

GUIDAP 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 GUIDAP 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 GUIDAP 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 GUIDAP 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 in 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 possibilities 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 use 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
1  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
2  %   GUP ver. 1.50      Helsinki March 2, 1994
3  %   copyright Asko Huuskonen, Markku Lehtinen,
4  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
5
6  start_GUP
7
8  name_expr='CP1K'; % Experiment name
9  name_site='T';    % Site
10 data_path=[path_GUP 'demodata/CP1K/']; % data directory
11 result_path=path_tmp; % result directory
12
13 analysis_start=[1993 2 16 11 00 0]; % Start time
14 analysis_end= [1993 2 16 11 15 0]; % End time
15
16 analysis_integr=[180];
17 analysis_skip =[ 0 ];
18
19 analysis_altit=[90:3:130, 135:5:170, 180:10:240 260:20:600];
20 % analysis_range=[90:3:130, 135:5:170, 180:10:240 260:20:600];
21
22 analysis_control=[.50 .01 6 1];
23 % analysis_control(1) : if error of Me exceeds this, no fit done
24 % analysis_control(2) : iteration stopped, when step below this
25 % analysis_control(3) : max number of iterations
26 % analysis_control(4) : variance calculation
27
28 display_figures=[1 1 1 1];
29 % display_figures(1) : data dump shown
30 % display_figures(2) : raw electron density shown
31 % display_figures(3) : data and fitted curve for each fit shown
32 % display_figures(4) : results shown after each dump
33
34 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 explicitly 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
1  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```

2      %   GUP ver. 1.50      Helsinki March 2, 1994      %
3      %   copyright Asko Huuskonen, Markku Lehtinen,    %
4      %   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
5
6      start_GUP
7
8      name_expr='CP1K'; % Experiment name
9      name_site='T';    % Site
10     data_path=[path_GUP 'demodata/CP1K/']; % data directory
11     result_path=path_tmp; % result directory
12
13     analysis_start=[1993  2   16 11 00  0]; % Start time
14     analysis_end= [1993  2   16 11 15  0]; % End time
15
16     analysis_integr=[180 120 120];
17     analysis_skip =[ 0   30 30];
18
19     analysis_altit=[110:5:180 140:20:600];
20     analysis_code= [12*ones(1,14),0, 3*ones(1,23)];
21     % description of code numbers
22     % 1: alternating code ACF including the zero lag from power profile
23     % 2: F-region power profile
24     % 3: long pulse
25
26     display_figures=[1 1 1 1];
27
28     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
1  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
2  %   GUP ver. 1.50      Helsinki March 2, 1994      %
3  %   copyright Asko Huuskonen, Markku Lehtinen,    %
4  %   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
5
6  start_GUP
7
8  name_expr='CP1K'; % Experiment name
9  name_site='T';    % Site
10 data_path=[path_GUP 'demodata/CP1K/']; % data directory
11 result_path=path_tmp; % result directory
12
13 analysis_start=[1993  2   16 11 00  0]; % Start time
14 analysis_end= [1993  2   16 11 15  0]; % End time
15
16 analysis_integr=[180];
17 analysis_skip =[ 0 0];
18
19 analysis_altit=[90 600];
20 analysis_classic=1;
21
22 display_figures=[1 1 0 1];
23
24 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 the integration 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
1  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
2  %   GUP ver. 1.50      Helsinki March 2, 1994
3  %   copyright Asko Huuskonen, Markku Lehtinen,
4  %   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
5
6  start_GUP
7
8  name_expr='CP1K'; % Experiment name
9  name_site='T';   % Site
10
11  integ_deffile=[path_GUP 'exps/CP1K/integdef'];
12  result_path=path_tmp; % result directory
13
14  analysis_altit=[90 600];
15  analysis_classic=1;
16
17  display_figures=[1 1 0 1];
18
19  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 products 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 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
1  % GUISDAP v1.50   94-03-10 Copyright Markku Lehtinen, Asko Huuskonen
2  %
3  % 'ionomodel' is a user-supplied function which for a given set of heights
4  % outputs the plasma parameters (Ne, Ti, Te/Ti, coll, [O+]/Ne, velocity)
5  % with their a priori uncertainties. This simple example is static, a more complete

```

```

5      % model might use the date and time values to modify the model.
6      %
7      % function [apriori, apriorierror]=ionomodel(heights);
8
9
10     function [apriori, apriorierror]=ionomodel(heights);
11
12     global ionomodel_control
13
14     maxCollAltit=107; % Below this collision frequency is given freedom to change
15     minRatioAltit=107; % Above this temperature ratio is given freedom to change
16     % Note: This above need not be equal, but setting minRatioAltit less than
17     % maxCollAltit will produce unconvergent fits, because both parameters will change.
18     % It is completely OK to have a range where neither is fitted.
19
20     len=length(heights);
21     heights=col(heights);
22
23     if nargin==2, error=1; end
24     if length(ionomodel_control)==0, ionomodel_control=0; end
25
26     % The global variable ionomodel_control affects the a priori electron density model.
27     % This feature is needed in the experiment design, where a model ionosphere is needed
28     % to calculate a correlator dump.
29     if ionomodel_control==2; % constant SNR
30         ne=1e11*(heights/100).^2;
31     else
32         if ionomodel_control==1; % Strong ionosphere
33             Elowerwidth=15; Eupperwidth=50; Emaxheight=105; Emaxdensity=6e11;
34             Flowerwidth=75; Fupperwidth=125; Fmaxheight=300; Fmaxdensity=1e12;
35         elseif ionomodel_control==-1; % Weak ionosphere
36             Elowerwidth=15; Eupperwidth=60; Emaxheight=115; Emaxdensity=5e10;
37             Flowerwidth=75; Fupperwidth=125; Fmaxheight=300; Fmaxdensity=2e11;
38         elseif ionomodel_control==0; % Standard ionosphere
39             Elowerwidth=15; Eupperwidth=60; Emaxheight=115; Emaxdensity=2e11;
40             Flowerwidth=75; Fupperwidth=125; Fmaxheight=300; Fmaxdensity=5e11;
41         end
42         ne=zeros(size(heights));
43         ind=find(heights<=Emaxheight);
44         ne(ind)=ne(ind)+Emaxdensity*exp(-((heights(ind)-Emaxheight)/Elowerwidth).^2);
45         ind=find(heights>Emaxheight);
46         ne(ind)=ne(ind)+Emaxdensity*exp(-((heights(ind)-Emaxheight)/Eupperwidth).^2);
47         ind=find(heights<=Fmaxheight);
48         ne(ind)=ne(ind)+Fmaxdensity*exp(-((heights(ind)-Fmaxheight)/Flowerwidth).^2);
49         ind=find(heights>Fmaxheight);
50         ne(ind)=ne(ind)+Fmaxdensity*exp(-((heights(ind)-Fmaxheight)/Fupperwidth).^1.3);
51     end
52     par=1; % Electron density
53     apriori(:,par)=ne;
54     if error, apriorierror(:,par)=10*apriori(:,par); end
55
56     par=2; % Ion temperature
57     ExosphericTemp=1000;
58     CenterHeight=140;
59     ScaleLength=30;
60     ti=ExosphericTemp*(1+(atan((heights-CenterHeight)/ScaleLength))/(pi/2))/2;
61     apriori(:,par)=ti;
62     if error, apriorierror(:,par)=10*apriori(:,par); end
63
64     par=3; % Temperature ratio
65     MaxRatio=1.6;
66     CenterHeight=140;
67     ScaleLength=30;
68     ratio=0.5+(MaxRatio-0.5)*(1+(atan((heights-CenterHeight)/ScaleLength))/(pi/2))/2;
69     apriori(:,par)=max(1,ratio);
70     if error,
71         apriorierror(:,par)=10*apriori(:,par);
72         ind=find(heights<=minRatioAltit);
73         if length(ind)>0, apriorierror(ind,par)=apriori(ind,par)/100; end
74     end
75
76     par=4; % Ion-neutral collision frequency
77     apriori(:,par)=max(3578*(exp(-(heights-100)/5.8)),1);
78     if error,
79         apriorierror(:,par)=apriori(:,par)/100;
80         ind=find(heights<=maxCollAltit);
81         if length(ind)>0, apriorierror(ind,par)=apriori(ind,par); end
82     end
83
84     par=5; % Ion velocity
85     apriori(:,par)=zeros(len,1);
86     if error, apriorierror(:,par)=1000*ones(len,1); end
87
88     par=6; % Ion composition
89     xi=[149 150 250 251];
90     yi=[0.0 0.0 1.0 1.0];

```

```
91      apriori(:,par)=inter3(heights',xi,yi)';  
92      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 GUIDAP variables (`decodeparblock`)

if the radar controller program has changed

load in the initialization 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/guidap/m152/an_start.m
1 % GUIDAP v1.50 94-03-10 Copyright Markku Lehtinen, Asko Huuskonen
2 %
3 % Main program for the data analysis. The most important operations performed are
4 % routine name: action:
5 % chk_par1 1) transforms the user supplied control parameters to internal parameters
6 % 2) checks the data source (matlab files/EISCAT .dtst files)
7 % init_graphics opens a sufficient number of figure windows and defines there sizes etc.
8 % integr_NW calls Wigil Wade's integration package, when EISCAT .dtst files are used
9 % integr_data integrates data from Matlab files
10 % decodeparblock transfers the radar parameters (power etc) to internal GUIDAP parameters
11 % load_initfile loads the ambiguity functions etc
12 % scale_lpgwom scales the spectral amb. function with the correlator algorithm factors (lpg_ND)
13 % radar_eq radar equation
14 % scale_data scales the data with the correlator algorithm factors (lpg_ND)
15 % subr_backgr background subtraction to the data
16 % get_apriori the a priori model, electron density obtained from the data
17 % half_prof performs the gated analysis to the data
18 % save_results results stored to the disk
19 % plot_results displays the results
20 %
21 % Other routines called: globals nat_const get_ADDRSHIFT load_GUPvar GUIZARD GUIPERT form_adpar
22 % spektri_init constants chk_par2 simul_dump clear_results
23
24 t_init
25 t_start(1)
26
27 globals % Defines (nearly) all global variables
28 chk_par1
29 nat_const
30 get_ADDRSHIFT
```

```

31  init_graphics
32
33  rcprog_old=-1;
34  EOF=0;
35  while ~EOF,
36
37      if a_rawdata,
38          [OK,EOF]=integr_NW(a_integdeffile);
39      elseif any(a_simul)
40          UK=1;EOF=1;
41      else
42          [OK,EOF]=integr_data;
43      end
44      if OK, decodeparblock; end
45
46  %*****
47  %
48  % At this point, an integrated complex data dump is stored in variable d_data
49  % It is possible to study the contents of the data by graphical and other means, e.g.
50  % ind=0:length(d_data)-2; plot(ind,real(d_data(ind)));
51  %
52  %*****
53
54  if OK,
55      if d_rcprog~=rcprog_old
56          load_initfile
57          if a_control(4)==2, load_GUPvar, end
58          if exist('GUIZARD')==2, GUIZARD, end
59          scale_lpgwom % scales the spectral ambiguity function with lpg_WD factors
60          form_adpar
61          spektri_init % Loads in plasma dispersion function table
62          constants
63          rcprog_old=d_rcprog;
64      end
65      if exist('GUISPERT')==2, GUISPERT, end
66      radar_eq % calculates the radar constant
67      chk_par2
68
69      if any(a_simul),
70          simul_dump
71      else
72          scale_data
73      end
74
75  %*****
76  %
77  % Now the data has been scaled by calibration, so that it appears in units of K
78  % ind=0:length(d_data)-2; plot(ind,real(d_data(ind))); % This would show the data
79  %
80  %*****
81
82  subr_backgr
83
84  %*****
85  %
86  % At this point one finds background subtracted data in vector d_data
87  %
88  %*****
89
90  if di_figures(1),
91      figure(di_figures(1)); clf; ind=find(real(d_data)>-200 & d_data<2000));
92      plot(ind-1,real(d_data(ind))), title('Correlator dump, real part');
93      xlabel('Address'); ylabel('Power [K]'); grid; drawnow
94  end
95
96  get_apriori(any(a_simul))
97
98  %*****
99  %
100 % The get_apriori call calculated the raw electron density profile.
101 % It is stored in variables
102 % pp_range : range to power measurements
103 % pp_profile : Me with a priori temperature ratio model
104 % pp_sigma : Me with Te=Ti
105 %
106 %*****
107
108  clear_results
109  %*****
110  half_prof
111  %*****
112  save_results
113  if di_figures(4), plot_result(di_figures(4),1); end
114  end
115

```

