AIRS Code Collection

Generated by Doxygen 1.8.11

Contents

1 Main Page

The JUelich RApid Spectral SImulation Code (JURASSIC) is a fast radiative transfer model for the mid-infrared spectral region.This reference manual provides information on the algorithms and data structures used in the code. Further information can be found at: `http://www.fz-juelich.de/ias/jsc/jurassic`

2 Data Structure Index

2.1 Data Structures

Here are the data structures with brief descriptions:

3 File Index

3.1 File List

Here is a list of all files with brief descriptions:

4 Data Structure Documentation

4.1 airs_l1_t Struct Reference

AIRS Level-1 data.

```
#include <libairs.h>
```

Data Fields

- double time [L1_NTRACK][L1_NXTRACK]

    Time (seconds since 2000-01-01T00:00Z).
- double lon [L1_NTRACK][L1_NXTRACK]

    Footprint longitude [deg].
- double lat [L1_NTRACK][L1_NXTRACK]

    Footprint latitude [deg].
- double sat_z [L1_NTRACK]

    Satellite altitude [km].
- double sat_lon [L1_NTRACK]

    Satellite longitude [deg].
- double sat_lat [L1_NTRACK]

    Satellite latitude [deg].
- double nu [L1_NCHAN]

    Channel frequencies [cm^-1].
- float rad [L1_NTRACK][L1_NXTRACK][L1_NCHAN]

    Radiance [W/(m^2 sr cm^-1)].

4.1.1 Detailed Description

AIRS Level-1 data.

Definition at line 72 of file libairs.h.

4.1.2 Field Documentation

4.1.2.1 double airs_l1_t::time[L1_NTRACK][L1_NXTRACK]

Time (seconds since 2000-01-01T00:00Z).

Definition at line 75 of file libairs.h.

4.1.2.2 double airs_l1_t::lon[L1_NTRACK][L1_NXTRACK]

Footprint longitude [deg].

Definition at line 78 of file libairs.h.

4.1.2.3 double airs_l1_t::lat[L1_NTRACK][L1_NXTRACK]

Footprint latitude [deg].

Definition at line 81 of file libairs.h.

4.1.2.4 double airs_l1_t::sat_z[L1_NTRACK]

Satellite altitude [km].

Definition at line 84 of file libairs.h.

4.1.2.5 double airs_l1_t::sat_lon[L1_NTRACK]

Satellite longitude [deg].

Definition at line 87 of file libairs.h.

4.1.2.6 double airs_l1_t::sat_lat[L1_NTRACK]

Satellite latitude [deg].

Definition at line 90 of file libairs.h.

4.1.2.7 double airs_l1_t::nu[L1_NCHAN]

Channel frequencies [cm^-1].

Definition at line 93 of file libairs.h.

4.1.2.8 float airs_l1_t::rad[L1_NTRACK][L1_NXTRACK][L1_NCHAN]

Radiance [W/(m^2 sr cm^-1)].

Definition at line 96 of file libairs.h.

The documentation for this struct was generated from the following file:

- libairs.h

4.2 airs_l2_t Struct Reference

AIRS Level-2 data.

```
#include <libairs.h>
```

Data Fields

- double time [L2_NTRACK][L2_NXTRACK]

  Time (seconds since 2000-01-01T00:00Z).
- double z [L2_NTRACK][L2_NXTRACK][L2_NLAY]

  Geopotential height [km].
- double lon [L2_NTRACK][L2_NXTRACK]

  Longitude [deg].
- double lat [L2_NTRACK][L2_NXTRACK]

  Latitude [deg].
- double p [L2_NLAY]

  Pressure [hPa].
- double t [L2_NTRACK][L2_NXTRACK][L2_NLAY]

  Temperature [K].

4.2.1 Detailed Description

AIRS Level-2 data.

Definition at line 101 of file libairs.h.

4.2.2 Field Documentation

4.2.2.1 double airs_l2_t::time[L2_NTRACK][L2_NXTRACK]

Time (seconds since 2000-01-01T00:00Z).

Definition at line 104 of file libairs.h.

4.2.2.2 double airs_l2_t::z[L2_NTRACK][L2_NXTRACK][L2_NLAY]

Geopotential height [km].

Definition at line 107 of file libairs.h.

4.2.2.3 double airs_l2_t::lon[L2_NTRACK][L2_NXTRACK]

Longitude [deg].

Definition at line 110 of file libairs.h.

4.2.2.4 double airs_l2_t::lat[L2_NTRACK][L2_NXTRACK]

Latitude [deg].

Definition at line 113 of file libairs.h.

4.2.2.5 double airs_l2_t::p[L2_NLAY]

Pressure [hPa].

Definition at line 116 of file libairs.h.

4.2.2.6 double airs_l2_t::t[L2_NTRACK][L2_NXTRACK][L2_NLAY]

Temperature [K].

Definition at line 119 of file libairs.h.

The documentation for this struct was generated from the following file:

- libairs.h

4.3 atm_t Struct Reference

Atmospheric data.

```
#include <jurassic.h>
```

Data Fields

- int np
  Number of data points.
- double time [NP]
  Time (seconds since 2000-01-01T00:00Z).
- double z [NP]
  Altitude [km].
- double lon [NP]
  Longitude [deg].
- double lat [NP]
  Latitude [deg].
- double p [NP]
  Pressure [hPa].
- double t [NP]
  Temperature [K].
- double q [NG][NP]
  Volume mixing ratio.
- double k [NW][NP]
  Extinction [1/km].

4.3.1 Detailed Description

Atmospheric data.

Definition at line 206 of file jurassic.h.

4.3.2 Field Documentation

4.3.2.1 int atm_t::np

Number of data points.

Definition at line 209 of file jurassic.h.

4.3.2.2 double atm_t::time[NP]

Time (seconds since 2000-01-01T00:00Z).

Definition at line 212 of file jurassic.h.

4.3.2.3 double atm_t::z[NP]

Altitude [km].

Definition at line 215 of file jurassic.h.

4.3.2.4 double atm_t::lon[NP]

Longitude [deg].

Definition at line 218 of file jurassic.h.

4.3.2.5 double atm_t::lat[NP]

Latitude [deg].

Definition at line 221 of file jurassic.h.

4.3.2.6 double atm_t::p[NP]

Pressure [hPa].

Definition at line 224 of file jurassic.h.

4.3.2.7 double atm_t::t[NP]

Temperature [K].

Definition at line 227 of file jurassic.h.

4.3.2.8 double atm_t::q[NG][NP]

Volume mixing ratio.

Definition at line 230 of file jurassic.h.

4.3.2.9 double atm_t::k[NW][NP]

Extinction [1/km].

Definition at line 233 of file jurassic.h.

The documentation for this struct was generated from the following file:

- jurassic.h

4.4 ctl_t Struct Reference

Forward model control parameters.

```
#include <jurassic.h>
```

Data Fields

- int ng

  Number of emitters.
- char emitter [NG][LEN]

  Name of each emitter.
- int nd

  Number of radiance channels.
- int nw

  Number of spectral windows.
- double nu [ND]

  Centroid wavenumber of each channel [cm^-1].
- int window [ND]

  Window index of each channel.
- char tblbase [LEN]

  Basename for table files and filter function files.
- double hydz

  Reference height for hydrostatic pressure profile (-999 to skip) [km].
- int ctm_co2

  Compute CO2 continuum (0=no, 1=yes).
- int ctm_h2o

  Compute H2O continuum (0=no, 1=yes).
- int ctm_n2

  Compute N2 continuum (0=no, 1=yes).
- int ctm_o2

  Compute O2 continuum (0=no, 1=yes).
- int refrac

  Take into account refractivity (0=no, 1=yes).

- double rayds

    Maximum step length for raytracing [km].
- double raydz

    Vertical step length for raytracing [km].
- char fov [LEN]

    Field-of-view data file.
- double retp_zmin

    Minimum altitude for pressure retrieval [km].
- double retp_zmax

    Maximum altitude for pressure retrieval [km].
- double rett_zmin

    Minimum altitude for temperature retrieval [km].
- double rett_zmax

    Maximum altitude for temperature retrieval [km].
- double retq_zmin [NG]

    Minimum altitude for volume mixing ratio retrieval [km].
- double retq_zmax [NG]

    Maximum altitude for volume mixing ratio retrieval [km].
- double retk_zmin [NW]

    Minimum altitude for extinction retrieval [km].
- double retk_zmax [NW]

    Maximum altitude for extinction retrieval [km].
- int write_bbt

    Use brightness temperature instead of radiance (0=no, 1=yes).
- int write_matrix

    Write matrix file (0=no, 1=yes).

4.4.1 Detailed Description

Forward model control parameters.

Definition at line 238 of file jurassic.h.

4.4.2 Field Documentation

4.4.2.1 int ctl_t::ng

Number of emitters.

Definition at line 241 of file jurassic.h.

4.4.2.2 char ctl_t::emitter[NG][LEN]

Name of each emitter.

Definition at line 244 of file jurassic.h.

4.4.2.3 int ctl_t::nd

Number of radiance channels.

Definition at line 247 of file jurassic.h.

4.4.2.4 int ctl_t::nw

Number of spectral windows.

Definition at line 250 of file jurassic.h.

4.4.2.5 double ctl_t::nu[ND]

Centroid wavenumber of each channel [cm^-1].

Definition at line 253 of file jurassic.h.

4.4.2.6 int ctl_t::window[ND]

Window index of each channel.

Definition at line 256 of file jurassic.h.

4.4.2.7 char ctl_t::tblbase[LEN]

Basename for table files and filter function files.

Definition at line 259 of file jurassic.h.

4.4.2.8 double ctl_t::hydz

Reference height for hydrostatic pressure profile (-999 to skip) [km].

Definition at line 262 of file jurassic.h.

4.4.2.9 int ctl_t::ctm_co2

Compute CO2 continuum (0=no, 1=yes).

Definition at line 265 of file jurassic.h.

4.4.2.10 int ctl_t::ctm_h2o

Compute H2O continuum (0=no, 1=yes).

Definition at line 268 of file jurassic.h.

4.4.2.11 int ctl_t::ctm_n2

Compute N2 continuum (0=no, 1=yes).

Definition at line 271 of file jurassic.h.

4.4.2.12 int ctl_t::ctm_o2

Compute O2 continuum (0=no, 1=yes).

Definition at line 274 of file jurassic.h.

4.4.2.13 int ctl_t::refrac

Take into account refractivity (0=no, 1=yes).

Definition at line 277 of file jurassic.h.

4.4.2.14 double ctl_t::rayds

Maximum step length for raytracing [km].

Definition at line 280 of file jurassic.h.

4.4.2.15 double ctl_t::raydz

Vertical step length for raytracing [km].

Definition at line 283 of file jurassic.h.

4.4.2.16 char ctl_t::fov[LEN]

Field-of-view data file.

Definition at line 286 of file jurassic.h.

4.4.2.17 double ctl_t::retp_zmin

Minimum altitude for pressure retrieval [km].

Definition at line 289 of file jurassic.h.

4.4.2.18 double ctl_t::retp_zmax

Maximum altitude for pressure retrieval [km].

Definition at line 292 of file jurassic.h.

4.4.2.19 double ctl_t::rett_zmin

Minimum altitude for temperature retrieval [km].

Definition at line 295 of file jurassic.h.

4.4.2.20 double ctl_t::rett_zmax

Maximum altitude for temperature retrieval [km].

Definition at line 298 of file jurassic.h.

4.4.2.21 double ctl_t::retq_zmin[NG]

Minimum altitude for volume mixing ratio retrieval [km].

Definition at line 301 of file jurassic.h.

4.4.2.22 double ctl_t::retq_zmax[NG]

Maximum altitude for volume mixing ratio retrieval [km].

Definition at line 304 of file jurassic.h.

4.4.2.23 double ctl_t::retk_zmin[NW]

Minimum altitude for extinction retrieval [km].

Definition at line 307 of file jurassic.h.

4.4.2.24 double ctl_t::retk_zmax[NW]

Maximum altitude for extinction retrieval [km].

Definition at line 310 of file jurassic.h.

4.4.2.25 int ctl_t::write_bbt

Use brightness temperature instead of radiance (0=no, 1=yes).

Definition at line 313 of file jurassic.h.

4.4.2.26 int ctl_t::write_matrix

Write matrix file (0=no, 1=yes).

Definition at line 316 of file jurassic.h.

The documentation for this struct was generated from the following file:

- jurassic.h

4.5 los_t Struct Reference

Line-of-sight data.

```
#include <jurassic.h>
```

Data Fields

- int np
  Number of LOS points.
- double z [NLOS]
  Altitude [km].
- double lon [NLOS]
  Longitude [deg].
- double lat [NLOS]
  Latitude [deg].
- double p [NLOS]
  Pressure [hPa].
- double t [NLOS]
  Temperature [K].
- double q [NG][NLOS]
  Volume mixing ratio.
- double k [NW][NLOS]
  Extinction [1/km].
- double tsurf
  Surface temperature [K].
- double ds [NLOS]
  Segment length [km].
- double u [NG][NLOS]
  Column density [molecules/cm^2].

4.5.1 Detailed Description

Line-of-sight data.

Definition at line 321 of file jurassic.h.

4.5.2 Field Documentation

4.5.2.1 int los_t::np

Number of LOS points.

Definition at line 324 of file jurassic.h.

4.5.2.2 double los_t::z[NLOS]

Altitude [km].

Definition at line 327 of file jurassic.h.

4.5.2.3 double los_t::lon[NLOS]

Longitude [deg].

Definition at line 330 of file jurassic.h.

4.5.2.4 double los_t::lat[NLOS]

Latitude [deg].

Definition at line 333 of file jurassic.h.

4.5.2.5 double los_t::p[NLOS]

Pressure [hPa].

Definition at line 336 of file jurassic.h.

4.5.2.6 double los_t::t[NLOS]

Temperature [K].

Definition at line 339 of file jurassic.h.

4.5.2.7 double los_t::q[NG][NLOS]

Volume mixing ratio.

Definition at line 342 of file jurassic.h.

4.5.2.8 double los_t::k[NW][NLOS]

Extinction [1/km].

Definition at line 345 of file jurassic.h.

4.5.2.9 double los_t::tsurf

Surface temperature [K].

Definition at line 348 of file jurassic.h.

4.5.2.10 double los_t::ds[NLOS]

Segment length [km].

Definition at line 351 of file jurassic.h.

4.5.2.11 double los_t::u[NG][NLOS]

Column density [molecules/cm^2].

Definition at line 354 of file jurassic.h.

The documentation for this struct was generated from the following file:

- jurassic.h

4.6 met_t Struct Reference

Meteorological data.

Data Fields

- double time
  Time [s].
- int nx
  Number of longitudes.
- int ny
  Number of latitudes.
- int np
  Number of pressure levels.
- double lon [EX]
  Longitude [deg].
- double lat [EY]
  Latitude [deg].
- double p [EP]
  Pressure [hPa].
- float t [EX][EY][EP]
  Temperature [K].

4.6.1 Detailed Description

Meteorological data.

Definition at line 38 of file erafm.c.

4.6.2 Field Documentation

4.6.2.1 double met_t::time

Time [s].

Definition at line 41 of file erafm.c.

4.6.2.2 int met_t::nx

Number of longitudes.

Definition at line 44 of file erafm.c.

4.6.2.3 int met_t::ny

Number of latitudes.

Definition at line 47 of file erafm.c.

4.6.2.4 int met_t::np

Number of pressure levels.

Definition at line 50 of file erafm.c.

4.6.2.5 double met_t::lon[EX]

Longitude [deg].

Definition at line 53 of file erafm.c.

4.6.2.6 double met_t::lat[EY]

Latitude [deg].

Definition at line 56 of file erafm.c.

4.6.2.7 double met_t::p[EP]

Pressure [hPa].

Definition at line 59 of file erafm.c.

4.6.2.8 float met_t::t[EX][EY][EP]

Temperature [K].

Definition at line 62 of file erafm.c.

The documentation for this struct was generated from the following file:

- erafm.c

4.7 ncd_t Struct Reference

Data Fields

- int ncid
- int np
- double l1_time [L1_NTRACK][L1_NXTRACK]
- double l1_lon [L1_NTRACK][L1_NXTRACK]
- double l1_lat [L1_NTRACK][L1_NXTRACK]
- double l1_sat_z [L1_NTRACK]
- double l1_sat_lon [L1_NTRACK]
- double l1_sat_lat [L1_NTRACK]
- double l1_nu [L1_NCHAN]
- float l1_rad [L1_NTRACK][L1_NXTRACK][L1_NCHAN]
- double l2_z [L2_NTRACK][L2_NXTRACK][L2_NLAY]
- double l2_p [L2_NLAY]
- double l2_t [L2_NTRACK][L2_NXTRACK][L2_NLAY]
- float ret_z [NP]
- float ret_p [L1_NTRACK ∗L1_NXTRACK]
- float ret_t [L1_NTRACK ∗L1_NXTRACK ∗NP]

4.7.1 Detailed Description

Definition at line 43 of file retrieval.c.

4.7.2 Field Documentation

4.7.2.1 int ncd_t::ncid

Definition at line 46 of file retrieval.c.

4.7.2.2 int ncd_t::np

Definition at line 49 of file retrieval.c.

4.7.2.3 double ncd_t::l1_time[L1_NTRACK][L1_NXTRACK]

Definition at line 52 of file retrieval.c.

4.7.2.4 double ncd_t::l1_lon[L1_NTRACK][L1_NXTRACK]

Definition at line 55 of file retrieval.c.

4.7.2.5 double ncd_t::l1_lat[L1_NTRACK][L1_NXTRACK]

Definition at line 58 of file retrieval.c.

4.7.2.6 double ncd_t::l1_sat_z[L1_NTRACK]

Definition at line 61 of file retrieval.c.

4.7.2.7 double ncd_t::l1_sat_lon[L1_NTRACK]

Definition at line 64 of file retrieval.c.

4.7.2.8 double ncd_t::l1_sat_lat[L1_NTRACK]

Definition at line 67 of file retrieval.c.

4.7.2.9 double ncd_t::l1_nu[L1_NCHAN]

Definition at line 70 of file retrieval.c.

4.7.2.10 float ncd_t::l1_rad[L1_NTRACK][L1_NXTRACK][L1_NCHAN]

Definition at line 73 of file retrieval.c.

4.7.2.11 double ncd_t::l2_z[L2_NTRACK][L2_NXTRACK][L2_NLAY]

Definition at line 76 of file retrieval.c.

4.7.2.12 double ncd_t::l2_p[L2_NLAY]

Definition at line 79 of file retrieval.c.

4.7.2.13 double ncd_t::l2_t[L2_NTRACK][L2_NXTRACK][L2_NLAY]

Definition at line 82 of file retrieval.c.

4.7.2.14 float ncd_t::ret_z[NP]

Definition at line 85 of file retrieval.c.

4.7.2.15 float ncd_t::ret_p[L1_NTRACK ∗L1_NXTRACK]

Definition at line 88 of file retrieval.c.

4.7.2.16 float ncd_t::ret_t[L1_NTRACK ∗L1_NXTRACK ∗NP]

Definition at line 91 of file retrieval.c.

The documentation for this struct was generated from the following file:

- retrieval.c

4.8 obs_t Struct Reference

Observation geometry and radiance data.

```
#include <jurassic.h>
```

Data Fields

- int nr

  Number of ray paths.
- double time [NR]

  Time (seconds since 2000-01-01T00:00Z).
- double obsz [NR]

  Observer altitude [km].
- double obslon [NR]

  Observer longitude [deg].
- double obslat [NR]

  Observer latitude [deg].
- double vpz [NR]

  View point altitude [km].
- double vplon [NR]

  View point longitude [deg].
- double vplat [NR]

  View point latitude [deg].
- double tpz [NR]

  Tangent point altitude [km].
- double tplon [NR]

  Tangent point longitude [deg].
- double tplat [NR]

  Tangent point latitude [deg].
- double tau [ND][NR]

  Transmittance of ray path.
- double rad [ND][NR]

  Radiance [W/(m^2 sr cm^-1)].

4.8.1 Detailed Description

Observation geometry and radiance data.

Definition at line 359 of file jurassic.h.

4.8.2 Field Documentation

4.8.2.1 int obs_t::nr

Number of ray paths.

Definition at line 362 of file jurassic.h.

4.8.2.2 double obs_t::time[NR]

Time (seconds since 2000-01-01T00:00Z).

Definition at line 365 of file jurassic.h.

4.8.2.3 double obs_t::obsz[NR]

Observer altitude [km].

Definition at line 368 of file jurassic.h.

4.8.2.4 double obs_t::obslon[NR]

Observer longitude [deg].

Definition at line 371 of file jurassic.h.

4.8.2.5 double obs_t::obslat[NR]

Observer latitude [deg].

Definition at line 374 of file jurassic.h.

4.8.2.6 double obs_t::vpz[NR]

View point altitude [km].

Definition at line 377 of file jurassic.h.

4.8.2.7 double obs_t::vplon[NR]

View point longitude [deg].

Definition at line 380 of file jurassic.h.

4.8.2.8 double obs_t::vplat[NR]

View point latitude [deg].

Definition at line 383 of file jurassic.h.

4.8.2.9 double obs_t::tpz[NR]

Tangent point altitude [km].

Definition at line 386 of file jurassic.h.

4.8.2.10 double obs_t::tplon[NR]

Tangent point longitude [deg].

Definition at line 389 of file jurassic.h.

4.8.2.11 double obs_t::tplat[NR]

Tangent point latitude [deg].

Definition at line 392 of file jurassic.h.

4.8.2.12 double obs_t::tau[ND][NR]

Transmittance of ray path.

Definition at line 395 of file jurassic.h.

4.8.2.13 double obs_t::rad[ND][NR]

Radiance [W/(m^2 sr cm^-1)].

Definition at line 398 of file jurassic.h.

The documentation for this struct was generated from the following file:

- jurassic.h

4.9 pert_t Struct Reference

Perturbation data.

```
#include <libairs.h>
```

Data Fields

- int ntrack

  Number of along-track values.
- int nxtrack

  Number of across-track values.
- double time [PERT_NTRACK][PERT_NXTRACK]

  Time (seconds since 2000-01-01T00:00Z).
- double lon [PERT_NTRACK][PERT_NXTRACK]

  Longitude [deg].
- double lat [PERT_NTRACK][PERT_NXTRACK]

  Latitude [deg].
- double dc [PERT_NTRACK][PERT_NXTRACK]

  Brightness temperature (8 micron) [K].
- double bt [PERT_NTRACK][PERT_NXTRACK]

  Brightness temperature (4 or 15 micron) [K].
- double pt [PERT_NTRACK][PERT_NXTRACK]

  Brightness temperature perturbation (4 or 15 micron) [K].
- double var [PERT_NTRACK][PERT_NXTRACK]

  Brightness temperature variance (4 or 15 micron) [K].

4.9.1 Detailed Description

Perturbation data.

Definition at line 124 of file libairs.h.

4.9.2 Field Documentation

4.9.2.1 int pert_t::ntrack

Number of along-track values.

Definition at line 127 of file libairs.h.

4.9.2.2 int pert_t::nxtrack

Number of across-track values.

Definition at line 130 of file libairs.h.

4.9.2.3 double pert_t::time[PERT_NTRACK][PERT_NXTRACK]

Time (seconds since 2000-01-01T00:00Z).

Definition at line 133 of file libairs.h.

4.9.2.4 double pert_t::lon[PERT_NTRACK][PERT_NXTRACK]

Longitude [deg].

Definition at line 136 of file libairs.h.

4.9.2.5 double pert_t::lat[PERT_NTRACK][PERT_NXTRACK]

Latitude [deg].

Definition at line 139 of file libairs.h.

4.9.2.6 double pert_t::dc[PERT_NTRACK][PERT_NXTRACK]

Brightness temperature (8 micron) [K].

Definition at line 142 of file libairs.h.

4.9.2.7 double pert_t::bt[PERT_NTRACK][PERT_NXTRACK]

Brightness temperature (4 or 15 micron) [K].

Definition at line 145 of file libairs.h.

4.9.2.8 double pert_t::pt[PERT_NTRACK][PERT_NXTRACK]

Brightness temperature perturbation (4 or 15 micron) [K].

Definition at line 148 of file libairs.h.

4.9.2.9 double pert_t::var[PERT_NTRACK][PERT_NXTRACK]

Brightness temperature variance (4 or 15 micron) [K].

Definition at line 151 of file libairs.h.

The documentation for this struct was generated from the following file:

- libairs.h

4.10 ret_t Struct Reference

Retrieval results.

```
#include <libairs.h>
```

Data Fields

- int nds

  Number of data sets.
- int np

  Number of data points.
- double time [NDS][NPG]

  Time (seconds since 2000-01-01T00:00Z).
- double z [NDS][NPG]

  Altitude [km].
- double lon [NDS][NPG]

  Longitude [deg].
- double lat [NDS][NPG]

  Latitude [deg].
- double p [NDS][NPG]

  Pressure [hPa].
- double t [NDS][NPG]

  Temperature [K].
- double t_apr [NDS][NPG]

  Temperature (a priori data) [K].
- double t_tot [NDS][NPG]

  Temperature (total error) [K].
- double t_noise [NDS][NPG]

  Temperature (noise error) [K].
- double t_fm [NDS][NPG]

  Temperature (forward model error) [K].
- double t_cont [NDS][NPG]

  Temperature (measurement content).
- double t_res [NDS][NPG]

  Temperature (resolution).
- double chisq [NDS]

  Chi^2.
- int kernel_recomp
- int conv_itmax
- double conv_dmin
- double err_formod [ND]
- double err_noise [ND]
- double err_press
- double err_press_cz
- double err_press_ch
- double err_temp
- double err_temp_cz
- double err_temp_ch
- double err_q [NG]
- double err_q_cz [NG]
- double err_q_ch [NG]
- double err_k [NW]
- double err_k_cz [NW]
- double err_k_ch [NW]

4.10.1 Detailed Description

Retrieval results.

Definition at line 156 of file libairs.h.

4.10.2 Field Documentation

4.10.2.1 int ret_t::nds

Number of data sets.

Definition at line 159 of file libairs.h.

4.10.2.2 int ret_t::np

Number of data points.

Definition at line 162 of file libairs.h.

4.10.2.3 double ret_t::time[NDS][NPG]

Time (seconds since 2000-01-01T00:00Z).

Definition at line 165 of file libairs.h.

4.10.2.4 double ret_t::z[NDS][NPG]

Altitude [km].

Definition at line 168 of file libairs.h.

4.10.2.5 double ret_t::lon[NDS][NPG]

Longitude [deg].

Definition at line 171 of file libairs.h.

4.10.2.6 double ret_t::lat[NDS][NPG]

Latitude [deg].

Definition at line 174 of file libairs.h.

4.10.2.7 double ret_t::p[NDS][NPG]

Pressure [hPa].

Definition at line 177 of file libairs.h.

4.10.2.8 double ret_t::t[NDS][NPG]

Temperature [K].

Definition at line 180 of file libairs.h.

4.10.2.9 double ret_t::t_apr[NDS][NPG]

Temperature (a priori data) [K].

Definition at line 183 of file libairs.h.

4.10.2.10 double ret_t::t_tot[NDS][NPG]

Temperature (total error) [K].

Definition at line 186 of file libairs.h.

4.10.2.11 double ret_t::t_noise[NDS][NPG]

Temperature (noise error) [K].

Definition at line 189 of file libairs.h.

4.10.2.12 double ret_t::t_fm[NDS][NPG]

Temperature (forward model error) [K].

Definition at line 192 of file libairs.h.

4.10.2.13 double ret_t::t_cont[NDS][NPG]

Temperature (measurement content).

Definition at line 195 of file libairs.h.

4.10.2.14 double ret_t::t_res[NDS][NPG]

Temperature (resolution).

Definition at line 198 of file libairs.h.

4.10.2.15 double ret_t::chisq[NDS]

Chi^2.

Definition at line 201 of file libairs.h.

4.10.2.16 int ret_t::kernel_recomp

Definition at line 99 of file retrieval.c.

4.10.2.17 int ret_t::conv_itmax

Definition at line 102 of file retrieval.c.

4.10.2.18 double ret_t::conv_dmin

Definition at line 105 of file retrieval.c.

4.10.2.19 double ret_t::err_formod[ND]

Definition at line 108 of file retrieval.c.

4.10.2.20 double ret_t::err_noise[ND]

Definition at line 111 of file retrieval.c.

4.10.2.21 double ret_t::err_press

Definition at line 114 of file retrieval.c.

4.10.2.22 double ret_t::err_press_cz

Definition at line 117 of file retrieval.c.

4.10.2.23 double ret_t::err_press_ch

Definition at line 120 of file retrieval.c.

4.10.2.24 double ret_t::err_temp

Definition at line 123 of file retrieval.c.

4.10.2.25 double ret_t::err_temp_cz

Definition at line 126 of file retrieval.c.

4.10.2.26 double ret_t::err_temp_ch

Definition at line 129 of file retrieval.c.

4.10.2.27 double ret_t::err_q[NG]

Definition at line 132 of file retrieval.c.

4.10.2.28 double ret_t::err_q_cz[NG]

Definition at line 135 of file retrieval.c.

4.10.2.29 double ret_t::err_q_ch[NG]

Definition at line 138 of file retrieval.c.

4.10.2.30 double ret_t::err_k[NW]

Definition at line 141 of file retrieval.c.

4.10.2.31 double ret_t::err_k_cz[NW]

Definition at line 144 of file retrieval.c.

4.10.2.32 double ret_t::err_k_ch[NW]

Definition at line 147 of file retrieval.c.

The documentation for this struct was generated from the following files:

- libairs.h
- retrieval.c

4.11 tbl_t Struct Reference

Emissivity look-up tables.

```
#include <jurassic.h>
```

Data Fields

- int np [NG][ND]
  Number of pressure levels.
- int nt [NG][ND][TBLNP]
  Number of temperatures.
- int nu [NG][ND][TBLNP][TBLNT]
  Number of column densities.
- double p [NG][ND][TBLNP]
  Pressure [hPa].
- double t [NG][ND][TBLNP][TBLNT]
  Temperature [K].
- float u [NG][ND][TBLNP][TBLNT][TBLNU]
  Column density [molecules/cm^2].
- float eps [NG][ND][TBLNP][TBLNT][TBLNU]
  Emissivity.
- double st [TBLNS]
  Source function temperature [K].
- double sr [ND][TBLNS]
  Source function radiance [W/(m^2 sr cm^-1)].

4.11.1 Detailed Description

Emissivity look-up tables.

Definition at line 403 of file jurassic.h.

4.11.2 Field Documentation

4.11.2.1 int tbl_t::np[NG][ND]

Number of pressure levels.

Definition at line 406 of file jurassic.h.

4.11.2.2 int tbl_t::nt[NG][ND][TBLNP]

Number of temperatures.

Definition at line 409 of file jurassic.h.

4.11.2.3 int tbl_t::nu[NG][ND][TBLNP][TBLNT]

Number of column densities.

Definition at line 412 of file jurassic.h.

4.11.2.4 double tbl_t::p[NG][ND][TBLNP]

Pressure [hPa].

Definition at line 415 of file jurassic.h.

4.11.2.5 double tbl_t::t[NG][ND][TBLNP][TBLNT]

Temperature [K].

Definition at line 418 of file jurassic.h.

4.11.2.6 float tbl_t::u[NG][ND][TBLNP][TBLNT][TBLNU]

Column density [molecules/cm^2].

Definition at line 421 of file jurassic.h.

4.11.2.7 float tbl_t::eps[NG][ND][TBLNP][TBLNT][TBLNU]

Emissivity.

Definition at line 424 of file jurassic.h.

4.11.2.8 double tbl_t::st[TBLNS]

Source function temperature [K].

Definition at line 427 of file jurassic.h.

4.11.2.9 double tbl_t::sr[ND][TBLNS]

Source function radiance [W/(m^2 sr cm^-1)].

Definition at line 430 of file jurassic.h.

