Position',[.3 .1 .19 .8],'NextPlot','replace')
p=plot(r_Ti,h,'bo',[r_Ti-r_dTi,r_Ti+r_dTi]',[h,h]','b-',r_Ti.*r_ratio,h,'rx');
set(p,'MarkerSize',4)
set(gca,'YTickLabels',[' '],'FontSize',12)
xlabel('Ti(o), Te(x) [K]')
set(get(gca,'Xlabel'),'FontSize',12)
ax=axis;ax(1)=0;ax(3:4)=[minheight maxheight];axis(ax);

axes('Position',[.5 .1 .19 .8],'NextPlot','replace')
p=plot(r_ratio,h,'go',[r_ratio-r_dratio,r_ratio+r_dratio]',[h,h]','g-',...
       [1,1],[minheight maxheight],'Linewidth',0.5);
set(p,'MarkerSize',4)
set(gca,'YTickLabels',[' '],'FontSize',12)
xlabel('Te/Ti')
set(get(gca,'Xlabel'),'FontSize',12)
ax=axis;ax(3:4)=[minheight maxheight];ax(1)=0;ax(2)=max(min([ceil(max(r_ratio)),4]),2);axis(ax);

axes('Position',[.7 .1 .19 .8],'NextPlot','replace')
p=plot(r_vel,h,'go',[r_vel-r_dvel,r_vel+r_dvel]',[h,h]','g-',...
       [0,0],[minheight maxheight],'Linewidth',0.5);
set(p,'MarkerSize',4)
set(gca,'YTickLabels',[' '],'FontSize',12)
xlabel('Vi')
set(get(gca,'Xlabel'),'FontSize',12)
ax=axis;ax(3:4)=[minheight maxheight];axis(ax);

axes('Position',[.9 .1 .09 .8],'NextPlot','replace')
p=plot(r_comp,h,'ro');
set(p,'MarkerSize',4)
set(gca,'YTickLabels',[' '],'FontSize',12)
xlabel('[O+]/Ne')
set(get(gca,'Xlabel'),'FontSize',12)
ax=[-0.1 1.1 minheight maxheight];axis(ax);

axes('Position',[.1 .1 .8 .8],'NextPlot','add')
set(gca,'Visible','off')
alku=r_time(1,:);loppu=r_time(2,:);
tit=[sprintf('%2.0f/%2.0f/%4.0f',alku(3),alku(2),alku(1)),...
     sprintf('  %2.0f:%2.0f:%2.0f', alku(4),alku(5),alku(6)),...
                sprintf('-%2.0f:%2.0f:%2.0f',loppu(4),loppu(5),loppu(6))];
                title(tit);
set(get(gca,'Title'),'FontSize',18,'FontWeight','bold')

drawnow
```

Other routines

All the result memory addresses belonging to a specified set of lag profile groups is given by

/geo/gmt/askoh/guisdap/m152/lpg_addr.m

```
% GUISDAP v1.50   94-03-10 Copyright Markku Lehtinen, Asko Huuskonen
%
% function returns the result memory addresses that belong to
% a lag profile group.
% function addr=lpg_addr(lpgs)

function addr=lpg_addr(lpgs)

global lpg_ra lpg_nt lpg_ri

addr=[];
for lpg=lpgs
 addr=[addr,lpg_ra(lpg)+(0:lpg_nt(lpg)-1)*lpg_ri(lpg)];
end
```