```

116     end
117     t_stop(1)
118     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
1  % GUISDAP v1.50  94-03-10 Copyright Markku Lehtinen, Asko Huuskonen
2  %
3  % Integration program for radar data in Matlab files. The files may either contain
4  % individual dumps or integrated dumps. The latter may or may not have
5  % the variance estimates of the data included.
6  % Each call to this routine performs the integration for one integration period
7  %
8  % NOTE: This is a EISCAT specific function
9  %
10 % Input parameters (all global):
11 % a_ind a_indold a_interval a_year a_start a_integr a_skip a_end
12 % : specify the integration periods
13 % d_filelist : contains the names of the data files
14 % Output parameters:
15 % OK : if true, the integration was succesful
16 % EOF : if true, the end of file was found during integration
17 % Output parameters (global):
18 % d_parbl : the parameter block returned by the integration program
19 % d_data : the integrated data vector
20 % d_var1 d_var2: data variances
21 %
22 % See also: an_start integr_NW integrate
23 % function [OK,EOF]=integr_data
24 function [OK,EOF]=integr_data
25
26 global d_parbl d_data d_var1 d_var2 data_path d_filelist a_control
27 global a_ind a_indold a_interval a_year a_start a_integr a_skip a_end
28
29 OK=0;EOF=0;
30 if a_ind==0
31     a_ind=1;
32     a_interval=a_start+[0 ,a_integr(1)];
33 else
34     a_indold=a_ind;a_ind=a_ind+1; if (a_ind>length(a_integr)), a_ind=1;end

```

```

35     a_interval=a_interval(2)+a_skip(a_indold)+[0 ,a_integr(a_ind)];
36 end
37 if a_interval(2)>=a_end; EOF=1; end
38 if (d_filelist(length(d_filelist))<a_interval(1)), EOF=1; return, end,
39
40 fixed=[1, 5:90 92:93 127:128]; % parameters which are not allowed to change
41 averaged=[96:99]; % parameters which are averaged
42 ORed=95; % parameter which is OR'ed
43
44 for file=d_filelist(find(d_filelist>a_interval(1) & d_filelist<=a_interval(2)))'
45     filename='00000000';str=int2str(file);filename(9-length(str):8)=str;
46     load(canon([data_path, filename]))
47     % For compatibility with some old integrated files
48     if exist('n_preint'), i_averaged=n_preint; end
49
50     [Md,Md]=size(d_data);
51     if Md==1, % For compatibility with some incorrect data file, data as row vectors
52 %     fprintf(' The file contains row vectors instead of columns\n')
53     d_parbl=d_parbl(:); d_data=d_data(:);
54     end
55
56     [secs,year]=tosecs(d_parbl(2:4));
57     if (secs~=file | year~=a_year),
58         disp('Filename conflicts with file contents or years do not match'), keyboard
59     end
60
61     if d_parbl(95)~=0, fprintf(' Status word is %g\n',d_parbl(95)), end
62     dumpOK= rem(d_parbl(95),2)==0 & d_parbl(95)~=64;
63     if ~OK & dumpOK, % initialize with the first good dump
64         first_parbl=d_parbl; % save the first parameter block
65         aver=d_parbl(averaged); % initialize averaging
66         status=d_parbl(ORed); % save the status word
67         starttime=secs-d_parbl(94); % calculate starttime of first dump
68         if (exist('pre_integrated')== 1),
69             aver=d_parbl(averaged)*i_averaged; % initialize averaging
70             data=d_data;
71             if exist('i_var1') % The integrated file need not have variances in it
72                 d_var1=i_var1(:);
73                 d_var2=i_var2(:);
74                 N_averaged=i_averaged;
75             else
76                 d_var1=d_data.*d_data;
77                 d_var2=d_data.*conj(d_data);
78                 N_averaged=1;
79             end
80         else
81             aver=d_parbl(averaged); % initialize averaging
82             lendata=length(d_data);
83             d_data(lendata-1:lendata)=abs(d_data(lendata-1:lendata));
84             d_data=d_data(1:2:lendata)+sqrt(-1)*d_data(2:2:lendata);
85             data=d_data; % data is now a complex column vector
86             d_var1=d_data.*d_data; % for the variance calculations
87             d_var2=d_data.*conj(d_data);
88             N_averaged=1;
89         end
90         UK=1;
91     elseif OK & dumpOK, % update with the following files
92         if any(d_parbl(fixed)~=first_parbl(fixed)),
93             disp(' changes in the parameter block, continuing')
94             indfixed=find(d_parbl(fixed)~=first_parbl(fixed));
95             disp(' # , original , last dump')
96             indfixed=fixed(indfixed);
97             disp([indfixed',first_parbl(indfixed),d_parbl(indfixed)])
98         end
99         status=bitwiseor(status,d_parbl(ORed),16);
100         if (exist('pre_integrated')== 1),
101             aver=aver+d_parbl(averaged)*i_averaged;
102             data=data+d_data;
103             if exist('i_var1')
104                 d_var1=d_var1+i_var1(:);
105                 d_var2=d_var2+i_var2(:);
106                 N_averaged=N_averaged+i_averaged;
107             else
108                 d_var1=d_var1+d_data.*d_data;
109                 d_var2=d_var2+d_data.*conj(d_data);
110                 N_averaged=N_averaged+1;
111             end
112         else
113             aver=aver+d_parbl(averaged);
114             d_data(lendata-1:lendata)=abs(d_data(lendata-1:lendata));
115             d_data=d_data(1:2:lendata)+sqrt(-1)*d_data(2:2:lendata);
116             data=data+d_data;
117             d_var1=d_var1+d_data.*d_data;
118             d_var2=d_var2+d_data.*conj(d_data);
119             N_averaged=N_averaged+1;

```

```

120         end
121     end
122 end
123
124 if OK, % if at least one good data dump was found
125     if a_control(4)==1 & N_averaged<2,
126         fprintf(' One file is not enough for variance determination\n')
127         fprintf(' Skipping this integration period\n')
128         fprintf(' command ''analysi_control(4)=2'' in the startup file will enable the analysis\n')
129         OK=0;
130         return
131     elseif a_control(4)==1 & N_averaged<5,
132         fprintf(' ***** WARNING *****\n')
133         fprintf(' %.Of files may not be enough for reliable variance determination\n',N_averaged)
134         fprintf(' ***** WARNING *****\n')
135     end
136
137     % update parameter block, accept the last parameter block as starting point
138     d_parbl(averaged)=aver/N_averaged;
139     d_parbl(ORed)=status;
140     d_parbl(94)=secs-starttime;
141     d_data=data;
142     d_var1=d_var1-data.*data/N_averaged;
143     d_var2=d_var2-data.*conj(data)/N_averaged;
144
145 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
1 % GUISDAP v1.50 94-03-10 Copyright Markku Lehtinen, Asko Huuskonen
2 %
3 % Interface to Nigel Wade's integration package (mex call)
4 % Parameters:
5 % a_integdeffile : the integration definition file
6 % OK : if true, the integration was succesful
7 % EOF : if true, the end of file was found during integration
8 % global parameters:
9 % d_parbl : the parameter block returned by the integration program
10 % d_data : the integrated data vector
11 % d_var1 d_var2: data variances
12 %
13 % See also: an_start integr_data integrate
14 %
15 % function [OK,EOF]=integr_NW(a_integdeffile)
16 function [OK,EOF]=integr_NW(a_integdeffile)
17
18 global d_parbl d_data d_var1 d_var2 a_control
19
20 [status,d1,N_averaged,par,d_parbl,d_data,d_var1,d_var2,scan]=...
21     integrate(['-id ',a_integdeffile]);
22
23     if status==0, OK=1; EOF=0;
24     elseif status>0, OK=1; EOF=1;
25     elseif status<0, OK=0; EOF=0; fprintf(' Error in integration\n'); return, end
26
27 % If no files found, the routine returns empty matrices, exit here
28 if length(N_averaged)==0, OK=0; return; end
29
30 d_data=d_data(:);
31 d_var1=d_var1(:)-d_data.*d_data/N_averaged;
32 d_var2=d_var2(:)-d_data.*conj(d_data)/N_averaged;
33
34 if a_control(4)==1 & N_averaged<2,
35     fprintf(' One file is not enough for variance determination\n')
36     fprintf(' Skipping this integration period\n')
37     fprintf(' command ''analysi_control(4)=2'' in the startup file will enable the analysis\n')
38     OK=0;
39     return
40 elseif a_control(4)==1 & N_averaged<5,
41     fprintf(' ***** WARNING *****\n')
42     fprintf(' %.Of files may not be enough for reliable variance determination\n',N_averaged)
43     fprintf(' ***** WARNING *****\n')
44 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
1 % GUISDAP v1.50 94-03-10 Copyright Markku Lehtinen, Asko Huuskonen
2 %
3 % function decodeparblock
4 % The relevant information in the parameter block is now transferred to
5 % GUISDAP variables
6 % NOTE: This is a EISCAT specific script
7 %
8 % See also: an_start, integr_data integr_NW
9
10 %global d_parbl d_rcprog d_time p_RECloc
11 %global ch_az ch_el ch_f ch_filt ch_adc ch_Pt ch_range name_site
12
13 if any(a_simul),
14     d_rcprog=1;
15     d_time(1,:)=toYMDHMS(a_year,a_start);
16     d_time(2,:)=toYMDHMS(a_year,a_end);
17     ch_Pt=a_simul(3)*ones(1,8);
18     ch_range=a_simul(5)*ones(1,8);
19     ch_az=a_simul(6)*ones(1,8);
20     ch_el=a_simul(7)*ones(1,8);
21     return
22 end
23
24 if d_parbl(128)>4 & d_parbl(128)<=10,
25     % The site information
26     names=['K';'T';'V';'S'];
27     if name_site=='V'
28         name_site=names(d_parbl(1));
29     end
30
31     % Put the remote receiver locations to GUP variables
32     % Why? Because the remote sites may have common initialization file
33     if name_site=='K'
34         p_RECloc=[67.863, 20.44, .412];
35     elseif name_site=='S'
36         p_RECloc=[67.367, 26.65, .180];
37     end
38
39
40     % The radar controller program number
41     d_rcprog=d_parbl(92);
42     % endtime of integration
43     [time,year]=tosecs(d_parbl(2:4));
44     d_time(2,:)=toYMDHMS(year,time);
45     % starttime of integration
46     time=time-d_parbl(94);
47     d_time(1,:)=toYMDHMS(year,time);
48     fprintf('%4.0f/%2.0f/%2.0f %2.0f:%2.0f:%2.0f',d_time(1,1:6))
49     fprintf('-%2.0f:%2.0f:%2.0f integrated\n',d_time(2,4:6))
50
51     % Antenna pointing direction
52     ch_az=(d_parbl(6)/10)*ones(1,8);
53     ch_el=(d_parbl(9)/10)*ones(1,8);
54
55     % transmit frequencies for channels
56     if d_parbl(13)>1000, % upper first local oscillator for UHF
57         ch_f=d_parbl(13)/10-d_parbl(19:26)'/100-30;
58     elseif d_parbl(13)>800, % lower first local oscillator for UHF
59         ch_f=d_parbl(13)/10+d_parbl(19:26)'/100-30;
60     else
61         error(' VHF experiment?, update decodeparblock')
62     end
end
```

```

53
54 % filter widths
55 ch_filt(2,:)=d_parbl(35:42)'/10;
56 % filter types are stored in two words, 4 bits reserved for each channel
57 for i=1:4, ch_filt(1,[4+i,i])=rem(floor(d_parbl(87:88)'/16^(i-1)),16); end
58
59 % analog-to-digital conversion intervals
60 ch_adc=[d_parbl(78:79)/20;d_parbl(80:85)/10]';
61
62 % transmitter power
63 ch_Pt=ones(1,8)*d_parbl(99)*1000;
64 if rem(d_parbl(127),2)==1,
65     ch_Pt=ones(1,8)*max(d_parbl([99,101]))*1000;
66 end
67
68 % Range to the common volume (for remotes on multistatic radars only)
69 range=d_parbl(11);
70 if d_parbl(11)==0,
71     if (name_site=='K' | name_site=='S')
72         fprintf(' Distance to the common volume is not stored at the parameter block\n')
73         fprintf(' Skipping this integration period\n')
74         UK=0; return
75     else
76         range=1000;
77     end
78 end
79 ch_range=range*ones(1,8)/10;
80
81 else
82     error('Unknown parameter block version, update decodeparblock')
83 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

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

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 definition of 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

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

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 comparison with ionosonde and dynasound 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