The documentation for this struct was generated from the following file:

- jurassic.h

4.12 wave_t Struct Reference

Wave analysis data.

```
#include <libairs.h>
```

Data Fields

- int nx
  Number of across-track values.
- int ny
  Number of along-track values.
- double time
  Time (seconds since 2000-01-01T00:00Z).
- double z
  Altitude [km].
- double lon [WX][WY]
  Longitude [deg].
- double lat [WX][WY]
  Latitude [deg].
- double x [WX]
  Across-track distance [km].
- double y [WY]
  Along-track distance [km].
- double temp [WX][WY]
  Temperature [K].
- double bg [WX][WY]
  Background [K].
- double pt [WX][WY]
  Perturbation [K].
- double var [WX][WY]
  Variance [K].

4.12.1 Detailed Description

Wave analysis data.

Definition at line 206 of file libairs.h.

4.12.2 Field Documentation

4.12.2.1 int wave_t::nx

Number of across-track values.

Definition at line 209 of file libairs.h.

4.12.2.2 int wave_t::ny

Number of along-track values.

Definition at line 212 of file libairs.h.

4.12.2.3 double wave_t::time

Time (seconds since 2000-01-01T00:00Z).

Definition at line 215 of file libairs.h.

4.12.2.4 double wave_t::z

Altitude [km].

Definition at line 218 of file libairs.h.

4.12.2.5 double wave_t::lon[WX][WY]

Longitude [deg].

Definition at line 221 of file libairs.h.

4.12.2.6 double wave_t::lat[WX][WY]

Latitude [deg].

Definition at line 224 of file libairs.h.

4.12.2.7 double wave_t::x[WX]

Across-track distance [km].

Definition at line 227 of file libairs.h.

4.12.2.8 double wave_t::y[WY]

Along-track distance [km].

Definition at line 230 of file libairs.h.

4.12.2.9 double wave_t::temp[WX][WY]

Temperature [K].

Definition at line 233 of file libairs.h.

4.12.2.10 double wave_t::bg[WX][WY]

Background [K].

Definition at line 236 of file libairs.h.

4.12.2.11 double wave_t::pt[WX][WY]

Perturbation [K].

Definition at line 239 of file libairs.h.

4.12.2.12 double wave_t::var[WX][WY]

Variance [K].

Definition at line 242 of file libairs.h.

The documentation for this struct was generated from the following file:

- libairs.h

5 File Documentation

5.1 arpp.c File Reference

Functions

- void fill_gaps (double x[L2_NTRACK][L2_NXTRACK][L2_NLAY], double dx, double dy)
- int init_l2 (airs_l2_t ∗l2, int track, int xtrack, ctl_t ∗ctl, atm_t ∗atm)
- int main (int argc, char ∗argv[ ])

5.1.1 Function Documentation

5.1.1.1 void fill_gaps (double *x[L2_NTRACK][L2_NXTRACK][L2_NLAY]*, double *dx*, double *dy*)

Definition at line 189 of file arpp.c.

```
               {

  static double help[L2_NTRACK][L2_NXTRACK], w, wsum;

  int lay, track, track2, xtrack, xtrack2;

  /* Loop over layers... */
  for (lay = 0; lay < L2_NLAY; lay++) {

    /* Loop over grid points... */
    for (track = 0; track < L2_NTRACK; track++)
      for (xtrack = 0; xtrack < L2_NXTRACK; xtrack++) {

        /* Init... */
        help[track][xtrack] = 0;
        wsum = 0;

        /* Averrage data points... */
        for (track2 = 0; track2 < L2_NTRACK; track2++)
          for (xtrack2 = 0; xtrack2 < L2_NXTRACK; xtrack2++)
            if (gsl_finite(x[track2][xtrack2][lay])
              && x[track2][xtrack2][lay] > 0) {
              w = exp(-gsl_pow_2((xtrack - xtrack2) / dx)
                      - gsl_pow_2((track - track2) / dy));
              help[track][xtrack] += w * x[track2][xtrack2][lay];
              wsum += w;
            }

        /* Normalize... */
        if (wsum > 0)
          help[track][xtrack] /= wsum;
        else
          help[track][xtrack] = GSL_NAN;
      }

    /* Copy grid points... */
    for (track = 0; track < L2_NTRACK; track++)
      for (xtrack = 0; xtrack < L2_NXTRACK; xtrack++)
        x[track][xtrack][lay] = help[track][xtrack];
  }
}
```

5.1.1.2 int init_l2 (airs_l2_t * *l2*, int *track*, int *xtrack*, ctl_t * *ctl*, atm_t * *atm*)

Definition at line 236 of file arpp.c.

```
                 {

  static atm_t atm_airs;

  double k[NW], p, q[NG], t, w, zmax = 0, zmin = 1000;

  int ip, lay;

  /* Reset track- and xtrack-index to match Level-2 data... */
  track /= 3;
  xtrack /= 3;

  /* Store AIRS data in atmospheric data struct... */
  atm_airs.np = 0;
  for (lay = 0; lay < L2_NLAY; lay++)
    if (gsl_finite(l2->z[track][xtrack][lay])) {
      atm_airs.time[atm_airs.np] = l2->time[track][xtrack];
      atm_airs.z[atm_airs.np] = l2->z[track][xtrack][lay];
      atm_airs.lon[atm_airs.np] = l2->lon[track][xtrack];
      atm_airs.lat[atm_airs.np] = l2->lat[track][xtrack];
      atm_airs.p[atm_airs.np] = l2->p[lay];
      atm_airs.t[atm_airs.np] = l2->t[track][xtrack][lay];
      atm_airs.np++;
    }
```

```

  /* Check number of levels... */
  if (atm_airs.np <= 0)
    return 0;

  /* Get height range of AIRS data... */
  for (ip = 0; ip < atm_airs.np; ip++) {
    zmax = GSL_MAX(zmax, atm_airs.z[ip]);
    zmin = GSL_MIN(zmin, atm_airs.z[ip]);
  }

  /* Merge AIRS data... */
  for (ip = 0; ip < atm->np; ip++) {

    /* Interpolate AIRS data... */
    intpol_atm(ctl, &atm_airs, atm->z[ip], &p, &t, q, k);

    /* Weighting factor... */
    w = 1;
    if (atm->z[ip] > zmax)
      w = GSL_MAX(1 - (atm->z[ip] - zmax) / 50, 0);
    if (atm->z[ip] < zmin)
      w = GSL_MAX(1 - (zmin - atm->z[ip]) / 50, 0);

    /* Merge... */
    atm->t[ip] = w * t + (1 - w) * atm->t[ip];
    atm->p[ip] = w * p + (1 - w) * atm->p[ip];
  }

  return 1;
}
```

Here is the call graph for this function:

5.2 arpp.c

```
#include "libairs.h"

/* ------------------------------------------------------------
   Functions...
   ------------------------------------------------------------ */

/* Fill data gaps in L2 data. */
void fill_gaps(
  double x[L2_NTRACK][L2_NXTRACK][L2_NLAY],
  double dx,
  double dy);

/* Initialize with AIRS Level-2 data. */
int init_l2(
  airs_l2_t * l2,
  int track,
  int xtrack,
  ctl_t * ctl,
  atm_t * atm);

/* ------------------------------------------------------------
   Main...
   ------------------------------------------------------------ */

int main(
  int argc,
  char *argv[]) {

  static atm_t atm, atm_clim;
  static ctl_t ctl;
  static obs_t obs;

  static airs_l1_t l1;
  static airs_l2_t l2;

  FILE *dirlist;

  char cmd[LEN], dirname[LEN];

  double cov = 0, cov_thresh, dx, dy, lat0, lat1, lon0, lon1,
    sza_thresh, z[NP];

  int channel[ND], i, id, ip, iz, nz,
    track, track0, track1, xtrack, xtrack0, xtrack1;

```

```
  /* ------------------------------------------------------------
     Read control parameters...
     ------------------------------------------------------------ */

  /* Check arguments... */
  if (argc < 5)
    ERRMSG("Give parameters: <ctl> <airs.nc> <basedir> <dirlist>");

  /* Read control parameters... */
  read_ctl(argc, argv, &ctl);

  /* Read retrieval grid... */
  nz = (int) scan_ctl(argc, argv, "NZ", -1, "", NULL);
  for (iz = 0; iz < nz; iz++)
    z[iz] = scan_ctl(argc, argv, "Z", iz, "", NULL);

  /* Read track range and sampling step... */
  track0 = (int) scan_ctl(argc, argv, "TRACK_MIN", -1, "1", NULL);
  track1 = (int) scan_ctl(argc, argv, "TRACK_MAX", -1, "135", NULL);

  /* Read xtrack range and sampling step... */
  xtrack0 = (int) scan_ctl(argc, argv, "XTRACK_MIN", -1, "1", NULL);
  xtrack1 = (int) scan_ctl(argc, argv, "XTRACK_MAX", -1, "90", NULL);

  /* Read box coordinates... */
  lon0 = scan_ctl(argc, argv, "LON_MIN", -1, "-180", NULL);
  lon1 = scan_ctl(argc, argv, "LON_MAX", -1, "180", NULL);
  lat0 = scan_ctl(argc, argv, "LAT_MIN", -1, "-90", NULL);
  lat1 = scan_ctl(argc, argv, "LAT_MAX", -1, "90", NULL);
  cov_thresh = scan_ctl(argc, argv, "COV_THRESH", -1, "50", NULL);

  /* Smoothing of background... */
  dx = scan_ctl(argc, argv, "DX", -1, "8", NULL);
  dy = scan_ctl(argc, argv, "DY", -1, "2", NULL);

  /* Read SZA threshold... */
  sza_thresh = scan_ctl(argc, argv, "SZA_THRESH", -1, "96", NULL);

  /* ------------------------------------------------------------
     Read AIRS data and initialize...
     ------------------------------------------------------------ */

  /* Read AIRS data... */
  read_l1(argv[2], &l1);
  read_l2(argv[2], &l2);

  /* Check coverage of box... */
  for (track = 0; track < L1_NTRACK; track++)
    for (xtrack = 0; xtrack < L1_NXTRACK; xtrack++)
      if (l1.lon[track][xtrack] >= lon0 && l1.lon[track][xtrack] <= lon1 &&
          l1.lat[track][xtrack] >= lat0 && l1.lat[track][xtrack] <= lat1)
        cov += 100. / (L1_NTRACK * L1_NXTRACK);
  if (cov < cov_thresh)
    return EXIT_SUCCESS;

  /* Identify radiance channels... */
  for (id = 0; id < ctl.nd; id++) {
    channel[id] = -999;
    for (i = 0; i < L1_NCHAN; i++)
      if (fabs(ctl.nu[id] - l1.nu[i]) < 0.1)
        channel[id] = i;
    if (channel[id] < 0)
      ERRMSG("Cannot identify radiance channel!");
  }

  /* Fill data gaps... */
  fill_gaps(l2.z, dx, dy);
  fill_gaps(l2.t, dx, dy);

  /* Set climatological data for center of granule... */
  atm_clim.np = nz;
  for (iz = 0; iz < nz; iz++) {
    atm_clim.time[iz] = l1.time[L1_NTRACK / 2][L1_NXTRACK / 2];
    atm_clim.z[iz] = z[iz];
    atm_clim.lon[iz] = l1.lon[L1_NTRACK / 2][L1_NXTRACK / 2];
    atm_clim.lat[iz] = l1.lat[L1_NTRACK / 2][L1_NXTRACK / 2];
  }
  climatology(&ctl, &atm_clim);

  /* ------------------------------------------------------------
     Prepare atmospheric data and observation data...
     ------------------------------------------------------------ */

  /* Create directory list... */
  if (!(dirlist = fopen(argv[4], "w")))
    ERRMSG("Cannot create directory list!");

```

```
  /* Loop over swaths and scans... */
  for (track = track0 - 1; track < track1; track++)
    for (xtrack = xtrack0 - 1; xtrack < xtrack1; xtrack++) {

      /* Store observation data... */
      obs.nr = 1;
      obs.time[0] = l1.time[track][xtrack];
      obs.obsz[0] = l1.sat_z[track];
      obs.obslon[0] = l1.sat_lon[track];
      obs.obslat[0] = l1.sat_lat[track];
      obs.vpz[0] = 0;
      obs.vplon[0] = l1.lon[track][xtrack];
      obs.vplat[0] = l1.lat[track][xtrack];
      for (id = 0; id < ctl.nd; id++)
        obs.rad[id][0] = l1.rad[track][xtrack][channel[id]];

      /* Flag out 4 micron channels for daytime measurements... */
      if (sza(obs.time[0], obs.obslon[0], obs.obslat[0]) < sza_thresh)
        for (id = 0; id < ctl.nd; id++)
          if (ctl.nu[id] >= 2000)
            obs.rad[id][0] = GSL_NAN;

      /* Prepare atmospheric data... */
      copy_atm(&ctl, &atm, &atm_clim, 0);
      for (ip = 0; ip < atm.np; ip++) {
        atm.time[ip] = obs.time[0];
        atm.lon[ip] = obs.vplon[0];
        atm.lat[ip] = obs.vplat[0];
      }

      /* Merge Level-2 data... */
      if (!init_l2(&l2, track, xtrack, &ctl, &atm))
        continue;

      /* Create directory... */
      sprintf(dirname, "%s/swath_%d/scan_%d", argv[3], track + 1, xtrack + 1);
      sprintf(cmd, "mkdir -p %s", dirname);
      if (system(cmd))
        ERRMSG("Cannot create directory!");
      fprintf(dirlist, "%s\n", dirname);

      /* Write observation data... */
      write_obs(dirname, "obs_meas.tab", &ctl, &obs);

      /* Write atmospheric data... */
      write_atm(dirname, "atm_apr.tab", &ctl, &atm);
    }

  /* Close directory list... */
  fclose(dirlist);

  return EXIT_SUCCESS;
}

/*****************************************************************************/

void fill_gaps(
  double x[L2_NTRACK][L2_NXTRACK][L2_NLAY],
  double dx,
  double dy) {

  static double help[L2_NTRACK][L2_NXTRACK], w, wsum;

  int lay, track, track2, xtrack, xtrack2;

  /* Loop over layers... */
  for (lay = 0; lay < L2_NLAY; lay++) {

    /* Loop over grid points... */
    for (track = 0; track < L2_NTRACK; track++)
      for (xtrack = 0; xtrack < L2_NXTRACK; xtrack++) {

        /* Init... */
        help[track][xtrack] = 0;
        wsum = 0;

        /* Averrage data points... */
        for (track2 = 0; track2 < L2_NTRACK; track2++)
          for (xtrack2 = 0; xtrack2 < L2_NXTRACK; xtrack2++)
            if (gsl_finite(x[track2][xtrack2][lay])
                && x[track2][xtrack2][lay] > 0) {
              w = exp(-gsl_pow_2((xtrack - xtrack2) / dx)
                      - gsl_pow_2((track - track2) / dy));
              help[track][xtrack] += w * x[track2][xtrack2][lay];
              wsum += w;
            }

```

```
      /* Normalize... */
      if (wsum > 0)
        help[track][xtrack] /= wsum;
      else
        help[track][xtrack] = GSL_NAN;
    }

  /* Copy grid points... */
  for (track = 0; track < L2_NTRACK; track++)
    for (xtrack = 0; xtrack < L2_NXTRACK; xtrack++)
      x[track][xtrack][lay] = help[track][xtrack];
  }
}

/*****************************************************************************/

int init_l2(
  airs_l2_t * l2,
  int track,
  int xtrack,
  ctl_t * ctl,
  atm_t * atm) {

  static atm_t atm_airs;

  double k[NW], p, q[NG], t, w, zmax = 0, zmin = 1000;

  int ip, lay;

  /* Reset track- and xtrack-index to match Level-2 data... */
  track /= 3;
  xtrack /= 3;

  /* Store AIRS data in atmospheric data struct... */
  atm_airs.np = 0;
  for (lay = 0; lay < L2_NLAY; lay++)
    if (gsl_finite(l2->z[track][xtrack][lay])) {
      atm_airs.time[atm_airs.np] = l2->time[track][xtrack];
      atm_airs.z[atm_airs.np] = l2->z[track][xtrack][lay];
      atm_airs.lon[atm_airs.np] = l2->lon[track][xtrack];
      atm_airs.lat[atm_airs.np] = l2->lat[track][xtrack];
      atm_airs.p[atm_airs.np] = l2->p[lay];
      atm_airs.t[atm_airs.np] = l2->t[track][xtrack][lay];
      atm_airs.np++;
    }

  /* Check number of levels... */
  if (atm_airs.np <= 0)
    return 0;

  /* Get height range of AIRS data... */
  for (ip = 0; ip < atm_airs.np; ip++) {
    zmax = GSL_MAX(zmax, atm_airs.z[ip]);
    zmin = GSL_MIN(zmin, atm_airs.z[ip]);
  }

  /* Merge AIRS data... */
  for (ip = 0; ip < atm->np; ip++) {

    /* Interpolate AIRS data... */
    intpol_atm(ctl, &atm_airs, atm->z[ip], &p, &t, q, k);

    /* Weighting factor... */
    w = 1;
    if (atm->z[ip] > zmax)
      w = GSL_MAX(1 - (atm->z[ip] - zmax) / 50, 0);
    if (atm->z[ip] < zmin)
      w = GSL_MAX(1 - (zmin - atm->z[ip]) / 50, 0);

    /* Merge... */
    atm->t[ip] = w * t + (1 - w) * atm->t[ip];
    atm->p[ip] = w * p + (1 - w) * atm->p[ip];
  }

  return 1;
}
```

5.3 autocorr.c File Reference

Functions

- int main (int argc, char *argv[ ])

5.3.1 Function Documentation

5.3.1.1 int main (int *argc*, char ∗ *argv[ ]*)

Definition at line 14 of file autocorr.c.

```
00016                 {
00017
00018   FILE *in;
00019
00020   static double x[NMAX], y[NMAX], y2[NMAX], y3[NMAX],
00021     t, work[2 * NMAX], v0, t0, t1;
00022
00023   static int i, idx, it, n, n2, nt, rm;
00024
00025   /* Write info... */
00026   if (argc != 6)
00027     ERRMSG("Give parameters: <xy.tab> <rm> var0> <t0> <t1>");
00028   rm = atoi(argv[2]);
00029   v0 = atof(argv[3]);
00030   t0 = atof(argv[4]);
00031   t1 = atof(argv[5]);
00032
00033   /* Read data... */
00034   printf("# Read time series: %s\n\n", argv[1]);
00035   if (!(in = fopen(argv[1], "r")))
00036     ERRMSG("Cannot open file!");
00037   while (fscanf(in, "%lg %lg", &x[n], &y[n]) == 2)
00038     if (x[n] >= t0 && x[n] <= t1)
00039       if (++n >= NMAX)
00040         ERRMSG("Too many data points!");
00041   fclose(in);
00042
00043   /* Interpolate time series... */
00044   for (t = x[0]; t <= x[n - 1]; t += 86400) {
00045     idx = locate(x, n, t);
00046     y2[n2] = LIN(x[idx], y[idx], x[idx + 1], y[idx + 1], t);
00047     if (++n2 >= NMAX)
00048       ERRMSG("Too many data points!");
00049   }
00050
00051   /* Remove running mean... */
00052   for (i = 0; i < n2; i++) {
00053     y3[i] = 0;
00054     nt = 0;
00055     for (it = -rm; it <= rm; it++)
00056       if (i + it >= 0 && i + it < n2) {
00057         y3[i] += y2[i + it];
00058         nt++;
00059       }
00060     y3[i] /= nt;
00061   }
00062   nt = n2;
00063   n2 = 0;
00064   for (i = 0; i < nt; i++)
00065     if (y3[i] > v0) {
00066       y2[n2] = y2[i] - y3[i];
00067       n2++;
00068     }
00069
00070   /* Loop over time lag... */
00071   for (it = 0; it < n2; it++) {
00072
00073     /* Shift time series... */
00074     for (i = 0; i < n2; i++)
00075       y3[i] = y2[i + it < n2 ? i + it : i + it - n2];
00076
00077     /* Get correlation coefficient... */
00078     printf("%d %g %g\n", it,
00079            gsl_stats_correlation(y2, 1, y3, 1, (size_t) n2),
00080            gsl_stats_spearman(y2, 1, y3, 1, (size_t) n2, work));
00081   }
00082
00083   return (EXIT_SUCCESS);
00084 }
```

Here is the call graph for this function:

5.4 autocorr.c

```c
#include "libairs.h"

/* ------------------------------------------------------------
   Definitions...
   ------------------------------------------------------------ */

/* Maximum number of data points. */
#define NMAX 1000000

/* ------------------------------------------------------------
   Macros...
   ------------------------------------------------------------ */

int main(
  int argc,
  char *argv[]) {

  FILE *in;

  static double x[NMAX], y[NMAX], y2[NMAX], y3[NMAX],
    t, work[2 * NMAX], v0, t0, t1;

  static int i, idx, it, n, n2, nt, rm;

  /* Write info... */
  if (argc != 6)
    ERRMSG("Give parameters: <xy.tab> <rm> var0> <t0> <t1>");
  rm = atoi(argv[2]);
  v0 = atof(argv[3]);
  t0 = atof(argv[4]);
  t1 = atof(argv[5]);

  /* Read data... */
  printf("# Read time series: %s\n\n", argv[1]);
  if (!(in = fopen(argv[1], "r")))
    ERRMSG("Cannot open file!");
  while (fscanf(in, "%lg %lg", &x[n], &y[n]) == 2)
    if (x[n] >= t0 && x[n] <= t1)
      if (++n >= NMAX)
        ERRMSG("Too many data points!");
  fclose(in);

  /* Interpolate time series... */
  for (t = x[0]; t <= x[n - 1]; t += 86400) {
    idx = locate(x, n, t);
    y2[n2] = LIN(x[idx], y[idx], x[idx + 1], y[idx + 1], t);
    if (++n2 >= NMAX)
      ERRMSG("Too many data points!");
  }

  /* Remove running mean... */
  for (i = 0; i < n2; i++) {
    y3[i] = 0;
    nt = 0;
    for (it = -rm; it <= rm; it++)
      if (i + it >= 0 && i + it < n2) {
        y3[i] += y2[i + it];
        nt++;
      }
    y3[i] /= nt;
  }
  nt = n2;
  n2 = 0;
  for (i = 0; i < nt; i++)
    if (y3[i] > v0) {
      y2[n2] = y2[i] - y3[i];
      n2++;
    }

  /* Loop over time lag... */
  for (it = 0; it < n2; it++) {

    /* Shift time series... */
    for (i = 0; i < n2; i++)
      y3[i] = y2[i + it < n2 ? i + it : i + it - n2];

    /* Get correlation coefficient... */
    printf("%d %g %g\n", it,
           gsl_stats_correlation(y2, 1, y3, 1, (size_t) n2),
           gsl_stats_spearman(y2, 1, y3, 1, (size_t) n2, work));
  }

  return (EXIT_SUCCESS);
}
```

5.5 bands.c File Reference

Functions

- int main (int argc, char ∗argv[ ])

5.5.1 Function Documentation

5.5.1.1 int main (int *argc*, char ∗ *argv[ ]*)

Definition at line 14 of file bands.c.

```
                  {

  FILE *out;

  static airs_rad_gran_t airs_rad_gran;

  static double rad[NB];

  static int chan_min[NB], chan_max[NB], iarg, ib, ichan, n, nb, track,
    xtrack;

  /* Check arguments... */
  if (argc < 4)
    ERRMSG("Give parameters: <ctl> <out.tab> <l1b_file1> [<l1b_file2> ...]");

  /* Get control parameters... */
  nb = (int) scan_ctl(argc, argv, "NB", -1, "1", NULL);
  if (nb > NB)
    ERRMSG("Too many bands!");
  for (ib = 0; ib < nb; ib++) {
    chan_min[ib] = (int) scan_ctl(argc, argv, "CHAN_MIN", ib, "", NULL);
    if (chan_min[ib] < 0 || chan_min[ib] >= AIRS_RAD_CHANNEL)
      ERRMSG("Channel index out of range!");
    chan_max[ib] = (int) scan_ctl(argc, argv, "CHAN_MAX", ib, "", NULL);
    if (chan_max[ib] < 0 || chan_max[ib] >= AIRS_RAD_CHANNEL)
      ERRMSG("Channel index out of range!");
  }

  /* Create file... */
  printf("Write band data: %s\n", argv[2]);
  if (!(out = fopen(argv[2], "w")))
    ERRMSG("Cannot create file!");

  /* Loop over HDF files... */
  for (iarg = 3; iarg < argc; iarg++) {

    /* Read AIRS data... */
    printf("Read AIRS Level-1B data file: %s\n", argv[iarg]);
    airs_rad_rdr(argv[iarg], &airs_rad_gran);

    /* Write header... */
    if (iarg == 3) {
      fprintf(out,
              "# $1 = time [s]\n"
              "# $2 = footprint longitude [deg]\n"
              "# $3 = footprint latitude [deg]\n"
              "# $4 = satellite altitude [km]\n"
              "# $5 = satellite longitude [deg]\n"
              "# $6 = satellite latitude [deg]\n");
      for (ib = 0; ib < nb; ib++)
        fprintf(out,
                "# $%d = BT(%.2f/cm...%.2f/cm) [K]\n",
                7 + ib, airs_rad_gran.nominal_freq[chan_min[ib]],
                airs_rad_gran.nominal_freq[chan_max[ib]]);
    }

    /* Flag bad observations... */
    for (track = 0; track < AIRS_RAD_GEOTRACK; track++)
      for (xtrack = 0; xtrack < AIRS_RAD_GEOXTRACK; xtrack++)
        for (ichan = 0; ichan < AIRS_RAD_CHANNEL; ichan++)
          if ((airs_rad_gran.state[track][xtrack] != 0)
              || (airs_rad_gran.ExcludedChans[ichan] > 2)
              || (airs_rad_gran.CalChanSummary[ichan] & 8)
              || (airs_rad_gran.CalChanSummary[ichan] & (32 + 64))
              || (airs_rad_gran.CalFlag[track][ichan] & 16))
```

```
            airs_rad_gran.radiances[track][xtrack][ichan] = GSL_NAN;

    /* Loop over scans... */
    for (track = 0; track < AIRS_RAD_GEOTRACK; track++) {

      /* Write output... */
      fprintf(out, "\n");

      /* Loop over footprints... */
      for (xtrack = 0; xtrack < AIRS_RAD_GEOXTRACK; xtrack++) {

        /* Write output... */
        fprintf(out, "%.2f %.4f %.4f %.3f %.4f %.4f",
                airs_rad_gran.Time[track][xtrack] - 220838400,
                airs_rad_gran.Longitude[track][xtrack],
                airs_rad_gran.Latitude[track][xtrack],
                airs_rad_gran.satheight[track],
                airs_rad_gran.sat_lon[track], airs_rad_gran.sat_lat[track]);

        /* Loop over bands... */
        for (ib = 0; ib < nb; ib++) {

          /* Get mean radiance... */
          n = 0;
          rad[ib] = 0;
          for (ichan = chan_min[ib]; ichan <= chan_max[ib]; ichan++)
            if (gsl_finite(airs_rad_gran.radiances[track][xtrack][ichan])) {
              rad[ib] += airs_rad_gran.radiances[track][xtrack][ichan];
              n++;
            }
          if (n > 0)
            rad[ib] /= n;
          else
            rad[ib] = GSL_NAN;

          /* Convert to brightness temperature... */
          rad[ib] = brightness(rad[ib] * 0.001,
                               0.5 *
                               (airs_rad_gran.nominal_freq[chan_min[ib]] +
                                airs_rad_gran.nominal_freq[chan_max[ib]]));

          /* Write output... */
          fprintf(out, " %.3f", rad[ib]);
        }

        /* Write output... */
        fprintf(out, "\n");
      }
    }
  }

  /* Close file... */
  fclose(out);

  return EXIT_SUCCESS;
}
```

Here is the call graph for this function:

5.6 bands.c

```
#include "libairs.h"

/* ------------------------------------------------------------
   Dimensions...
   ------------------------------------------------------------ */

/* Maximum number of bands... */
#define NB 100

/* ------------------------------------------------------------
   Main...
   ------------------------------------------------------------ */

int main(
  int argc,
  char *argv[]) {

  FILE *out;

  static airs_rad_gran_t airs_rad_gran;
```

```
static double rad[NB];

static int chan_min[NB], chan_max[NB], iarg, ib, ichan, n, nb, track,
  xtrack;

/* Check arguments... */
if (argc < 4)
  ERRMSG("Give parameters: <ctl> <out.tab> <l1b_file1> [<l1b_file2> ...]");

/* Get control parameters... */
nb = (int) scan_ctl(argc, argv, "NB", -1, "1", NULL);
if (nb > NB)
  ERRMSG("Too many bands!");
for (ib = 0; ib < nb; ib++) {
  chan_min[ib] = (int) scan_ctl(argc, argv, "CHAN_MIN", ib, "", NULL);
  if (chan_min[ib] < 0 || chan_min[ib] >= AIRS_RAD_CHANNEL)
    ERRMSG("Channel index out of range!");
  chan_max[ib] = (int) scan_ctl(argc, argv, "CHAN_MAX", ib, "", NULL);
  if (chan_max[ib] < 0 || chan_max[ib] >= AIRS_RAD_CHANNEL)
    ERRMSG("Channel index out of range!");
}

/* Create file... */
printf("Write band data: %s\n", argv[2]);
if (!(out = fopen(argv[2], "w")))
  ERRMSG("Cannot create file!");

/* Loop over HDF files... */
for (iarg = 3; iarg < argc; iarg++) {

  /* Read AIRS data... */
  printf("Read AIRS Level-1B data file: %s\n", argv[iarg]);
  airs_rad_rdr(argv[iarg], &airs_rad_gran);

  /* Write header... */
  if (iarg == 3) {
    fprintf(out,
            "# $1 = time [s]\n"
            "# $2 = footprint longitude [deg]\n"
            "# $3 = footprint latitude [deg]\n"
            "# $4 = satellite altitude [km]\n"
            "# $5 = satellite longitude [deg]\n"
            "# $6 = satellite latitude [deg]\n");
    for (ib = 0; ib < nb; ib++)
      fprintf(out,
              "# $%d = BT(%.2f/cm...%.2f/cm) [K]\n",
              7 + ib, airs_rad_gran.nominal_freq[chan_min[ib]],
              airs_rad_gran.nominal_freq[chan_max[ib]]);
  }

  /* Flag bad observations... */
  for (track = 0; track < AIRS_RAD_GEOTRACK; track++)
    for (xtrack = 0; xtrack < AIRS_RAD_GEOXTRACK; xtrack++)
      for (ichan = 0; ichan < AIRS_RAD_CHANNEL; ichan++)
        if ((airs_rad_gran.state[track][xtrack] != 0)
            || (airs_rad_gran.ExcludedChans[ichan] > 2)
            || (airs_rad_gran.CalChanSummary[ichan] & 8)
            || (airs_rad_gran.CalChanSummary[ichan] & (32 + 64))
            || (airs_rad_gran.CalFlag[track][ichan] & 16))
          airs_rad_gran.radiances[track][xtrack][ichan] = GSL_NAN;

  /* Loop over scans... */
  for (track = 0; track < AIRS_RAD_GEOTRACK; track++) {

    /* Write output... */
    fprintf(out, "\n");

    /* Loop over footprints... */
    for (xtrack = 0; xtrack < AIRS_RAD_GEOXTRACK; xtrack++) {

      /* Write output... */
      fprintf(out, "%.2f %.4f %.4f %.3f %.4f %.4f",
              airs_rad_gran.Time[track][xtrack] - 220838400,
              airs_rad_gran.Longitude[track][xtrack],
              airs_rad_gran.Latitude[track][xtrack],
              airs_rad_gran.satheight[track],
              airs_rad_gran.sat_lon[track], airs_rad_gran.sat_lat[track]);

      /* Loop over bands... */
      for (ib = 0; ib < nb; ib++) {

        /* Get mean radiance... */
        n = 0;
        rad[ib] = 0;
        for (ichan = chan_min[ib]; ichan <= chan_max[ib]; ichan++)
          if (gsl_finite(airs_rad_gran.radiances[track][xtrack][ichan])) {
```

```
              rad[ib] += airs_rad_gran.radiances[track][xtrack][ichan];
              n++;
            }
          if (n > 0)
            rad[ib] /= n;
          else
            rad[ib] = GSL_NAN;

          /* Convert to brightness temperature... */
          rad[ib] = brightness(rad[ib] * 0.001,
                               0.5 *
                               (airs_rad_gran.nominal_freq[chan_min[ib]] +
                                airs_rad_gran.nominal_freq[chan_max[ib]]));

          /* Write output... */
          fprintf(out, " %.3f", rad[ib]);
        }

        /* Write output... */
        fprintf(out, "\n");
      }
    }
  }

  /* Close file... */
  fclose(out);

  return EXIT_SUCCESS;
}
```

5.7 cfc.c File Reference

Functions

- double get_noise (double bt, double dt250, double nu)
- int main (int argc, char *argv[ ])

5.7.1 Function Documentation

5.7.1.1 double get_noise (double *bt,* double *dt250,* double *nu*)

Definition at line 177 of file cfc.c.

```
             {

  double nesr;

  nesr = planck(250.0 + dt250, nu) - planck(250.0, nu);

  return brightness(planck(bt, nu) + nesr, nu) - bt;
}
```

Here is the call graph for this function:

5.7.1.2 int main (int *argc,* char * *argv[ ]*)

Definition at line 17 of file cfc.c.

```
              {

  FILE *out;

  static airs_rad_gran_t airs_rad_gran;

  static double ci, ci_err, ci_nedt = 0.35, cimax,
    f11_low, f11_low_err, f11_low_bt1, f11_low_bt1_nedt =
    0.35, f11_low_bt2, f11_low_bt2_nedt =
    0.32, f11_high, f11_high_err, f11_high_bt1, f11_high_bt1_nedt =
    0.34, f11_high_bt2, f11_high_bt2_nedt = 0.32;

  static int ichan, track, xtrack, iarg, f11_low_nu1 = 558, f11_low_nu2 =
    596, f11_high_nu1 = 624, f11_high_nu2 = 596, ci_nu = 558;

  /* Check arguments... */
  if (argc < 3)
    ERRMSG("Give parameters: <out.tab> <l1b_file1> [<l1b_file2> ...]");

  /* Create file... */
  printf("Write CFC-11 emission data: %s\n", argv[1]);
  if (!(out = fopen(argv[1], "w")))
    ERRMSG("Cannot create file!");

  /* Loop over HDF files... */
  for (iarg = 2; iarg < argc; iarg++) {

    /* Read AIRS data... */
    printf("Read AIRS Level-1B data file: %s\n", argv[iarg]);
    airs_rad_rdr(argv[iarg], &airs_rad_gran);

    /* Write header... */
    if (iarg == 2) {
      fprintf(out,
              "# $1  = time [s]\n"
              "# $2  = footprint longitude [deg]\n"
              "# $3  = footprint latitude [deg]\n"
              "# $4  = satellite altitude [km]\n"
              "# $5  = satellite longitude [deg]\n"
              "# $6  = satellite latitude [deg]\n");
      fprintf(out,
              "# $7  = cloud index, BT(%.2f/cm) [K]\n"
              "# $8  = cloud index error [K]\n"
              "# $9  = CFC-11 index (low wavenumbers),"
              " BT(%.2f/cm) - BT(%.2f/cm) [K]\n"
              "# $10 = CFC-11 index (low wavenumbers) error [K]\n"
              "# $11 = CFC-11 index (high wavenumbers),"
              " BT(%.2f/cm) - BT(%.2f/cm) [K]\n"
              "# $12 = CFC-11 index (high wavenumbers) error [K]\n",
              airs_rad_gran.nominal_freq[ci_nu],
              airs_rad_gran.nominal_freq[f11_low_nu1],
              airs_rad_gran.nominal_freq[f11_low_nu2],
              airs_rad_gran.nominal_freq[f11_high_nu1],
              airs_rad_gran.nominal_freq[f11_high_nu2]);
    }

    /* Flag bad observations... */
    for (track = 0; track < AIRS_RAD_GEOTRACK; track++)
      for (xtrack = 0; xtrack < AIRS_RAD_GEOXTRACK; xtrack++)
        for (ichan = 0; ichan < AIRS_RAD_CHANNEL; ichan++)
          if ((airs_rad_gran.state[track][xtrack] != 0)
              || (airs_rad_gran.ExcludedChans[ichan] > 2)
              || (airs_rad_gran.CalChanSummary[ichan] & 8)
              || (airs_rad_gran.CalChanSummary[ichan] & (32 + 64))
              || (airs_rad_gran.CalFlag[track][ichan] & 16))
            airs_rad_gran.radiances[track][xtrack][ichan] = GSL_NAN;

    /* Get maximum cloud index... */
    cimax = -999;
    for (track = 0; track < AIRS_RAD_GEOTRACK; track++)
      for (xtrack = 0; xtrack < AIRS_RAD_GEOXTRACK; xtrack++) {
        ci = brightness(airs_rad_gran.radiances[track][xtrack][ci_nu] * 0.001,
                        airs_rad_gran.nominal_freq[ci_nu]);
        if (ci > cimax)
          cimax = ci;
      }

    /* Loop over scans... */
    for (track = 0; track < AIRS_RAD_GEOTRACK; track++) {

      /* Write output... */
      fprintf(out, "\n");

      /* Loop over footprints... */
      for (xtrack = 0; xtrack < AIRS_RAD_GEOXTRACK; xtrack++) {

        /* Skip daytime measurements... */
```

```
        if (sza(airs_rad_gran.Time[track][xtrack] - 220838400,
                airs_rad_gran.Longitude[track][xtrack],
                airs_rad_gran.Latitude[track][xtrack]) < 96.0)
          continue;

        /* cloud index... */
        ci = brightness(airs_rad_gran.radiances[track][xtrack][ci_nu] * 0.001,
                        airs_rad_gran.nominal_freq[ci_nu]);
        ci_err = get_noise(ci, ci_nedt, airs_rad_gran.nominal_freq[ci_nu]);

        /* Check cloud index... */
        if (ci < 0.95 * cimax || ci <= 270.)
          continue;

        /* CFC-11 index (low wavenumbers)... */
        f11_low_bt1 =
          brightness(airs_rad_gran.radiances[track][xtrack][f11_low_nu1] *
                     0.001, airs_rad_gran.nominal_freq[f11_low_nu1]);
        f11_low_bt2 =
          brightness(airs_rad_gran.radiances[track][xtrack][f11_low_nu2] *
                     0.001, airs_rad_gran.nominal_freq[f11_low_nu2]);
        f11_low = f11_low_bt1 - f11_low_bt2;
        f11_low_err = sqrt(gsl_pow_2(get_noise(f11_low_bt1, f11_low_bt1_nedt,
                                               airs_rad_gran.nominal_freq
                                               [f11_low_nu1]))
                           +
                           gsl_pow_2(get_noise
                                     (f11_low_bt2, f11_low_bt2_nedt,
                                      airs_rad_gran.nominal_freq
                                      [f11_low_nu2])));

        /* CFC-11 index (high wavenumbers)... */
        f11_high_bt1 =
          brightness(airs_rad_gran.radiances[track][xtrack][f11_high_nu1] *
                     0.001, airs_rad_gran.nominal_freq[f11_high_nu1]);
        f11_high_bt2 =
          brightness(airs_rad_gran.radiances[track][xtrack][f11_high_nu2] *
                     0.001, airs_rad_gran.nominal_freq[f11_high_nu2]);
        f11_high = f11_high_bt1 - f11_high_bt2;
        f11_high_err =
          sqrt(gsl_pow_2
               (get_noise
                (f11_high_bt1, f11_high_bt1_nedt,
                 airs_rad_gran.nominal_freq[f11_high_nu1]))
               +
               gsl_pow_2(get_noise
                         (f11_high_bt2, f11_high_bt2_nedt,
                          airs_rad_gran.nominal_freq[f11_high_nu2])));

        /* Write output... */
        fprintf(out,
                "%.2f %.4f %.4f %.3f %.4f %.4f %.2f %.2f %.2f %.2f %.2f %.2f\n",
                airs_rad_gran.Time[track][xtrack] - 220838400,
                airs_rad_gran.Longitude[track][xtrack],
                airs_rad_gran.Latitude[track][xtrack],
                airs_rad_gran.satheight[track],
                airs_rad_gran.sat_lon[track],
                airs_rad_gran.sat_lat[track],
                ci, ci_err, f11_low, f11_low_err, f11_high, f11_high_err);
      }
    }
  }

  /* Close file... */
  fclose(out);

  return EXIT_SUCCESS;
}
```

Here is the call graph for this function:

5.8 cfc.c

```
#include "libairs.h"

/* ------------------------------------------------------------
   Functions...
   ------------------------------------------------------------ */

/* Estimate noise. */
double get_noise(
```

```
  double bt,
  double dt250,
  double nu);

/* ------------------------------------------------------------
   Main...
   ------------------------------------------------------------ */

int main(
  int argc,
  char *argv[]) {

  FILE *out;

  static airs_rad_gran_t airs_rad_gran;

  static double ci, ci_err, ci_nedt = 0.35, cimax,
    f11_low, f11_low_err, f11_low_bt1, f11_low_bt1_nedt =
    0.35, f11_low_bt2, f11_low_bt2_nedt =
    0.32, f11_high, f11_high_err, f11_high_bt1, f11_high_bt1_nedt =
    0.34, f11_high_bt2, f11_high_bt2_nedt = 0.32;

  static int ichan, track, xtrack, iarg, f11_low_nu1 = 558, f11_low_nu2 =
    596, f11_high_nu1 = 624, f11_high_nu2 = 596, ci_nu = 558;

  /* Check arguments... */
  if (argc < 3)
    ERRMSG("Give parameters: <out.tab> <l1b_file1> [<l1b_file2> ...]");

  /* Create file... */
  printf("Write CFC-11 emission data: %s\n", argv[1]);
  if (!(out = fopen(argv[1], "w")))
    ERRMSG("Cannot create file!");

  /* Loop over HDF files... */
  for (iarg = 2; iarg < argc; iarg++) {

    /* Read AIRS data... */
    printf("Read AIRS Level-1B data file: %s\n", argv[iarg]);
    airs_rad_rdr(argv[iarg], &airs_rad_gran);

    /* Write header... */
    if (iarg == 2) {
      fprintf(out,
              "# $1  = time [s]\n"
              "# $2  = footprint longitude [deg]\n"
              "# $3  = footprint latitude [deg]\n"
              "# $4  = satellite altitude [km]\n"
              "# $5  = satellite longitude [deg]\n"
              "# $6  = satellite latitude [deg]\n");
      fprintf(out,
              "# $7  = cloud index, BT(%.2f/cm) [K]\n"
              "# $8  = cloud index error [K]\n"
              "# $9  = CFC-11 index (low wavenumbers),"
              " BT(%.2f/cm) - BT(%.2f/cm) [K]\n"
              "# $10 = CFC-11 index (low wavenumbers) error [K]\n"
              "# $11 = CFC-11 index (high wavenumbers),"
              " BT(%.2f/cm) - BT(%.2f/cm) [K]\n"
              "# $12 = CFC-11 index (high wavenumbers) error [K]\n",
              airs_rad_gran.nominal_freq[ci_nu],
              airs_rad_gran.nominal_freq[f11_low_nu1],
              airs_rad_gran.nominal_freq[f11_low_nu2],
              airs_rad_gran.nominal_freq[f11_high_nu1],
              airs_rad_gran.nominal_freq[f11_high_nu2]);
    }

    /* Flag bad observations... */
    for (track = 0; track < AIRS_RAD_GEOTRACK; track++)
      for (xtrack = 0; xtrack < AIRS_RAD_GEOXTRACK; xtrack++)
        for (ichan = 0; ichan < AIRS_RAD_CHANNEL; ichan++)
          if ((airs_rad_gran.state[track][xtrack] != 0)
              || (airs_rad_gran.ExcludedChans[ichan] > 2)
              || (airs_rad_gran.CalChanSummary[ichan] & 8)
              || (airs_rad_gran.CalChanSummary[ichan] & (32 + 64))
              || (airs_rad_gran.CalFlag[track][ichan] & 16))
            airs_rad_gran.radiances[track][xtrack][ichan] = GSL_NAN;

    /* Get maximum cloud index... */
    cimax = -999;
    for (track = 0; track < AIRS_RAD_GEOTRACK; track++)
      for (xtrack = 0; xtrack < AIRS_RAD_GEOXTRACK; xtrack++) {
        ci = brightness(airs_rad_gran.radiances[track][xtrack][ci_nu] * 0.001,
                        airs_rad_gran.nominal_freq[ci_nu]);
        if (ci > cimax)
          cimax = ci;
      }

```

```
    /* Loop over scans... */
    for (track = 0; track < AIRS_RAD_GEOTRACK; track++) {

      /* Write output... */
      fprintf(out, "\n");

      /* Loop over footprints... */
      for (xtrack = 0; xtrack < AIRS_RAD_GEOXTRACK; xtrack++) {

        /* Skip daytime measurements... */
        if (sza(airs_rad_gran.Time[track][xtrack] - 220838400,
                airs_rad_gran.Longitude[track][xtrack],
                airs_rad_gran.Latitude[track][xtrack]) < 96.0)
          continue;

        /* cloud index... */
        ci = brightness(airs_rad_gran.radiances[track][xtrack][ci_nu] * 0.001,
                        airs_rad_gran.nominal_freq[ci_nu]);
        ci_err = get_noise(ci, ci_nedt, airs_rad_gran.nominal_freq[ci_nu]);

        /* Check cloud index... */
        if (ci < 0.95 * cimax || ci <= 270.)
          continue;

        /* CFC-11 index (low wavenumbers)... */
        f11_low_bt1 =
          brightness(airs_rad_gran.radiances[track][xtrack][f11_low_nu1] *
                     0.001, airs_rad_gran.nominal_freq[f11_low_nu1]);
        f11_low_bt2 =
          brightness(airs_rad_gran.radiances[track][xtrack][f11_low_nu2] *
                     0.001, airs_rad_gran.nominal_freq[f11_low_nu2]);
        f11_low = f11_low_bt1 - f11_low_bt2;
        f11_low_err = sqrt(gsl_pow_2(get_noise(f11_low_bt1, f11_low_bt1_nedt,
                                               airs_rad_gran.nominal_freq
                                               [f11_low_nu1]))
                           +
                           gsl_pow_2(get_noise
                                     (f11_low_bt2, f11_low_bt2_nedt,
                                      airs_rad_gran.nominal_freq
                                      [f11_low_nu2])));

        /* CFC-11 index (high wavenumbers)... */
        f11_high_bt1 =
          brightness(airs_rad_gran.radiances[track][xtrack][f11_high_nu1] *
                     0.001, airs_rad_gran.nominal_freq[f11_high_nu1]);
        f11_high_bt2 =
          brightness(airs_rad_gran.radiances[track][xtrack][f11_high_nu2] *
                     0.001, airs_rad_gran.nominal_freq[f11_high_nu2]);
        f11_high = f11_high_bt1 - f11_high_bt2;
        f11_high_err =
          sqrt(gsl_pow_2
               (get_noise
                (f11_high_bt1, f11_high_bt1_nedt,
                 airs_rad_gran.nominal_freq[f11_high_nu1]))
               +
               gsl_pow_2(get_noise
                         (f11_high_bt2, f11_high_bt2_nedt,
                          airs_rad_gran.nominal_freq[f11_high_nu2])));

        /* Write output... */
        fprintf(out,
                "%.2f %.4f %.4f %.3f %.4f %.4f %.2f %.2f %.2f %.2f %.2f %.2f\n",
                airs_rad_gran.Time[track][xtrack] - 220838400,
                airs_rad_gran.Longitude[track][xtrack],
                airs_rad_gran.Latitude[track][xtrack],
                airs_rad_gran.satheight[track],
                airs_rad_gran.sat_lon[track],
                airs_rad_gran.sat_lat[track],
                ci, ci_err, f11_low, f11_low_err, f11_high, f11_high_err);
      }
    }
  }

  /* Close file... */
  fclose(out);

  return EXIT_SUCCESS;
}

/*****************************************************************************/

double get_noise(
  double bt,
  double dt250,
  double nu) {

  double nesr;
```

```
  nesr = planck(250.0 + dt250, nu) - planck(250.0, nu);

  return brightness(planck(bt, nu) + nesr, nu) - bt;
}
```

5.9 collect.c File Reference

Functions

- void addatt (int ncid, int varid, const char ∗unit, const char ∗long_name)
- void addvar (int ncid, const char ∗var_name, const char ∗unit, const char ∗longname, int type, int dimid[ ], int full, int ∗varid, int ndims)
- void analyze (ctl_t ∗ctl, const char ∗dirlistdat, size_t ∗ndir, size_t ∗max_np, int ∗press_full, int ∗temp_full, int ∗q_full, int ∗k_full)
- void collect_read_atm (char ∗dirname, char ∗filename, ctl_t ∗ctl, atm_t ∗atm_dest, atm_t ∗atm_src)
- void writevar (int ncid, int varid[ ], int full, double ∗atm, double ∗tot, double ∗noi, double ∗fm, double ∗cont, double ∗res, double ∗apr, size_t ind_start[ ], size_t ind_end[ ])
- double read_chisq (const char ∗dirname)
- int main (int argc, char ∗argv[ ])

5.9.1 Function Documentation

5.9.1.1 void addatt (int *ncid,* int *varid,* const char ∗ *unit,* const char ∗ *long_name*)

Definition at line 239 of file collect.c.

```
                            {

  /* Set long name... */
  NC(nc_put_att_text(ncid, varid, "long_name", strlen(long_name), long_name));

  /* Set units... */
  NC(nc_put_att_text(ncid, varid, "units", strlen(unit), unit));
}
```

5.9.1.2 void addvar (int *ncid,* const char ∗ *var_name,* const char ∗ *unit,* const char ∗ *longname,* int *type,* int *dimid[ ],* int *full,* int ∗ *varid,* int *ndims*)

Definition at line 254 of file collect.c.

```
              {

  char lname[LEN], var[LEN];

  /* Retrieval results... */
  NC(nc_def_var(ncid, var_name, type, ndims, dimid, &varid[0]));
  sprintf(lname, "%s", longname);
  addatt(ncid, varid[0], unit, lname);

  /* Write full output? */
  if (full) {

    /* Total error... */
    sprintf(var, "%s_total", var_name);
    NC(nc_def_var(ncid, var, NC_FLOAT, 2, dimid, &varid[1]));
    sprintf(lname, "%s total error", longname);
    addatt(ncid, varid[1], unit, lname);

    /* Noise error... */
    sprintf(var, "%s_noise", var_name);
    NC(nc_def_var(ncid, var, NC_FLOAT, 2, dimid, &varid[2]));
```

```
    sprintf(lname, "%s noise error", longname);
    addatt(ncid, varid[2], unit, lname);

    /* Forward model error... */
    sprintf(var, "%s_formod", var_name);
    NC(nc_def_var(ncid, var, NC_FLOAT, 2, dimid, &varid[3]));
    sprintf(lname, "%s forward model error", longname);
    addatt(ncid, varid[3], unit, lname);

    /* Measurement content... */
    sprintf(var, "%s_cont", var_name);
    NC(nc_def_var(ncid, var, NC_FLOAT, 2, dimid, &varid[4]));
    sprintf(lname, "%s measurement content", longname);
    addatt(ncid, varid[4], "1", lname);

    /* Resolution... */
    sprintf(var, "%s_res", var_name);
    NC(nc_def_var(ncid, var, NC_FLOAT, 2, dimid, &varid[5]));
    sprintf(lname, "%s resolution", longname);
    addatt(ncid, varid[5], "1", lname);

    /* A priori data... */
    sprintf(var, "%s_apr", var_name);
    NC(nc_def_var(ncid, var, NC_FLOAT, 2, dimid, &varid[6]));
    sprintf(lname, "%s a priori", longname);
    addatt(ncid, varid[6], unit, lname);
  }
}
```

Here is the call graph for this function:

5.9.1.3 void analyze (ctl_t * *ctl*, const char * *dirlistdat*, size_t * *ndir*, size_t * *max_np*, int * *press_full*, int * *temp_full*, int * *q_full*, int * *k_full*)

Definition at line 315 of file collect.c.

```
                {

  static atm_t atm;

  FILE *dirlist;

  char dir[LEN];

  int ig, ip, iw;

  /* Initialize... */
  *ndir = *max_np = 0;
  *press_full = *temp_full = 0;
  for (ig = 0; ig < ctl->ng; ig++)
    q_full[ig] = 0;
  for (iw = 0; iw < ctl->nw; iw++)
    k_full[iw] = 0;

  /* Open directory list... */
  if (!(dirlist = fopen(dirlistdat, "r")))
    ERRMSG("Cannot open directory list!");

  /* Loop over directories... */
  while (fscanf(dirlist, "%s", dir) != EOF) {

    /* Increment profile counter... */
    ++(*ndir);

    /* Read atmospheric data... */
    read_atm(dir, "atm_apr.tab", ctl, &atm);

    /* Analyze atmospheric data... */
    *max_np = GSL_MAX((size_t) atm.np, *max_np);
    for (ip = 0; ip < atm.np; ip++) {
      *press_full = *press_full ||
        (atm.z[ip] >= ctl->retp_zmin && atm.z[ip] <= ctl->retp_zmax);
      *temp_full = *temp_full ||
        (atm.z[ip] >= ctl->rett_zmin && atm.z[ip] <= ctl->rett_zmax);
      for (ig = 0; ig < ctl->ng; ig++)
        q_full[ig] = q_full[ig] || (atm.z[ip] >= ctl->retq_zmin[ig]
                                    && atm.z[ip] <= ctl->retq_zmax[ig]);
      for (iw = 0; iw < ctl->nw; iw++)
        k_full[iw] = k_full[iw] || (atm.z[ip] >= ctl->retk_zmin[iw]
                                    && atm.z[ip] <= ctl->retk_zmax[iw]);
```

```
    }
  }

  /* Close directory list... */
  fclose(dirlist);
}
```

Here is the call graph for this function:

5.9.1.4 void collect_read_atm (char * *dirname,* char * *filename,* ctl_t * *ctl,* atm_t * *atm_dest,* atm_t * *atm_src*)

Definition at line 376 of file collect.c.

```
                  {

  FILE *in;

  char file[LEN];

  /* Set filename... */
  sprintf(file, "%s/%s", dirname, filename);

  /* Try to read file... */
  if (!(in = fopen(file, "r")))
    copy_atm(ctl, atm_dest, atm_src, 1);
  else {
    read_atm(dirname, filename, ctl, atm_dest);
    fclose(in);
  }
}
```

Here is the call graph for this function:

5.9.1.5 void writevar (int *ncid,* int *varid[ ],* int *full,* double * *atm,* double * *tot,* double * *noi,* double * *fm,* double * *cont,* double * *res,* double * *apr,* size_t *ind_start[ ],* size_t *ind_end[ ]*)

Definition at line 401 of file collect.c.

```
                   {

  /* Retrieval results... */
  NC(nc_put_vara_double(ncid, varid[0], ind_start, ind_end, atm));

  /* Write full output? */
  if (full) {

    /* Total error... */
    NC(nc_put_vara_double(ncid, varid[1], ind_start, ind_end, tot));

    /* Noise error... */
    NC(nc_put_vara_double(ncid, varid[2], ind_start, ind_end, noi));

    /* Forward model error... */
    NC(nc_put_vara_double(ncid, varid[3], ind_start, ind_end, fm));

    /* Measurement content... */
    NC(nc_put_vara_double(ncid, varid[4], ind_start, ind_end, cont));

    /* Resolution... */
    NC(nc_put_vara_double(ncid, varid[5], ind_start, ind_end, res));

    /* A priori... */
    NC(nc_put_vara_double(ncid, varid[6], ind_start, ind_end, apr));
  }
}
```

5.9.1.6 double read_chisq (const char * *dirname*)

Definition at line 443 of file collect.c.

```
                         {

  FILE *in;

  char file[LEN], line[LEN], *tok;

  double chisq, dummy;

  /* Set filename... */
  sprintf(file, "%s/costs.tab", dirname);

  /* Write info... */
  printf("Read cost function data: %s\n", file);

  /* Open file... */
  if (!(in = fopen(file, "r")))
    return GSL_NAN;

  /* Read data... */
  while (fgets(line, LEN, in)) {
    TOK(line, tok, "%lg", dummy);
    TOK(NULL, tok, "%lg", chisq);
  }

  /* Close file... */
  fclose(in);

  return chisq;
}
```

5.9.1.7 int main (int *argc*, char * *argv[ ]*)

Definition at line 78 of file collect.c.

```
                  {

  static atm_t atm, atm_noise, atm_fm, atm_tot, atm_cont, atm_res, atm_apr;
  static ctl_t ctl;

  FILE *dirlist;

  double chisq;

  char dir[LEN], longname[LEN], varname[LEN];

  int chisq_id[NVAR], cnt = -1, dimid[2], dimid_chisq[1], ig, iw,
    k_full[ND] = { 0 }, k_id[ND][NVAR], lat_id[NVAR], lon_id[NVAR],
    na[NDSMAX], ncid, nds_id, np_id, np_dimid, nds_dimid,
    press_full = 0, press_id[NVAR], q_full[NG] = {
  0}, q_id[NG][NVAR], temp_full = 0, temp_id[NVAR], time_id[NVAR], z_id[NVAR];

  size_t i, ind_start[] = { 0, 0 }, ind_end[] = {
  0, 0}, ind_start_chisq[] = {
  0}, ind_end_chisq[] = {
  0}, max_np, ndir = 0;

  /* Check arguments... */
  if (argc < 4)
    ERRMSG("Give parameters: <ctl> <dirlist> <out.nc>");

  /* Read control parameters... */
  read_ctl(argc, argv, &ctl);

  /* Get maximum size of variable arrays... */
  analyze(&ctl, argv[2], &ndir, &max_np,
          &press_full, &temp_full, q_full, k_full);

  /* Check number of data sets... */
  if (ndir >= NDSMAX)
    ERRMSG("Too many data sets!");

  /* Create netCDF file... */
  NC(nc_create(argv[3], NC_CLOBBER, &ncid));

```

```
  /* Set dimensions... */
  NC(nc_def_dim(ncid, "np", max_np, &np_dimid));
  NC(nc_def_dim(ncid, "nds", ndir, &nds_dimid));

  /* Set dimids... */
  dimid[0] = np_dimid;
  dimid[1] = nds_dimid;
  dimid_chisq[0] = nds_dimid;

  /* Coordinate variables... */
  addvar(ncid, "nds", "1", "data set index",
         NC_INT, dimid_chisq, 0, &nds_id, 1);
  addvar(ncid, "np", "1", "data point index", NC_INT, dimid, 0, &np_id, 1);

  /* Atmospheric data... */
  addvar(ncid, "time", "s", "time (seconds since 2000-01-01T00:00Z)",
         NC_DOUBLE, dimid, 0, time_id, 2);
  addvar(ncid, "z", "km", "altitude", NC_DOUBLE, dimid, 0, z_id, 2);
  addvar(ncid, "lon", "deg", "longitude", NC_DOUBLE, dimid, 0, lon_id, 2);
  addvar(ncid, "lat", "deg", "latitude", NC_DOUBLE, dimid, 0, lat_id, 2);
  addvar(ncid, "press", "hPa", "pressure",
         NC_FLOAT, dimid, press_full, press_id, 2);
  addvar(ncid, "temp", "K", "temperature",
         NC_FLOAT, dimid, temp_full, temp_id, 2);
  for (ig = 0; ig < ctl.ng; ig++) {
    sprintf(longname, "%s %s", ctl.emitter[ig], "volume mixing ratio");
    addvar(ncid, ctl.emitter[ig], "1", longname,
           NC_FLOAT, dimid, q_full[ig], q_id[ig], 2);
  }
  for (iw = 0; iw < ctl.nw; iw++) {
    sprintf(varname, "extinct_win%d", iw);
    sprintf(longname, "extinction (window %d)", iw);
    addvar(ncid, varname, "km^-1", longname,
           NC_FLOAT, dimid, k_full[iw], k_id[iw], 2);
  }

  /* Cost function data... */
  addvar(ncid, "chisq", "1", "normalized cost function",
         NC_FLOAT, dimid_chisq, 0, chisq_id, 1);

  /* Leave define mode... */
  NC(nc_enddef(ncid));

  /* Write coordinate variables... */
  ind_start[0] = 0;
  ind_end[0] = ndir;
  for (i = 0; i < ind_end[0]; i++)
    na[i] = (int) i + 1;
  NC(nc_put_vara_int(ncid, nds_id, ind_start, ind_end, na));

  ind_start[0] = 0;
  ind_end[0] = max_np;
  for (i = 0; i < ind_end[0]; i++)
    na[i] = (int) i + 1;
  NC(nc_put_vara_int(ncid, np_id, ind_start, ind_end, na));

  /* Open directory list... */
  if (!(dirlist = fopen(argv[2], "r")))
    ERRMSG("Cannot open directory list!");

  /* Loop over directories... */
  while (fscanf(dirlist, "%s", dir) != EOF) {

    /* Increment counter... */
    cnt++;

    /* Read atmospheric data... */
    read_atm(dir, "atm_apr.tab", &ctl, &atm_apr);
    collect_read_atm(dir, "atm_final.tab", &ctl, &atm, &atm_apr);
    collect_read_atm(dir, "atm_err_total.tab", &ctl, &atm_tot, &atm_apr);
    collect_read_atm(dir, "atm_err_noise.tab", &ctl, &atm_noise, &atm_apr);
    collect_read_atm(dir, "atm_err_formod.tab", &ctl, &atm_fm, &atm_apr);
    collect_read_atm(dir, "atm_cont.tab", &ctl, &atm_cont, &atm_apr);
    collect_read_atm(dir, "atm_res.tab", &ctl, &atm_res, &atm_apr);
    chisq = read_chisq(dir);

    /* Set indices... */
    ind_start[0] = 0;
    ind_start[1] = (size_t) cnt;
    ind_end[0] = (size_t) atm.np;
    ind_end[1] = 1;
    ind_start_chisq[0] = (size_t) cnt;
    ind_end_chisq[0] = 1;

    /* Write atmospheric data... */
    writevar(ncid, time_id, 0, atm.time, NULL, NULL, NULL, NULL, NULL, NULL,
             ind_start, ind_end);
```

```
    writevar(ncid, z_id, 0, atm.z, NULL, NULL, NULL, NULL, NULL, NULL,
             ind_start, ind_end);
    writevar(ncid, lon_id, 0, atm.lon, NULL, NULL, NULL, NULL, NULL, NULL,
             ind_start, ind_end);
    writevar(ncid, lat_id, 0, atm.lat, NULL, NULL, NULL, NULL, NULL, NULL,
             ind_start, ind_end);
    writevar(ncid, press_id, press_full, atm.p, atm_tot.p, atm_noise.p,
             atm_fm.p, atm_cont.p, atm_res.p, atm_apr.p, ind_start, ind_end);
    writevar(ncid, temp_id, temp_full, atm.t, atm_tot.t, atm_noise.t,
             atm_fm.t, atm_cont.t, atm_res.t, atm_apr.t, ind_start, ind_end);
    for (ig = 0; ig < ctl.ng; ig++)
      writevar(ncid, q_id[ig], q_full[ig], atm.q[ig], atm_tot.q[ig],
               atm_noise.q[ig], atm_fm.q[ig], atm_cont.q[ig],
               atm_res.q[ig], atm_apr.q[ig], ind_start, ind_end);
    for (iw = 0; iw < ctl.nw; iw++)
      writevar(ncid, k_id[iw], k_full[iw], atm.k[iw], atm_tot.k[iw],
               atm_noise.k[iw], atm_fm.k[iw], atm_cont.k[iw], atm_res.k[iw],
               atm_apr.k[iw], ind_start, ind_end);

    /* Write cost function data... */
    writevar(ncid, chisq_id, 0, &chisq, NULL, NULL, NULL, NULL, NULL, NULL,
             ind_start_chisq, ind_end_chisq);
  }

  /* Close file... */
  NC(nc_close(ncid));

  return EXIT_SUCCESS;
}
```