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

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 product of the last two factors in the radar equation ($l_{t,t}$ and $N_e/N_0 \dots$) 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 to multiply the products by the addresswise factor $C(addr)$.

```

/geo/gmt/askoh/guidap/m152/radar_eq.m
1 % GUIDAP v1.50 94-03-10 Copyright Markku Lehtinen, Asko Huuskonen
2 %
3 % radar equation for monostatic and bistatic cases
4 % function radar_eq
5 function radar_eq
6
7 global ch_fradar ch_gain ch_Pt ch_az ch_el p_XMITloc p_RECloc
8 global v_lightspeed v_electronradius v_Boltzmann p_dtau p_R0 p_N0
9 global ad_coeff ad_range ad_w
10 global lpg_h lpg_w lpg_code lpg_bcs lp_vc vc_ch
11 global ADDR_SHIFT name_site
12 global sc_angle sc_R0 sc_R1 ch_range k_radar k_radar0
13
14 scale=(p_dtau*1e-6*v_lightspeed/2); % Scale factor from p_dtau units to meters
15
16 Cbeam=0.460; % This factor depends on the beam geometry
17 eff=0.651;
18 % Antenna efficiency is the ratio of the true gain and the
19 % theoretical gain 4*pi*A/lambda^2
20
21
22 % First find the ranges and the scattering angles from the common volume
23 % For monostatic case quite simply
24 % Check that distance from transmitter to receiver less than 100 m
25 if max(abs(gg2gc(p_XMITloc)-gg2gc(p_RECloc)))<.1
26     sc_angle=pi;
27     sc_R0=p_R0*scale;
28     sc_R1=p_R0*scale;
29     k_radar=k_radar0;
30
31     % Effective beam cross section
32     Aeff=Cbeam*eff*(4*pi*sc_R0^2)./ch_gain;
33
34     % Volume for unit pulse length
35     Veff=Aeff*scale;
36     % Comment: Pulse lengths are also expressed in units of p_dtau
37     % Multiplication by scale is included here so that
38     % is is not needed in functions dirthe and power_prof
39 else
40     ch=1;
41     [gg_sp,angle,ranges]=loc2gg(p_RECloc,[ch_el(ch),ch_az(ch),ch_range(ch)],p_XMITloc);
42     sc_angle=angle;
43     sc_R0=ranges(2)*1E3;
44     sc_R1=ranges(1)*1E3;
45
46     % Effective scattering volume in m^3 for the remotes
47     P=(32*pi/0.3215)^2/3.63;
48     Veff=(pi/P)^1.5*(sc_R0^2*sc_R1^2)/sqrt(sc_R0^2+sc_R1^2)/sin(sc_angle);
49     fprintf(' Veff is %.4g\n',Veff)
50     % Update range variables for the new geometry
51     ind=find(lpg_bcs=='s');
52     range=sc_R1/scale;
53     lpg_h(ind)=range*ones(size(ind));
54     lpg_w(ind)=(range/100)*ones(size(ind));
55     ad=ADDR_SHIFT+lpg_addr(ind);
56     ad_range(ad)=range*ones(size(ad));
57     ad_w(ad)=(range/100)*ones(size(ad));
58     % Update now the radar k
59     k_radar=k_radar0*sin(sc_angle/2);
60
61 % Now testing the Uppsala remote equation
62 if 0,
63     lpg=find(lpg_bcs=='s'); lpg=lpg(1);
64     w=wr_lpg(lpg_lp(lpg));
65     r=1:length(w);
66     pp=round(sum(w'.*r)/sum(w));
67     rr=round((sc_R0+sc_R1)/scale/2);
68     r=r-pp+rr;
69     global lpg_WD
70     A=sqrt(3.63/2)*(0.3215/pi/32);

```

```

71     TX=[ch_fradar(ch),ch_gain(ch),A,A];
72     [Cs,r2,leff,wwr]=radar_MB(TX,TX,sc_R0/1E3,sc_R1/1E3,sc_angle,r*1e-6,w);
73     leff=leff/lpg_WD(lpg)*1e6;
74     Aeff=Cbeam*eff*(4*pi*sc_R0^2)./ch_gain; % This is for monostatic
75     Veff=Aeff(1)*leff*scale;
76     fprintf(' Veff is %.4g\n',Veff)
77     end
78 end
79
80 lambda=v_lightspeed./ch_fradar;
81
82 % Single electron cross section at the beam intersection for unit power
83 polfac=1-0.5*sin(sc_angle)^2;
84 fprintf(' Polarization factor is %.2f\n',polfac)
85 POperPt=4*pi*(v_electronradius^2)*polfac*...
86     (ch_gain./(4*pi*sc_R0^2)).*(ch_gain./(4*pi*sc_R1^2)).*(lambda.^2/(4*pi));
87
88 % Multiplier for scale electron density and true power
89 p_coeff0=POperPt.*ch_Pt.*Veff*p_M0;
90 % Calculate now the factor for virtual channels
91 vc=find(vc_ch>0); % These virtual channels in use
92 p_coeff0=p_coeff0(vc_ch(vc))./(v_Boltzmann*Ap(vc,0)/(p_dtau*1e-6));
93
94 % For monostatic case calculate for each point the range factor
95 for sig=find(lpg_bcs=='s')
96     % Must find the virtual channel now
97     lp=lpg_lp(sig);
98     vc=lp_vc(lp(1));
99     addr=ADDR_SHIFT+lpg_addr(sig);
100     ad_coeff(addr)=p_coeff0(vc)*(sc_R1./(scale*ad_range(addr))).^2;
101 end
102
103 % Finally factor to compensate for all the inaccuracies in the numeric constants
104 %Magic_constant=1.11;
105 %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_WD`). 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
1 % GUISDAP v1.50 94-03-10 Copyright Markku Lehtinen, Asko Huuskonen
2 %
3 % Here the data points are divided by the correlator algorithm factors lpg_WD.
4 % By this procedure different signal/background/calibration algorithms are
5 % handled automatically. The spectral ambiguity functions are scaled
6 % accordingly by routine 'scale_lpgwom'
7 %
8 % Next the data is transformed to units of K with the calibration data
9 % This is not an optimal solution, and will be reconsidered in later releases
10 %
11 % Warning: NEVER perform this operation twice, as the results is returned
12 % in the input variables 'd_data', 'd_var1' and 'd_var2'
13 % See also: an_start, scale_lpgwom
14
15 function scale_data
16
17 global lpg_ra lpg_WD lpg_cal lpg_bac lpg_T d_data d_var1 d_var2 ADDR_SHIFT
18
19 %***** SCALING BY CORRELATOR ALGORITHM FACTORS *****
20 data=d_data;
21 d_data=zeros(size(data));
22 for lpg=1:length(lpg_ra) % go through all lag profile groups
23     addr=lpg_addr(lpg); % result memory addresses for lpg;
24     addr=addr+ADDR_SHIFT; % To change from radar to Matlab addressing
25     d_data(addr)=data(addr)/lpg_WD(lpg);
26     d_var1(addr)=d_var1(addr)/(lpg_WD(lpg)*lpg_WD(lpg));
27     d_var2(addr)=d_var2(addr)/(lpg_WD(lpg)*lpg_WD(lpg));

```

```

28     end
29
30     %***** AND BY CALIBRATION POWER *****
31     %calculate first scale for all calibration measurements
32     calibs=diff_val(lpg_cal); % find all different values
33     calibs=calibs(find(calibs>0)); % Accept non-zero values
34     scale=zeros(size(lpg_cal));
35     for cal=calibs
36         bac=lpg_bac(cal);
37         bac_power=mean(d_data(lpg_addr(bac)+ADDR_SHIFT));
38         cal_power=mean(d_data(lpg_addr(cal)+ADDR_SHIFT));
39         scale(cal)=(cal_power-bac_power)/lpg_T(cal);
40     end
41
42     for lpg=1:length(lpg_ra) % go through all lag profile groups
43         cal=lpg_cal(lpg);
44         addr=lpg_addr(lpg)+ADDR_SHIFT; % To change from radar to Matlab addressing
45         d_data(addr)=d_data(addr)/scale(cal);
46         d_var1(addr)=d_var1(addr)/(scale(cal)*scale(cal));
47         d_var2(addr)=d_var2(addr)/(scale(cal)*scale(cal));
48     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
1 % GUISDAP v1.50 94-03-10 Copyright Markku Lehtinen, Asko Huuskonen
2 %
3 % function to estimate and subtract the background component in the data
4 % uses the stored background lag profile group numbers in 'lpg_bac'
5 % Tries to include the subtraction effect also in the variance estimate
6 %
7 % See also: an_start
8 %function subr_backgr
9 function subr_backgr
10
11 global d_data d_var1 d_var2 lpg_bac lpg_nt lpg_background ADDR_SHIFT
12
13 data=d_data;
14 var1=d_var1;
15 var2=d_var2;
16
17 %***** SUBTRACTING BACKGROUND *****
18 bacs=lpg_bac(lpg_bac>0);
19 bacs=diff_val(bacs); % find all different values
20
21 for bac=bacs,
22     addr=lpg_addr(bac)+ADDR_SHIFT; % To change from radar to Matlab addressing
23     background=mean(data(addr));
24     variance1=mean(var1(addr))/lpg_nt(bac);
25     variance2=mean(var2(addr))/lpg_nt(bac);
26     for lpg=find(lpg_bac==bac)
27         addr=lpg_addr(lpg)+ADDR_SHIFT; % To change from radar to Matlab addressing
28         lpg_background(lpg)=background;
29         d_data(addr)=data(addr)-background;
30         d_var1(addr)=var1(addr)+variance1;
31         d_var2(addr)=var2(addr)+variance2;
32     end
33 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
1 % GUIDAP v1.50 94-03-10 Copyright Markku Lehtinen, Asko Huuskonen
2 %
3 % This function find the center altitudes of the analysis gates and calls
4 % ionomodel to get the model ionosphere for these altitudes.
5 % Next, all zero lag measurements are used to get an initial estimate for
6 % the electron density
7 %
8 % See also: ionomodel, power_prof range_to_height
9 function get_apriori(simul)
10
11 global a_addr a_adstart a_adend a_range a_priori a_priorierror ADDR_SHIFT
12 global lpg_bcs lpg_lag ch_el p_W0 ad_w ad_range di_results di_figures
13 global pp_profile pp_range pp_sigma
14
15 if nargin==0, simul=0; end
16
17 for gate=1:length(a_adstart)
18     range(gate)=mean(ad_range(a_addr(a_adstart(gate):a_adend(gate))+ADDR_SHIFT));
19 end
20 height=range_to_height(range,ch_el(1));
21 % form the a priori model for the analysis
22 [a_priori,a_priorierror]=ionomodel(height');
23 % change from physical to scaled variables
24 a_priori=real_to_scaled(a_priori);
25 a_priorierror=real_to_scaled(a_priorierror);
26
27 % Exit here, if the a_priori model is loaded for simulation purposes,
28 % or it should not be updated with the measured raw electron density
29 if simul, return, end
30
31 % Calculate electron density estimates from zero lags, if available.
32 ind=find(lpg_bcs=='s' & lpg_lag==0);
33 if length(ind)>0,
34     [pp_profile,pp_range,pp_sigma]=power_prof(lpg_addr(ind)',0);
35     pp_height=range_to_height(pp_range,ch_el(1));
36
37     if di_figures(2),
38         figure(di_figures(2));set(gcf,'NextPlot','replace');clf reset
39         p=plot(pp_profile*(p_W0/1e11),pp_height,'o');
40         set(p,'MarkerSize',2)
41         title('Ne with model Te/Ti')
42         ylabel('Altitude/km'); xlabel('Raw electron density/1e11'), drawnow
43     end
44
45     % Update the a priori electron density
46     for gate=1:length(a_adstart)
47         ind=find(abs(pp_range-range(gate))<ad_w(ADDR_SHIFT+a_addr(a_adstart(gate))));
48         if length(ind)>0,
49             a_priori(gate,1)=mean(pp_profile(ind));
50             a_priorierror(gate,1)=10*a_priori(gate,1);
51         end
52     end
53 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

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

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
1 % GUISDAP v1.50 94-03-10 Copyright Markku Lehtinen, Asko Huuskonen
2 %
3 % Function calculates the electron density from a given set of measurements
4 % using the temperature ratio values in the model ionosphere
5 % Note that this calculation operates in physical units instead of scaled units
6 % Input parameters:
7 % addr: addresses to use
8 % Debyecorr: Two different options are available
9 % The first one is faster but neglects Debye correction (Debyecorr=0)
10 % The other one is slower but includes Debye correction (Debyecorr=1)
11 % output: pp_prof : Raw electron density with the a priori Te/Ti model
12 % pp_range: The range to the center of gate
13 % pp_sigma: Raw power profile with Te/Ti=1, excl. Debye term
14 %
15 % See also: get_apriori
16 %
17 % function [pp_prof,pp_range,sigma]=power_prof(addr,Debyecorr)
18 function [pp_prof,pp_range,sigma]=power_prof(addr,Debyecorr)
19
20 global ad_range ch_el d_data lpg_womscalled ad_lpg p_om ad_coeff
21 global v_epsilon0 v_Boltzmann v_elemcharge k_radar p_N0 ADDR_SHIFT
22
23 addr=addr+ADDR_SHIFT; % To change from radar to Matlab addressing
24 pp_range=ad_range(addr);
25 pp_height=range_to_height(pp_range,ch_el(1));
26 apriori=ionomodel(pp_height);
27
28 signal_power=real(d_data(addr));
29 len_eff=max(real(lpg_womscalled(ad_lpg(addr),:)))';
30 sigma=p_N0*signal_power./(ad_coeff(addr)'.*len_eff);
31
32 % Here one can choose of two different solutions
33 % The first one is faster but neglects Debye correction (Debyecorr=0)
34 % The other one is slower but includes Debye correction (Debyecorr=1)
35 % Note that this calculation operates in physical units instead of scaled
36 if Debyecorr,
37 % solve the third order equation for electron density
38 ratio=apriori(:,3);
39 Te=apriori(:,2).*ratio;
40 ch=1; % hyi hyi
41 A=(k_radar(ch))^2*v_epsilon0*v_Boltzmann*Te/(v_elemcharge)^2;
42 B=-sigma.*(1+ratio);
43 C=-sigma.*A.*(2+ratio);
44 D=-sigma.*A.*A;
45 for i=1:length(A);
46 apu=roots([1,B(i),C(i),D(i)]);
47 % choose the root closest to the first order solution equal to -B.
48 [hups,ind]=min(abs(apu+B(i)));
49 res(i,1)=apu(ind);
50 end
51 else
52 % Solve only the first order equation (neglect Debye effect)
53 ratio=apriori(:,3);
54 res=sigma.*(1+ratio);
55 end
56
57 pp_prof=res/p_N0;
58 pp_sigma=2*sigma;
59