Here is the call graph for this function:

5.10 collect.c

```
#include "libairs.h"

/* ------------------------------------------------------------
   Constants...
   ------------------------------------------------------------ */

/* Number of file types for each variable. */
#define NVAR 7

/* Maximum number of data sets. */
#define NDSMAX 100000

/* ------------------------------------------------------------
   Functions...
   ------------------------------------------------------------ */

/* Add variable attributes to netCDF file. */
void addatt(
  int ncid,
  int varid,
  const char *unit,
  const char *long_name);

/* Create variable in netCDF file. */
void addvar(
  int ncid,
  const char *var_name,
  const char *unit,
  const char *longname,
  int type,
  int dimid[],
  int full,
  int *varid,
  int ndims);

/* Gather retrieval results. */
void analyze(
  ctl_t * ctl,
  const char *dirlistdat,
  size_t * ndir,
  size_t * max_np,
  int *press_full,
  int *temp_full,
  int *q_full,
  int *k_full);

/* Read atmospheric data. */
```

```
void collect_read_atm(
  char *dirname,
  char *filename,
  ctl_t * ctl,
  atm_t * atm_dest,
  atm_t * atm_src);

/* Write variable data to netcdf file. */
void writevar(
  int ncid,
  int varid[],
  int full,
  double *atm,
  double *tot,
  double *noi,
  double *fm,
  double *cont,
  double *res,
  double *apr,
  size_t ind_start[],
  size_t ind_end[]);

/* Read cost function data. */
double read_chisq(
  const char *dirname);

/* ------------------------------------------------------------
   Main...
   ------------------------------------------------------------ */

int main(
  int argc,
  char *argv[]) {

  static atm_t atm, atm_noise, atm_fm, atm_tot, atm_cont, atm_res, atm_apr;
  static ctl_t ctl;

  FILE *dirlist;

  double chisq;

  char dir[LEN], longname[LEN], varname[LEN];

  int chisq_id[NVAR], cnt = -1, dimid[2], dimid_chisq[1], ig, iw,
    k_full[ND] = { 0 }, k_id[ND][NVAR], lat_id[NVAR], lon_id[NVAR],
    na[NDSMAX], ncid, nds_id, np_id, np_dimid, nds_dimid,
    press_full = 0, press_id[NVAR], q_full[NG] = {
  0}, q_id[NG][NVAR], temp_full = 0, temp_id[NVAR], time_id[NVAR], z_id[NVAR];

  size_t i, ind_start[] = { 0, 0 }, ind_end[] = {
  0, 0}, ind_start_chisq[] = {
  0}, ind_end_chisq[] = {
  0}, max_np, ndir = 0;

  /* Check arguments... */
  if (argc < 4)
    ERRMSG("Give parameters: <ctl> <dirlist> <out.nc>");

  /* Read control parameters... */
  read_ctl(argc, argv, &ctl);

  /* Get maximum size of variable arrays... */
  analyze(&ctl, argv[2], &ndir, &max_np,
          &press_full, &temp_full, q_full, k_full);

  /* Check number of data sets... */
  if (ndir >= NDSMAX)
    ERRMSG("Too many data sets!");

  /* Create netCDF file... */
  NC(nc_create(argv[3], NC_CLOBBER, &ncid));

  /* Set dimensions... */
  NC(nc_def_dim(ncid, "np", max_np, &np_dimid));
  NC(nc_def_dim(ncid, "nds", ndir, &nds_dimid));

  /* Set dimids... */
  dimid[0] = np_dimid;
  dimid[1] = nds_dimid;
  dimid_chisq[0] = nds_dimid;

  /* Coordinate variables... */
  addvar(ncid, "nds", "1", "data set index",
         NC_INT, dimid_chisq, 0, &nds_id, 1);
  addvar(ncid, "np", "1", "data point index", NC_INT, dimid, 0, &np_id, 1);

  /* Atmospheric data... */
```

```
    addvar(ncid, "time", "s", "time (seconds since 2000-01-01T00:00Z)",
           NC_DOUBLE, dimid, 0, time_id, 2);
    addvar(ncid, "z", "km", "altitude", NC_DOUBLE, dimid, 0, z_id, 2);
    addvar(ncid, "lon", "deg", "longitude", NC_DOUBLE, dimid, 0, lon_id, 2);
    addvar(ncid, "lat", "deg", "latitude", NC_DOUBLE, dimid, 0, lat_id, 2);
    addvar(ncid, "press", "hPa", "pressure",
           NC_FLOAT, dimid, press_full, press_id, 2);
    addvar(ncid, "temp", "K", "temperature",
           NC_FLOAT, dimid, temp_full, temp_id, 2);
    for (ig = 0; ig < ctl.ng; ig++) {
      sprintf(longname, "%s %s", ctl.emitter[ig], "volume mixing ratio");
      addvar(ncid, ctl.emitter[ig], "1", longname,
             NC_FLOAT, dimid, q_full[ig], q_id[ig], 2);
    }
    for (iw = 0; iw < ctl.nw; iw++) {
      sprintf(varname, "extinct_win%d", iw);
      sprintf(longname, "extinction (window %d)", iw);
      addvar(ncid, varname, "km^-1", longname,
             NC_FLOAT, dimid, k_full[iw], k_id[iw], 2);
    }

    /* Cost function data... */
    addvar(ncid, "chisq", "1", "normalized cost function",
           NC_FLOAT, dimid_chisq, 0, chisq_id, 1);

    /* Leave define mode... */
    NC(nc_enddef(ncid));

    /* Write coordinate variables... */
    ind_start[0] = 0;
    ind_end[0] = ndir;
    for (i = 0; i < ind_end[0]; i++)
      na[i] = (int) i + 1;
    NC(nc_put_vara_int(ncid, nds_id, ind_start, ind_end, na));

    ind_start[0] = 0;
    ind_end[0] = max_np;
    for (i = 0; i < ind_end[0]; i++)
      na[i] = (int) i + 1;
    NC(nc_put_vara_int(ncid, np_id, ind_start, ind_end, na));

    /* Open directory list... */
    if (!(dirlist = fopen(argv[2], "r")))
      ERRMSG("Cannot open directory list!");

    /* Loop over directories... */
    while (fscanf(dirlist, "%s", dir) != EOF) {

      /* Increment counter... */
      cnt++;

      /* Read atmospheric data... */
      read_atm(dir, "atm_apr.tab", &ctl, &atm_apr);
      collect_read_atm(dir, "atm_final.tab", &ctl, &atm, &atm_apr);
      collect_read_atm(dir, "atm_err_total.tab", &ctl, &atm_tot, &atm_apr);
      collect_read_atm(dir, "atm_err_noise.tab", &ctl, &atm_noise, &atm_apr);
      collect_read_atm(dir, "atm_err_formod.tab", &ctl, &atm_fm, &atm_apr);
      collect_read_atm(dir, "atm_cont.tab", &ctl, &atm_cont, &atm_apr);
      collect_read_atm(dir, "atm_res.tab", &ctl, &atm_res, &atm_apr);
      chisq = read_chisq(dir);

      /* Set indices... */
      ind_start[0] = 0;
      ind_start[1] = (size_t) cnt;
      ind_end[0] = (size_t) atm.np;
      ind_end[1] = 1;
      ind_start_chisq[0] = (size_t) cnt;
      ind_end_chisq[0] = 1;

      /* Write atmospheric data... */
      writevar(ncid, time_id, 0, atm.time, NULL, NULL, NULL, NULL, NULL, NULL,
               ind_start, ind_end);
      writevar(ncid, z_id, 0, atm.z, NULL, NULL, NULL, NULL, NULL, NULL,
               ind_start, ind_end);
      writevar(ncid, lon_id, 0, atm.lon, NULL, NULL, NULL, NULL, NULL, NULL,
               ind_start, ind_end);
      writevar(ncid, lat_id, 0, atm.lat, NULL, NULL, NULL, NULL, NULL, NULL,
               ind_start, ind_end);
      writevar(ncid, press_id, press_full, atm.p, atm_tot.p, atm_noise.p,
               atm_fm.p, atm_cont.p, atm_res.p, atm_apr.p, ind_start, ind_end);
      writevar(ncid, temp_id, temp_full, atm.t, atm_tot.t, atm_noise.t,
               atm_fm.t, atm_cont.t, atm_res.t, atm_apr.t, ind_start, ind_end);
      for (ig = 0; ig < ctl.ng; ig++)
        writevar(ncid, q_id[ig], q_full[ig], atm.q[ig], atm_tot.q[ig],
                 atm_noise.q[ig], atm_fm.q[ig], atm_cont.q[ig],
                 atm_res.q[ig], atm_apr.q[ig], ind_start, ind_end);
      for (iw = 0; iw < ctl.nw; iw++)
```

```
      writevar(ncid, k_id[iw], k_full[iw], atm.k[iw], atm_tot.k[iw],
               atm_noise.k[iw], atm_fm.k[iw], atm_cont.k[iw], atm_res.k[iw],
               atm_apr.k[iw], ind_start, ind_end);

    /* Write cost function data... */
    writevar(ncid, chisq_id, 0, &chisq, NULL, NULL, NULL, NULL, NULL, NULL,
             ind_start_chisq, ind_end_chisq);
  }

  /* Close file... */
  NC(nc_close(ncid));

  return EXIT_SUCCESS;
}

/*****************************************************************************/

void addatt(
  int ncid,
  int varid,
  const char *unit,
  const char *long_name) {

  /* Set long name... */
  NC(nc_put_att_text(ncid, varid, "long_name", strlen(long_name), long_name));

  /* Set units... */
  NC(nc_put_att_text(ncid, varid, "units", strlen(unit), unit));
}

/*****************************************************************************/

void addvar(
  int ncid,
  const char *var_name,
  const char *unit,
  const char *longname,
  int type,
  int dimid[],
  int full,
  int *varid,
  int ndims) {

  char lname[LEN], var[LEN];

  /* Retrieval results... */
  NC(nc_def_var(ncid, var_name, type, ndims, dimid, &varid[0]));
  sprintf(lname, "%s", longname);
  addatt(ncid, varid[0], unit, lname);

  /* Write full output? */
  if (full) {

    /* Total error... */
    sprintf(var, "%s_total", var_name);
    NC(nc_def_var(ncid, var, NC_FLOAT, 2, dimid, &varid[1]));
    sprintf(lname, "%s total error", longname);
    addatt(ncid, varid[1], unit, lname);

    /* Noise error... */
    sprintf(var, "%s_noise", var_name);
    NC(nc_def_var(ncid, var, NC_FLOAT, 2, dimid, &varid[2]));
    sprintf(lname, "%s noise error", longname);
    addatt(ncid, varid[2], unit, lname);

    /* Forward model error... */
    sprintf(var, "%s_formod", var_name);
    NC(nc_def_var(ncid, var, NC_FLOAT, 2, dimid, &varid[3]));
    sprintf(lname, "%s forward model error", longname);
    addatt(ncid, varid[3], unit, lname);

    /* Measurement content... */
    sprintf(var, "%s_cont", var_name);
    NC(nc_def_var(ncid, var, NC_FLOAT, 2, dimid, &varid[4]));
    sprintf(lname, "%s measurement content", longname);
    addatt(ncid, varid[4], "1", lname);

    /* Resolution... */
    sprintf(var, "%s_res", var_name);
    NC(nc_def_var(ncid, var, NC_FLOAT, 2, dimid, &varid[5]));
    sprintf(lname, "%s resolution", longname);
    addatt(ncid, varid[5], "1", lname);

    /* A priori data... */
    sprintf(var, "%s_apr", var_name);
    NC(nc_def_var(ncid, var, NC_FLOAT, 2, dimid, &varid[6]));
    sprintf(lname, "%s a priori", longname);
```

```
    addatt(ncid, varid[6], unit, lname);
  }
}

/*****************************************************************************/

void analyze(
  ctl_t * ctl,
  const char *dirlistdat,
  size_t * ndir,
  size_t * max_np,
  int *press_full,
  int *temp_full,
  int *q_full,
  int *k_full) {

  static atm_t atm;

  FILE *dirlist;

  char dir[LEN];

  int ig, ip, iw;

  /* Initialize... */
  *ndir = *max_np = 0;
  *press_full = *temp_full = 0;
  for (ig = 0; ig < ctl->ng; ig++)
    q_full[ig] = 0;
  for (iw = 0; iw < ctl->nw; iw++)
    k_full[iw] = 0;

  /* Open directory list... */
  if (!(dirlist = fopen(dirlistdat, "r")))
    ERRMSG("Cannot open directory list!");

  /* Loop over directories... */
  while (fscanf(dirlist, "%s", dir) != EOF) {

    /* Increment profile counter... */
    ++(*ndir);

    /* Read atmospheric data... */
    read_atm(dir, "atm_apr.tab", ctl, &atm);

    /* Analyze atmospheric data... */
    *max_np = GSL_MAX((size_t) atm.np, *max_np);
    for (ip = 0; ip < atm.np; ip++) {
      *press_full = *press_full ||
        (atm.z[ip] >= ctl->retp_zmin && atm.z[ip] <= ctl->retp_zmax);
      *temp_full = *temp_full ||
        (atm.z[ip] >= ctl->rett_zmin && atm.z[ip] <= ctl->rett_zmax);
      for (ig = 0; ig < ctl->ng; ig++)
        q_full[ig] = q_full[ig] || (atm.z[ip] >= ctl->retq_zmin[ig]
                                    && atm.z[ip] <= ctl->retq_zmax[ig]);
      for (iw = 0; iw < ctl->nw; iw++)
        k_full[iw] = k_full[iw] || (atm.z[ip] >= ctl->retk_zmin[iw]
                                    && atm.z[ip] <= ctl->retk_zmax[iw]);
    }
  }

  /* Close directory list... */
  fclose(dirlist);
}

/*****************************************************************************/

void collect_read_atm(
  char *dirname,
  char *filename,
  ctl_t * ctl,
  atm_t * atm_dest,
  atm_t * atm_src) {

  FILE *in;

  char file[LEN];

  /* Set filename... */
  sprintf(file, "%s/%s", dirname, filename);

  /* Try to read file... */
  if (!(in = fopen(file, "r")))
    copy_atm(ctl, atm_dest, atm_src, 1);
  else {
    read_atm(dirname, filename, ctl, atm_dest);
    fclose(in);
```

```
  }
}

/*****************************************************************************/

void writevar(
  int ncid,
  int varid[],
  int full,
  double *atm,
  double *tot,
  double *noi,
  double *fm,
  double *cont,
  double *res,
  double *apr,
  size_t ind_start[],
  size_t ind_end[]) {

  /* Retrieval results... */
  NC(nc_put_vara_double(ncid, varid[0], ind_start, ind_end, atm));

  /* Write full output? */
  if (full) {

    /* Total error... */
    NC(nc_put_vara_double(ncid, varid[1], ind_start, ind_end, tot));

    /* Noise error... */
    NC(nc_put_vara_double(ncid, varid[2], ind_start, ind_end, noi));

    /* Forward model error... */
    NC(nc_put_vara_double(ncid, varid[3], ind_start, ind_end, fm));

    /* Measurement content... */
    NC(nc_put_vara_double(ncid, varid[4], ind_start, ind_end, cont));

    /* Resolution... */
    NC(nc_put_vara_double(ncid, varid[5], ind_start, ind_end, res));

    /* A priori... */
    NC(nc_put_vara_double(ncid, varid[6], ind_start, ind_end, apr));
  }
}

/*****************************************************************************/

double read_chisq(
  const char *dirname) {

  FILE *in;

  char file[LEN], line[LEN], *tok;

  double chisq, dummy;

  /* Set filename... */
  sprintf(file, "%s/costs.tab", dirname);

  /* Write info... */
  printf("Read cost function data: %s\n", file);

  /* Open file... */
  if (!(in = fopen(file, "r")))
    return GSL_NAN;

  /* Read data... */
  while (fgets(line, LEN, in)) {
    TOK(line, tok, "%lg", dummy);
    TOK(NULL, tok, "%lg", chisq);
  }

  /* Close file... */
  fclose(in);

  return chisq;
}
```

5.11 day2doy.c File Reference

Functions

- int main (int argc, char *argv[ ])

5.11.1 Function Documentation

5.11.1.1 int main (int *argc,* char ∗ *argv[ ]*)

Definition at line 3 of file day2doy.c.

```
00005                 {
00006 
00007   int day, doy, mon, year;
00008 
00009   /* Check arguments... */
00010   if (argc < 4)
00011     ERRMSG("Give parameters: <year> <mon> <day>");
00012 
00013   /* Read arguments... */
00014   year = atoi(argv[1]);
00015   mon = atoi(argv[2]);
00016   day = atoi(argv[3]);
00017 
00018   /* Convert... */
00019   day2doy(year, mon, day, &doy);
00020   printf("%d %d\n", year, doy);
00021 
00022   return EXIT_SUCCESS;
00023 }
```

Here is the call graph for this function:

5.12 day2doy.c

```
00001 #include "libairs.h"
00002 
00003 int main(
00004   int argc,
00005   char *argv[]) {
00006 
00007   int day, doy, mon, year;
00008 
00009   /* Check arguments... */
00010   if (argc < 4)
00011     ERRMSG("Give parameters: <year> <mon> <day>");
00012 
00013   /* Read arguments... */
00014   year = atoi(argv[1]);
00015   mon = atoi(argv[2]);
00016   day = atoi(argv[3]);
00017 
00018   /* Convert... */
00019   day2doy(year, mon, day, &doy);
00020   printf("%d %d\n", year, doy);
00021 
00022   return EXIT_SUCCESS;
00023 }
```

5.13 distance.c File Reference

Functions

- int main (int argc, char ∗argv[ ])

5.13.1 Function Documentation

5.13.1.1 int main (int *argc*, char ∗ *argv[ ]*)

Definition at line 3 of file distance.c.

```
00005                  {
00006
00007   double lat0, lat1, lon0, lon1, x0[3], x1[3];
00008
00009   /* Check arguments... */
00010   if (argc < 5)
00011     ERRMSG("Give parameters: <lon0> <lat0> <lon1> <lat1>");
00012
00013   /* Read geolocations... */
00014   lon0 = atof(argv[1]);
00015   lat0 = atof(argv[2]);
00016   lon1 = atof(argv[3]);
00017   lat1 = atof(argv[4]);
00018
00019   /* Write distance to stdout... */
00020   geo2cart(0, lon0, lat0, x0);
00021   geo2cart(0, lon1, lat1, x1);
00022   printf("%g\n", DIST(x0, x1));
00023
00024   return EXIT_SUCCESS;
00025 }
```

Here is the call graph for this function:

5.14 distance.c

```
00001 #include "jurassic.h"
00002
00003 int main(
00004   int argc,
00005   char *argv[]) {
00006
00007   double lat0, lat1, lon0, lon1, x0[3], x1[3];
00008
00009   /* Check arguments... */
00010   if (argc < 5)
00011     ERRMSG("Give parameters: <lon0> <lat0> <lon1> <lat1>");
00012
00013   /* Read geolocations... */
00014   lon0 = atof(argv[1]);
00015   lat0 = atof(argv[2]);
00016   lon1 = atof(argv[3]);
00017   lat1 = atof(argv[4]);
00018
00019   /* Write distance to stdout... */
00020   geo2cart(0, lon0, lat0, x0);
00021   geo2cart(0, lon1, lat1, x1);
00022   printf("%g\n", DIST(x0, x1));
00023
00024   return EXIT_SUCCESS;
00025 }
```

5.15 doy2day.c File Reference

Functions

- int main (int argc, char ∗argv[ ])

5.15.1 Function Documentation

5.15.1.1 int main (int *argc*, char ∗ *argv[ ]*)

Definition at line 3 of file doy2day.c.

```
00005                 {
00006
00007   int day, doy, mon, year;
00008
00009   /* Check arguments... */
00010   if (argc < 3)
00011     ERRMSG("Give parameters: <year> <doy>");
00012
00013   /* Read arguments... */
00014   year = atoi(argv[1]);
00015   doy = atoi(argv[2]);
00016
00017   /* Convert... */
00018   doy2day(year, doy, &mon, &day);
00019   printf("%d %d %d\n", year, mon, day);
00020
00021   return EXIT_SUCCESS;
00022 }
```

Here is the call graph for this function:

5.16 doy2day.c

```
00001 #include "libairs.h"
00002
00003 int main(
00004   int argc,
00005   char *argv[]) {
00006
00007   int day, doy, mon, year;
00008
00009   /* Check arguments... */
00010   if (argc < 3)
00011     ERRMSG("Give parameters: <year> <doy>");
00012
00013   /* Read arguments... */
00014   year = atoi(argv[1]);
00015   doy = atoi(argv[2]);
00016
00017   /* Convert... */
00018   doy2day(year, doy, &mon, &day);
00019   printf("%d %d %d\n", year, mon, day);
00020
00021   return EXIT_SUCCESS;
00022 }
```

5.17 erafm.c File Reference

Data Structures

- struct met_t

  Meteorological data.

Functions

- void addatt (int ncid, int varid, const char *unit, const char *long_name)
- void intpol_met_3d (float array[EX][EY][EP], int ip, int ix, int iy, double wp, double wx, double wy, double *var)

  Auxilary function for interpolation of meteorological data.
- void intpol_met_space (met_t *met, double p, double lon, double lat, double *t)

  Spatial interpolation of meteorological data.
- void read_met (char *filename, met_t *met)

  Read meteorological data file.
- void read_met_extrapolate (met_t *met)

  Extrapolate meteorological data at lower boundary.
- void read_met_help (int ncid, char *varname, char *varname2, met_t *met, int np, float dest[EX][EY][EP], float scl)

  Read and convert variable from meteorological data file.
- int main (int argc, char *argv[ ])

5.17.1 Function Documentation

5.17.1.1 void addatt (int *ncid*, int *varid*, const char * *unit*, const char * *long_name*)

Definition at line 289 of file erafm.c.

```
                           {

  /* Set long name... */
  NC(nc_put_att_text(ncid, varid, "long_name", strlen(long_name), long_name));

  /* Set units... */
  NC(nc_put_att_text(ncid, varid, "units", strlen(unit), unit));
}
```

5.17.1.2 void intpol_met_3d (float *array[EX][EY][EP]*, int *ip*, int *ix*, int *iy*, double *wp*, double *wx*, double *wy*, double * *var*)

Auxilary function for interpolation of meteorological data.

Definition at line 304 of file erafm.c.

```
                 {

  double aux00, aux01, aux10, aux11;

  /* Interpolate vertically... */
  aux00 = wp * (array[ix][iy][ip] - array[ix][iy][ip + 1])
    + array[ix][iy][ip + 1];
  aux01 = wp * (array[ix][iy + 1][ip] - array[ix][iy + 1][ip + 1])
    + array[ix][iy + 1][ip + 1];
  aux10 = wp * (array[ix + 1][iy][ip] - array[ix + 1][iy][ip + 1])
    + array[ix + 1][iy][ip + 1];
  aux11 = wp * (array[ix + 1][iy + 1][ip] - array[ix + 1][iy + 1][ip + 1])
    + array[ix + 1][iy + 1][ip + 1];

  /* Interpolate horizontally... */
  aux00 = wy * (aux00 - aux01) + aux01;
  aux11 = wy * (aux10 - aux11) + aux11;
  *var = wx * (aux00 - aux11) + aux11;
}
```

5.17.1.3 void intpol_met_space (met_t * *met*, double *p*, double *lon*, double *lat*, double * *t*)

Spatial interpolation of meteorological data.

Definition at line 334 of file erafm.c.

```
              {

  double wp, wx, wy;

  int ip, ix, iy;

  /* Check longitude... */
  if (lon < 0)
    lon += 360;

  /* Get indices... */
  ip = locate(met->p, met->np, p);
  ix = locate(met->lon, met->nx, lon);
  iy = locate(met->lat, met->ny, lat);

  /* Get weights... */
  wp = (met->p[ip + 1] - p) / (met->p[ip + 1] - met->p[ip]);
  wx = (met->lon[ix + 1] - lon) / (met->lon[ix + 1] - met->lon[ix]);
  wy = (met->lat[iy + 1] - lat) / (met->lat[iy + 1] - met->lat[iy]);

  /* Interpolate... */
  intpol_met_3d(met->t, ip, ix, iy, wp, wx, wy, t);
}
```

Here is the call graph for this function:

5.17.1.4 void read_met (char * *filename*, met_t * *met*)

Read meteorological data file.

Definition at line 365 of file erafm.c.

```
                {

  int ip, dimid, ncid, varid, year, mon, day, hour;

  size_t np, nx, ny;

  /* Write info... */
  printf("Read meteorological data: %s\n", filename);

  /* Open netCDF file... */
  NC(nc_open(filename, NC_NOWRITE, &ncid));

  /* Get dimensions... */
  NC(nc_inq_dimid(ncid, "lon", &dimid));
  NC(nc_inq_dimlen(ncid, dimid, &nx));
  if (nx > EX)
    ERRMSG("Too many longitudes!");

  NC(nc_inq_dimid(ncid, "lat", &dimid));
  NC(nc_inq_dimlen(ncid, dimid, &ny));
  if (ny > EY)
    ERRMSG("Too many latitudes!");

  NC(nc_inq_dimid(ncid, "lev", &dimid));
  NC(nc_inq_dimlen(ncid, dimid, &np));
  if (np > EP)
    ERRMSG("Too many pressure levels!");

  /* Store dimensions... */
  met->np = (int) np;
  met->nx = (int) nx;
  met->ny = (int) ny;

  /* Read geolocations... */
  NC(nc_inq_varid(ncid, "time", &varid));
  NC(nc_get_var_double(ncid, varid, &met->time));

```

```
  NC(nc_inq_varid(ncid, "lev", &varid));
  NC(nc_get_var_double(ncid, varid, met->p));

  NC(nc_inq_varid(ncid, "lon", &varid));
  NC(nc_get_var_double(ncid, varid, met->lon));

  NC(nc_inq_varid(ncid, "lat", &varid));
  NC(nc_get_var_double(ncid, varid, met->lat));

  /* Convert time... */
  year = (int) met->time / 10000;
  met->time -= year * 10000;
  mon = (int) met->time / 100;
  met->time -= mon * 100;
  day = (int) (met->time);
  met->time -= day;
  hour = (int) (met->time * 24.);
  time2jsec(year, mon, day, hour, 0, 0, 0, &met->time);

  /* Check and convert pressure levels... */
  for (ip = 0; ip < met->np; ip++) {
    if (ip > 0 && met->p[ip - 1] > met->p[ip])
      ERRMSG("Pressure levels must be in descending order!");
    met->p[ip] /= 100.;
  }

  /* Read meteorological data... */
  read_met_help(ncid, "T", "t", met, met->np, met->t, 1.0);

  /* Extrapolate data for lower boundary... */
  read_met_extrapolate(met);

  /* Close file... */
  NC(nc_close(ncid));
}
```

Here is the call graph for this function:

5.17.1.5 void read_met_extrapolate (met_t * *met*)

Extrapolate meteorological data at lower boundary.

Definition at line 442 of file erafm.c.

```
                 {

  int ip, ip0, ix, iy;

  /* Loop over columns... */
  for (ix = 0; ix < met->nx; ix++)
    for (iy = 0; iy < met->ny; iy++) {

      /* Find lowest valid data point... */
      for (ip0 = met->np - 1; ip0 >= 0; ip0--)
        if (!gsl_finite(met->t[ix][iy][ip0]))
          break;

      /* Extrapolate... */
      for (ip = ip0; ip >= 0; ip--)
        met->t[ix][iy][ip]
          = (float) LIN(met->p[ip + 1], met->t[ix][iy][ip + 1],
                        met->p[ip + 2], met->t[ix][iy][ip + 2], met->p[ip]);
    }
}
```

5.17.1.6 void read_met_help (int *ncid*, char * *varname*, char * *varname2*, met_t * *met*, int *np*, float *dest[EX][EY][EP]*, float *scl*)

Read and convert variable from meteorological data file.

Definition at line 466 of file erafm.c.

```
                {

  static float *help;

  int ip, ix, iy, n = 0, varid;

  /* Alloc... */
  ALLOC(help, float,
        EP * EX * EY);

  /* Check if variable exists... */
  if (nc_inq_varid(ncid, varname, &varid) != NC_NOERR)
    if (nc_inq_varid(ncid, varname2, &varid) != NC_NOERR)
      ERRMSG("Cannot read variable!");

  /* Read data... */
  NC(nc_get_var_float(ncid, varid, help));

  /* Copy and check data... */
  for (ip = 0; ip < np; ip++)
    for (iy = 0; iy < met->ny; iy++)
      for (ix = 0; ix < met->nx; ix++) {
        dest[ix][iy][ip] = scl * help[n++];
        if (dest[ix][iy][ip] < -1e10 || dest[ix][iy][ip] > 1e10)
          dest[ix][iy][ip] = GSL_NAN;
      }

  /* Free... */
  free(help);
}
```

5.17.1.7 int main (int *argc*, char ∗ *argv[ ]*)

Definition at line 119 of file erafm.c.

```
                   {

  met_t *met;

  static pert_t *pert, *pert2;

  static wave_t wave;

  char pertname[LEN];

  double temp, var_dh, wsum, kp[NSHAPE], kw[NSHAPE];

  int bg_poly_x, itrack, ixtrack, ix, iy, iz, nz,
    ncid, bt_varid, pt_varid, var_varid, dimid[2];

  size_t start[2], count[2];

  /* ------------------------------------------------------------
     Initialize...
     ------------------------------------------------------------ */

  /* Check arguments... */
  if (argc < 5)
    ERRMSG("Give parameters: <ctl> <era.nc> <airs.nc> <kernel.tab>");

  /* Get control parameters... */
  scan_ctl(argc, argv, "PERTNAME", -1, "4mu", pertname);
  bg_poly_x = (int) scan_ctl(argc, argv, "BG_POLY_X", -1, "5", NULL);
  var_dh = scan_ctl(argc, argv, "VAR_DH", -1, "100", NULL);

  /* Alloc... */
  ALLOC(met, met_t, 1);
  ALLOC(pert, pert_t, 1);
  ALLOC(pert2, pert_t, 1);

  /* Read meteorological data... */
  read_met(argv[2], met);

  /* Read AIRS perturbation data... */
  read_pert(argv[3], pertname, pert);

  /* Copy perturbation data... */
  memcpy(pert2, pert, sizeof(pert_t));

  /* Read kernel function... */
  read_shape(argv[4], kp, kw, &nz);
```

```
  for (iz = 0; iz < nz; iz++)
    kp[iz] = P(kp[iz]);

  /* ------------------------------------------------------------
     Simulate AIRS data...
     ------------------------------------------------------------ */

  /* Write info... */
  printf("Simulate measurements...\n");

  /* Loop over scans... */
  for (itrack = 0; itrack < pert->ntrack; itrack++) {

    /* Loop over footprints... */
    for (ixtrack = 0; ixtrack < pert->nxtrack; ixtrack++) {

      /* Check measured data... */
      if (pert->time[itrack][ixtrack] < 0
          || pert->lon[itrack][ixtrack] < -180
          || pert->lon[itrack][ixtrack] > 180
          || pert->lat[itrack][ixtrack] < -90
          || pert->lat[itrack][ixtrack] > 90
          || pert->pt[itrack][ixtrack] < -100
          || pert->pt[itrack][ixtrack] > 100
          || !gsl_finite(pert->bt[itrack][ixtrack])
          || !gsl_finite(pert->pt[itrack][ixtrack])
          || !gsl_finite(pert->var[itrack][ixtrack])
          || !gsl_finite(pert->dc[itrack][ixtrack]))
        continue;

      /* Estimate brightness temperature... */
      pert2->bt[itrack][ixtrack] = wsum = 0;
      for (iz = 0; iz < nz; iz++) {
        intpol_met_space(met, kp[iz], pert->lon[itrack][ixtrack],
                         pert->lat[itrack][ixtrack], &temp);
        pert2->bt[itrack][ixtrack] += kw[iz] * temp;
        wsum += kw[iz];
      }
      pert2->bt[itrack][ixtrack] /= wsum;
    }
  }

  /* ------------------------------------------------------------
     Calculate perturbations and variances...
     ------------------------------------------------------------ */

  /* Write info... */
  printf("Get perturbations and variances...\n");

  /* Convert to wave analysis struct... */
  pert2wave(pert2, &wave, 0, pert2->ntrack - 1, 0, pert2->nxtrack - 1);

  /* Estimate background... */
  background_poly(&wave, bg_poly_x, 0);

  /* Compute variance... */
  variance(&wave, var_dh);

  /* Copy data... */
  for (ix = 0; ix < wave.nx; ix++)
    for (iy = 0; iy < wave.ny; iy++) {
      pert2->pt[iy][ix] = wave.pt[ix][iy];
      pert2->var[iy][ix] = wave.var[ix][iy];
    }

  /* ------------------------------------------------------------
     Write to netCDF file...
     ------------------------------------------------------------ */

  /* Write info... */
  printf("Add data to netCDF file...\n");

  /* Open netCDF file... */
  NC(nc_open(argv[3], NC_WRITE, &ncid));

  /* Get dimensions... */
  NC(nc_inq_dimid(ncid, "NTRACK", &dimid[0]));
  NC(nc_inq_dimid(ncid, "NXTRACK", &dimid[1]));

  /* Enter define mode... */
  NC(nc_redef(ncid));

  /* Add variables... */
  NC(nc_def_var(ncid, "bt_sim", NC_FLOAT, 2, dimid, &bt_varid));
  addatt(ncid, bt_varid, "K", "simulated brightness temperature");
  NC(nc_def_var(ncid, "bt_sim_pt", NC_FLOAT, 2, dimid, &pt_varid));
  addatt(ncid, pt_varid, "K",
```

```
          "simulated brightness temperature perturbation");
  NC(nc_def_var(ncid, "bt_sim_var", NC_FLOAT, 2, dimid, &var_varid));
  addatt(ncid, var_varid, "K^2", "simulated brightness temperature variance");

  /* Leave define mode... */
  NC(nc_enddef(ncid));

  /* Loop over tracks... */
  for (itrack = 0; itrack < pert2->ntrack; itrack++) {

    /* Set array sizes... */
    start[0] = (size_t) itrack;
    start[1] = 0;
    count[0] = 1;
    count[1] = (size_t) pert2->nxtrack;

    /* Write data... */
    NC(nc_put_vara_double(ncid, bt_varid, start, count, pert2->bt[itrack]));
    NC(nc_put_vara_double(ncid, pt_varid, start, count, pert2->pt[itrack]));
    NC(nc_put_vara_double(ncid, var_varid, start, count, pert2->var[itrack]));
  }

  /* Close file... */
  NC(nc_close(ncid));

  /* Free... */
  free(met);
  free(pert);
  free(pert2);

  return EXIT_SUCCESS;
}
```

Here is the call graph for this function:

5.18 erafm.c

```
#include "libairs.h"

/* ------------------------------------------------------------
   Dimensions...
   ------------------------------------------------------------ */

#define EP 91

#define EX 2880

#define EY 1441

/* ------------------------------------------------------------
   Constants...
   ------------------------------------------------------------ */

#define H0 7.0

#define P0 1013.25

/* ------------------------------------------------------------
   Macros...
   ------------------------------------------------------------ */

#define P(z) (P0*exp(-(z)/H0))

/* ------------------------------------------------------------
   Structs...
   ------------------------------------------------------------ */

typedef struct {

  double time;

  int nx;

  int ny;

  int np;

  double lon[EX];

  double lat[EY];

```

```
  double p[EP];

  float t[EX][EY][EP];

} met_t;

/* ------------------------------------------------------------
   Functions...
   ------------------------------------------------------------ */

/* Add variable defintions to netCDF file. */
void addatt(
  int ncid,
  int varid,
  const char *unit,
  const char *long_name);

void intpol_met_3d(
  float array[EX][EY][EP],
  int ip,
  int ix,
  int iy,
  double wp,
  double wx,
  double wy,
  double *var);

void intpol_met_space(
  met_t * met,
  double p,
  double lon,
  double lat,
  double *t);

void read_met(
  char *filename,
  met_t * met);

void read_met_extrapolate(
  met_t * met);

void read_met_help(
  int ncid,
  char *varname,
  char *varname2,
  met_t * met,
  int np,
  float dest[EX][EY][EP],
  float scl);

/* ------------------------------------------------------------
   Main...
   ------------------------------------------------------------ */

int main(
  int argc,
  char *argv[]) {

  met_t *met;

  static pert_t *pert, *pert2;

  static wave_t wave;

  char pertname[LEN];

  double temp, var_dh, wsum, kp[NSHAPE], kw[NSHAPE];

  int bg_poly_x, itrack, ixtrack, ix, iy, iz, nz,
    ncid, bt_varid, pt_varid, var_varid, dimid[2];

  size_t start[2], count[2];

  /* ------------------------------------------------------------
     Initialize...
     ------------------------------------------------------------ */

  /* Check arguments... */
  if (argc < 5)
    ERRMSG("Give parameters: <ctl> <era.nc> <airs.nc> <kernel.tab>");

  /* Get control parameters... */
  scan_ctl(argc, argv, "PERTNAME", -1, "4mu", pertname);
  bg_poly_x = (int) scan_ctl(argc, argv, "BG_POLY_X", -1, "5", NULL);
  var_dh = scan_ctl(argc, argv, "VAR_DH", -1, "100", NULL);

  /* Alloc... */
```

```
  ALLOC(met, met_t, 1);
  ALLOC(pert, pert_t, 1);
  ALLOC(pert2, pert_t, 1);

  /* Read meteorological data... */
  read_met(argv[2], met);

  /* Read AIRS perturbation data... */
  read_pert(argv[3], pertname, pert);

  /* Copy perturbation data... */
  memcpy(pert2, pert, sizeof(pert_t));

  /* Read kernel function... */
  read_shape(argv[4], kp, kw, &nz);
  for (iz = 0; iz < nz; iz++)
    kp[iz] = P(kp[iz]);

  /* ------------------------------------------------------------
     Simulate AIRS data...
     ------------------------------------------------------------ */

  /* Write info... */
  printf("Simulate measurements...\n");

  /* Loop over scans... */
  for (itrack = 0; itrack < pert->ntrack; itrack++) {

    /* Loop over footprints... */
    for (ixtrack = 0; ixtrack < pert->nxtrack; ixtrack++) {

      /* Check measured data... */
      if (pert->time[itrack][ixtrack] < 0
          || pert->lon[itrack][ixtrack] < -180
          || pert->lon[itrack][ixtrack] > 180
          || pert->lat[itrack][ixtrack] < -90
          || pert->lat[itrack][ixtrack] > 90
          || pert->pt[itrack][ixtrack] < -100
          || pert->pt[itrack][ixtrack] > 100
          || !gsl_finite(pert->bt[itrack][ixtrack])
          || !gsl_finite(pert->pt[itrack][ixtrack])
          || !gsl_finite(pert->var[itrack][ixtrack])
          || !gsl_finite(pert->dc[itrack][ixtrack]))
        continue;

      /* Estimate brightness temperature... */
      pert2->bt[itrack][ixtrack] = wsum = 0;
      for (iz = 0; iz < nz; iz++) {
        intpol_met_space(met, kp[iz], pert->lon[itrack][ixtrack],
                         pert->lat[itrack][ixtrack], &temp);
        pert2->bt[itrack][ixtrack] += kw[iz] * temp;
        wsum += kw[iz];
      }
      pert2->bt[itrack][ixtrack] /= wsum;
    }
  }

  /* ------------------------------------------------------------
     Calculate perturbations and variances...
     ------------------------------------------------------------ */

  /* Write info... */
  printf("Get perturbations and variances...\n");

  /* Convert to wave analysis struct... */
  pert2wave(pert2, &wave, 0, pert2->ntrack - 1, 0, pert2->nxtrack - 1);

  /* Estimate background... */
  background_poly(&wave, bg_poly_x, 0);

  /* Compute variance... */
  variance(&wave, var_dh);

  /* Copy data... */
  for (ix = 0; ix < wave.nx; ix++)
    for (iy = 0; iy < wave.ny; iy++) {
      pert2->pt[iy][ix] = wave.pt[ix][iy];
      pert2->var[iy][ix] = wave.var[ix][iy];
    }

  /* ------------------------------------------------------------
     Write to netCDF file...
     ------------------------------------------------------------ */

  /* Write info... */
  printf("Add data to netCDF file...\n");

```

```
  /* Open netCDF file... */
  NC(nc_open(argv[3], NC_WRITE, &ncid));

  /* Get dimensions... */
  NC(nc_inq_dimid(ncid, "NTRACK", &dimid[0]));
  NC(nc_inq_dimid(ncid, "NXTRACK", &dimid[1]));

  /* Enter define mode... */
  NC(nc_redef(ncid));

  /* Add variables... */
  NC(nc_def_var(ncid, "bt_sim", NC_FLOAT, 2, dimid, &bt_varid));
  addatt(ncid, bt_varid, "K", "simulated brightness temperature");
  NC(nc_def_var(ncid, "bt_sim_pt", NC_FLOAT, 2, dimid, &pt_varid));
  addatt(ncid, pt_varid, "K",
         "simulated brightness temperature perturbation");
  NC(nc_def_var(ncid, "bt_sim_var", NC_FLOAT, 2, dimid, &var_varid));
  addatt(ncid, var_varid, "K^2", "simulated brightness temperature variance");

  /* Leave define mode... */
  NC(nc_enddef(ncid));

  /* Loop over tracks... */
  for (itrack = 0; itrack < pert2->ntrack; itrack++) {

    /* Set array sizes... */
    start[0] = (size_t) itrack;
    start[1] = 0;
    count[0] = 1;
    count[1] = (size_t) pert2->nxtrack;

    /* Write data... */
    NC(nc_put_vara_double(ncid, bt_varid, start, count, pert2->bt[itrack]));
    NC(nc_put_vara_double(ncid, pt_varid, start, count, pert2->pt[itrack]));
    NC(nc_put_vara_double(ncid, var_varid, start, count, pert2->var[itrack]));
  }

  /* Close file... */
  NC(nc_close(ncid));

  /* Free... */
  free(met);
  free(pert);
  free(pert2);

  return EXIT_SUCCESS;
}

/*****************************************************************************/

void addatt(
  int ncid,
  int varid,
  const char *unit,
  const char *long_name) {

  /* Set long name... */
  NC(nc_put_att_text(ncid, varid, "long_name", strlen(long_name), long_name));

  /* Set units... */
  NC(nc_put_att_text(ncid, varid, "units", strlen(unit), unit));
}

/*****************************************************************************/

void intpol_met_3d(
  float array[EX][EY][EP],
  int ip,
  int ix,
  int iy,
  double wp,
  double wx,
  double wy,
  double *var) {

  double aux00, aux01, aux10, aux11;

  /* Interpolate vertically... */
  aux00 = wp * (array[ix][iy][ip] - array[ix][iy][ip + 1])
    + array[ix][iy][ip + 1];
  aux01 = wp * (array[ix][iy + 1][ip] - array[ix][iy + 1][ip + 1])
    + array[ix][iy + 1][ip + 1];
  aux10 = wp * (array[ix + 1][iy][ip] - array[ix + 1][iy][ip + 1])
    + array[ix + 1][iy][ip + 1];
  aux11 = wp * (array[ix + 1][iy + 1][ip] - array[ix + 1][iy + 1][ip + 1])
    + array[ix + 1][iy + 1][ip + 1];

```

```
  /* Interpolate horizontally... */
  aux00 = wy * (aux00 - aux01) + aux01;
  aux11 = wy * (aux10 - aux11) + aux11;
  *var = wx * (aux00 - aux11) + aux11;
}

/*****************************************************************************/

void intpol_met_space(
  met_t * met,
  double p,
  double lon,
  double lat,
  double *t) {

  double wp, wx, wy;

  int ip, ix, iy;

  /* Check longitude... */
  if (lon < 0)
    lon += 360;

  /* Get indices... */
  ip = locate(met->p, met->np, p);
  ix = locate(met->lon, met->nx, lon);
  iy = locate(met->lat, met->ny, lat);

  /* Get weights... */
  wp = (met->p[ip + 1] - p) / (met->p[ip + 1] - met->p[ip]);
  wx = (met->lon[ix + 1] - lon) / (met->lon[ix + 1] - met->lon[ix]);
  wy = (met->lat[iy + 1] - lat) / (met->lat[iy + 1] - met->lat[iy]);

  /* Interpolate... */
  intpol_met_3d(met->t, ip, ix, iy, wp, wx, wy, t);
}

/*****************************************************************************/

void read_met(
  char *filename,
  met_t * met) {

  int ip, dimid, ncid, varid, year, mon, day, hour;

  size_t np, nx, ny;

  /* Write info... */
  printf("Read meteorological data: %s\n", filename);

  /* Open netCDF file... */
  NC(nc_open(filename, NC_NOWRITE, &ncid));

  /* Get dimensions... */
  NC(nc_inq_dimid(ncid, "lon", &dimid));
  NC(nc_inq_dimlen(ncid, dimid, &nx));
  if (nx > EX)
    ERRMSG("Too many longitudes!");

  NC(nc_inq_dimid(ncid, "lat", &dimid));
  NC(nc_inq_dimlen(ncid, dimid, &ny));
  if (ny > EY)
    ERRMSG("Too many latitudes!");

  NC(nc_inq_dimid(ncid, "lev", &dimid));
  NC(nc_inq_dimlen(ncid, dimid, &np));
  if (np > EP)
    ERRMSG("Too many pressure levels!");

  /* Store dimensions... */
  met->np = (int) np;
  met->nx = (int) nx;
  met->ny = (int) ny;

  /* Read geolocations... */
  NC(nc_inq_varid(ncid, "time", &varid));
  NC(nc_get_var_double(ncid, varid, &met->time));

  NC(nc_inq_varid(ncid, "lev", &varid));
  NC(nc_get_var_double(ncid, varid, met->p));

  NC(nc_inq_varid(ncid, "lon", &varid));
  NC(nc_get_var_double(ncid, varid, met->lon));

  NC(nc_inq_varid(ncid, "lat", &varid));
  NC(nc_get_var_double(ncid, varid, met->lat));

```

```
  /* Convert time... */
  year = (int) met->time / 10000;
  met->time -= year * 10000;
  mon = (int) met->time / 100;
  met->time -= mon * 100;
  day = (int) (met->time);
  met->time -= day;
  hour = (int) (met->time * 24.);
  time2jsec(year, mon, day, hour, 0, 0, 0, &met->time);

  /* Check and convert pressure levels... */
  for (ip = 0; ip < met->np; ip++) {
    if (ip > 0 && met->p[ip - 1] > met->p[ip])
      ERRMSG("Pressure levels must be in descending order!");
    met->p[ip] /= 100.;
  }

  /* Read meteorological data... */
  read_met_help(ncid, "T", "t", met, met->np, met->t, 1.0);

  /* Extrapolate data for lower boundary... */
  read_met_extrapolate(met);

  /* Close file... */
  NC(nc_close(ncid));
}

/*****************************************************************************/

void read_met_extrapolate(
  met_t * met) {

  int ip, ip0, ix, iy;

  /* Loop over columns... */
  for (ix = 0; ix < met->nx; ix++)
    for (iy = 0; iy < met->ny; iy++) {

      /* Find lowest valid data point... */
      for (ip0 = met->np - 1; ip0 >= 0; ip0--)
        if (!gsl_finite(met->t[ix][iy][ip0]))
          break;

      /* Extrapolate... */
      for (ip = ip0; ip >= 0; ip--)
        met->t[ix][iy][ip]
          = (float) LIN(met->p[ip + 1], met->t[ix][iy][ip + 1],
                        met->p[ip + 2], met->t[ix][iy][ip + 2], met->p[ip]);
    }
}

/*****************************************************************************/

void read_met_help(
  int ncid,
  char *varname,
  char *varname2,
  met_t * met,
  int np,
  float dest[EX][EY][EP],
  float scl) {

  static float *help;

  int ip, ix, iy, n = 0, varid;

  /* Alloc... */
  ALLOC(help, float,
        EP * EX * EY);

  /* Check if variable exists... */
  if (nc_inq_varid(ncid, varname, &varid) != NC_NOERR)
    if (nc_inq_varid(ncid, varname2, &varid) != NC_NOERR)
      ERRMSG("Cannot read variable!");

  /* Read data... */
  NC(nc_get_var_float(ncid, varid, help));

  /* Copy and check data... */
  for (ip = 0; ip < np; ip++)
    for (iy = 0; iy < met->ny; iy++)
      for (ix = 0; ix < met->nx; ix++) {
        dest[ix][iy][ip] = scl * help[n++];
        if (dest[ix][iy][ip] < -1e10 || dest[ix][iy][ip] > 1e10)
          dest[ix][iy][ip] = GSL_NAN;
      }

```

```
  /* Free... */
  free(help);
}
```

5.19 events.c File Reference

Functions

- int main (int argc, char ∗argv[ ])

5.19.1 Function Documentation

5.19.1.1 int main (int *argc*, char ∗ *argv[ ]*)

Definition at line 3 of file events.c.

```
                  {

  static pert_t *pert;

  static wave_t *wave;

  static FILE *in, *out;

  static char pertname[LEN];

  static double gauss_fwhm, var_dh, varmin, varmax, nu, t230 = 230.0,
    dt230, tbg, nesr, nedt = 0;

  static int iarg, ix, iy, bg_poly_x, bg_poly_y, bg_smooth_x, bg_smooth_y,
    itrack, itrack2, itrackmax, ixtrack, ixtrack2, ixtrackmax, dtrack = 15,
    dxtrack = 15;

  /* Check arguments... */
  if (argc < 4)
    ERRMSG("Give parameters: <ctl> <events.tab> <pert1.nc> [<pert2.nc> ...]");

  /* Get control parameters... */
  scan_ctl(argc, argv, "PERTNAME", -1, "4mu", pertname);
  bg_poly_x = (int) scan_ctl(argc, argv, "BG_POLY_X", -1, "0", NULL);
  bg_poly_y = (int) scan_ctl(argc, argv, "BG_POLY_Y", -1, "0", NULL);
  bg_smooth_x = (int) scan_ctl(argc, argv, "BG_SMOOTH_X", -1, "0", NULL);
  bg_smooth_y = (int) scan_ctl(argc, argv, "BG_SMOOTH_Y", -1, "0", NULL);
  gauss_fwhm = scan_ctl(argc, argv, "GAUSS_FWHM", -1, "0", NULL);
  var_dh = scan_ctl(argc, argv, "VAR_DH", -1, "0", NULL);
  varmin = scan_ctl(argc, argv, "VARMIN", -1, "", NULL);
  dt230 = scan_ctl(argc, argv, "DT230", -1, "0.16", NULL);
  nu = scan_ctl(argc, argv, "NU", -1, "2345.0", NULL);

  /* Alloc... */
  ALLOC(pert, pert_t, 1);

  /* Create file... */
  printf("Write event data: %s\n", argv[2]);
  if (!(out = fopen(argv[2], "w")))
    ERRMSG("Cannot create file!");

  /* Write header... */
  fprintf(out,
          "# $1 = time [s]\n"
          "# $2 = longitude [deg]\n"
          "# $3 = latitude [deg]\n" "# $4 = maximum variance [K^2]\n\n");

  /* Loop over perturbation files... */
  for (iarg = 3; iarg < argc; iarg++) {

    /* Read perturbation data... */
    if (!(in = fopen(argv[iarg], "r")))
      continue;
    else {
      fclose(in);
      read_pert(argv[iarg], pertname, pert);
    }

```

```
    /* Recalculate background and perturbations... */
    if (bg_poly_x > 0 || bg_poly_y > 0 ||
        bg_smooth_x > 0 || bg_smooth_y > 0 || gauss_fwhm > 0 || var_dh > 0) {

      /* Allocate... */
      ALLOC(wave, wave_t, 1);

      /* Convert to wave analysis struct... */
      pert2wave(pert, wave, 0, pert->ntrack - 1, 0, pert->nxtrack - 1);

      /* Estimate background... */
      background_poly(wave, bg_poly_x, bg_poly_y);
      background_smooth(wave, bg_smooth_x, bg_smooth_y);

      /* Gaussian filter... */
      gauss(wave, gauss_fwhm);

      /* Compute variance... */
      variance(wave, var_dh);

      /* Copy data... */
      for (ix = 0; ix < wave->nx; ix++)
        for (iy = 0; iy < wave->ny; iy++) {
          pert->pt[iy][ix] = wave->pt[ix][iy];
          pert->var[iy][ix] = wave->var[ix][iy];
        }

      /* Free... */
      free(wave);
    }

    /* Apply noise correction... */
    if (dt230 > 0)
      for (itrack = 0; itrack < pert->ntrack; itrack++)
        for (ixtrack = 0; ixtrack < pert->nxtrack; ixtrack++) {
          nesr = planck(t230 + dt230, nu) - planck(t230, nu);
          tbg = pert->bt[itrack][ixtrack] - pert->pt[itrack][ixtrack];
          nedt = brightness(planck(tbg, nu) + nesr, nu) - tbg;
          pert->var[itrack][ixtrack] -= gsl_pow_2(nedt);
        }

    /* Find local maxima... */
    for (itrack = 0; itrack < pert->ntrack; itrack += 2 * dtrack)
      for (ixtrack = dxtrack / 2; ixtrack < pert->nxtrack;
           ixtrack += 2 * dxtrack) {

        /* Init... */
        varmax = 0;
        itrackmax = -999;
        ixtrackmax = -999;

        /* Loop over box... */
        for (itrack2 = itrack;
             itrack2 < GSL_MIN(itrack + dtrack, pert->ntrack); itrack2++)
          for (ixtrack2 = ixtrack;
               ixtrack2 < GSL_MIN(ixtrack + dxtrack, pert->nxtrack);
               ixtrack2++)
            if (pert->var[itrack2][ixtrack2] >= varmax) {
              varmax = pert->var[itrack2][ixtrack2];
              itrackmax = itrack2;
              ixtrackmax = ixtrack2;
            }

        /* Report event... */
        if (itrackmax >= 0 && ixtrackmax >= 0 && varmax >= varmin)
          fprintf(out, "%.2f %g %g %g\n",
                  pert->time[itrackmax][ixtrackmax],
                  pert->lon[itrackmax][ixtrackmax],
                  pert->lat[itrackmax][ixtrackmax],
                  pert->var[itrackmax][ixtrackmax]);
      }
  }

  /* Close file... */
  fclose(out);

  /* Free... */
  free(pert);

  return EXIT_SUCCESS;
}
```

Here is the call graph for this function:

5.20 events.c

```c
#include "libairs.h"

int main(
  int argc,
  char *argv[]) {

  static pert_t *pert;

  static wave_t *wave;

  static FILE *in, *out;

  static char pertname[LEN];

  static double gauss_fwhm, var_dh, varmin, varmax, nu, t230 = 230.0,
    dt230, tbg, nesr, nedt = 0;

  static int iarg, ix, iy, bg_poly_x, bg_poly_y, bg_smooth_x, bg_smooth_y,
    itrack, itrack2, itrackmax, ixtrack, ixtrack2, ixtrackmax, dtrack = 15,
    dxtrack = 15;

  /* Check arguments... */
  if (argc < 4)
    ERRMSG("Give parameters: <ctl> <events.tab> <pert1.nc> [<pert2.nc> ...]");

  /* Get control parameters... */
  scan_ctl(argc, argv, "PERTNAME", -1, "4mu", pertname);
  bg_poly_x = (int) scan_ctl(argc, argv, "BG_POLY_X", -1, "0", NULL);
  bg_poly_y = (int) scan_ctl(argc, argv, "BG_POLY_Y", -1, "0", NULL);
  bg_smooth_x = (int) scan_ctl(argc, argv, "BG_SMOOTH_X", -1, "0", NULL);
  bg_smooth_y = (int) scan_ctl(argc, argv, "BG_SMOOTH_Y", -1, "0", NULL);
  gauss_fwhm = scan_ctl(argc, argv, "GAUSS_FWHM", -1, "0", NULL);
  var_dh = scan_ctl(argc, argv, "VAR_DH", -1, "0", NULL);
  varmin = scan_ctl(argc, argv, "VARMIN", -1, "", NULL);
  dt230 = scan_ctl(argc, argv, "DT230", -1, "0.16", NULL);
  nu = scan_ctl(argc, argv, "NU", -1, "2345.0", NULL);

  /* Alloc... */
  ALLOC(pert, pert_t, 1);

  /* Create file... */
  printf("Write event data: %s\n", argv[2]);
  if (!(out = fopen(argv[2], "w")))
    ERRMSG("Cannot create file!");

  /* Write header... */
  fprintf(out,
          "# $1 = time [s]\n"
          "# $2 = longitude [deg]\n"
          "# $3 = latitude [deg]\n" "# $4 = maximum variance [K^2]\n\n");

  /* Loop over perturbation files... */
  for (iarg = 3; iarg < argc; iarg++) {

    /* Read perturbation data... */
    if (!(in = fopen(argv[iarg], "r")))
      continue;
    else {
      fclose(in);
      read_pert(argv[iarg], pertname, pert);
    }

    /* Recalculate background and perturbations... */
    if (bg_poly_x > 0 || bg_poly_y > 0 ||
        bg_smooth_x > 0 || bg_smooth_y > 0 || gauss_fwhm > 0 || var_dh > 0) {

      /* Allocate... */
      ALLOC(wave, wave_t, 1);

      /* Convert to wave analysis struct... */
      pert2wave(pert, wave, 0, pert->ntrack - 1, 0, pert->nxtrack - 1);

      /* Estimate background... */
      background_poly(wave, bg_poly_x, bg_poly_y);
      background_smooth(wave, bg_smooth_x, bg_smooth_y);

      /* Gaussian filter... */
      gauss(wave, gauss_fwhm);

      /* Compute variance... */
      variance(wave, var_dh);

      /* Copy data... */
      for (ix = 0; ix < wave->nx; ix++)
```

```
        for (iy = 0; iy < wave->ny; iy++) {
          pert->pt[iy][ix] = wave->pt[ix][iy];
          pert->var[iy][ix] = wave->var[ix][iy];
        }

      /* Free... */
      free(wave);
    }

    /* Apply noise correction... */
    if (dt230 > 0)
      for (itrack = 0; itrack < pert->ntrack; itrack++)
        for (ixtrack = 0; ixtrack < pert->nxtrack; ixtrack++) {
          nesr = planck(t230 + dt230, nu) - planck(t230, nu);
          tbg = pert->bt[itrack][ixtrack] - pert->pt[itrack][ixtrack];
          nedt = brightness(planck(tbg, nu) + nesr, nu) - tbg;
          pert->var[itrack][ixtrack] -= gsl_pow_2(nedt);
        }

    /* Find local maxima... */
    for (itrack = 0; itrack < pert->ntrack; itrack += 2 * dtrack)
      for (ixtrack = dxtrack / 2; ixtrack < pert->nxtrack;
           ixtrack += 2 * dxtrack) {

        /* Init... */
        varmax = 0;
        itrackmax = -999;
        ixtrackmax = -999;

        /* Loop over box... */
        for (itrack2 = itrack;
             itrack2 < GSL_MIN(itrack + dtrack, pert->ntrack); itrack2++)
          for (ixtrack2 = ixtrack;
               ixtrack2 < GSL_MIN(ixtrack + dxtrack, pert->nxtrack);
               ixtrack2++)
            if (pert->var[itrack2][ixtrack2] >= varmax) {
              varmax = pert->var[itrack2][ixtrack2];
              itrackmax = itrack2;
              ixtrackmax = ixtrack2;
            }

        /* Report event... */
        if (itrackmax >= 0 && ixtrackmax >= 0 && varmax >= varmin)
          fprintf(out, "%.2f %g %g %g\n",
                  pert->time[itrackmax][ixtrackmax],
                  pert->lon[itrackmax][ixtrackmax],
                  pert->lat[itrackmax][ixtrackmax],
                  pert->var[itrackmax][ixtrackmax]);
      }
  }

  /* Close file... */
  fclose(out);

  /* Free... */
  free(pert);

  return EXIT_SUCCESS;
}
```

5.21 extract.c File Reference

Functions

- double gph2z (double gph, double lat)
- int main (int argc, char *argv[ ])

Variables

- int airs_chan [L1_NCHAN]

5.21.1 Function Documentation

5.21.1.1 double gph2z (double *gph*, double *lat*)

Definition at line 141 of file extract.c.

```
                {

  double a = 3.086e-3, g0 = gravity(0.0, 45.0), glat = gravity(0.0, lat);

  return glat / a - sqrt(gsl_pow_2(glat / a) - 2 * g0 * gph / a);
}
```

Here is the call graph for this function:

5.21.1.2 int main (int *argc*, char ∗ *argv[ ]*)

Definition at line 27 of file extract.c.

```
                  {

  static airs_rad_gran_t airs_rad_gran;
  static airs_ret_gran_t airs_ret_gran;

  static airs_l1_t l1;
  static airs_l2_t l2;

  int ichan, lay, track, xtrack;

  /* Check arguments... */
  if (argc != 4)
    ERRMSG("Give parameters: <airs_l1_file> <airs_l2_file> <out.nc>");

  /* Check Level-1 filename... */
  if (argv[1][0] != '-') {

    /* Read data... */
    printf("Read AIRS Level-1 file: %s\n", argv[1]);
    airs_rad_rdr(argv[1], &airs_rad_gran);

    /* Flag bad data... */
    for (track = 0; track < AIRS_RAD_GEOTRACK; track++)
      for (xtrack = 0; xtrack < AIRS_RAD_GEOXTRACK; xtrack++)
        for (ichan = 0; ichan < L1_NCHAN; ichan++)
          if ((airs_rad_gran.state[track][xtrack] != 0)
              || (airs_rad_gran.ExcludedChans[airs_chan[ichan]] > 2)
              || (airs_rad_gran.CalChanSummary[airs_chan[ichan]] & 8)
              || (airs_rad_gran.CalChanSummary[airs_chan[ichan]] & (32 + 64))
              || (airs_rad_gran.CalFlag[track][airs_chan[ichan]] & 16))
            airs_rad_gran.radiances[track][xtrack][airs_chan[ichan]]
              = GSL_NAN;

    /* Copy data to struct... */
    for (track = 0; track < AIRS_RAD_GEOTRACK; track++)
      for (xtrack = 0; xtrack < AIRS_RAD_GEOXTRACK; xtrack++) {
        l1.time[track][xtrack]
          = airs_rad_gran.Time[track][xtrack] - 220838400.;
        l1.lon[track][xtrack]
          = airs_rad_gran.Longitude[track][xtrack];
        l1.lat[track][xtrack]
          = airs_rad_gran.Latitude[track][xtrack];
        l1.sat_z[track]
          = airs_rad_gran.satheight[track];
        l1.sat_lon[track]
          = airs_rad_gran.sat_lon[track];
        l1.sat_lat[track]
          = airs_rad_gran.sat_lat[track];
        for (ichan = 0; ichan < L1_NCHAN; ichan++) {
          l1.nu[ichan]
            = airs_rad_gran.nominal_freq[airs_chan[ichan]];
          l1.rad[track][xtrack][ichan]
            = airs_rad_gran.radiances[track][xtrack][airs_chan[ichan]] *
            0.001f;
        }
      }

```

```
    /* Write netCDF file... */
    write_l1(argv[3], &l1);
  }

  /* Check Level-2 filename... */
  if (argv[2][0] != '-') {

    /* Read data... */
    printf("Read AIRS Level-2 file: %s\n", argv[2]);
    airs_ret_rdr(argv[2], &airs_ret_gran);

    /* Flag bad data... */
    for (track = 0; track < AIRS_RET_GEOTRACK; track++)
      for (xtrack = 0; xtrack < AIRS_RET_GEOXTRACK; xtrack++)
        for (lay = 1; lay < AIRS_RET_STDPRESSURELAY; lay++)
          if (airs_ret_gran.GP_Height[track][xtrack][lay] <= -9000.
              || airs_ret_gran.TAirStd[track][xtrack][lay] <= -9000.) {
            airs_ret_gran.GP_Height[track][xtrack][lay] = GSL_NAN;
            airs_ret_gran.TAirStd[track][xtrack][lay] = GSL_NAN;
          }

    /* Save data in struct... */
    for (track = 0; track < AIRS_RET_GEOTRACK; track++)
      for (xtrack = 0; xtrack < AIRS_RET_GEOXTRACK; xtrack++)
        for (lay = 1; lay < AIRS_RET_STDPRESSURELAY; lay++) {
          l2.time[track][xtrack]
            = airs_ret_gran.Time[track][xtrack] - 220838400.;
          l2.z[track][xtrack][lay - 1]
            = airs_ret_gran.GP_Height[track][xtrack][lay] / 1000.;
          l2.lon[track][xtrack]
            = airs_ret_gran.Longitude[track][xtrack];
          l2.lat[track][xtrack]
            = airs_ret_gran.Latitude[track][xtrack];
          l2.p[lay - 1]
            = airs_ret_gran.pressStd[lay];
          l2.t[track][xtrack][lay - 1]
            = airs_ret_gran.TAirStd[track][xtrack][lay];
        }

    /* Convert geopotential heights to geometric heights... */
    for (track = 0; track < L2_NTRACK; track++)
      for (xtrack = 0; xtrack < L2_NXTRACK; xtrack++)
        for (lay = 0; lay < L2_NLAY; lay++)
          l2.z[track][xtrack][lay]
            = gph2z(l2.z[track][xtrack][lay], l2.lat[track][xtrack]);

    /* Write netCDF file... */
    write_l2(argv[3], &l2);
  }

  return EXIT_SUCCESS;
}
```

Here is the call graph for this function:

5.21.2 Variable Documentation

5.21.2.1 int airs_chan[L1_NCHAN]

Initial value:

```
= { 54, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82,
  2035, 2036, 2040, 2041, 2052, 2053, 2054, 2055,
  2067, 2075, 2076, 2077, 2078, 2079, 2080, 2081,
  2082, 2086, 2088, 2089, 2091, 2092, 2093
}
```

Definition at line 8 of file extract.c.