```

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
1 % GUISDAP v1.50 94-03-10 Copyright Markku Lehtinen, Asko Huuskonen
2 %
3 % Main analysis routine, NOT documented yet, sorry
4 %function half_prof
5 function half_prof
6     t_start(2)
7
8     global a_addr a_adstart a_adend a_control ad_lpg ad_coeff ad_range ADDR_SHIFT
9     global d_data d_var1 d_var2 d_time p_rep p_dtau ch_el dp_comp di_fit dp_fit
10    global lpg_WD lpg_lag lpg_womscalled lpg_bac lpg_cal lpg_background
11    global a_priori a_priorierror f_womega p_coeffg p_WD
12    global r_range r_status r_ind g_ind
13    global pldfvv p_DO p_om k_radar k_radar0
14
15    r_ind=0;
16    for g_ind=1:length(a_adstart)
17        r_ind=r_ind+1;
18        % ADDR_SHIFT is added to result memory addresses
19        % in order to transfer from radar indexing to Matlab indexing
20        addr=a_addr(a_adstart(g_ind):a_adend(g_ind));
21        lpgs=ad_lpg(addr+ADDR_SHIFT); % These are the lag profile groups of the data points
22
23        f_womega=[real(lpg_womscalled(lpgs,:));imag(lpg_womscalled(lpgs,:))];
24        p_coeffg=[ad_coeff(addr+ADDR_SHIFT),ad_coeff(addr+ADDR_SHIFT)]';
25
26        signal_acf=[real(d_data(addr+ADDR_SHIFT));imag(d_data(addr+ADDR_SHIFT))];
27        measurement=[signal_acf;col(a_priori(g_ind,1:5))];
28
29        % Take starting point from the a priori model, temperature values are taken from
30        % the previous gate (if previous fit was successful and gate below the present one).
31        aa=a_priori(g_ind,:);
32        if r_ind>1, if mean(ad_range(addr+ADDR_SHIFT))>=r_range(r_ind-1) & r_status(r_ind-1)==0,
33            aa(2:3)=result(2:3); aa(1)=aa(1)*(1+result(3))/(1+a_priori(r_ind,3));
34        end, end
35
36        if a_control(4)==1, % Using variance estimates calculated from data
37            signal_var=real([d_var1(addr+ADDR_SHIFT)+d_var2(addr+ADDR_SHIFT);...
38                d_var2(addr+ADDR_SHIFT)-d_var1(addr+ADDR_SHIFT)]/2);
39        elseif a_control(4)==2 % calculating variance estimates from ambiguity functions
40            %
41            t=clock;
42            lpgbac=lpg_bac(diff_val(lpg_cal(lpgs)));
43            ind=find(lpg_lag(lpgbac)==0);
44            Tback=mean(lpg_background(lpgbac(ind)));
45            [covRe,covIm]=adgr_var(addr,Tback,aa);
46            WD2=lpg_WD(lpgs).^2;
47            % The variance scaling is calculated from the integration time here
48            % In fact, it should be based on the loop counter value stored to the data dump
49            Int_time=(d_time(2,3:6)-d_time(1,3:6))*[24*3600; 3600; 60 ; 1];
50            if Int_time<0, Int_time=Int_time+24*3600; end
51            Var_scale=(p_rep*p_dtau*1e-6/Int_time)/p_WD;
52            signal_var=[covRe./WD2,covIm./WD2]'.*Var_scale;

```

```

52 %      fprintf(' Time used in variance calculation %.1f s\n',etime(clock,t))
53 end
54 % The variance for zero lag imaginary parts is zero. These must be corrected
55 % to some non-zero value in order to program to work.
56 ind=find(signal_var==0); % Assume here diagonal variance
57 signal_var(ind)=eps*ones(size(ind));
58 variance=[signal_var; (a_priorierror(g_ind,1:5)).^2];
59 r_range(r_ind)=sum(ad_range([addr,addr]+ADDR_SHIFT)./signal_var)/sum(1 ./signal_var);
60
61 ch=1; kd2=k_radar(ch)^2*p_D0^2; Fscale=k_radar0(ch)/k_radar(ch); % hyi hyi
62 [small_f_womega,small_p_om]=find_om_grid(aa,p_coeffg,f_womega,kd2,p_om,pldfvv);
63
64 errorlim=a_control(1); status=0;
65 if errorlim>0 & errorlim<10000 % To prevent unnecessary error estimation
66 % Check if the error of Ne larger than given limit when the fit is started
67 [error,correl,alpha]=error_estimate(aa,variance,kd2,p_coeffg,...
68                                   small_f_womega,Fscale*small_p_om,pldfvv);
69 if error(1)/aa(1) > errorlim, result=aa; chi2=inf; status=2; end % No fit done
70 end
71 if status==0, % Now proceed to the fitting routine
72 tol=a_control(2); maxiter=a_control(3);
73 t_start(3),
74 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
75 [result,chi2,iter,alpha]=mrqmdiag(aa,measurement,variance,tol,maxiter,...
76                                   kd2,p_coeffg,small_f_womega,Fscale*small_p_om,pldfvv);
77 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
78 status=(iter>maxiter);
79 t_stop(3),
80 end
81 store_results(aa,measurement,variance,result(1:5),alpha,chi2,status)
82 end
83
84 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
1 % GUISDAP v1.50 94-03-10 Copyright Markku Lehtinen, Asko Huuskonen
2 %
3 % Internal routine to print and plot analysis results after each gate
4 %
5 % See also: half_prof
6 % function store_results(aa,meas,var,result,alpha,chi2,status)
7 function store_results(aa,meas,var,result,alpha,chi2,status)
8
9 global a_priori a_priorierror ad_lpg p_RECloc
10 global ch_el di_fit g_ind
11 global lpg_lag di_figures
12 global r_range r_status r_param r_dp r_error r_res r_apriori r_apriorierror r_ind
13 global k_radar k_radar0 p_D0 p_coeffg f_womega p_om pldfvv
14
15 % Scale residual and Xfer results to physical units
16 comp=aa(6);
17 aa=aa(1:5);
18 len=length(meas)-length(aa);
19 chi2=chi2/len;
20 err=covm2vec(alpha);
21 res=scaled_to_real(result);
22 er=err;er(1:5)=scaled_to_real(err(1:5));
23 height=range_to_height(r_range(r_ind),ch_el(1));
24
25 % Print results to the console
26 if rem(r_ind,20)==1,
27 fprintf(' alt Ne/1e11 Ti Te/Ti coll velocity [0+]/Ne resid status\n')
28 end
29 fprintf('%5.1f',height);
30 fprintf(' %4.2f:%4.2f',res(1)/1e11,er(1)/1e11);
31 fprintf(' %4.0f:%3.0f',res(2),er(2));
32 fprintf(' %4.2f:%4.2f',res(3),er(3));
33 fprintf(' %4.1f:%4.1f',res(4)/1e3,er(4)/1e3);
34 fprintf(' %6.1f:%4.1f',res(5),er(5));
35 fprintf(' %5.2f ',comp);
36 fprintf(' %4.2f ',chi2);
37 if status==0, str='OK';
38 elseif status==1 str='Max iter';
39 elseif status==2 str='No fit done';
40 end
41 fprintf(str);
42 fprintf('\n')
43

```

```

44 % Store results to result variables
45 r_param(r_ind,:)=res;
46 r_dp(r_ind,:)=comp;
47 r_error(r_ind,:)=er;
48 r_res(r_ind,:)=chi2,sqrt(2/len)];
49 r_status(r_ind,:)=status;
50 r_apriori(r_ind,1:6)=scaled_to_real(a_priori(g_ind,:));
51 r_apriorierror(r_ind,1:6)=scaled_to_real(a_apriorierror(g_ind,:));
52
53 if di_figures(3),
54     ch=1; kd2=k_radar(ch)^2*p_D0^2; Fscale=k_radar0(ch)/k_radar(ch); % hyi hyi
55     l_om=length(p_om);
56     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)]';
57     [M,N]=size(f_womega);
58     womega=f_womega.*dom(ones(M,1),:);
59     theo=dirthe([result,comp],p_coeffg,womega,kd2,Fscale*p_om,pldfvv);
60     res_err=err(1:5)';
61     sig_err=sqrt(var);
62     indr=1:len/2;
63     indi=len/2+1:len;
64     indp=len+(1:5);
65     sig_r=meas(indr);err_r=sig_err(indr);
66     sig_i=meas(indi);err_i=sig_err(indi);
67     sig_p=meas(indp);err_p=sig_err(indp);
68     fitted_r=theo(indr);
69     fitted_i=theo(indi);
70     fitted_p=theo(indp);
71     indi=indi-length(indi);
72     indp=1:5;
73
74     figure(di_figures(3));clf
75
76     set(gca,'Position',[.1 .1 .7 .8],'NextPlot','replace');
77     plot(0,-1,'b.',indr,sig_r,'ro',indr,fitted_r,'g-'),hold on
78     plot(indi+.2,sig_i,'bo',indi+.2,fitted_i,'g-');
79     plot([indr;indr],[sig_r-err_r,sig_r+err_r'],'r-')
80     plot([indi+.2;indi+.2],[sig_i-err_i,sig_i+err_i'],'b-')
81     set(get(gca,'Children'),'MarkerSize',4)
82     title(' Data (o) and fit results (solid line)');
83     ylabel('Power [K]'); xlabel(' # of data point');
84
85     axes('Position',[.85 .1 .12 .8],'NextPlot','replace');
86     plot(indp-0.15,sig_p,'ro',indp+0.15,fitted_p,'go'),hold on
87     plot([indp;indp]-0.15,[sig_p-err_p,sig_p+err_p'],'r:',...
88         [indp;indp]+0.15,[fitted_p-res_err,fitted_p+res_err'],'g-')
89     set(get(gca,'Children'),'MarkerSize',4)
90     axis([0 6 -1 max([5,ceil(fitted_p)])]);
91     set(gca,'Xtick',1:5,'XTickLabels',['W';'T';'r';'c';'v']);
92     title('parameters')
93     drawnow
94 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
1 % GUISDAP v1.50 94-03-10 Copyright Markku Lehtinen, Asko Huuskonen
2 %
3 % DIRect THEory for Inoherent Scatter radar measurements
4 % Available also as a fast mex-version
5 % Input parameters:
6 % param: plasma parameters in scaled units
7 % p_coeffg: radar factor for each measurement
8 % f_womega: spectral ambiguity function for each measurement
9 % kd2: (k D)^2, k is radar k-vector and D is Debye length for scale parameters
10 % p_om: frequency axis for f_womega
11 % pldfvv: plasma dispersion function interpolation table
12 % Output parameter:
13 % theo: theoretical values for the measurements
14 % See also: spec, transf
15 function theo=dirthe(param,p_coeffg,f_womega,kd2,p_om,pldfvv);
16
17 t_start(4)
18
19 param=real(param); % hyi hyi
20
21 % The allowed ranges of the scaled parameters are defined here
22 param(1)=max(0.005,param(1));
23 param(2)=min(100,max(0.1,param(2)));
24 param(3)=min(8,max(0.20,param(3)));
25 param(4)=max(param(4),0);
26
27 [nin0,tit0,mim0,psi,vi]=transf(param);
28 s=spec(nin0,tit0,mim0,psi,vi,kd2,p_om,pldfvv);
29
30 theo=[p_coeffg.*(f_womega*s);col(param(1:5))];
31 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
1 % GUISDAP v1.50 94-03-10 Copyright Markku Lehtinen, Asko Huuskonen
2 %
3 % ADDR_SHIFT is needed as indices to Matlab matrices start from 1 and the first address
4 % used by the radar can be different. For instance it is 0 for the EISCAT radars
5 % GUISDAP uses the true result memory addresses, until a reference to a Matlab-matrix
6 % is needed. At that point ADDR_SHIFT is added to the address.
7 % The routine assumes that the base address is zero. If that is not the case the
8 % value of the base address must be supplied using the global parameter p_baseaddr
9 % This parameter is meaningful only for data analysis
10 %
11 % function get_ADDRSHIFT.m
12 function get_ADDRSHIFT
13
14 global ADDR_SHIFT p_baseaddr
15
16 if length(p_baseaddr)==0,
17     p_baseaddr=0; % This is the first result memory address used by EISCAT
18 end
19 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
1 % GUISDAP v1.50 94-03-10 Copyright Markku Lehtinen, Asko Huuskonen
2 %
3 % Routine to check whether data comes in EISCAT .dtst format of binary Matlab files
4 % Output parameters:
5 % rawdata: true if .dtst files found on the data_path
6 % filelist: contains the dump times, if matlab files were found
7 %
```