5.22 extract.c

```
#include "libairs.h"

/* ------------------------------------------------------------
   Global variables...
   ------------------------------------------------------------ */

/* List of AIRS channels (don't change). */
int airs_chan[L1_NCHAN] = { 54, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82,
  2035, 2036, 2040, 2041, 2052, 2053, 2054, 2055,
  2067, 2075, 2076, 2077, 2078, 2079, 2080, 2081,
  2082, 2086, 2088, 2089, 2091, 2092, 2093
};

/* ------------------------------------------------------------
   Functions...
   ------------------------------------------------------------ */

/* Convert geopotential height to geometric altitude. */
double gph2z(
  double gph,
  double lat);

/* ------------------------------------------------------------
   Main...
   ------------------------------------------------------------ */

int main(
  int argc,
  char *argv[]) {

  static airs_rad_gran_t airs_rad_gran;
  static airs_ret_gran_t airs_ret_gran;

  static airs_l1_t l1;
  static airs_l2_t l2;

  int ichan, lay, track, xtrack;

  /* Check arguments... */
  if (argc != 4)
    ERRMSG("Give parameters: <airs_l1_file> <airs_l2_file> <out.nc>");

  /* Check Level-1 filename... */
  if (argv[1][0] != '-') {

    /* Read data... */
    printf("Read AIRS Level-1 file: %s\n", argv[1]);
    airs_rad_rdr(argv[1], &airs_rad_gran);

    /* Flag bad data... */
    for (track = 0; track < AIRS_RAD_GEOTRACK; track++)
      for (xtrack = 0; xtrack < AIRS_RAD_GEOXTRACK; xtrack++)
        for (ichan = 0; ichan < L1_NCHAN; ichan++)
          if ((airs_rad_gran.state[track][xtrack] != 0)
              || (airs_rad_gran.ExcludedChans[airs_chan[ichan]] > 2)
              || (airs_rad_gran.CalChanSummary[airs_chan[ichan]] & 8)
              || (airs_rad_gran.CalChanSummary[airs_chan[ichan]] & (32 + 64))
              || (airs_rad_gran.CalFlag[track][airs_chan[ichan]] & 16))
            airs_rad_gran.radiances[track][xtrack][airs_chan[ichan]]
              = GSL_NAN;

    /* Copy data to struct... */
    for (track = 0; track < AIRS_RAD_GEOTRACK; track++)
      for (xtrack = 0; xtrack < AIRS_RAD_GEOXTRACK; xtrack++) {
        l1.time[track][xtrack]
          = airs_rad_gran.Time[track][xtrack] - 220838400.;
        l1.lon[track][xtrack]
          = airs_rad_gran.Longitude[track][xtrack];
        l1.lat[track][xtrack]
          = airs_rad_gran.Latitude[track][xtrack];
        l1.sat_z[track]
          = airs_rad_gran.satheight[track];
        l1.sat_lon[track]
          = airs_rad_gran.sat_lon[track];
        l1.sat_lat[track]
          = airs_rad_gran.sat_lat[track];
        for (ichan = 0; ichan < L1_NCHAN; ichan++) {
          l1.nu[ichan]
            = airs_rad_gran.nominal_freq[airs_chan[ichan]];
          l1.rad[track][xtrack][ichan]
            = airs_rad_gran.radiances[track][xtrack][airs_chan[ichan]] *
            0.001f;
        }
      }
```

```

    /* Write netCDF file... */
    write_l1(argv[3], &l1);
  }

  /* Check Level-2 filename... */
  if (argv[2][0] != '-') {

    /* Read data... */
    printf("Read AIRS Level-2 file: %s\n", argv[2]);
    airs_ret_rdr(argv[2], &airs_ret_gran);

    /* Flag bad data... */
    for (track = 0; track < AIRS_RET_GEOTRACK; track++)
      for (xtrack = 0; xtrack < AIRS_RET_GEOXTRACK; xtrack++)
        for (lay = 1; lay < AIRS_RET_STDPRESSURELAY; lay++)
          if (airs_ret_gran.GP_Height[track][xtrack][lay] <= -9000.
              || airs_ret_gran.TAirStd[track][xtrack][lay] <= -9000.) {
            airs_ret_gran.GP_Height[track][xtrack][lay] = GSL_NAN;
            airs_ret_gran.TAirStd[track][xtrack][lay] = GSL_NAN;
          }

    /* Save data in struct... */
    for (track = 0; track < AIRS_RET_GEOTRACK; track++)
      for (xtrack = 0; xtrack < AIRS_RET_GEOXTRACK; xtrack++)
        for (lay = 1; lay < AIRS_RET_STDPRESSURELAY; lay++) {
          l2.time[track][xtrack]
            = airs_ret_gran.Time[track][xtrack] - 220838400.;
          l2.z[track][xtrack][lay - 1]
            = airs_ret_gran.GP_Height[track][xtrack][lay] / 1000.;
          l2.lon[track][xtrack]
            = airs_ret_gran.Longitude[track][xtrack];
          l2.lat[track][xtrack]
            = airs_ret_gran.Latitude[track][xtrack];
          l2.p[lay - 1]
            = airs_ret_gran.pressStd[lay];
          l2.t[track][xtrack][lay - 1]
            = airs_ret_gran.TAirStd[track][xtrack][lay];
        }

    /* Convert geopotential heights to geometric heights... */
    for (track = 0; track < L2_NTRACK; track++)
      for (xtrack = 0; xtrack < L2_NXTRACK; xtrack++)
        for (lay = 0; lay < L2_NLAY; lay++)
          l2.z[track][xtrack][lay]
            = gph2z(l2.z[track][xtrack][lay], l2.lat[track][xtrack]);

    /* Write netCDF file... */
    write_l2(argv[3], &l2);
  }

  return EXIT_SUCCESS;
}

/*****************************************************************************/

double gph2z(
  double gph,
  double lat) {

  double a = 3.086e-3, g0 = gravity(0.0, 45.0), glat = gravity(0.0, lat);

  return glat / a - sqrt(gsl_pow_2(glat / a) - 2 * g0 * gph / a);
}
```

5.23 get_wave_pert.c File Reference

Functions

- int main (int argc, char ∗argv[ ])

5.23.1 Function Documentation

5.23.1.1 int main (int *argc,* char ∗ *argv[ ]*)

Definition at line 3 of file get_wave_pert.c.

```
                {

  static wave_t wave;
  static pert_t *pert;

  char method[LEN], pertname[LEN];

  double var_dh, Amax, phimax, lhmax, alphamax, betamax;

  int bg_poly_x, bg_poly_y, bg_smooth_x, bg_smooth_y, inter_x,
    dtrack, dxtrack, track0, xtrack0;

  /* Check arguments... */
  if (argc < 3)
    ERRMSG("Give parameters: <ctl> <pert.nc>");

  /* Get control parameters... */
  scan_ctl(argc, argv, "PERTNAME", -1, "4mu", pertname);
  track0 = (int) scan_ctl(argc, argv, "TRACK0", -1, "", NULL);
  xtrack0 = (int) scan_ctl(argc, argv, "XTRACK0", -1, "", NULL);
  dtrack = (int) scan_ctl(argc, argv, "DTRACK", -1, "20", NULL);
  dxtrack = (int) scan_ctl(argc, argv, "DXTRACK", -1, "20", NULL);
  inter_x = (int) scan_ctl(argc, argv, "INTER_X", -1, "0", NULL);
  bg_poly_x = (int) scan_ctl(argc, argv, "BG_POLY_X", -1, "5", NULL);
  bg_poly_y = (int) scan_ctl(argc, argv, "BG_POLY_Y", -1, "0", NULL);
  bg_smooth_x = (int) scan_ctl(argc, argv, "BG_SMOOTH_X", -1, "0", NULL);
  bg_smooth_y = (int) scan_ctl(argc, argv, "BG_SMOOTH_Y", -1, "7", NULL);
  var_dh = scan_ctl(argc, argv, "VAR_DH", -1, "100", NULL);
  scan_ctl(argc, argv, "METHOD", -1, "P", method);

  /* Allocate... */
  ALLOC(pert, pert_t, 1);

  /* Read perturbation data... */
  read_pert(argv[2], pertname, pert);

  /* Check indices... */
  if (track0 < 0 || track0 >= pert->ntrack)
    ERRMSG("Along-track index out of range!");
  if (xtrack0 < 0 || xtrack0 >= pert->nxtrack)
    ERRMSG("Across-track index out of range!");

  /* Convert to wave analysis struct... */
  pert2wave(pert, &wave,
            track0 - dtrack, track0 + dtrack,
            xtrack0 - dxtrack, xtrack0 + dxtrack);

  /* Interpolate to regular grid... */
  intpol_x(&wave, inter_x);

  /* Estimate background... */
  background_poly(&wave, bg_poly_x, bg_poly_y);
  background_smooth(&wave, bg_smooth_x, bg_smooth_y);

  /* Compute variance... */
  variance(&wave, var_dh);

  /* Get wave characteristics... */
  if (method[0] == 'p' || method[0] == 'P')
    period(&wave, &Amax, &phimax, &lhmax, &alphamax, &betamax, "period.tab");
  if (method[0] == 'f' || method[0] == 'F')
    fft(&wave, &Amax, &phimax, &lhmax, &alphamax, &betamax, "period.tab");

  /* Save wave struct... */
  write_wave("wave.tab", &wave);

  /* Write results... */
  PRINT("%g", Amax);
  PRINT("%g", phimax);
  PRINT("%g", lhmax);
  PRINT("%g", alphamax);
  PRINT("%g", betamax);

  /* Free... */
  free(pert);

  return EXIT_SUCCESS;
}
```

Here is the call graph for this function:

5.24 get_wave_pert.c

```
#include "libairs.h"
```

```

int main(
  int argc,
  char *argv[]) {

  static wave_t wave;
  static pert_t *pert;

  char method[LEN], pertname[LEN];

  double var_dh, Amax, phimax, lhmax, alphamax, betamax;

  int bg_poly_x, bg_poly_y, bg_smooth_x, bg_smooth_y, inter_x,
    dtrack, dxtrack, track0, xtrack0;

  /* Check arguments... */
  if (argc < 3)
    ERRMSG("Give parameters: <ctl> <pert.nc>");

  /* Get control parameters... */
  scan_ctl(argc, argv, "PERTNAME", -1, "4mu", pertname);
  track0 = (int) scan_ctl(argc, argv, "TRACK0", -1, "", NULL);
  xtrack0 = (int) scan_ctl(argc, argv, "XTRACK0", -1, "", NULL);
  dtrack = (int) scan_ctl(argc, argv, "DTRACK", -1, "20", NULL);
  dxtrack = (int) scan_ctl(argc, argv, "DXTRACK", -1, "20", NULL);
  inter_x = (int) scan_ctl(argc, argv, "INTER_X", -1, "0", NULL);
  bg_poly_x = (int) scan_ctl(argc, argv, "BG_POLY_X", -1, "5", NULL);
  bg_poly_y = (int) scan_ctl(argc, argv, "BG_POLY_Y", -1, "0", NULL);
  bg_smooth_x = (int) scan_ctl(argc, argv, "BG_SMOOTH_X", -1, "0", NULL);
  bg_smooth_y = (int) scan_ctl(argc, argv, "BG_SMOOTH_Y", -1, "7", NULL);
  var_dh = scan_ctl(argc, argv, "VAR_DH", -1, "100", NULL);
  scan_ctl(argc, argv, "METHOD", -1, "P", method);

  /* Allocate... */
  ALLOC(pert, pert_t, 1);

  /* Read perturbation data... */
  read_pert(argv[2], pertname, pert);

  /* Check indices... */
  if (track0 < 0 || track0 >= pert->ntrack)
    ERRMSG("Along-track index out of range!");
  if (xtrack0 < 0 || xtrack0 >= pert->nxtrack)
    ERRMSG("Across-track index out of range!");

  /* Convert to wave analysis struct... */
  pert2wave(pert, &wave,
            track0 - dtrack, track0 + dtrack,
            xtrack0 - dxtrack, xtrack0 + dxtrack);

  /* Interpolate to regular grid... */
  intpol_x(&wave, inter_x);

  /* Estimate background... */
  background_poly(&wave, bg_poly_x, bg_poly_y);
  background_smooth(&wave, bg_smooth_x, bg_smooth_y);

  /* Compute variance... */
  variance(&wave, var_dh);

  /* Get wave characteristics... */
  if (method[0] == 'p' || method[0] == 'P')
    period(&wave, &Amax, &phimax, &lhmax, &alphamax, &betamax, "period.tab");
  if (method[0] == 'f' || method[0] == 'F')
    fft(&wave, &Amax, &phimax, &lhmax, &alphamax, &betamax, "period.tab");

  /* Save wave struct... */
  write_wave("wave.tab", &wave);

  /* Write results... */
  PRINT("%g", Amax);
  PRINT("%g", phimax);
  PRINT("%g", lhmax);
  PRINT("%g", alphamax);
  PRINT("%g", betamax);

  /* Free... */
  free(pert);

  return EXIT_SUCCESS;
}
```

5.25 get_wave_synth.c File Reference

Functions

- int main (int argc, char ∗argv[ ])

5.25.1 Function Documentation

5.25.1.1 int main (int *argc,* char ∗ *argv[ ]*)

Definition at line 3 of file get_wave_synth.c.

```
                              {

  static wave_t wave;

  char method[LEN];

  double amp, dx, dy, lx, ly, phi, fwhm, var_dh,
    nedt, Amax, phimax, lhmax, alphamax, betamax;

  int bg_poly_x, bg_poly_y, bg_smooth_x, bg_smooth_y, inter_x, ix, iy, nx, ny;

  /* Check arguments... */
  if (argc < 2)
    ERRMSG("Give parameters: <ctl>");

  /* Get control parameters... */
  nx = (int) scan_ctl(argc, argv, "NX", -1, "90", NULL);
  ny = (int) scan_ctl(argc, argv, "NY", -1, "135", NULL);
  dx = scan_ctl(argc, argv, "DX", -1, "18", NULL);
  dy = scan_ctl(argc, argv, "DY", -1, "18", NULL);
  amp = scan_ctl(argc, argv, "AMP", -1, "1", NULL);
  phi = scan_ctl(argc, argv, "PHI", -1, "0", NULL);
  lx = scan_ctl(argc, argv, "LX", -1, "0", NULL);
  ly = scan_ctl(argc, argv, "LY", -1, "0", NULL);
  fwhm = scan_ctl(argc, argv, "FWHM", -1, "0", NULL);
  nedt = scan_ctl(argc, argv, "NOISE", -1, "0", NULL);
  inter_x = (int) scan_ctl(argc, argv, "INTER_X", -1, "0", NULL);
  bg_poly_x = (int) scan_ctl(argc, argv, "BG_POLY_X", -1, "5", NULL);
  bg_poly_y = (int) scan_ctl(argc, argv, "BG_POLY_Y", -1, "0", NULL);
  bg_smooth_x = (int) scan_ctl(argc, argv, "BG_SMOOTH_X", -1, "0", NULL);
  bg_smooth_y = (int) scan_ctl(argc, argv, "BG_SMOOTH_Y", -1, "7", NULL);
  var_dh = scan_ctl(argc, argv, "VAR_DH", -1, "100", NULL);
  scan_ctl(argc, argv, "METHOD", -1, "P", method);

  /* Set grid... */
  wave.nx = nx;
  wave.ny = ny;
  for (ix = 0; ix < nx; ix++)
    wave.x[ix] = (ix - nx / 2) * dx;
  for (iy = 0; iy < ny; iy++)
    wave.y[iy] = (iy - ny / 2) * dy;

  /* Init wave... */
  create_background(&wave);
  create_wave(&wave, amp, lx, ly, phi, fwhm);
  create_noise(&wave, nedt);

  /* Interpolate to regular grid... */
  intpol_x(&wave, inter_x);

  /* Estimate background... */
  background_poly(&wave, bg_poly_x, bg_poly_y);
  background_smooth(&wave, bg_smooth_x, bg_smooth_y);

  /* Compute variance... */
  variance(&wave, var_dh);

  /* Get wave characteristics... */
  if (method[0] == 'p' || method[0] == 'P')
    period(&wave, &Amax, &phimax, &lhmax, &alphamax, &betamax, "period.tab");
  if (method[0] == 'f' || method[0] == 'F')
    fft(&wave, &Amax, &phimax, &lhmax, &alphamax, &betamax, "period.tab");

  /* Save wave struct... */
  write_wave("wave.tab", &wave);

  /* Write results... */
  PRINT("%g", Amax);
```

```
  PRINT("%g", phimax);
  PRINT("%g", lhmax);
  PRINT("%g", alphamax);
  PRINT("%g", betamax);

  return EXIT_SUCCESS;
}
```

Here is the call graph for this function:

5.26 get_wave_synth.c

```
#include "libairs.h"

int main(
  int argc,
  char *argv[]) {

  static wave_t wave;

  char method[LEN];

  double amp, dx, dy, lx, ly, phi, fwhm, var_dh,
    nedt, Amax, phimax, lhmax, alphamax, betamax;

  int bg_poly_x, bg_poly_y, bg_smooth_x, bg_smooth_y, inter_x, ix, iy, nx, ny;

  /* Check arguments... */
  if (argc < 2)
    ERRMSG("Give parameters: <ctl>");

  /* Get control parameters... */
  nx = (int) scan_ctl(argc, argv, "NX", -1, "90", NULL);
  ny = (int) scan_ctl(argc, argv, "NY", -1, "135", NULL);
  dx = scan_ctl(argc, argv, "DX", -1, "18", NULL);
  dy = scan_ctl(argc, argv, "DY", -1, "18", NULL);
  amp = scan_ctl(argc, argv, "AMP", -1, "1", NULL);
  phi = scan_ctl(argc, argv, "PHI", -1, "0", NULL);
  lx = scan_ctl(argc, argv, "LX", -1, "0", NULL);
  ly = scan_ctl(argc, argv, "LY", -1, "0", NULL);
  fwhm = scan_ctl(argc, argv, "FWHM", -1, "0", NULL);
  nedt = scan_ctl(argc, argv, "NOISE", -1, "0", NULL);
  inter_x = (int) scan_ctl(argc, argv, "INTER_X", -1, "0", NULL);
  bg_poly_x = (int) scan_ctl(argc, argv, "BG_POLY_X", -1, "5", NULL);
  bg_poly_y = (int) scan_ctl(argc, argv, "BG_POLY_Y", -1, "0", NULL);
  bg_smooth_x = (int) scan_ctl(argc, argv, "BG_SMOOTH_X", -1, "0", NULL);
  bg_smooth_y = (int) scan_ctl(argc, argv, "BG_SMOOTH_Y", -1, "7", NULL);
  var_dh = scan_ctl(argc, argv, "VAR_DH", -1, "100", NULL);
  scan_ctl(argc, argv, "METHOD", -1, "P", method);

  /* Set grid... */
  wave.nx = nx;
  wave.ny = ny;
  for (ix = 0; ix < nx; ix++)
    wave.x[ix] = (ix - nx / 2) * dx;
  for (iy = 0; iy < ny; iy++)
    wave.y[iy] = (iy - ny / 2) * dy;

  /* Init wave... */
  create_background(&wave);
  create_wave(&wave, amp, lx, ly, phi, fwhm);
  create_noise(&wave, nedt);

  /* Interpolate to regular grid... */
  intpol_x(&wave, inter_x);

  /* Estimate background... */
  background_poly(&wave, bg_poly_x, bg_poly_y);
  background_smooth(&wave, bg_smooth_x, bg_smooth_y);

  /* Compute variance... */
  variance(&wave, var_dh);

  /* Get wave characteristics... */
  if (method[0] == 'p' || method[0] == 'P')
    period(&wave, &Amax, &phimax, &lhmax, &alphamax, &betamax, "period.tab");
  if (method[0] == 'f' || method[0] == 'F')
    fft(&wave, &Amax, &phimax, &lhmax, &alphamax, &betamax, "period.tab");

  /* Save wave struct... */
  write_wave("wave.tab", &wave);
```

```

  /* Write results... */
  PRINT("%g", Amax);
  PRINT("%g", phimax);
  PRINT("%g", lhmax);
  PRINT("%g", alphamax);
  PRINT("%g", betamax);

  return EXIT_SUCCESS;
}
```

5.27 hurricane.c File Reference

Functions

- int get_storm_pos (int nobs, double time_wmo[NTIME], double lon_wmo[NTIME], double lat_wmo[NTIME], double wind_wmo[NTIME], double pres_wmo[NTIME], double t, int dt, int st, double x[3], double ∗wind, double ∗dwind, double ∗pres, double ∗dpres)
- void read_var (int ncid, const char varname[ ], size_t nstorm, int nobs[NSTORM], double x[NSTORM][NTI↩ME])
- int main (int argc, char ∗argv[ ])

5.27.1 Function Documentation

5.27.1.1 int get_storm_pos (int *nobs*, double *time_wmo[NTIME]*, double *lon_wmo[NTIME]*, double *lat_wmo[NTIME]*, double *wind_wmo[NTIME]*, double *pres_wmo[NTIME]*, double *t*, int *dt*, int *st*, double *x[3]*, double ∗ *wind*, double ∗ *dwind*, double ∗ *pres*, double ∗ *dpres*)

Definition at line 341 of file hurricane.c.

```
                 {

  double w, x0[3], x1[3];

  int i;

  /* Check time range... */
  if (t < time_wmo[0] || t > time_wmo[nobs - 1])
    return 0;

  /* Interpolate position... */
  i = locate(time_wmo, nobs, t);
  w = (t - time_wmo[i]) / (time_wmo[i + 1] - time_wmo[i]);
  geo2cart(0, lon_wmo[i], lat_wmo[i], x0);
  geo2cart(0, lon_wmo[i + 1], lat_wmo[i + 1], x1);
  x[0] = (1 - w) * x0[0] + w * x1[0];
  x[1] = (1 - w) * x0[1] + w * x1[1];
  x[2] = (1 - w) * x0[2] + w * x1[2];

  /* Interpolate wind and pressure... */
  *pres = (1 - w) * pres_wmo[i] + w * pres_wmo[i + 1];
  *wind = (1 - w) * wind_wmo[i] + w * wind_wmo[i + 1];

  /* Get pressure and wind change... */
  *dpres = (pres_wmo[i + 1 + st] - pres_wmo[GSL_MAX(i - dt + st, 0)])
    / (time_wmo[i + 1 + st] - time_wmo[GSL_MAX(i - dt + st, 0)]) * 3600.;
  *dwind = (wind_wmo[i + 1 + st] - wind_wmo[GSL_MAX(i - dt + st, 0)])
    / (time_wmo[i + 1 + st] - time_wmo[GSL_MAX(i - dt + st, 0)]) * 3600.;

  return 1;
}
```

Here is the call graph for this function:

5.27.1.2 void read_var (int *ncid*, const char *varname[ ]*, size_t *nstorm*, int *nobs[NSTORM]*, double *x[NSTORM][NTIME]*)

Definition at line 389 of file hurricane.c.

```
                            {

  int varid;

  size_t count[2], istorm, start[2];

  /* Read pressure... */
  NC(nc_inq_varid(ncid, varname, &varid));
  for (istorm = 0; istorm < nstorm; istorm++) {
    start[0] = istorm;
    start[1] = 0;
    count[0] = 1;
    count[1] = (size_t) nobs[istorm];
    NC(nc_get_vara_double(ncid, varid, start, count, x[istorm]));
  }
}
```

5.27.1.3 int main (int *argc*, char ∗ *argv[ ]*)

Definition at line 46 of file hurricane.c.

```
                 {

  static pert_t *pert;

  static FILE *in, *out;

  static char filter[LEN], pertname[LEN], set[LEN];

  static double bt4_mean, bt4_var, bt8_min, dpres, dpresbest, dt230, dwind,
    dwindbest, lat_wmo[NSTORM][NTIME], latbest, lon_wmo[NSTORM][NTIME],
    lonbest, lonsat, lonstorm, nedt, nesr, nu, pmin, pres_wmo[NSTORM][NTIME],
    pres, presbest, r2, r2best = 1e100, rmax, wind_wmo[NSTORM][NTIME], wind,
    windbest, wmax, time_max_pres[NSTORM], time_max_wind[NSTORM],
    time_wmo[NSTORM][NTIME], timebest, xf[PERT_NTRACK][PERT_NXTRACK][3],
    xs[3], z;

  static int asc, dimid, dt, iarg, iobs, itrack, itrack2, ixtrack2, n,
    ncid, nobs[NSTORM], st, varid;

  static size_t istorm, nstorm, ntime;

  /* Check arguments... */
  if (argc < 5)
    ERRMSG("Give parameters: <ctl> <hurr.tab> <ibtracs.nc>"
           " <pert1.nc> [<pert2.nc> ...]");

  /* Get control parameters... */
  scan_ctl(argc, argv, "SET", -1, "full", set);
  scan_ctl(argc, argv, "PERTNAME", -1, "4mu", pertname);
  scan_ctl(argc, argv, "FILTER", -1, "both", filter);
  dt230 = scan_ctl(argc, argv, "DT230", -1, "0.16", NULL);
  nu = scan_ctl(argc, argv, "NU", -1, "2345.0", NULL);
  rmax = scan_ctl(argc, argv, "RMAX", -1, "500", NULL);
  dt = (int) scan_ctl(argc, argv, "DT", -1, "0", NULL);
  st = (int) scan_ctl(argc, argv, "ST", -1, "0", NULL);

  /* Allocate... */
  ALLOC(pert, pert_t, 1);

  /* ------------------------------------------------------------
     Read hurricane tracks...
     ------------------------------------------------------------ */

  /* Write info... */
  printf("Read hurricane tracks: %s\n", argv[3]);

  /* Open netCDF file... */
  NC(nc_open(argv[3], NC_NOWRITE, &ncid));

  /* Get dimensions... */
  NC(nc_inq_dimid(ncid, "storm", &dimid));
  NC(nc_inq_dimlen(ncid, dimid, &nstorm));
  NC(nc_inq_dimid(ncid, "time", &dimid));
```

```
  NC(nc_inq_dimlen(ncid, dimid, &ntime));
  if (nstorm > NSTORM)
    ERRMSG("Too many storms!");
  if (ntime > NTIME)
    ERRMSG("Too many time steps!");

  /* Read number of observations per storm... */
  NC(nc_inq_varid(ncid, "numObs", &varid));
  NC(nc_get_var_int(ncid, varid, nobs));

  /* Read data... */
  read_var(ncid, "lat_wmo", nstorm, nobs, lat_wmo);
  read_var(ncid, "lon_wmo", nstorm, nobs, lon_wmo);
  read_var(ncid, "time_wmo", nstorm, nobs, time_wmo);
  read_var(ncid, "wind_wmo", nstorm, nobs, wind_wmo);
  read_var(ncid, "pres_wmo", nstorm, nobs, pres_wmo);

  /* Convert units.. */
  for (istorm = 0; istorm < nstorm; istorm++)
    for (iobs = 0; iobs < nobs[istorm]; iobs++) {
      time_wmo[istorm][iobs] *= 86400.;
      time_wmo[istorm][iobs] -= 4453401600.00;
      lon_wmo[istorm][iobs] *= 0.01;
      lat_wmo[istorm][iobs] *= 0.01;
      wind_wmo[istorm][iobs] *= 0.0514444;
      pres_wmo[istorm][iobs] *= 0.1;
    }

  /* Check data... */
  for (istorm = 0; istorm < nstorm; istorm++)
    for (iobs = 0; iobs < nobs[istorm]; iobs++) {
      if (pres_wmo[istorm][iobs] <= 800 || pres_wmo[istorm][iobs] >= 1200)
        pres_wmo[istorm][iobs] = GSL_NAN;
      if (wind_wmo[istorm][iobs] <= 0.1)
        wind_wmo[istorm][iobs] = GSL_NAN;
    }

  /* Find time of maximum intensity (lowest pressure)... */
  for (istorm = 0; istorm < nstorm; istorm++) {
    pmin = 1e100;
    time_max_pres[istorm] = GSL_NAN;
    for (iobs = 0; iobs < nobs[istorm]; iobs++)
      if (gsl_finite(pres_wmo[istorm][iobs]) && pres_wmo[istorm][iobs] < pmin) {
        pmin = pres_wmo[istorm][iobs];
        time_max_pres[istorm] = time_wmo[istorm][iobs];
      }
  }

  /* Find time of maximum intensity (maximum wind)... */
  for (istorm = 0; istorm < nstorm; istorm++) {
    wmax = -1e100;
    time_max_wind[istorm] = GSL_NAN;
    for (iobs = 0; iobs < nobs[istorm]; iobs++)
      if (gsl_finite(wind_wmo[istorm][iobs]) && wind_wmo[istorm][iobs] > wmax) {
        wmax = wind_wmo[istorm][iobs];
        time_max_wind[istorm] = time_wmo[istorm][iobs];
      }
  }

  /* Close netCDF file... */
  NC(nc_close(ncid));

  /* ------------------------------------------------------------
     Analyze AIRS data...
     ------------------------------------------------------------ */

  /* Create file... */
  printf("Write hurricane data: %s\n", argv[2]);
  if (!(out = fopen(argv[2], "w")))
    ERRMSG("Cannot create file!");

  /* Write header... */
  fprintf(out,
          "# $1  = storm number\n"
          "# $2  = storm time since first report [hr]\n"
          "# $3  = storm time since wind maximum [hr]\n"
          "# $4  = storm time since pressure minimum [hr]\n"
          "# $5  = match time [s]\n"
          "# $6  = match longitude [deg]\n"
          "# $7  = match latitude [deg]\n"
          "# $8  = match distance [km]\n"
          "# $9  = wind speed [m/s]\n"
          "# $10 = wind speed change [m/s/hr]\n");
  fprintf(out,
          "# $11 = pressure [hPa]\n"
          "# $12 = pressure change [hPa/hr]\n"
          "# $13 = 8.1 micron BT minimum [K]\n"
```

```
        "# $14 = 4.3 micron BT variance [K^2]\n"
        "# $15 = 4.3 micron BT variance (noise-corrected) [K^2]\n"
        "# $16 = number of footprints\n\n");

/* Loop over perturbation files... */
for (iarg = 4; iarg < argc; iarg++) {

  /* Read perturbation data... */
  if (!(in = fopen(argv[iarg], "r")))
    continue;
  else {
    fclose(in);
    read_pert(argv[iarg], pertname, pert);
  }

  /* Get Cartesian coordinates... */
  for (itrack2 = 0; itrack2 < pert->ntrack; itrack2++)
    for (ixtrack2 = 0; ixtrack2 < pert->nxtrack; ixtrack2++)
      geo2cart(0, pert->lon[itrack2][ixtrack2],
               pert->lat[itrack2][ixtrack2], xf[itrack2][ixtrack2]);

  /* Loop over storms... */
  for (istorm = 0; istorm < nstorm; istorm++) {

    /* Loop along AIRS center track... */
    for (itrack = 0; itrack < pert->ntrack; itrack++) {

      /* Get storm position... */
      if (get_storm_pos(nobs[istorm], time_wmo[istorm], lon_wmo[istorm],
                        lat_wmo[istorm], wind_wmo[istorm], pres_wmo[istorm],
                        pert->time[itrack][pert->nxtrack / 2], dt, st, xs,
                        &wind, &dwind, &pres, &dpres)) {

        /* Get distance... */
        r2 = DIST2(xs, xf[itrack][pert->nxtrack / 2]);

        /* Find best match... */
        if (r2 < r2best) {

          /* Save position... */
          r2best = r2;
          timebest = pert->time[itrack][pert->nxtrack / 2];
          cart2geo(xs, &z, &lonbest, &latbest);

          /* Save wind... */
          windbest = wind;
          dwindbest = dwind;
          presbest = pres;
          dpresbest = dpres;

          /* Get BT data... */
          n = 0;
          bt8_min = 1e100;
          bt4_mean = 0;
          bt4_var = 0;
          for (itrack2 = GSL_MAX(itrack - ((int) (rmax / 17) + 1), 0);
               itrack2 <= GSL_MIN(itrack + ((int) (rmax / 17) + 1),
                                  pert->ntrack - 1); itrack2++)
            for (ixtrack2 = 0; ixtrack2 < pert->nxtrack; ixtrack2++) {

              /* Check data... */
              if (pert->time[itrack2][ixtrack2] < 0
                  || pert->lon[itrack2][ixtrack2] < -180
                  || pert->lon[itrack2][ixtrack2] > 180
                  || pert->lat[itrack2][ixtrack2] < -90
                  || pert->lat[itrack2][ixtrack2] > 90
                  || pert->pt[itrack2][ixtrack2] < -100
                  || pert->pt[itrack2][ixtrack2] > 100
                  || !gsl_finite(pert->bt[itrack2][ixtrack2])
                  || !gsl_finite(pert->pt[itrack2][ixtrack2])
                  || !gsl_finite(pert->var[itrack2][ixtrack2])
                  || !gsl_finite(pert->dc[itrack2][ixtrack2]))
                continue;

              /* Check east/west filter... */
              lonsat = pert->lon[itrack2][ixtrack2];
              while (lonsat < 20)
                lonsat += 360;
              lonstorm = lonbest;
              while (lonstorm < 20)
                lonstorm += 360;
              if ((filter[0] == 'e' || filter[0] == 'E')
                  && lonsat < lonstorm)
                continue;
              if ((filter[0] == 'w' || filter[0] == 'W')
                  && lonsat > lonstorm)
                continue;
```

```

                /* Get distance... */
                if (DIST2(xs, xf[itrack2][ixtrack2]) < rmax * rmax) {
                  bt8_min = GSL_MIN(bt8_min, pert->dc[itrack2][ixtrack2]);
                  bt4_mean += pert->bt[itrack2][ixtrack2];
                  bt4_var += gsl_pow_2(pert->pt[itrack2][ixtrack2]);
                  n++;
                }
              }
          }
        }

        /* Output over poles... */
        if (fabs(pert->lat[itrack][pert->nxtrack / 2]) > 80.) {

          /* Get and check ascending/descending flag... */
          asc =
            (pert->lat[itrack > 0 ? itrack : itrack + 1][pert->nxtrack / 2]
             > pert->lat[itrack >
                         0 ? itrack - 1 : itrack][pert->nxtrack / 2]);
          if ((set[0] == 'f' || set[0] == 'F')
              || ((set[0] == 'a' || set[0] == 'A') && asc)
              || ((set[0] == 'd' || set[0] == 'D') && !asc)) {

            /* Check for match... */
            if (r2best < 890. * 890.) {

              /* Estimate noise... */
              if (dt230 > 0) {
                nesr = planck(230.0 + dt230, nu) - planck(230.0, nu);
                nedt =
                  brightness(planck(bt4_mean / n, nu) + nesr,
                             nu) - bt4_mean / n;
              }

              /* Write output... */
              if (n > 0)
                fprintf(out,
                        "%lu %g %g %g %.2f %g %g %g %g %g %g %g %g %g %g %d\n",
                        istorm, (timebest - time_wmo[istorm][0]) / 3600.,
                        (timebest - time_max_wind[istorm]) / 3600.,
                        (timebest - time_max_pres[istorm]) / 3600.,
                        timebest, lonbest, latbest, sqrt(r2best), windbest,
                        dwindbest, presbest, dpresbest, bt8_min, bt4_var / n,
                        bt4_var / n - gsl_pow_2(nedt), n);
            }
          }

          /* Reset... */
          r2best = 1e100;
        }
      }
    }
  }

  /* Close file... */
  fclose(out);

  /* Free... */
  free(pert);

  return EXIT_SUCCESS;
}
```

Here is the call graph for this function:

5.28 hurricane.c

```
#include "libairs.h"

/* ------------------------------------------------------------
   Dimensions...
   ------------------------------------------------------------ */

/* Maximum number of storms. */
#define NSTORM 9000

/* Maximum number of observation times. */
#define NTIME 140

/* ------------------------------------------------------------
```

```
   Functions...
   ------------------------------------------------------------ */

/* Get storm position at given time... */
int get_storm_pos(
  int nobs,
  double time_wmo[NTIME],
  double lon_wmo[NTIME],
  double lat_wmo[NTIME],
  double wind_wmo[NTIME],
  double pres_wmo[NTIME],
  double t,
  int dt,
  int st,
  double x[3],
  double *wind,
  double *dwind,
  double *pres,
  double *dpres);

/* Read variable from netCDF file... */
void read_var(
  int ncid,
  const char varname[],
  size_t nstorm,
  int nobs[NSTORM],
  double x[NSTORM][NTIME]);

/* ------------------------------------------------------------
   Main...
   ------------------------------------------------------------ */

int main(
  int argc,
  char *argv[]) {

  static pert_t *pert;

  static FILE *in, *out;

  static char filter[LEN], pertname[LEN], set[LEN];

  static double bt4_mean, bt4_var, bt8_min, dpres, dpresbest, dt230, dwind,
    dwindbest, lat_wmo[NSTORM][NTIME], latbest, lon_wmo[NSTORM][NTIME],
    lonbest, lonsat, lonstorm, nedt, nesr, nu, pmin, pres_wmo[NSTORM][NTIME],
    pres, presbest, r2, r2best = 1e100, rmax, wind_wmo[NSTORM][NTIME], wind,
    windbest, wmax, time_max_pres[NSTORM], time_max_wind[NSTORM],
    time_wmo[NSTORM][NTIME], timebest, xf[PERT_NTRACK][PERT_NXTRACK][3],
    xs[3], z;

  static int asc, dimid, dt, iarg, iobs, itrack, itrack2, ixtrack2, n,
    ncid, nobs[NSTORM], st, varid;

  static size_t istorm, nstorm, ntime;

  /* Check arguments... */
  if (argc < 5)
    ERRMSG("Give parameters: <ctl> <hurr.tab> <ibtracs.nc>"
           " <pert1.nc> [<pert2.nc> ...]");

  /* Get control parameters... */
  scan_ctl(argc, argv, "SET", -1, "full", set);
  scan_ctl(argc, argv, "PERTNAME", -1, "4mu", pertname);
  scan_ctl(argc, argv, "FILTER", -1, "both", filter);
  dt230 = scan_ctl(argc, argv, "DT230", -1, "0.16", NULL);
  nu = scan_ctl(argc, argv, "NU", -1, "2345.0", NULL);
  rmax = scan_ctl(argc, argv, "RMAX", -1, "500", NULL);
  dt = (int) scan_ctl(argc, argv, "DT", -1, "0", NULL);
  st = (int) scan_ctl(argc, argv, "ST", -1, "0", NULL);

  /* Allocate... */
  ALLOC(pert, pert_t, 1);

  /* ------------------------------------------------------------
     Read hurricane tracks...
     ------------------------------------------------------------ */

  /* Write info... */
  printf("Read hurricane tracks: %s\n", argv[3]);

  /* Open netCDF file... */
  NC(nc_open(argv[3], NC_NOWRITE, &ncid));

  /* Get dimensions... */
  NC(nc_inq_dimid(ncid, "storm", &dimid));
  NC(nc_inq_dimlen(ncid, dimid, &nstorm));
  NC(nc_inq_dimid(ncid, "time", &dimid));
```

```
  NC(nc_inq_dimlen(ncid, dimid, &ntime));
  if (nstorm > NSTORM)
    ERRMSG("Too many storms!");
  if (ntime > NTIME)
    ERRMSG("Too many time steps!");

  /* Read number of observations per storm... */
  NC(nc_inq_varid(ncid, "numObs", &varid));
  NC(nc_get_var_int(ncid, varid, nobs));

  /* Read data... */
  read_var(ncid, "lat_wmo", nstorm, nobs, lat_wmo);
  read_var(ncid, "lon_wmo", nstorm, nobs, lon_wmo);
  read_var(ncid, "time_wmo", nstorm, nobs, time_wmo);
  read_var(ncid, "wind_wmo", nstorm, nobs, wind_wmo);
  read_var(ncid, "pres_wmo", nstorm, nobs, pres_wmo);

  /* Convert units.. */
  for (istorm = 0; istorm < nstorm; istorm++)
    for (iobs = 0; iobs < nobs[istorm]; iobs++) {
      time_wmo[istorm][iobs] *= 86400.;
      time_wmo[istorm][iobs] -= 4453401600.00;
      lon_wmo[istorm][iobs] *= 0.01;
      lat_wmo[istorm][iobs] *= 0.01;
      wind_wmo[istorm][iobs] *= 0.0514444;
      pres_wmo[istorm][iobs] *= 0.1;
    }

  /* Check data... */
  for (istorm = 0; istorm < nstorm; istorm++)
    for (iobs = 0; iobs < nobs[istorm]; iobs++) {
      if (pres_wmo[istorm][iobs] <= 800 || pres_wmo[istorm][iobs] >= 1200)
        pres_wmo[istorm][iobs] = GSL_NAN;
      if (wind_wmo[istorm][iobs] <= 0.1)
        wind_wmo[istorm][iobs] = GSL_NAN;
    }

  /* Find time of maximum intensity (lowest pressure)... */
  for (istorm = 0; istorm < nstorm; istorm++) {
    pmin = 1e100;
    time_max_pres[istorm] = GSL_NAN;
    for (iobs = 0; iobs < nobs[istorm]; iobs++)
      if (gsl_finite(pres_wmo[istorm][iobs]) && pres_wmo[istorm][iobs] < pmin) {
        pmin = pres_wmo[istorm][iobs];
        time_max_pres[istorm] = time_wmo[istorm][iobs];
      }
  }

  /* Find time of maximum intensity (maximum wind)... */
  for (istorm = 0; istorm < nstorm; istorm++) {
    wmax = -1e100;
    time_max_wind[istorm] = GSL_NAN;
    for (iobs = 0; iobs < nobs[istorm]; iobs++)
      if (gsl_finite(wind_wmo[istorm][iobs]) && wind_wmo[istorm][iobs] > wmax) {
        wmax = wind_wmo[istorm][iobs];
        time_max_wind[istorm] = time_wmo[istorm][iobs];
      }
  }

  /* Close netCDF file... */
  NC(nc_close(ncid));

  /* ------------------------------------------------------------
     Analyze AIRS data...
     ------------------------------------------------------------ */

  /* Create file... */
  printf("Write hurricane data: %s\n", argv[2]);
  if (!(out = fopen(argv[2], "w")))
    ERRMSG("Cannot create file!");

  /* Write header... */
  fprintf(out,
          "# $1  = storm number\n"
          "# $2  = storm time since first report [hr]\n"
          "# $3  = storm time since wind maximum [hr]\n"
          "# $4  = storm time since pressure minimum [hr]\n"
          "# $5  = match time [s]\n"
          "# $6  = match longitude [deg]\n"
          "# $7  = match latitude [deg]\n"
          "# $8  = match distance [km]\n"
          "# $9  = wind speed [m/s]\n"
          "# $10 = wind speed change [m/s/hr]\n");
  fprintf(out,
          "# $11 = pressure [hPa]\n"
          "# $12 = pressure change [hPa/hr]\n"
          "# $13 = 8.1 micron BT minimum [K]\n"
```

```
        "# $14 = 4.3 micron BT variance [K^2]\n"
        "# $15 = 4.3 micron BT variance (noise-corrected) [K^2]\n"
        "# $16 = number of footprints\n\n");

  /* Loop over perturbation files... */
  for (iarg = 4; iarg < argc; iarg++) {

    /* Read perturbation data... */
    if (!(in = fopen(argv[iarg], "r")))
      continue;
    else {
      fclose(in);
      read_pert(argv[iarg], pertname, pert);
    }

    /* Get Cartesian coordinates... */
    for (itrack2 = 0; itrack2 < pert->ntrack; itrack2++)
      for (ixtrack2 = 0; ixtrack2 < pert->nxtrack; ixtrack2++)
        geo2cart(0, pert->lon[itrack2][ixtrack2],
                 pert->lat[itrack2][ixtrack2], xf[itrack2][ixtrack2]);

    /* Loop over storms... */
    for (istorm = 0; istorm < nstorm; istorm++) {

      /* Loop along AIRS center track... */
      for (itrack = 0; itrack < pert->ntrack; itrack++) {

        /* Get storm position... */
        if (get_storm_pos(nobs[istorm], time_wmo[istorm], lon_wmo[istorm],
                          lat_wmo[istorm], wind_wmo[istorm], pres_wmo[istorm],
                          pert->time[itrack][pert->nxtrack / 2], dt, st, xs,
                          &wind, &dwind, &pres, &dpres)) {

          /* Get distance... */
          r2 = DIST2(xs, xf[itrack][pert->nxtrack / 2]);

          /* Find best match... */
          if (r2 < r2best) {

            /* Save position... */
            r2best = r2;
            timebest = pert->time[itrack][pert->nxtrack / 2];
            cart2geo(xs, &z, &lonbest, &latbest);

            /* Save wind... */
            windbest = wind;
            dwindbest = dwind;
            presbest = pres;
            dpresbest = dpres;

            /* Get BT data... */
            n = 0;
            bt8_min = 1e100;
            bt4_mean = 0;
            bt4_var = 0;
            for (itrack2 = GSL_MAX(itrack - ((int) (rmax / 17) + 1), 0);
                 itrack2 <= GSL_MIN(itrack + ((int) (rmax / 17) + 1),
                                    pert->ntrack - 1); itrack2++)
              for (ixtrack2 = 0; ixtrack2 < pert->nxtrack; ixtrack2++) {

                /* Check data... */
                if (pert->time[itrack2][ixtrack2] < 0
                    || pert->lon[itrack2][ixtrack2] < -180
                    || pert->lon[itrack2][ixtrack2] > 180
                    || pert->lat[itrack2][ixtrack2] < -90
                    || pert->lat[itrack2][ixtrack2] > 90
                    || pert->pt[itrack2][ixtrack2] < -100
                    || pert->pt[itrack2][ixtrack2] > 100
                    || !gsl_finite(pert->bt[itrack2][ixtrack2])
                    || !gsl_finite(pert->pt[itrack2][ixtrack2])
                    || !gsl_finite(pert->var[itrack2][ixtrack2])
                    || !gsl_finite(pert->dc[itrack2][ixtrack2]))
                  continue;

                /* Check east/west filter... */
                lonsat = pert->lon[itrack2][ixtrack2];
                while (lonsat < 20)
                  lonsat += 360;
                lonstorm = lonbest;
                while (lonstorm < 20)
                  lonstorm += 360;
                if ((filter[0] == 'e' || filter[0] == 'E')
                    && lonsat < lonstorm)
                  continue;
                if ((filter[0] == 'w' || filter[0] == 'W')
                    && lonsat > lonstorm)
                  continue;
```

```
                    /* Get distance... */
                    if (DIST2(xs, xf[itrack2][ixtrack2]) < rmax * rmax) {
                      bt8_min = GSL_MIN(bt8_min, pert->dc[itrack2][ixtrack2]);
                      bt4_mean += pert->bt[itrack2][ixtrack2];
                      bt4_var += gsl_pow_2(pert->pt[itrack2][ixtrack2]);
                      n++;
                    }
                  }
              }
            }

            /* Output over poles... */
            if (fabs(pert->lat[itrack][pert->nxtrack / 2]) > 80.) {

              /* Get and check ascending/descending flag... */
              asc =
                (pert->lat[itrack > 0 ? itrack : itrack + 1][pert->nxtrack / 2]
                 > pert->lat[itrack >
                             0 ? itrack - 1 : itrack][pert->nxtrack / 2]);
              if ((set[0] == 'f' || set[0] == 'F')
                  || ((set[0] == 'a' || set[0] == 'A') && asc)
                  || ((set[0] == 'd' || set[0] == 'D') && !asc)) {

                /* Check for match... */
                if (r2best < 890. * 890.) {

                  /* Estimate noise... */
                  if (dt230 > 0) {
                    nesr = planck(230.0 + dt230, nu) - planck(230.0, nu);
                    nedt =
                      brightness(planck(bt4_mean / n, nu) + nesr,
                                 nu) - bt4_mean / n;
                  }

                  /* Write output... */
                  if (n > 0)
                    fprintf(out,
                            "%lu %g %g %g %.2f %g %g %g %g %g %g %g %g %g %g %d\n",
                            istorm, (timebest - time_wmo[istorm][0]) / 3600.,
                            (timebest - time_max_wind[istorm]) / 3600.,
                            (timebest - time_max_pres[istorm]) / 3600.,
                            timebest, lonbest, latbest, sqrt(r2best), windbest,
                            dwindbest, presbest, dpresbest, bt8_min, bt4_var / n,
                            bt4_var / n - gsl_pow_2(nedt), n);
                }
              }

              /* Reset... */
              r2best = 1e100;
            }
          }
        }
      }

      /* Close file... */
      fclose(out);

      /* Free... */
      free(pert);

      return EXIT_SUCCESS;
    }

    /*****************************************************************************/

    int get_storm_pos(
      int nobs,
      double time_wmo[NTIME],
      double lon_wmo[NTIME],
      double lat_wmo[NTIME],
      double wind_wmo[NTIME],
      double pres_wmo[NTIME],
      double t,
      int dt,
      int st,
      double x[3],
      double *wind,
      double *dwind,
      double *pres,
      double *dpres) {

      double w, x0[3], x1[3];

      int i;

      /* Check time range... */
```

```
  if (t < time_wmo[0] || t > time_wmo[nobs - 1])
    return 0;

  /* Interpolate position... */
  i = locate(time_wmo, nobs, t);
  w = (t - time_wmo[i]) / (time_wmo[i + 1] - time_wmo[i]);
  geo2cart(0, lon_wmo[i], lat_wmo[i], x0);
  geo2cart(0, lon_wmo[i + 1], lat_wmo[i + 1], x1);
  x[0] = (1 - w) * x0[0] + w * x1[0];
  x[1] = (1 - w) * x0[1] + w * x1[1];
  x[2] = (1 - w) * x0[2] + w * x1[2];

  /* Interpolate wind and pressure... */
  *pres = (1 - w) * pres_wmo[i] + w * pres_wmo[i + 1];
  *wind = (1 - w) * wind_wmo[i] + w * wind_wmo[i + 1];

  /* Get pressure and wind change... */
  *dpres = (pres_wmo[i + 1 + st] - pres_wmo[GSL_MAX(i - dt + st, 0)])
    / (time_wmo[i + 1 + st] - time_wmo[GSL_MAX(i - dt + st, 0)]) * 3600.;
  *dwind = (wind_wmo[i + 1 + st] - wind_wmo[GSL_MAX(i - dt + st, 0)])
    / (time_wmo[i + 1 + st] - time_wmo[GSL_MAX(i - dt + st, 0)]) * 3600.;

  return 1;
}

/*****************************************************************************/

void read_var(
  int ncid,
  const char varname[],
  size_t nstorm,
  int nobs[NSTORM],
  double x[NSTORM][NTIME]) {

  int varid;

  size_t count[2], istorm, start[2];

  /* Read pressure... */
  NC(nc_inq_varid(ncid, varname, &varid));
  for (istorm = 0; istorm < nstorm; istorm++) {
    start[0] = istorm;
    start[1] = 0;
    count[0] = 1;
    count[1] = (size_t) nobs[istorm];
    NC(nc_get_vara_double(ncid, varid, start, count, x[istorm]));
  }
}
```

5.29 island.c File Reference

Functions

- void addatt (int ncid, int varid, const char ∗unit, const char ∗long_name)
- int main (int argc, char ∗argv[ ])

5.29.1 Function Documentation

5.29.1.1 void addatt (int *ncid,* int *varid,* const char ∗ *unit,* const char ∗ *long_name*)

Definition at line 360 of file island.c.

```
                              {

  /* Set long name... */
  NC(nc_put_att_text(ncid, varid, "long_name", strlen(long_name), long_name));

  /* Set units... */
  NC(nc_put_att_text(ncid, varid, "units", strlen(unit), unit));
}
```

5.29.1.2 int main (int *argc*, char ∗ *argv[]*)

Definition at line 18 of file island.c.

```
                  {

  static pert_t *pert;

  static wave_t *wave;

  static FILE *in, *out;

  static char pertname[LEN], ncfile[LEN];

  static double gauss_fwhm, var_dh, orblat, lon0, lat0, dlon, dlat, offset,
    ebt, emu, enoise, evar, wbt, wmu, wnoise, wvar, etime, wtime,
    dt230, nu, nesr, aux;

  static int iarg, ix, iy, itrack, itrack2, ixtrack, bg_poly_x, bg_poly_y,
    bg_smooth_x, bg_smooth_y, orb, orb_old = -1, en, wn, ncid, dimid[2],
    time_varid, track_varid, np_east_varid, var_east_varid,
    np_west_varid, var_west_varid, year_varid, doy_varid,
    track, year, mon, day, doy, iaux;

  static size_t count[2] = { 1, 1 }, start[2];

  /* Check arguments... */
  if (argc < 4)
    ERRMSG("Give parameters: <ctl> <var.tab> <pert1.nc> [<pert2.nc> ...]");

  /* Get control parameters... */
  scan_ctl(argc, argv, "PERTNAME", -1, "4mu", pertname);
  lon0 = scan_ctl(argc, argv, "LON0", -1, "", NULL);
  lat0 = scan_ctl(argc, argv, "LAT0", -1, "", NULL);
  dlon = scan_ctl(argc, argv, "DLON", -1, "", NULL);
  dlat = scan_ctl(argc, argv, "DLAT", -1, "", NULL);
  offset = scan_ctl(argc, argv, "OFFSET", -1, "1", NULL);
  bg_poly_x = (int) scan_ctl(argc, argv, "BG_POLY_X", -1, "0", NULL);
  bg_poly_y = (int) scan_ctl(argc, argv, "BG_POLY_Y", -1, "0", NULL);
  bg_smooth_x = (int) scan_ctl(argc, argv, "BG_SMOOTH_X", -1, "0", NULL);
  bg_smooth_y = (int) scan_ctl(argc, argv, "BG_SMOOTH_Y", -1, "0", NULL);
  gauss_fwhm = scan_ctl(argc, argv, "GAUSS_FWHM", -1, "0", NULL);
  var_dh = scan_ctl(argc, argv, "VAR_DH", -1, "0", NULL);
  orblat = scan_ctl(argc, argv, "ORBLAT", -1, "0", NULL);
  dt230 = scan_ctl(argc, argv, "DT230", -1, "0.16", NULL);
  nu = scan_ctl(argc, argv, "NU", -1, "2345.0", NULL);
  scan_ctl(argc, argv, "NCFILE", -1, "-", ncfile);

  /* Allocate... */
  ALLOC(pert, pert_t, 1);

  /* Create file... */
  printf("Write variance statistics: %s\n", argv[2]);
  if (!(out = fopen(argv[2], "w")))
    ERRMSG("Cannot create file!");

  /* Write header... */
  fprintf(out,
          "# $1  = time [s]\n"
          "# $2  = orbit number\n"
          "# $3  = eastern box: number of footprints\n"
          "# $4  = eastern box: variance [K^2]\n"
          "# $5  = eastern box: mean background temperature [K]\n"
          "# $6  = eastern box: noise estimate [K]\n"
          "# $7  = western box: number of footprints\n"
          "# $8  = western box: variance [K^2]\n"
          "# $9  = western box: mean background temperature [K]\n"
          "# $10 = western box: noise estimate [K]\n\n");

  /* Create netCDF file... */
  if (ncfile[0] != '-') {

    /* Create file... */
    printf("Write variance statistics: %s\n", ncfile);
    NC(nc_create(ncfile, NC_CLOBBER, &ncid));

    /* Set dimensions... */
    NC(nc_def_dim(ncid, "NP", NC_UNLIMITED, &dimid[0]));

    /* Add attributes... */
    aux = lon0;
    nc_put_att_double(ncid, NC_GLOBAL, "box_east_lon0", NC_DOUBLE, 1, &aux);
    aux = lon0 + dlon;
    nc_put_att_double(ncid, NC_GLOBAL, "box_east_lon1", NC_DOUBLE, 1, &aux);
```

```
    aux = lat0 - 0.5 * dlat;
    nc_put_att_double(ncid, NC_GLOBAL, "box_east_lat0", NC_DOUBLE, 1, &aux);
    aux = lat0 + 0.5 * dlat;
    nc_put_att_double(ncid, NC_GLOBAL, "box_east_lat1", NC_DOUBLE, 1, &aux);
    aux = lon0 - dlon - offset;
    nc_put_att_double(ncid, NC_GLOBAL, "box_west_lon0", NC_DOUBLE, 1, &aux);
    aux = lon0 - offset;
    nc_put_att_double(ncid, NC_GLOBAL, "box_west_lon1", NC_DOUBLE, 1, &aux);
    aux = lat0 - 0.5 * dlat;
    nc_put_att_double(ncid, NC_GLOBAL, "box_west_lat0", NC_DOUBLE, 1, &aux);
    aux = lat0 + 0.5 * dlat;
    nc_put_att_double(ncid, NC_GLOBAL, "box_west_lat1", NC_DOUBLE, 1, &aux);

    /* Add variables... */
    NC(nc_def_var(ncid, "time", NC_DOUBLE, 1, dimid, &time_varid));
    addatt(ncid, time_varid, "s", "time (seconds since 2000-01-01T00:00Z)");
    NC(nc_def_var(ncid, "year", NC_INT, 1, dimid, &year_varid));
    addatt(ncid, year_varid, "1", "year");
    NC(nc_def_var(ncid, "doy", NC_INT, 1, dimid, &doy_varid));
    addatt(ncid, doy_varid, "1", "day of year");
    NC(nc_def_var(ncid, "track", NC_INT, 1, dimid, &track_varid));
    addatt(ncid, track_varid, "1", "along-track index");
    NC(nc_def_var(ncid, "var_east", NC_DOUBLE, 1, dimid, &var_east_varid));
    addatt(ncid, var_east_varid, "K^2", "BT variance (east)");
    NC(nc_def_var(ncid, "var_west", NC_DOUBLE, 1, dimid, &var_west_varid));
    addatt(ncid, var_west_varid, "K^2", "BT variance (west)");
    NC(nc_def_var(ncid, "np_east", NC_INT, 1, dimid, &np_east_varid));
    addatt(ncid, np_east_varid, "1", "number of footprints (east)");
    NC(nc_def_var(ncid, "np_west", NC_INT, 1, dimid, &np_west_varid));
    addatt(ncid, np_west_varid, "1", "number of footprints (west)");

    /* Leave define mode... */
    NC(nc_enddef(ncid));
  }

  /* Loop over perturbation files... */
  for (iarg = 3; iarg < argc; iarg++) {

    /* Check filename... */
    if (!strcmp(argv[iarg], ncfile))
      continue;

    /* Initialize... */
    orb = 0;

    /* Read perturbation data... */
    if (!(in = fopen(argv[iarg], "r")))
      continue;
    else {
      fclose(in);
      read_pert(argv[iarg], pertname, pert);
    }

    /* Recalculate background and perturbations... */
    if (bg_poly_x > 0 || bg_poly_y > 0 ||
        bg_smooth_x > 0 || bg_smooth_y > 0 || gauss_fwhm > 0 || var_dh > 0) {

      /* Allocate... */
      ALLOC(wave, wave_t, 1);

      /* Convert to wave analysis struct... */
      pert2wave(pert, wave, 0, pert->ntrack - 1, 0, pert->nxtrack - 1);

      /* Estimate background... */
      background_poly(wave, bg_poly_x, bg_poly_y);
      background_smooth(wave, bg_smooth_x, bg_smooth_y);

      /* Gaussian filter... */
      gauss(wave, gauss_fwhm);

      /* Compute variance... */
      variance(wave, var_dh);

      /* Copy data... */
      for (ix = 0; ix < wave->nx; ix++)
        for (iy = 0; iy < wave->ny; iy++) {
          pert->pt[iy][ix] = wave->pt[ix][iy];
          pert->var[iy][ix] = wave->var[ix][iy];
        }

      /* Free... */
      free(wave);
    }

    /* Detection... */
    for (itrack = 0; itrack < pert->ntrack; itrack++)
      for (ixtrack = 0; ixtrack < pert->nxtrack; ixtrack++) {
```

```
      /* Check data... */
      if (pert->time[itrack][ixtrack] < 0
          || pert->lon[itrack][ixtrack] < -180
          || pert->lon[itrack][ixtrack] > 180
          || pert->lat[itrack][ixtrack] < -90
          || pert->lat[itrack][ixtrack] > 90
          || pert->pt[itrack][ixtrack] < -100
          || pert->pt[itrack][ixtrack] > 100
          || !gsl_finite(pert->bt[itrack][ixtrack])
          || !gsl_finite(pert->pt[itrack][ixtrack])
          || !gsl_finite(pert->var[itrack][ixtrack])
          || !gsl_finite(pert->dc[itrack][ixtrack]))
        continue;

      /* Count orbits... */
      if (itrack > 0 && ixtrack == pert->nxtrack / 2)
        if (pert->lat[itrack - 1][ixtrack] <= orblat
            && pert->lat[itrack][ixtrack] >= orblat)
          orb++;
      if (orb != orb_old) {

        /* Set orbit index... */
        orb_old = orb;

        /* Write output... */
        if (en > 0 && wn > 0) {

          /* Estimate noise... */
          if (dt230 > 0) {
            nesr = planck(230.0 + dt230, nu) - planck(230.0, nu);
            enoise = brightness(planck(ebt / en, nu) + nesr, nu) - ebt / en;
            wnoise = brightness(planck(wbt / wn, nu) + nesr, nu) - wbt / wn;
          }

          /* Write output... */
          fprintf(out, "%.2f %d %d %g %g %g %d %g %g %g\n", etime / en, orb,
                  en, evar / en - gsl_pow_2(emu / en), ebt / en, enoise,
                  wn, wvar / wn - gsl_pow_2(wmu / wn), wbt / wn, wnoise);

          /* Write to netCDF file... */
          if (ncfile[0] != '-') {

            /* Get year and doy... */
            jsec2time(etime / en, &year, &mon, &day, &iaux, &iaux, &iaux,
                      &aux);
            day2doy(year, mon, day, &doy);

            /* Find along-track index... */
            track = 0;
            for (itrack2 = 0; itrack2 < pert->ntrack; itrack2++)
              if (fabs(pert->time[itrack2][0] - etime / en)
                  < fabs(pert->time[track][0] - etime / en))
                track = itrack2;

            /* Write data... */
            aux = etime / en;
            NC(nc_put_vara_double(ncid, time_varid, start, count, &aux));
            NC(nc_put_vara_int(ncid, year_varid, start, count, &year));
            NC(nc_put_vara_int(ncid, doy_varid, start, count, &doy));
            NC(nc_put_vara_int(ncid, track_varid, start, count, &track));
            NC(nc_put_vara_int(ncid, np_east_varid, start, count, &en));
            aux = evar / en - gsl_pow_2(emu / en) - gsl_pow_2(enoise);
            NC(nc_put_vara_double
               (ncid, var_east_varid, start, count, &aux));
            NC(nc_put_vara_int(ncid, np_west_varid, start, count, &wn));
            aux = wvar / wn - gsl_pow_2(wmu / wn) - gsl_pow_2(wnoise);
            NC(nc_put_vara_double
               (ncid, var_west_varid, start, count, &aux));

            /* Increment data point counter... */
            start[0]++;
          }
        }

        /* Initialize... */
        etime = wtime = 0;
        evar = wvar = 0;
        emu = wmu = 0;
        ebt = wbt = 0;
        en = wn = 0;
      }

      /* Check if footprint is in eastern box... */
      if (pert->lon[itrack][ixtrack] >= lon0
          && pert->lon[itrack][ixtrack] <= lon0 + dlon
          && pert->lat[itrack][ixtrack] >= lat0 - dlat / 2.
```

```
            && pert->lat[itrack][ixtrack] <= lat0 + dlat / 2.) {

          etime += pert->time[itrack][ixtrack];
          emu += pert->pt[itrack][ixtrack];
          evar += gsl_pow_2(pert->pt[itrack][ixtrack]);
          ebt += pert->bt[itrack][ixtrack];
          en++;
        }

        /* Check if footprint is in western box... */
        if (pert->lon[itrack][ixtrack] >= lon0 - offset - dlon
            && pert->lon[itrack][ixtrack] <= lon0 - offset
            && pert->lat[itrack][ixtrack] >= lat0 - dlat / 2.
            && pert->lat[itrack][ixtrack] <= lat0 + dlat / 2.) {

          wtime += pert->time[itrack][ixtrack];
          wmu += pert->pt[itrack][ixtrack];
          wvar += gsl_pow_2(pert->pt[itrack][ixtrack]);
          wbt += pert->bt[itrack][ixtrack];
          wn++;
        }
      }

    /* Write output for last orbit... */
    if (en > 0 && wn > 0) {

      /* Estimate noise... */
      if (dt230 > 0) {
        nesr = planck(230.0 + dt230, nu) - planck(230.0, nu);
        enoise = brightness(planck(ebt / en, nu) + nesr, nu) - ebt / en;
        wnoise = brightness(planck(wbt / wn, nu) + nesr, nu) - wbt / wn;
      }

      /* Write output... */
      fprintf(out, "%.2f %d %d %g %g %g %d %g %g %g\n", etime / en, orb,
              en, evar / en - gsl_pow_2(emu / en), ebt / en, enoise,
              wn, wvar / wn - gsl_pow_2(wmu / wn), wbt / wn, wnoise);

      /* Write to netCDF file... */
      if (ncfile[0] != '-') {

        /* Get year and doy... */
        jsec2time(etime / en, &year, &mon, &day, &iaux, &iaux, &iaux, &aux);
        day2doy(year, mon, day, &doy);

        /* Find along-track index... */
        track = 0;
        for (itrack2 = 0; itrack2 < pert->ntrack; itrack2++)
          if (fabs(pert->time[itrack2][0] - etime / en)
              < fabs(pert->time[track][0] - etime / en))
            track = itrack2;

        /* Write data... */
        aux = etime / en;
        NC(nc_put_vara_double(ncid, time_varid, start, count, &aux));
        NC(nc_put_vara_int(ncid, year_varid, start, count, &year));
        NC(nc_put_vara_int(ncid, doy_varid, start, count, &doy));
        NC(nc_put_vara_int(ncid, track_varid, start, count, &track));
        NC(nc_put_vara_int(ncid, np_east_varid, start, count, &en));
        aux = evar / en - gsl_pow_2(emu / en) - gsl_pow_2(enoise);
        NC(nc_put_vara_double(ncid, var_east_varid, start, count, &aux));
        NC(nc_put_vara_int(ncid, np_west_varid, start, count, &wn));
        aux = wvar / wn - gsl_pow_2(wmu / wn) - gsl_pow_2(wnoise);
        NC(nc_put_vara_double(ncid, var_west_varid, start, count, &aux));

        /* Increment data point counter... */
        start[0]++;
      }
    }
  }

  /* Close file... */
  fclose(out);

  /* Close file... */
  if (ncfile[0] != '-')
    NC(nc_close(ncid));

  /* Free... */
  free(pert);

  return EXIT_SUCCESS;
}
```