```

8      % See also: integr_data integr_NW chk_par1
9      %
10     %function [filelist,rawdata]=load_filelist
11     function [filelist,rawdata]=load_filelist
12
13     global data_path
14
15     OK=0; rawdata=0;
16     file=canon([data_path 'filelist'],0);
17     if exist([file, '.mat'])==2,
18         OK=1;eval(canon(['load ', file, '.mat -ascii']))
19     elseif exist([file, '.dat'])==2,
20         OK=1;load([file, '.dat'])
21     else
22         files=ls(canon(data_path,0));
23         for ind=find(files=='d')
24             if files(ind-1:ind+3)=='dtst'
25                 rawdata=1; OK=1;break
26             end
27         end
28     end
29
30     if OK & rawdata
31         fprintf('*****\n')
32         fprintf(' Data directory contains EISCAT raw data files\n')
33         fprintf(' Data will be integrated by Nigel Wade''s integration front end\n')
34         fprintf('*****\n')
35     elseif OK & ~rawdata
36         fprintf('*****\n')
37         fprintf(' Data directory contains matlab files\n')
38         fprintf(' Data will be integrated by GUISDAP integration routine\n')
39         fprintf('*****\n')
40         filelist=diff_val(filelist);
41     else
42         fprintf('*****\n')
43         fprintf([' No data found in directory ', data_path, '\n Contents of the directory is:\n'])
44         fprintf(ls(canon(data_path,0)))
45         fprintf('*****\n')
46         error(' ')
47     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
1      % GUISDAP v1.50 94-03-10 Copyright Markku Lehtinen, Asko Huuskonen
2      %
3      % A script to load the ambiguity functions etc. into the workspace for analysis
4      % The script assumed that variables name_expr and name_site exist in the workspace
5      % The script contains reference to EISCAT remote sites. However, the script works
6      % without modifications for other radars as long as name_site is different from K and S
7      %
8      % See also: path_expr save_toinitfile
9      %
10     clear vcg_Aenv
11
12     temp=[path_expr name_expr name_site];
13     if name_site=='K' | name_site=='S';
14         temp=[path_expr name_expr 'R'];
15     end
16     if exist('d_rcprog')==1, rcp=d_rcprog; else rcp=0; end
17     if exist(canon([temp '_' int2str(rcp) 'init.mat'],0))==2,
18         eval(canon(['load ' temp '_' int2str(rcp) 'init']))
19     elseif exist(canon([temp 'init.mat'],0))==2,
20         eval(canon(['load ' temp 'init']))
21     else
22         fprintf(['\n\n File ', canon([temp 'init.mat'],0), ' not found \n\n'])
23         error(' ')
24     end
25     if GUP_iniver<1.52,
26         fprintf('*\n\n* Files produced by GUP version %.2f not usable\n', GUP_iniver)
27         fprintf('* Please, reinitialize the experiment with GUP 1.52 or later\n*\n')
28         error(' ')
29     end
30
31     if exist('vcg_Aenv')
32         vc_Aenv=vcg_Aenv(:,vc_group);
33         vc_Ap=vcg_Ap(:,vc_group);
34         vc_Apenv=vcg_Apenv(:,vc_group);

```

```

35     vc_penv=vcg_penv(:,vc_group);
36     vc_penvabs=vcg_penvabs(:,vc_group);
37     clear vcg_Aenv vcg_Ap vcg_Apenv vcg_penv vcg_penvabs
38     end
39     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:

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

```

/geo/gmt/askoh/guisdap/m152/form_adpar.m
1 % GUISDAP v1.50 94-03-10 Copyright Markku Lehtinen, Asko Huuskonen
2 %
3 % This internal routine calculates certain parameters for each memory location in advance
4 % so that they can be obtained rapidly later
5 % ad_range : Range to the center of ambiguity function
6 % ad_w : Width of the ambiguity functions
7 % ad_code : Code number
8 % ad_lpg : Lag profile group number
9 % ad_coeff : The radar factor (to be calculated in radar_eq)
10 %
11 function form_adpar
12
13 global lpg_ra lpg_nt lpg_ri lpg_bcs lpg_h lpg_dt lpg_w lpg_code
14 global ad_range ad_w ad_code ad_lpg ad_coeff ADDR_SHIFT
15
16 len=max(lpg_ra+(lpg_nt-1).*lpg_ri)+1;
17 ad_range=zeros(1,len); ad_w=zeros(1,len);
18 ad_code=zeros(1,len); ad_lpg=zeros(1,len);
19 ad_coeff=zeros(1,len); % used in radar_eq
20 for sig=find(lpg_bcs=='s')
21     addr=lpg_addr(sig)+ADDR_SHIFT; % To change from radar to Matlab addressing
22     len=length(addr);
23     ad_range(addr)=lpg_h(sig)+(0:lpg_nt(sig)-1).*lpg_dt(sig);
24     ad_w(addr)=lpg_w(sig)*ones(len,1);
25     ad_code(addr)=lpg_code(sig)*ones(len,1);
26     ad_lpg(addr)=sig*ones(len,1);
27 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
1 % GUISDAP v1.50 94-03-10 Copyright Markku Lehtinen, Asko Huuskonen
2 %
3 % See also: mrqmndiag
4 function [aa,chi2,its,alpha,OK]=...
5     mrqmn(a,ym,variance,ftol,itmax,kd2,p_coeffg,f_womega,p_om,pldfvv,errorlim)
6
7     comp=a(6);a=a(1:5);
8     if nargin==10, errorlim=inf; end
9     OK=1;
10    if min(size(variance))==1, diagonal=1;end
11    if diagonal,
12        invsigma=1./variance;
13        apu=invsigma*ones(1,length(a));
14    else
15        invsigma=inv(variance);
16    end
17    lambda=0.001;
18    aa=a; valldier=0;
19    ya=dirthe([aa,comp],p_coeffg,f_womega,kd2,p_om,pldfvv);
20    dyda=zeros(length(ya),length(aa));
21    if diagonal,
22        chi2=(ym-ya)'.*(invsigma.*(ym-ya));
23    else
24        chi2=(ym-ya)'.*invsigma*(ym-ya);
25    end
26    its=0;
27    while its<itmax
28        if ~valldier
29            its=its+1;
30            for i=1:length(aa)
31                aa2=aa; aa2(i)=aa(i)+0.0001;
32                dyda(:,i)=(ya-dirthe([aa2,comp],p_coeffg,f_womega,kd2,p_om,pldfvv))/0.0001; %calculate deriva-
tives;

```

```

33     end
34     if diagonal,
35         alpha=dyda'*(apu.*dyda);
36         beta=dyda'*(invsigma.*(ya-ym));
37     else
38         alpha=dyda'*invsigma*dyda;
39         beta=dyda'*invsigma*(ya-ym);
40     end
41     if its==1 error=sqrt(diag(inv(alpha)));
42     if error(1)>errorlim, OK=2; return, end
43     end
44     end
45     da=((alpha+lambda*eye(size(alpha)))\beta)';
46     % disp(da);
47     chi2old=chi2; yaold=ya; aaold=aa;
48     aa=aa+da;
49     ya=dirthe([aa,comp],p_coefffg,f_womega,kd2,p_om,pldfvv);
50     if diagonal,
51         chi2=(ym-ya)'*(invsigma.*(ym-ya));
52     else
53         chi2=(ym-ya)'*invsigma*(ym-ya);
54     end
55     if chi2>=chi2old
56         chi2=chi2old; aa=aaold; ya=yaold; lambda=lambda*10; validder=1;
57         if max(abs(da))<ftol, OK=0; break, end
58     else
59         lambda=lambda/10; validder=0;
60         if max(abs(da))<ftol, OK=0; break, end
61     end
62     end
63     end
64     aa=[aa,comp];
65     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
1 % GUISDAP v1.50 94-03-10 Copyright Markku Lehtinen, Asko Huuskonen
2 %
3 % script defining useful radar constants
4 %
5 % See also: nat_const
6
7 k_radar0=2*pi*2*ch_fradar/v_lightspeed;
8 p_D0=sqrt(v_epsilon0*v_Boltzmann*p_T0/(p_N0*v_elemcharge^2));
9 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
1 % GUISDAP v1.50 94-03-10 Copyright Markku Lehtinen, Asko Huuskonen
2 %
3 % This script defines global variables needed in the data analysis
4 %
5 % See also: glob_EISCAT, glob_GUP, start_GUP
6
7 % Define first the global variables used in the initialization
8 glob_GUP
9
10 % These globals are needed only in the analysis
11
12 global a_priori a_priorierror p_MD
13
14 global ad_range ad_w ad_code ad_lpg ad_coefff ADDR_SHIFT
15
16 global ch_az ch_el ch_f ch_filt ch_adc ch_Pt ch_scangle ch_range
17
18 global d_data d_parbl d_rcprog d_time d_filelist
19 global d_var1 d_var2 % sig_var1 sig_var2
20
21 global a_addr a_adstart a_adend a_control a_ind
22 global a_year a_start a_integr a_skip a_end
23 global di_figures figure_fit
24 global dp_comp dp_comppost dp_fit
25
26 global lpg_womscaled k_radar
27
28 % global p_coefff0 p_coefffg f_womega
29
30 global r_range r_param r_dp r_error r_res r_status
31 global r_apriori r_apriorierror r_h r_time r_XMITloc r_RECloc r_SCangle
32

```

```

33 % Global definitions for the spectrum calculations
34 global pldfv pldfv % tabc2powc powc
35
36 % Global variables for full profile analysis grids
37 %global ug_r ug_ip ug_p eg_r
38

```

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

```

/geo/gmt/askoh/guidap/m152/chk_par1.m
1 % GUIDAP v1.50 94-03-10 Copyright Markku Lehtinen, Asko Huuskonen
2
3 % chk_par1 is the main interface between the user supplied control parameters and
4 % the internal GUIDAP control parameters. There is mostly a one-to-one correspondence
5 % between the parameters e.g.
6 % User parameter GUIDAP parameter
7 % analysis_altit a_altit
8 % analysis_integr a_integr
9 % display_figures di_figures
10 % integ_deffile a_integdeffile
11 % The user parameters are local to the workspace whereas the GUIDAP parameters are global
12 % If any of the user parameters is not specified, or empty, the corresponding GUIDAP
13 % parameter will get the values specified in the routine.
14 %
15 % See also: an_start globals chk_par2 load_filelist
16
17 % The first if-block tries to locate the data source and produces variables
18 % necessary for integration:
19 a_simul=[]; % If this variable present, the data is produced from theory (So clear it)
20 % The first 'if' looks for the integration definition file for Wigel Wade's integration
21 % package. If this variable is present and the file exists, the data comes through that package.
22
23 if exist('integ_deffile')
24     if exist(canon(integ_deffile),0)
25         % The first branch looks for the integration definition file for Wigel Wade's integration
26         % package. If this variable is present and the file exists, the data comes through that package.
27         a_integdeffile=canon(integ_deffile);
28         a_rawdata=1; % This flags that WW's package is to be used
29         fprintf('*****\n')
30         fprintf(' An integration definition file found\n')
31         fprintf(' Data will be integrated by Wigel Wade's integration front end\n')
32         fprintf('*****\n')
33     else
34         fprintf('*****\n')
35         fprintf(' File ', integ_deffile, ' not found\n')
36         fprintf(' Stopping execution\n')
37         fprintf('*****\n')
38         error(' ')
39     end
40 elseif exist('analysis_simul')
41     % The second branch is for simulated data to be calculated from the theory
42     % The presence of the variable a_simul is the flag, which is used later. Contents
43     % a_simul(1): integration time
44     % a_simul(2): controls the start time in steps of 7200 s, i.e. 2 hours.
45     % a_simul(3): The transmitter power
46     % a_simul(4): The background temperature
47     % a_simul(5:6): Antenna azimuth and elevation
48     a_simul=[0 1 1.2e6 100 300 180 90];
49     a_simul(1:length(analysis_simul))=analysis_simul;
50     a_year=2222;
51     a_start=7200*a_simul(2);
52     a_end=7200*a_simul(2)+a_simul(1);
53 else
54     % The final branch is the normal case. Call to load_filelist checks whether EISCAT .dst
55     % files are present at the data_path (a_rawdata=1) or whether matlab files are there
56     [d_filelist,a_rawdata]=load_filelist;
57     a_start=tosecs(analysis_start); % the programs uses internally only seconds,
58     a_end =tosecs(analysis_end); % counted from the beginning of year
59     a_year=analysis_start(1);
60     a_ind=0;
61
62     if exist('analysis_integr'),
63         a_integr=analysis_integr;
64     else
65         % If analysis_integr not present, produce integration times from the dump times
66         % Then no (further) integration is performed by GUIDAP
67         if a_rawdata
68             a_integr=2; % This should make sure that all dumps are analyzed separately
69         else % This is for Matlab files. Note that the files may contain integrated data
70             a_integr=diff([a_start d_filelist(find(d_filelist>a_start & d_filelist<=a_end))]);
71         end
72         a_skip=zeros(size(a_integr));
73     end

```

```

74     if exist('analysis_skip'),
75         a_skip=analysis_skip;
76     else
77         a_skip=zeros(size(a_integr)); % if skips were not defined, assume
78     end % them to be zero
79
80     if a_rawdata
81         % If EISCAT .dtst files were found, the integration definition file is produced here
82         % The code transfers the contents of the GUISDAP control parameters to the control
83         % parameters of Nigél's analysis package. The standard header lines are read first:
84         fid=fopen(canon([path_GUP,'matfiles/integdefs'],'r'));
85         aa=fread(fid,inf,'char');
86         fclose(fid);
87         a_integdeffile=canon([path_tmp,'integdef'],0);
88         fid=fopen(a_integdeffile,'w');
89         fwrite(fid,aa,'char');
90         if name_site=='T'; fprintf(fid,['data-source tromso\n']);
91         elseif name_site=='K'; fprintf(fid,['data-source kiruna\n']);
92         elseif name_site=='S'; fprintf(fid,['data-source sodankyla\n']);
93         elseif name_site=='V'; fprintf(fid,['data-source vhf\n']);
94     end
95     fprintf(fid,['input-names ', canon(data_path),'\n']);
96     fprintf(fid,'start-time %0.f:%0.f:%0.f',analysis_start(4:6));
97     fprintf(fid,' %0.f/%0.f/%0.f\n', analysis_start(1:3));
98     fprintf(fid,'end-time %0.f:%0.f:%0.f',analysis_end(4:6));
99     fprintf(fid,' %0.f/%0.f/%0.f\n', analysis_end(1:3));
100    fprintf(fid,'cycle-time %0.f\n', sum(a_integr)+sum(a_skip));
101    starts=[0,cumsum(a_integr+a_skip)];
102    for i=1:length(a_integr)
103        fprintf(fid,'scan-pos timed %0.f %0.f\n',starts(i), a_integr(i));
104    end
105    fclose(fid);
106 end
107
108
109 a_control=[100000, 0.01, 6 1];
110 % a_control(1) No fit is tried, if the error of We is larger than (1) at the start
111 % a_control(2) Fitting is stopped when step for all parameters is less than (2)
112 % a_control(3) Maximum number of iterations
113 % a_control(4) Variance calculation
114 % = 1 when variance estimated from data
115 % = 2 when variance estimated using ambiguity functions
116 if exist('analysis_control')
117     ind=find(analysis_control>0);
118     a_control(ind)=analysis_control(ind);
119 end
120 if any(a_simul)
121     a_control(4)=2;
122 end
123
124 %dp_comp=[0:.1:1];
125 %dp_fit=0;
126 %if exist('discretepar_fit')==1,
127 % dp_fit=discretepar_fit;
128 %end
129
130 %if length(dp_comp)>36,
131 % fprintf('Too many discrete parameter values\n')
132 % error('Contact your local GUIspert')
133 %end
134
135 di_figures=[1 1 1 1];
136 if exist('display_figures')
137     di_figures(1:length(display_figures))=display_figures;
138 end
139 if exist('display_fit'), % For compatibility with GUISDAP 1.44 and earlier
140     di_figures(3)=display_fit;
141 end
142 if exist('display_results'), % For compatibility with GUISDAP 1.44 and earlier
143     di_figures(4)=display_results;
144 end
145
146 % check that the various path names have directory separators at the end
147 len=length(data_path); char=data_path(len);
148 if char=='.' & char~='/' & char~='\', data_path(len+1)='.'/;end
149
150 len=length(result_path); char=result_path(len);
151 if char=='.' & char~='/' & char~='\', result_path(len+1)='.'/;end
152
153 len=length(path_GUP); char=path_GUP(len);
154 if char=='.' & char~='/' & char~='\', path_GUP(len+1)='.'/;end
155
156 len=length(path_exps); char=path_exps(len);
157 if char=='.' & char~='/' & char~='\', path_exps(len+1)='.'/;end
158

```

```

159 len=length(path_tmp); char=path_tmp(len);
160 if char~=':' & char~='/' & char~='\', path_tmp(len+1)='/' ;end
161
162
/geo/gmt/askoh/guidap/m152/chk_par2.m
1 % GUIDAP v1.50 94-03-10 Copyright Markku Lehtinen, Asko Huuskonen
2 %
3
4 if exist('analysis_classic')==1,
5     a_classic=analysis_classic;
6 else
7     a_classic=0;
8 end
9
10 fac=1000/(v_lightspeed/2)/(p_dtau*1e-6);
11 if exist('analysis_range')==1,
12     a_range=analysis_range*fac;
13 elseif exist('analysis_altit')==1,
14     a_range=height_to_range(analysis_altit,ch_el(1));
15 else
16     a_range=[0 100000];
17 end
18
19 if exist('analysis_addr')==1,
20     a_addr=analysis_addr;
21     a_adstart=analysis_adstart;
22     a_adend=[a_adstart(2:length(a_adstart))-1,length(a_addr)];
23     return
24 elseif a_classic
25     minrange=min(a_range);
26     maxrange=max(a_range);
27     a_range=[];
28     for code=diff_val(lpg_code),
29         lags=lpg_lag(find(lpg_code==code & lpg_bcs=='s'));
30         if max(lags)>0, % ACF data, analyze these
31             lpg=find(lpg_code==code & lpg_bcs=='s' & lpg_lag==max(lags));
32             ranges=lpg_h(lpg)+(0:lpg_nt(lpg)-1)*lpg_dt(lpg);
33             ranges=ranges(find(ranges>=minrange & ranges<=maxrange));
34             lenr=length(ranges);
35             if lenr>0,
36                 len=length(a_range);
37                 a_range=[a_range ranges-lpg_dt(lpg)/2 max(ranges)+lpg_dt(lpg)/2 0];
38                 a_code(code,len+(1:lenr))=ones(1,lenr);
39                 a_code(code,len+lenr+1)=0;
40             end
41         end
42         a_minwidth=zeros(1,length(a_range));
43         a_maxwidth=1000000000*ones(1,length(a_range));
44     end
45 else
46
47     lenr=length(a_range);
48     a_minwidth=zeros(1,lenr-1);
49     a_maxwidth=inf*ones(1,lenr-1);
50
51     if exist('analysis_minwidth')==1,
52         len=length(analysis_minwidth);
53         a_minwidth(1:len)=analysis_minwidth*fac;
54     end
55
56     if exist('analysis_maxwidth')==1,
57         len=length(analysis_maxwidth);
58         a_maxwidth(1:len)=analysis_maxwidth*fac;
59     end
60
61     a_code=[];
62     gates=[];
63     codes=[];
64     if exist('analysis_code')==1,
65         if length(analysis_code)<length(a_range)-1;
66             fprintf(' Analysis_code variable is too short and neglected\n')
67         else
68             temp=analysis_code;
69             while any(temp>0);
70                 code=rem(temp,10);
71                 temp=floor(temp/10);
72                 ind=find(code>0);
73                 gates=[gates,ind];
74                 codes=[codes,code(ind)];
75             end
76             for i=1:max(codes)
77                 ind=find(codes==i);
78                 len=length(ind);
79                 if len>0; a_code(i,gates(ind))=ones(1,len); end

```

```

80         end
81         clear code temp ind gates codes i
82     end
83 end
84 end
85
86 % form the addresses for each gate
87 a_addr=[];
88 a_adstart=[];
89 a_adend=[];
90 diffran=diff(a_range);
91 for gate=find(diffran>0),
92     addr=find(ad_range>a_range(gate) & ad_range<=a_range(gate+1));
93     % Select suitable codes
94     if length(a_code)>0,
95         ind1=[];
96         for code=find(a_code(:,gate)==1)';
97             ind1=[ind1,find(ad_code(addr)==code)];
98         end
99         addr=addr(ind1);
100     end
101     % Remove too narrow or too broad responses
102     ind1=find(ad_w(addr)>a_minwidth(gate) & ad_w(addr)<a_maxwidth(gate));
103     addr=addr(ind1);
104
105     if length(addr)>1, % store addresses if at least two points found
106         len=length(a_addr);
107         a_adstart=[a_adstart,len+1];
108         a_addr=[a_addr,addr-ADDR_SHIFT]; % From matlab indexing to radar indexing
109         a_adend=[a_adend,len+length(addr)];
110     end
111 end
112
113 if length(a_addr)==0,
114     fprintf(' No correlator result memory locations have been selected\n');
115     fprintf(' Check the data selection parameters\n');
116     fprintf(' Execution will be stopped\n');
117     error(' ');
118 end
119
120 %clear fac minrange maxrange code codes lags lpg ranges
121 %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
1 % GUISDAP v1.50 94-03-10 Copyright Markku Lehtinen, Asko Huuskonen
2 %
3 % range_to_height.m
4 % function that calculates the height of a volume (in km)
5 % when the range (in us) and antenna direction (in deg) is known.
6 %
7 % See also: loc2gg height_to_range
8 %
9 % function height=range_to_height(range,el)
10 function height=range_to_height(range,el)
11
12 global p_dtau v_lightspeed
13
14 Earth_radius=6372;
15 ran=range*(p_dtau*1e-6*v_lightspeed/2)/1000/Earth_radius;
16 sin_el=sin(pi*el/180);
17 height=Earth_radius*(sqrt(1+2*ran*sin_el+ran.^2)-1);
18
19 % This code for elliptical earth
20 % scale=(p_dtau*1e-6*v_lightspeed/2);
21 % gg_site=[69,25,0];
22 % gg_sp=loc2gg(gg_site,[el,180,range*scale/1000])
23 % [height,gg_sp(3)]

```

```

/geo/gmt/askoh/guisdap/m152/height_to_range.m
1 % GUISDAP v1.50 94-03-10 Copyright Markku Lehtinen, Asko Huuskonen
2 %
3 % height_to_range.m
4 % function that calculates the range of a volume (in us)
5 % when the height (in km) and antenna direction (in deg) is known.
6 % assumes spherical earth for the moment
7 %

```

```

8      % See also: range_to_height
9      %
10     % function range=height_to_range(height,el)
11     function range=height_to_range(height,el)
12
13     global p_dtau v_lightspeed
14
15     Earth_radius=6372;
16     hei=height/Earth_radius;
17     sin_el=sin(pi*el/180);
18     range=1000*Earth_radius*(sqrt(sin_el^2+2*hei*hei.^2)-sin_el);
19     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
1      % GUIDAP v1.50 94-03-10
2      %
3      % This function transforms the scattering point location given in local coordinates
4      % loc [elevation, azimuth, range] at location
5      % site1 [latitude, longitude, height] to geographic coordinates
6      % Thanks to J. Murgin (KGI Reprot W0 80:2), EISCAT analysis package, and P.Pollari
7      %
8      % In bistatic case, where another reference site (site2) is given
9      % the routine returns the scattering angle and ranges.
10     %
11     % See also: gg2gc, gc2gg, radar_eq
12     %
13     %function [gg_sp,angle,ranges]=loc2gg(site1,loc,site2)
14     function [gg_sp,angle,ranges]=loc2gg(site1,loc,site2)
15
16     factor=pi/180; % conversion factor from degrees to radians
17     % earth radius (km) and flatness factor
18     r_earth=6378.135;
19     g=1.00673944;
20
21     % first calculate the transformation matrices
22     sinlat=sin(site1(1)*factor);
23     coslat=cos(site1(1)*factor);
24     tanlat=tan(site1(1)*factor);
25     sinlon=sin(site1(2)*factor);
26     coslon=cos(site1(2)*factor);
27
28     rlocgc= [ sinlat*coslon -sinlon coslat*coslon;
29              sinlat*sinlon coslon coslat*sinlon;
30              -coslat 0 sinlat ];
31
32     s1=loc(1);s2=loc(2);s3=loc(3);
33     loc=s3*[-cos(factor*s2)*cos(factor*s1),...
34             sin(factor*s2)*cos(factor*s1),...
35             sin(factor*s1)];
36
37     gc_site1=gg2gc(site1); % Site1 to geocentric
38     gc_sp=gc_site1+(rlocgc*loc)'; % Add scattering distance in geocentric
39     gg_sp=gc2gg(gc_sp); % Transform back to geographic
40
41     if nargin==3 % bistatic case
42         gc_site2=gg2gc(site2); % Site2 to geocentric
43         gc_site1=gc_site1-gc_sp; % Move origin to scattering point
44         gc_site2=gc_site2-gc_sp; % And same for site2
45         ranges=[sqrt(sum(gc_site1.^2)), sqrt(sum(gc_site2.^2))];
46         angle=pi-acos(sum(gc_site1.*gc_site2)./ranges(1)/ranges(2));
47     end

```

```

/geo/gmt/askoh/guisdap/m152/gc2gg.m
1      % GUIDAP v1.50 94-03-10
2      %
3      % This function transforms coordinates from geocentric to geographic (lat, lon, h)
4      % Thanks to J. Murgin (KGI Reprot W0 80:2), EISCAT analysis package, and P.Pollari
5      %
6      % See also: gg2gc, loc2gg
7      %
8      % function gg=gc2gg(gc)
9      function gg=gc2gg(gc)
10
11     factor=pi/180; % conversion factor from degrees to radians
12     % earth radius (km) and flatness factor
13     r_earth=6378.135;
14     g=1.00673944;
15
16     if gc(1)==0 & gc(2)==0,

```

```

17     fprintf('Beware of the spinning earth axis!\n');
18     gg=[90, 0, gc(3)-r_earth/gl];
19     else
20         gg(2)=atan2(gc(2),gc(1))/factor;
21         r0=sqrt(sum(gc(1:2).*gc(1:2)));
22         xi0=gc(3)/(r0*sqrt(g));
23         xi_iter=r_earth*(g-1)/(g*r0);
24         tanxi=xi0;
25         tanxi=xi0+xi_iter*tanxi/sqrt(1+tanxi^2);
26         tanxi=xi0+xi_iter*tanxi/sqrt(1+tanxi^2);
27         gg(1)=atan(sqrt(g)*tanxi)/factor;
28         gg(3)=sqrt(1+g*tanxi^2)*(r0-r_earth/sqrt(1+tanxi^2));
29     end;
30

/geo/gmt/askoh/guidap/m152/gg2gc.m
1  % GUIDAP v1.50  94-03-10
2  %
3  % This function transforms coordinates from geographic (lat, lon, h) to geocentric
4  % Thanks to J. Murgin (KGI Reprot NO 80:2), EISCAT analysis package, and P.Pollari
5  %
6  % See also: gc2gg, loc2gg
7  %
8  % function gc=gg2gc(gg)
9  function gc=gg2gc(gg)
10
11     factor=pi/180;      % conversion factor from degrees to radians
12     % earth radius (km) and flatness factor
13     r_earth=6378.135;
14     g=1.00673944;
15
16     lat=gg(1)*factor;
17     lon=gg(2)*factor;
18     h=gg(3);
19
20     hor=(r_earth/sqrt(1+tan(lat)^2/g)+h*cos(lat));
21     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 GUIDAP. The first one is based on calculating the estimates empirically during the data integration. This method 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/guidap/m152/calc_vcsignal.m
1  % GUIDAP v1.50  94-03-10 Copyright Markku Lehtinen, Asko Huuskonen
2  %
3  % A function to calculate the signal strength for all virtual channels and for all
4  % lag values up to the length of p*env
5  % Input parameters:
6  % addr: A reference address to give the radar factor
7  % T_noise: Background temperature in K
8  % param: plasma parameters in scaled units
9  % Output parameter: (global)
10 % vc_signal: signal strength for all virtual channels and lag values
11 % See also: ACF, real_to_scaled, adgr_covar, adgr_var, addr_covar
12 %function calc_vcsignal(addr,T_noise,param)
13 function calc_vcsignal(addr,T_noise,param)
14
15 global vc_signal vc_group vc_Apenv vc_Ap ad_coeff ADDR_SHIFT
16
17 % Faster calculation possible if results calculated for virtual channel groups
18 % One group contains all virtual channels with identical transmission and filters
19 [a,ind]=wind(diff_val(vc_group),vc_group);
20 vcg_Apenv=vc_Apenv(:,ind);
21 vcg_Ap=vc_Ap(:,ind);

```

```

22 [M,M]=size(vcg_Apenv);
23 len_vcsig=M;
24
25 % Calculate the plasma autocorrelation function for all lags up to length of p*env
26 ac=col(real(ACF(param,[0:M-1])));
27 % Multiply by effective pulse length (vcg_Apenv) and radar factor
28 K=ad_coeff(addr+ADDR_SHIFT)*(vcg_Apenv.*ac(:,ones(1,M))));
29
30 % Background power for all lags
31 B=zeros(size(vcg_Apenv));
32 [M2,M2]=size(vcg_Ap);B(1:M2,1:M2)=vcg_Ap;
33 apu=T_noise./B(1,:);
34 B=B.*apu(ones(M,1),:);
35
36 vcg_signal=K+B; % Result is sum of plasma and background contributions
37 vc_signal=vcg_signal(:,vc_group); % Store to all virtual channels for fast reference

/geo/gmt/askoh/guidap/m152/ACF.m
1 % GUIDAP v1.50 94-03-10 Copyright Markku Lehtinen, Asko Huuskonen
2 %
3 % this function returns the theoretical autocorrelation function without
4 % any pulse form or receiver effects. It is implemented by calling dirthe,
5 % and defining f_womega so that multiplying by it makes a Fourier transform
6 % Input parameters
7 % param: plasma parameters in scaled units
8 % lags: lag values (in p_dtau units)
9 % pldfvv: Plasma dispersion function in interpolation table (global)
10 % Output parameters:
11 % plasma_acf: plasma ACF for the specified parameters and lags
12 % See also: dirthe, real_to_scaled
13 %function plasma_acf=ACF(param,lags)
14 function plasma_acf=ACF(param,lags)
15
16 global p_om p_om0 p_dtau p_D0 k_radar pldfvv
17
18 lags=col(lags);
19 % Create complex exponential to make the Fourier transform
20 paino=exp( lags*p_om'*((p_dtau*1e-6)*p_om0(1)*sqrt(-1)) );
21 len=length(p_om);
22 dom=[0; 0.5*(p_om(3:len)-p_om(1:len-2)); 0]';
23 [M,M]=size(paino);
24 % Take the frequency bin widths into account
25 f_womega=paino.*(dom(ones(M,1),:));
26
27 p_coeffg=ones(size(lags));
28 ch=1; kd2=k_radar(ch)^2*p_D0^2;
29
30 plasma_acf=dirthe(param,p_coeffg,f_womega,kd2,p_om,pldfvv);
31 plasma_acf=plasma_acf(1:length(lags));

```

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

```

/geo/gmt/askoh/guidap/m152/addr_covar.m
1 % GUIDAP v1.50 94-03-10 Copyright Markku Lehtinen, Asko Huuskonen
2 %
3 % A low level routine to calculate the covariance of two result memory addresses.
4 % This matlab version is slow. The routine is available also as a mex-version.
5 % This routine is called by adgr_covar and adgr_var, through which the use
6 % of addr_covar is recommended.
7 % Input parameters:
8 % addr1, addr2 : result memory addresses
9 % vc_signal : signal strength as calculated by function calc_vcsignal
10 % lp_XXX : lag profile parameters
11 % Output parameters
12 % covarRe : Covariance between the real parts of addr1 and addr2
13 % covarIm : Covariance between the imaginary parts of addr1 and addr2
14 % See also adgr_covar, adgr_var, calc_vcsignal
15 %function [covarRe,covarIm]=addr_covar(addr1,addr2,vc_signal,lp_vc,lp_dt,lp_ra,...
16 % lp_ri,lp_nt,lp_t1,lp_t2,lp_dec,lp_nfir,lp_fir)
17 function [covarRe,covarIm]=addr_covar(addr1,addr2,vc_signal,lp_vc,lp_dt,lp_ra,...
18 lp_ri,lp_nt,lp_t1,lp_t2,lp_dec,lp_nfir,lp_fir)
19
20 [len,Mvc]=size(vc_signal);
21
22 % Find the lag profiles which contribute to address addr1
23 lps1=find( lp_ra<=addr1 & addr1<=lp_ra+((lp_nt-1).*lp_ri) ...
24 & round((addr1-lp_ra)./lp_ri)==(addr1-lp_ra)./lp_ri );
25
26 % Find the lag profiles which contribute to address addr2
27 lps2=find( lp_ra<=addr2 & addr2<=lp_ra+((lp_nt-1).*lp_ri) ...
28 & round((addr2-lp_ra)./lp_ri)==(addr2-lp_ra)./lp_ri );
29
30 cov1=0; cov2=0;
31 for lpi=lps1 % This is a loop over all the lag profiles in lps1

```

```

32     dt1=lp_dt(lp1); % integer
33     for lp2=lps2 % This is a loop over all the lag profiles in lps2
34         dt2=lp_dt(lp2); % integer
35         if lp_vc(lp1)==lp_vc(lp2) % Products correlate only if virtual channels are equal
36             vc=lp_vc(lp1);
37
38             apu=lp_dec(lp1)*dt1*(addr1-lp_ra(lp1))./lp_ri(lp1);
39             time1=lp_t1(lp1)+apu; % sampling time for the first product in lp1
40             time2=lp_t2(lp1)+apu; % sampling time for the second product in lp1
41             apu=lp_dec(lp2)*dt2*(addr2-lp_ra(lp2))./lp_ri(lp2);
42             tau1=lp_t1(lp2)+apu; % sampling time for the first product in lp2
43             tau2=lp_t2(lp2)+apu; % sampling time for the second product in lp2
44
45             for itime=0:lp_nfir(lp1)-1 % All the filter coeffs in lp1 are treated here
46                 for itau=0:lp_nfir(lp2)-1 % All the filter coeffs in lp2 are treated here
47                     ero1=abs(tau1+itau*dt2-(time1+itime*dt1)); %
48                     ero2=abs(time2+itime*dt1-(tau2+itau*dt2)); %
49                     if ero1<len & ero2<len
50                         cov1=cov1+lp_fir(itime+1,lp1)*lp_fir(itau+1,lp2)*...
51                             vc_signal(ero1+1,vc)*vc_signal(ero2+1,vc);
52                     end
53
54                     ero3=abs(tau2+itau*dt2-(time1+itime*dt1)); %
55                     ero4=abs(time2+itime*dt1-(tau1+itau*dt2)); %
56                     if ero3<len & ero4<len
57                         cov2=cov2+lp_fir(itime+1,lp1)*lp_fir(itau+1,lp2)*...
58                             vc_signal(ero3+1,vc)*vc_signal(ero4+1,vc);
59                     end
60                 end
61             end
62
63             end % End-if of virtual channels
64         end % End-loop over lps2
65     end % End-loop over lps2
66
67     covarRe=(cov1+cov2)/2;
68     covarIm=(cov1-cov2)/2;

```

Covariance matrices or variance vectors 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
1 % GUISDAP v1.50 94-03-10 Copyright Markku Lehtinen, Asko Huuskonen
2 %
3 % A function to calculate the error covariance matrix for two sets of result
4 % memory addresses. If the two sets are identical, only diagonal and upper
5 % (or lower) triangle is calculated. The covariance calculations are performed
6 % by the routine addr_covar
7 % Input parameters:
8 % addr1: first set of result memory addresses
9 % addr2: second set of result memory addresses
10 % T_noise: background temperature in K
11 % param: plasma parameters in scaled units
12 % vc_signal: signal strength for all virtual channels and for all lags (global)
13 % Output parameters
14 % covarRe: covariance between the real parts of the two sets
15 % covarIm: covariance between the imaginary parts of the two sets
16 % See also: addr_covar, adgr_var, calc_vcsignal, real_to_scaled
17 %function [covarRe,covarIm]=adgr_covar(addr1,addr2,T_noise,param)
18 function [covarRe,covarIm]=adgr_covar(addr1,addr2,T_noise,param)
19
20 global vc_signal lp_vc lp_dt lp_ra lp_ri lp_nt lp_t1 lp_t2 lp_dec lp_nfir lp_fir
21 calc_vcsignal(addr1(1),T_noise,param)
22
23 covarRe=zeros(length(addr1),length(addr2));
24 covarIm=zeros(length(addr1),length(addr2));
25
26 % Execution time saved if both sets are equal as Cov(A,B)=Cov(B,A)
27 % Then only diagonal and upper (or lower) triangle calculated.
28 if length(addr1)==length(addr2),
29     if all(addr1==addr2), identical=1; else, identical=0; end
30 end
31
32 for i=1:length(addr1)
33     if identical, first=i; else, first=1; end
34     fprintf(' %.0f',i);
35     for j=first:length(addr2)
36         [covarRe(i,j),covarIm(i,j)]=...
37             addr_covar(addr1(i),addr2(j),vc_signal,lp_vc,lp_dt,lp_ra,...
38                 lp_ri,lp_nt,lp_t1,lp_t2,lp_dec,lp_nfir,lp_fir);
39         if identical,
40             covarRe(j,i)=covarRe(i,j);
41             covarIm(j,i)=covarIm(i,j);
42         end

```

```

43     end
44     end
45     fprintf(' \n')

/geo/gmt/askoh/guisdap/m152/adgr_var.m
1 % GUISDAP v1.50 94-03-10 Copyright Markku Lehtinen, Asko Huuskonen
2 %
3 % A function to calculate the variance for a set of result memory addresses.
4 % The covariance calculations are performed by the routine addr_covar
5 % Input parameters:
6 % addr: set of result memory addresses
7 % T_noise: background temperature in K
8 % param: plasma parameters in scaled units
9 % vc_signal: signal strength for all virtual channels and for all lags (global)
10 % Output parameters
11 % varRe: covariance between the real parts of the two sets
12 % varIm: covariance between the imaginary parts of the two sets
13 % See also: addr_covar, adgr_covar, calc_vcsignal, real_to_scaled
14 %function [varRe,varIm]=adgr_var(addr,T_noise,param)
15 function [varRe,varIm]=adgr_var(addr,T_noise,param)
16
17 global vc_signal lp_vc lp_dt lp_ra lp_ri lp_nt lp_t1 lp_t2 lp_dec lp_nfir lp_fir
18 calc_vcsignal(addr(1),T_noise,param)
19
20 varRe=zeros(1,length(addr));
21 varIm=zeros(1,length(addr));
22
23 for i=1:length(addr)
24     [varRe(i),varIm(i)]=addr_covar(addr(i),addr(i),vc_signal,lp_vc,lp_dt,lp_ra,...
25         lp_ri,lp_nt,lp_t1,lp_t2,lp_dec,lp_nfir,lp_fir);
26
27 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
1 % GUISDAP v1.50 94-03-10 Copyright Markku Lehtinen, Asko Huuskonen
2 %
3 % Function to calculate the scaled variable values from physical ones
4 % Parameters:
5 % physical: plasma parameters in physical units
6 % scaled: plasma parameters in scaled units
7 % See also: scaled_to_real
8 %function scaled=real_to_scaled(physical)
9 function scaled=real_to_scaled(physical)
10
11 global p_m0 p_m0 p_TO p_om0 k_radar0
12
13 scaled=physical; % affects element 3 and also 6 (if specified on input)
14 scaled(:,1)=physical(:,1)/p_m0;
15 scaled(:,2)=physical(:,2)/p_TO;
16 ch=1; % hyi hyi
17 scaled(:,4)=physical(:,4)/(p_om0(ch));
18 scaled(:,5)=physical(:,5)/(p_om0(ch)/k_radar0(ch));

/geo/gmt/askoh/guisdap/m152/scaled_to_real.m
1 % GUISDAP v1.50 94-03-10 Copyright Markku Lehtinen, Asko Huuskonen
2 %
3 % Function to calculate the plasma parameters in physical units
4 % when input is in scaled values
5 % Parameters:
6 % physical: plasma parameters in physical units
7 % scaled: plasma parameters in scaled units
8 % See also: real_to_scaled
9 %function physical=scaled_to_real(scaled)
10 function physical=scaled_to_real(scaled)
11
12 global p_m0 p_m0 p_TO p_om0 k_radar0
13
14 physical=scaled; % affects element 3 and also 6 (if specified on input)
15 physical(:,1)=scaled(:,1)*p_m0;
16 physical(:,2)=scaled(:,2)*p_TO;
17 ch=1; % hyi hyi
18 physical(:,4)=scaled(:,4)*(p_om0(ch));
19 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
1 % GUIDAP v1.50 94-03-10 Copyright Markku Lehtinen, Asko Huuskonen
2 %
3 % defines the result variable as empty matrices
4 % function clear_results
5 function clear_results
6
7 global r_ind r_range r_param r_error r_res r_status r_dp
8 global r_apriori r_apriorierror
9
10 r_ind=0;
11 r_range=[];r_param=[];r_error=[];r_res=[];r_status=[];r_dp=[];
12 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
1 % GUIDAP v1.50 94-03-10 Copyright Markku Lehtinen, Asko Huuskonen
2 %
3 % function to store whole profile of results into a file
4 %
5 % function save_results
6 function save_results
7
8 global result_path name_expr name_site
9 global p_XMITloc p_RECloc sc_angle
10 global d_time GUP_ver ch_az ch_el ch_Pt p_m0 p_dtau v_lightspeed
11 global r_ver r_time r_az r_el r_Pt r_m0 r_h r_SCangle
12 global r_ind r_range r_param r_error r_res r_status r_dp
13 global r_apriori r_apriorierror
14 global pp_range pp_sigma
15
16 filename='00000000';
17 if length(d_time)>0,
18 file=int2str(tosecs(d_time(2,:)));len=length(file);
19 filename(9-len:8)=file;
20 end
21 file=canon([result_path, filename]);
22
23 r_ver=GUP_ver;
24 r_time=d_time;
25 r_az=ch_az(1);
26 r_el=ch_el(1);
27 r_Pt=ch_Pt(1);
28 r_m0=p_m0;
29 r_XMITloc=p_XMITloc;
30 r_RECloc=p_RECloc;
31 r_SCangle=sc_angle/2;
32
33 r_pp=pp_sigma;
34 r_pprange=col(pp_range)*(p_dtau*1e-6*v_lightspeed/2/1000);
35
36 r_h=col(range_to_height(r_range,ch_el(1)));
37 r_range=col(r_range)*(p_dtau*1e-6*v_lightspeed/2/1000);
38 variables=['r_ver name_expr name_site r_time r_az r_el r_Pt r_m0'];
39 variables=[variables ' r_range r_h r_param r_error r_res r_status r_dp'];
40 variables=[variables ' r_apriori r_apriorierror r_pp r_pprange'];
41 variables=[variables ' r_XMITloc r_RECloc r_SCangle'];
42
43 eval(['save ', file, '.mat ' variables])
44 % save file name to "filelist.dat"
45 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
1 % GUIDAP v1.50 94-03-10 Copyright Markku Lehtinen, Asko Huuskonen
2 %
3 % Init_graphics opens a sufficient number of figure windows.
4 % The sizes and locations are defined here.
5 % Parameter:
6 % di_figures (global): requested windows (on input) and figure handles (on output)
7
8 Positions=...
9 [ 50, 50,550,300; % Correlator dump
10 50, 50,200,400; % Raw electron density
11 600,300,600,400; % Fit results
12 580,280,600,400]; % Results
13
```

```

14 Names=...
15 ['Correlator dump, real part';
16  ' Raw electron density  ';
17  ' Fit results and quality ';
18  '      Results          '];
19
20 undefined=di_figures;
21
22 unused=[];
23 for F=get(0,'Children')
24     if strcmp(get(F,'type'),'figure')==1
25         UserData=get(F,'UserData');
26         if length(UserData)==1
27             if UserData>=1 & UserData<=4,
28                 undefined(UserData)=0;
29                 di_figures(UserData)=F;
30             else
31                 unused=[unused, F];
32             end
33         else
34             unused=[unused, F];
35         end
36     end
37 end
38
39 for fig=find(undefined);
40     if length(unused),
41         F=unused(1);
42         unused(1)=[];
43     else
44         F=figure;
45     end
46     figure(F), clf
47     di_figures(fig)=F;
48     set(F,'Position',Positions(fig,:), 'NumberTitle','off', 'Name',Names(fig,:), 'UserData',fig);
49     set(F,'DefaultAxesFontName','helvetica')
50     set(F,'DefaultTextFontName','helvetica')
51     set(F,'DefaultAxesFontWeight','bold')
52     set(F,'DefaultTextFontWeight','bold')
53     set(F,'DefaultAxesFontSize',14)
54     set(F,'DefaultTextFontSize',14)
55 end
56
57 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/guidap/m152/plot_result.m
1 % GUIDAP v1.50 94-03-10 Copyright Markku Lehtinen, Asko Huuskonen
2 %
3 % routine to produce a simple plot of the analysis results
4 %
5 % function plot_result(Handle,good_only)
6 function plot_result(Handle,good_only)
7
8 global r_param r_error r_dp r_status r_h r_el r_time
9
10 if nargin==2
11     ind=find(r_status==0);
12     if isempty(ind)
13         fprintf(' No succesfull fits in the profile, plotting it anyway\n')
14         ind=1:length(r_status);
15     end
16 else
17     ind=1:length(r_status);
18 end
19
20 r_We=r_param(ind,1)/1e11;
21 r_Ti=r_param(ind,2);
22 r_ratio=r_param(ind,3);
23 r_coll=r_param(ind,4);
24 r_vel=r_param(ind,5);
25 r_dWe=r_error(ind,1)/1e11;
26 r_dTi=r_error(ind,2);
27 r_dratio=r_error(ind,3);
28 r_dcoll=r_error(ind,4);
29 r_dvel=r_error(ind,5);
30 r_comp=r_dp(ind);
31 h=r_h(ind);
32 minheight=10*floor(min(h)/10);
33 maxheight=10*ceil(max(h)/10);
34
35 if nargin>=1,figure(Handle); else figure;end
36 set(gcf,'NextPlot','replace');clf reset

```

```

37 set(gcf,'PaperOrientation','landscape','PaperUnits','centim','PaperPos',[0 2 25 18])
38
39 axes('Position',[.1 .1 .19 .8],'NextPlot','replace')
40 p=plot(r_Ne,h,'go',[r_Ne-r_dNe,r_Ne+r_dNe],[h,h'],'g-');
41 set(p,'MarkerSize',4)
42 set(gca,'FontSize',12)
43 xlabel('Ne [1e11 m-3]')
44 ylabel('Altitude [km]')
45 set(get(gca,'XLabel'),'FontSize',12)
46 set(get(gca,'YLabel'),'FontWeight','bold')
47 ax=axis;ax(1)=0;ax(3:4)=[minheight maxheight];axis(ax);
48
49 axes('Position',[.3 .1 .19 .8],'NextPlot','replace')
50 p=plot(r_Ti,h,'bo',[r_Ti-r_dTi,r_Ti+r_dTi],[h,h'],'b-','r_Ti.*r_ratio,h','rx');
51 set(p,'MarkerSize',4)
52 set(gca,'YTickLabels',' ','FontSize',12)
53 xlabel('Ti(o), Te(x) [K]')
54 set(get(gca,'XLabel'),'FontSize',12)
55 ax=axis;ax(1)=0;ax(3:4)=[minheight maxheight];axis(ax);
56
57 axes('Position',[.5 .1 .19 .8],'NextPlot','replace')
58 p=plot(r_ratio,h,'go',[r_ratio-r_dratio,r_ratio+r_dratio],[h,h'],'g-',...
59 [1,1],[minheight maxheight],'Linewidth',0.5);
60 set(p,'MarkerSize',4)
61 set(gca,'YTickLabels',' ','FontSize',12)
62 xlabel('Te/Ti')
63 set(get(gca,'XLabel'),'FontSize',12)
64 ax=axis;ax(3:4)=[minheight maxheight];ax(1)=0;ax(2)=max(min([ceil(max(r_ratio)),4]),2);axis(ax);
65
66 axes('Position',[.7 .1 .19 .8],'NextPlot','replace')
67 p=plot(r_vel,h,'go',[r_vel-r_dvel,r_vel+r_dvel],[h,h'],'g-',...
68 [0,0],[minheight maxheight],'Linewidth',0.5);
69 set(p,'MarkerSize',4)
70 set(gca,'YTickLabels',' ','FontSize',12)
71 xlabel('Vi')
72 set(get(gca,'XLabel'),'FontSize',12)
73 ax=axis;ax(3:4)=[minheight maxheight];axis(ax);
74
75 axes('Position',[.9 .1 .09 .8],'NextPlot','replace')
76 p=plot(r_comp,h,'ro');
77 set(p,'MarkerSize',4)
78 set(gca,'YTickLabels',' ','FontSize',12)
79 xlabel('0+/Ne')
80 set(get(gca,'XLabel'),'FontSize',12)
81 ax=[-0.1 1.1 minheight maxheight];axis(ax);
82
83 axes('Position',[.1 .1 .8 .8],'NextPlot','add')
84 set(gca,'Visible','off')
85 alku=r_time(1,:);loppu=r_time(2,:);
86 tit=lsprintf('%2.0f/%2.0f/%4.0f',alku(3),alku(2),alku(1)),...
87 sprintf(' %2.0f:%2.0f:%2.0f', alku(4),alku(5),alku(6)),...
88 sprintf('-%2.0f:%2.0f:%2.0f',loppu(4),loppu(5),loppu(6));
89 title(tit);
90 set(get(gca,'Title'),'FontSize',18,'FontWeight','bold')
91
92 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
1 % GUISDAP v1.50 94-03-10 Copyright Markku Lehtinen, Asko Huuskonen
2 %
3 % function returns the result memory addresses that belong to
4 % a lag profile group.
5 % function addr=lpg_addr(lpgs)
6
7 function addr=lpg_addr(lpgs)
8
9 global lpg_ra lpg_nt lpg_ri
10
11 addr=[];
12 for lpg=lpgs
13     addr=[addr,lpg_ra(lpg)+(0:lpg_nt(lpg)-1)*lpg_ri(lpg)];
14 end

```