Here is the call graph for this function:

5.30 island.c

```
#include "libairs.h"

/* ------------------------------------------------------------
   Functions...
   ------------------------------------------------------------ */

/* Add variable defintions to netCDF file. */
void addatt(
  int ncid,
  int varid,
  const char *unit,
  const char *long_name);

/* ------------------------------------------------------------
   Main...
   ------------------------------------------------------------ */

int main(
  int argc,
  char *argv[]) {

  static pert_t *pert;

  static wave_t *wave;

  static FILE *in, *out;

  static char pertname[LEN], ncfile[LEN];

  static double gauss_fwhm, var_dh, orblat, lon0, lat0, dlon, dlat, offset,
    ebt, emu, enoise, evar, wbt, wmu, wnoise, wvar, etime, wtime,
    dt230, nu, nesr, aux;

  static int iarg, ix, iy, itrack, itrack2, ixtrack, bg_poly_x, bg_poly_y,
    bg_smooth_x, bg_smooth_y, orb, orb_old = -1, en, wn, ncid, dimid[2],
    time_varid, track_varid, np_east_varid, var_east_varid,
    np_west_varid, var_west_varid, year_varid, doy_varid,
    track, year, mon, day, doy, iaux;

  static size_t count[2] = { 1, 1 }, start[2];

  /* Check arguments... */
  if (argc < 4)
    ERRMSG("Give parameters: <ctl> <var.tab> <pert1.nc> [<pert2.nc> ...]");

  /* Get control parameters... */
  scan_ctl(argc, argv, "PERTNAME", -1, "4mu", pertname);
  lon0 = scan_ctl(argc, argv, "LON0", -1, "", NULL);
  lat0 = scan_ctl(argc, argv, "LAT0", -1, "", NULL);
  dlon = scan_ctl(argc, argv, "DLON", -1, "", NULL);
  dlat = scan_ctl(argc, argv, "DLAT", -1, "", NULL);
  offset = scan_ctl(argc, argv, "OFFSET", -1, "1", NULL);
  bg_poly_x = (int) scan_ctl(argc, argv, "BG_POLY_X", -1, "0", NULL);
  bg_poly_y = (int) scan_ctl(argc, argv, "BG_POLY_Y", -1, "0", NULL);
  bg_smooth_x = (int) scan_ctl(argc, argv, "BG_SMOOTH_X", -1, "0", NULL);
  bg_smooth_y = (int) scan_ctl(argc, argv, "BG_SMOOTH_Y", -1, "0", NULL);
  gauss_fwhm = scan_ctl(argc, argv, "GAUSS_FWHM", -1, "0", NULL);
  var_dh = scan_ctl(argc, argv, "VAR_DH", -1, "0", NULL);
  orblat = scan_ctl(argc, argv, "ORBLAT", -1, "0", NULL);
  dt230 = scan_ctl(argc, argv, "DT230", -1, "0.16", NULL);
  nu = scan_ctl(argc, argv, "NU", -1, "2345.0", NULL);
  scan_ctl(argc, argv, "NCFILE", -1, "-", ncfile);

  /* Allocate... */
  ALLOC(pert, pert_t, 1);

  /* Create file... */
  printf("Write variance statistics: %s\n", argv[2]);
  if (!(out = fopen(argv[2], "w")))
    ERRMSG("Cannot create file!");

  /* Write header... */
  fprintf(out,
          "# $1  = time [s]\n"
          "# $2  = orbit number\n"
          "# $3  = eastern box: number of footprints\n"
          "# $4  = eastern box: variance [K^2]\n"
          "# $5  = eastern box: mean background temperature [K]\n"
          "# $6  = eastern box: noise estimate [K]\n"
          "# $7  = western box: number of footprints\n"
          "# $8  = western box: variance [K^2]\n"
          "# $9  = western box: mean background temperature [K]\n"
          "# $10 = western box: noise estimate [K]\n\n");

```

```
/* Create netCDF file... */
if (ncfile[0] != '-') {

  /* Create file... */
  printf("Write variance statistics: %s\n", ncfile);
  NC(nc_create(ncfile, NC_CLOBBER, &ncid));

  /* Set dimensions... */
  NC(nc_def_dim(ncid, "NP", NC_UNLIMITED, &dimid[0]));

  /* Add attributes... */
  aux = lon0;
  nc_put_att_double(ncid, NC_GLOBAL, "box_east_lon0", NC_DOUBLE, 1, &aux);
  aux = lon0 + dlon;
  nc_put_att_double(ncid, NC_GLOBAL, "box_east_lon1", NC_DOUBLE, 1, &aux);
  aux = lat0 - 0.5 * dlat;
  nc_put_att_double(ncid, NC_GLOBAL, "box_east_lat0", NC_DOUBLE, 1, &aux);
  aux = lat0 + 0.5 * dlat;
  nc_put_att_double(ncid, NC_GLOBAL, "box_east_lat1", NC_DOUBLE, 1, &aux);
  aux = lon0 - dlon - offset;
  nc_put_att_double(ncid, NC_GLOBAL, "box_west_lon0", NC_DOUBLE, 1, &aux);
  aux = lon0 - offset;
  nc_put_att_double(ncid, NC_GLOBAL, "box_west_lon1", NC_DOUBLE, 1, &aux);
  aux = lat0 - 0.5 * dlat;
  nc_put_att_double(ncid, NC_GLOBAL, "box_west_lat0", NC_DOUBLE, 1, &aux);
  aux = lat0 + 0.5 * dlat;
  nc_put_att_double(ncid, NC_GLOBAL, "box_west_lat1", NC_DOUBLE, 1, &aux);

  /* Add variables... */
  NC(nc_def_var(ncid, "time", NC_DOUBLE, 1, dimid, &time_varid));
  addatt(ncid, time_varid, "s", "time (seconds since 2000-01-01T00:00Z)");
  NC(nc_def_var(ncid, "year", NC_INT, 1, dimid, &year_varid));
  addatt(ncid, year_varid, "1", "year");
  NC(nc_def_var(ncid, "doy", NC_INT, 1, dimid, &doy_varid));
  addatt(ncid, doy_varid, "1", "day of year");
  NC(nc_def_var(ncid, "track", NC_INT, 1, dimid, &track_varid));
  addatt(ncid, track_varid, "1", "along-track index");
  NC(nc_def_var(ncid, "var_east", NC_DOUBLE, 1, dimid, &var_east_varid));
  addatt(ncid, var_east_varid, "K^2", "BT variance (east)");
  NC(nc_def_var(ncid, "var_west", NC_DOUBLE, 1, dimid, &var_west_varid));
  addatt(ncid, var_west_varid, "K^2", "BT variance (west)");
  NC(nc_def_var(ncid, "np_east", NC_INT, 1, dimid, &np_east_varid));
  addatt(ncid, np_east_varid, "1", "number of footprints (east)");
  NC(nc_def_var(ncid, "np_west", NC_INT, 1, dimid, &np_west_varid));
  addatt(ncid, np_west_varid, "1", "number of footprints (west)");

  /* Leave define mode... */
  NC(nc_enddef(ncid));
}

/* Loop over perturbation files... */
for (iarg = 3; iarg < argc; iarg++) {

  /* Check filename... */
  if (!strcmp(argv[iarg], ncfile))
    continue;

  /* Initialize... */
  orb = 0;

  /* Read perturbation data... */
  if (!(in = fopen(argv[iarg], "r")))
    continue;
  else {
    fclose(in);
    read_pert(argv[iarg], pertname, pert);
  }

  /* Recalculate background and perturbations... */
  if (bg_poly_x > 0 || bg_poly_y > 0 ||
      bg_smooth_x > 0 || bg_smooth_y > 0 || gauss_fwhm > 0 || var_dh > 0) {

    /* Allocate... */
    ALLOC(wave, wave_t, 1);

    /* Convert to wave analysis struct... */
    pert2wave(pert, wave, 0, pert->ntrack - 1, 0, pert->nxtrack - 1);

    /* Estimate background... */
    background_poly(wave, bg_poly_x, bg_poly_y);
    background_smooth(wave, bg_smooth_x, bg_smooth_y);

    /* Gaussian filter... */
    gauss(wave, gauss_fwhm);

    /* Compute variance... */
    variance(wave, var_dh);
```

```
      /* Copy data... */
      for (ix = 0; ix < wave->nx; ix++)
        for (iy = 0; iy < wave->ny; iy++) {
          pert->pt[iy][ix] = wave->pt[ix][iy];
          pert->var[iy][ix] = wave->var[ix][iy];
        }

      /* Free... */
      free(wave);
    }

    /* Detection... */
    for (itrack = 0; itrack < pert->ntrack; itrack++)
      for (ixtrack = 0; ixtrack < pert->nxtrack; ixtrack++) {

        /* Check data... */
        if (pert->time[itrack][ixtrack] < 0
            || pert->lon[itrack][ixtrack] < -180
            || pert->lon[itrack][ixtrack] > 180
            || pert->lat[itrack][ixtrack] < -90
            || pert->lat[itrack][ixtrack] > 90
            || pert->pt[itrack][ixtrack] < -100
            || pert->pt[itrack][ixtrack] > 100
            || !gsl_finite(pert->bt[itrack][ixtrack])
            || !gsl_finite(pert->pt[itrack][ixtrack])
            || !gsl_finite(pert->var[itrack][ixtrack])
            || !gsl_finite(pert->dc[itrack][ixtrack]))
          continue;

        /* Count orbits... */
        if (itrack > 0 && ixtrack == pert->nxtrack / 2)
          if (pert->lat[itrack - 1][ixtrack] <= orblat
              && pert->lat[itrack][ixtrack] >= orblat)
            orb++;
        if (orb != orb_old) {

          /* Set orbit index... */
          orb_old = orb;

          /* Write output... */
          if (en > 0 && wn > 0) {

            /* Estimate noise... */
            if (dt230 > 0) {
              nesr = planck(230.0 + dt230, nu) - planck(230.0, nu);
              enoise = brightness(planck(ebt / en, nu) + nesr, nu) - ebt / en;
              wnoise = brightness(planck(wbt / wn, nu) + nesr, nu) - wbt / wn;
            }

            /* Write output... */
            fprintf(out, "%.2f %d %d %g %g %g %d %g %g %g\n", etime / en, orb,
                    en, evar / en - gsl_pow_2(emu / en), ebt / en, enoise,
                    wn, wvar / wn - gsl_pow_2(wmu / wn), wbt / wn, wnoise);

            /* Write to netCDF file... */
            if (ncfile[0] != '-') {

              /* Get year and doy... */
              jsec2time(etime / en, &year, &mon, &day, &iaux, &iaux, &iaux,
                        &aux);
              day2doy(year, mon, day, &doy);

              /* Find along-track index... */
              track = 0;
              for (itrack2 = 0; itrack2 < pert->ntrack; itrack2++)
                if (fabs(pert->time[itrack2][0] - etime / en)
                    < fabs(pert->time[track][0] - etime / en))
                  track = itrack2;

              /* Write data... */
              aux = etime / en;
              NC(nc_put_vara_double(ncid, time_varid, start, count, &aux));
              NC(nc_put_vara_int(ncid, year_varid, start, count, &year));
              NC(nc_put_vara_int(ncid, doy_varid, start, count, &doy));
              NC(nc_put_vara_int(ncid, track_varid, start, count, &track));
              NC(nc_put_vara_int(ncid, np_east_varid, start, count, &en));
              aux = evar / en - gsl_pow_2(emu / en) - gsl_pow_2(enoise);
              NC(nc_put_vara_double
                 (ncid, var_east_varid, start, count, &aux));
              NC(nc_put_vara_int(ncid, np_west_varid, start, count, &wn));
              aux = wvar / wn - gsl_pow_2(wmu / wn) - gsl_pow_2(wnoise);
              NC(nc_put_vara_double
                 (ncid, var_west_varid, start, count, &aux));

              /* Increment data point counter... */
              start[0]++;
```

```
          }
        }

        /* Initialize... */
        etime = wtime = 0;
        evar = wvar = 0;
        emu = wmu = 0;
        ebt = wbt = 0;
        en = wn = 0;
      }

      /* Check if footprint is in eastern box... */
      if (pert->lon[itrack][ixtrack] >= lon0
          && pert->lon[itrack][ixtrack] <= lon0 + dlon
          && pert->lat[itrack][ixtrack] >= lat0 - dlat / 2.
          && pert->lat[itrack][ixtrack] <= lat0 + dlat / 2.) {

        etime += pert->time[itrack][ixtrack];
        emu += pert->pt[itrack][ixtrack];
        evar += gsl_pow_2(pert->pt[itrack][ixtrack]);
        ebt += pert->bt[itrack][ixtrack];
        en++;
      }

      /* Check if footprint is in western box... */
      if (pert->lon[itrack][ixtrack] >= lon0 - offset - dlon
          && pert->lon[itrack][ixtrack] <= lon0 - offset
          && pert->lat[itrack][ixtrack] >= lat0 - dlat / 2.
          && pert->lat[itrack][ixtrack] <= lat0 + dlat / 2.) {

        wtime += pert->time[itrack][ixtrack];
        wmu += pert->pt[itrack][ixtrack];
        wvar += gsl_pow_2(pert->pt[itrack][ixtrack]);
        wbt += pert->bt[itrack][ixtrack];
        wn++;
      }
    }

  /* Write output for last orbit... */
  if (en > 0 && wn > 0) {

    /* Estimate noise... */
    if (dt230 > 0) {
      nesr = planck(230.0 + dt230, nu) - planck(230.0, nu);
      enoise = brightness(planck(ebt / en, nu) + nesr, nu) - ebt / en;
      wnoise = brightness(planck(wbt / wn, nu) + nesr, nu) - wbt / wn;
    }

    /* Write output... */
    fprintf(out, "%.2f %d %d %g %g %g %d %g %g %g\n", etime / en, orb,
            en, evar / en - gsl_pow_2(emu / en), ebt / en, enoise,
            wn, wvar / wn - gsl_pow_2(wmu / wn), wbt / wn, wnoise);

    /* Write to netCDF file... */
    if (ncfile[0] != '-') {

      /* Get year and doy... */
      jsec2time(etime / en, &year, &mon, &day, &iaux, &iaux, &iaux, &aux);
      day2doy(year, mon, day, &doy);

      /* Find along-track index... */
      track = 0;
      for (itrack2 = 0; itrack2 < pert->ntrack; itrack2++)
        if (fabs(pert->time[itrack2][0] - etime / en)
            < fabs(pert->time[track][0] - etime / en))
          track = itrack2;

      /* Write data... */
      aux = etime / en;
      NC(nc_put_vara_double(ncid, time_varid, start, count, &aux));
      NC(nc_put_vara_int(ncid, year_varid, start, count, &year));
      NC(nc_put_vara_int(ncid, doy_varid, start, count, &doy));
      NC(nc_put_vara_int(ncid, track_varid, start, count, &track));
      NC(nc_put_vara_int(ncid, np_east_varid, start, count, &en));
      aux = evar / en - gsl_pow_2(emu / en) - gsl_pow_2(enoise);
      NC(nc_put_vara_double(ncid, var_east_varid, start, count, &aux));
      NC(nc_put_vara_int(ncid, np_west_varid, start, count, &wn));
      aux = wvar / wn - gsl_pow_2(wmu / wn) - gsl_pow_2(wnoise);
      NC(nc_put_vara_double(ncid, var_west_varid, start, count, &aux));

      /* Increment data point counter... */
      start[0]++;
    }
  }
}

/* Close file... */
```

```
  fclose(out);

  /* Close file... */
  if (ncfile[0] != '-')
    NC(nc_close(ncid));

  /* Free... */
  free(pert);

  return EXIT_SUCCESS;
}

/*****************************************************************************/

void addatt(
  int ncid,
  int varid,
  const char *unit,
  const char *long_name) {

  /* Set long name... */
  NC(nc_put_att_text(ncid, varid, "long_name", strlen(long_name), long_name));

  /* Set units... */
  NC(nc_put_att_text(ncid, varid, "units", strlen(unit), unit));
}
```

5.31 jurassic.c File Reference

JURASSIC library definitions.

Functions

- size_t atm2x (ctl_t ∗ctl, atm_t ∗atm, gsl_vector ∗x, int ∗iqa, int ∗ipa)

  Compose state vector or parameter vector.
- void atm2x_help (atm_t ∗atm, double zmin, double zmax, double ∗value, int val_iqa, gsl_vector ∗x, int ∗iqa, int ∗ipa, size_t ∗n)

  Add elements to state vector.
- double brightness (double rad, double nu)

  Compute brightness temperature.
- void cart2geo (double ∗x, double ∗z, double ∗lon, double ∗lat)

  Convert Cartesian coordinates to geolocation.
- void climatology (ctl_t ∗ctl, atm_t ∗atm)

  Interpolate climatological data.
- double ctmco2 (double nu, double p, double t, double u)

  Compute carbon dioxide continuum (optical depth).
- double ctmh2o (double nu, double p, double t, double q, double u)

  Compute water vapor continuum (optical depth).
- double ctmn2 (double nu, double p, double t)

  Compute nitrogen continuum (absorption coefficient).
- double ctmo2 (double nu, double p, double t)

  Compute oxygen continuum (absorption coefficient).
- void copy_atm (ctl_t ∗ctl, atm_t ∗atm_dest, atm_t ∗atm_src, int init)

  Copy and initialize atmospheric data.
- void copy_obs (ctl_t ∗ctl, obs_t ∗obs_dest, obs_t ∗obs_src, int init)

  Copy and initialize observation data.
- int find_emitter (ctl_t ∗ctl, const char ∗emitter)

  Find index of an emitter.
- void formod (ctl_t ∗ctl, atm_t ∗atm, obs_t ∗obs)

  Determine ray paths and compute radiative transfer.

- void formod_continua (ctl_t *ctl, los_t *los, int ip, double *beta)

  Compute absorption coefficient of continua.
- void formod_fov (ctl_t *ctl, obs_t *obs)

  Apply field of view convolution.
- void formod_pencil (ctl_t *ctl, atm_t *atm, obs_t *obs, int ir)

  Compute radiative transfer for a pencil beam.
- void formod_srcfunc (ctl_t *ctl, tbl_t *tbl, double t, double *src)

  Compute Planck source function.
- void geo2cart (double z, double lon, double lat, double *x)

  Convert geolocation to Cartesian coordinates.
- double gravity (double z, double lat)

  Determine gravity of Earth.
- void hydrostatic (ctl_t *ctl, atm_t *atm)

  Set hydrostatic equilibrium.
- void idx2name (ctl_t *ctl, int idx, char *quantity)

  Determine name of state vector quantity for given index.
- void init_tbl (ctl_t *ctl, tbl_t *tbl)

  Initialize look-up tables.
- void intpol_atm (ctl_t *ctl, atm_t *atm, double z, double *p, double *t, double *q, double *k)

  Interpolate atmospheric data.
- void intpol_tbl (ctl_t *ctl, tbl_t *tbl, los_t *los, int ip, double tau_path[NG][ND], double tau_seg[ND])

  Get transmittance from look-up tables.
- double intpol_tbl_eps (tbl_t *tbl, int ig, int id, int ip, int it, double u)

  Interpolate emissivity from look-up tables.
- double intpol_tbl_u (tbl_t *tbl, int ig, int id, int ip, int it, double eps)

  Interpolate column density from look-up tables.
- void jsec2time (double jsec, int *year, int *mon, int *day, int *hour, int *min, int *sec, double *remain)

  Convert seconds to date.
- void kernel (ctl_t *ctl, atm_t *atm, obs_t *obs, gsl_matrix *k)

  Compute Jacobians.
- int locate (double *xx, int n, double x)

  Find array index.
- int locate_tbl (float *xx, int n, double x)

  Find array index in float array.
- size_t obs2y (ctl_t *ctl, obs_t *obs, gsl_vector *y, int *ida, int *ira)

  Compose measurement vector.
- double planck (double t, double nu)

  Compute Planck function.
- void raytrace (ctl_t *ctl, atm_t *atm, obs_t *obs, los_t *los, int ir)

  Do ray-tracing to determine LOS.
- void read_atm (const char *dirname, const char *filename, ctl_t *ctl, atm_t *atm)

  Read atmospheric data.
- void read_ctl (int argc, char *argv[ ], ctl_t *ctl)

  Read forward model control parameters.
- void read_matrix (const char *dirname, const char *filename, gsl_matrix *matrix)

  Read matrix.
- void read_obs (const char *dirname, const char *filename, ctl_t *ctl, obs_t *obs)

  Read observation data.
- void read_shape (const char *filename, double *x, double *y, int *n)

  Read shape function.
- double refractivity (double p, double t)

Compute refractivity (return value is n - 1).

- double scan_ctl (int argc, char ∗argv[ ], const char ∗varname, int arridx, const char ∗defvalue, char ∗value)

  Search control parameter file for variable entry.
- void tangent_point (los_t ∗los, double ∗tpz, double ∗tplon, double ∗tplat)

  Find tangent point of a given LOS.
- void time2jsec (int year, int mon, int day, int hour, int min, int sec, double remain, double ∗jsec)

  Convert date to seconds.
- void timer (const char ∗name, const char ∗file, const char ∗func, int line, int mode)

  Measure wall-clock time.
- void write_atm (const char ∗dirname, const char ∗filename, ctl_t ∗ctl, atm_t ∗atm)

  Write atmospheric data.
- void write_matrix (const char ∗dirname, const char ∗filename, ctl_t ∗ctl, gsl_matrix ∗matrix, atm_t ∗atm, obs_t ∗obs, const char ∗rowspace, const char ∗colspace, const char ∗sort)

  Write matrix.
- void write_obs (const char ∗dirname, const char ∗filename, ctl_t ∗ctl, obs_t ∗obs)

  Write observation data.
- void x2atm (ctl_t ∗ctl, gsl_vector ∗x, atm_t ∗atm)

  Decompose parameter vector or state vector.
- void x2atm_help (atm_t ∗atm, double zmin, double zmax, double ∗value, gsl_vector ∗x, size_t ∗n)

  Extract elements from state vector.
- void y2obs (ctl_t ∗ctl, gsl_vector ∗y, obs_t ∗obs)

  Decompose measurement vector.

5.31.1 Detailed Description

JURASSIC library definitions.

Definition in file jurassic.c.

5.31.2 Function Documentation

5.31.2.1 size_t atm2x (ctl_t ∗ *ctl,* atm_t ∗ *atm,* gsl_vector ∗ *x,* int ∗ *iqa,* int ∗ *ipa*)

Compose state vector or parameter vector.

Definition at line 29 of file jurassic.c.

```
              {

  int ig, iw;

  size_t n = 0;

  /* Add pressure... */
  atm2x_help(atm, ctl->retp_zmin, ctl->retp_zmax,
             atm->p, IDXP, x, iqa, ipa, &n);

  /* Add temperature... */
  atm2x_help(atm, ctl->rett_zmin, ctl->rett_zmax,
             atm->t, IDXT, x, iqa, ipa, &n);

  /* Add volume mixing ratios... */
  for (ig = 0; ig < ctl->ng; ig++)
    atm2x_help(atm, ctl->retq_zmin[ig], ctl->retq_zmax[ig],
               atm->q[ig], IDXQ(ig), x, iqa, ipa, &n);

  /* Add extinction... */
  for (iw = 0; iw < ctl->nw; iw++)
    atm2x_help(atm, ctl->retk_zmin[iw], ctl->retk_zmax[iw],
               atm->k[iw], IDXK(iw), x, iqa, ipa, &n);

  return n;
}
```

Here is the call graph for this function:

5.31.2.2 void atm2x_help (atm_t * *atm*, double *zmin*, double *zmax*, double * *value*, int *val_iqa*, gsl_vector * *x*, int * *iqa*, int * *ipa*, size_t * *n*)

Add elements to state vector.

Definition at line 63 of file jurassic.c.

```
                {

  int ip;

  /* Add elements to state vector... */
  for (ip = 0; ip < atm->np; ip++)
    if (atm->z[ip] >= zmin && atm->z[ip] <= zmax) {
      if (x != NULL)
        gsl_vector_set(x, *n, value[ip]);
      if (iqa != NULL)
        iqa[*n] = val_iqa;
      if (ipa != NULL)
        ipa[*n] = ip;
      (*n)++;
    }
}
```

5.31.2.3 double brightness (double *rad*, double *nu*)

Compute brightness temperature.

Definition at line 91 of file jurassic.c.

```
               {

  return C2 * nu / gsl_log1p(C1 * gsl_pow_3(nu) / rad);
}
```

5.31.2.4 void cart2geo (double * *x*, double * *z*, double * *lon*, double * *lat*)

Convert Cartesian coordinates to geolocation.

Definition at line 101 of file jurassic.c.

```
                 {

  double radius;

  radius = NORM(x);
  *lat = asin(x[2] / radius) * 180 / M_PI;
  *lon = atan2(x[1], x[0]) * 180 / M_PI;
  *z = radius - RE;
}
```

5.31.2.5 void climatology (ctl_t * ctl, atm_t * atm_mean)

Interpolate climatological data.

Definition at line 117 of file jurassic.c.

```
               {

static double z[121] = {
  0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19,
  20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37,
  38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55,
  56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73,
  74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91,
  92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107,
  108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 120
};

static double pre[121] = {
  1017, 901.083, 796.45, 702.227, 617.614, 541.644, 473.437, 412.288,
  357.603, 308.96, 265.994, 228.348, 195.619, 167.351, 143.039, 122.198,
  104.369, 89.141, 76.1528, 65.0804, 55.641, 47.591, 40.7233, 34.8637,
  29.8633, 25.5956, 21.9534, 18.8445, 16.1909, 13.9258, 11.9913,
  10.34, 8.92988, 7.72454, 6.6924, 5.80701, 5.04654, 4.39238, 3.82902,
  3.34337, 2.92413, 2.56128, 2.2464, 1.97258, 1.73384, 1.52519, 1.34242,
  1.18197, 1.04086, 0.916546, 0.806832, 0.709875, 0.624101, 0.548176,
  0.480974, 0.421507, 0.368904, 0.322408, 0.281386, 0.245249, 0.213465,
  0.185549, 0.161072, 0.139644, 0.120913, 0.104568, 0.0903249, 0.0779269,
  0.0671493, 0.0577962, 0.0496902, 0.0426736, 0.0366093, 0.0313743,
  0.0268598, 0.0229699, 0.0196206, 0.0167399, 0.0142646, 0.0121397,
  0.0103181, 0.00875775, 0.00742226, 0.00628076, 0.00530519, 0.00447183,
  0.00376124, 0.00315632, 0.00264248, 0.00220738, 0.00184003, 0.00153095,
  0.00127204, 0.00105608, 0.000876652, 0.00072798, 0.00060492,
  0.000503201, 0.000419226, 0.000349896, 0.000292659, 0.000245421,
  0.000206394, 0.000174125, 0.000147441, 0.000125333, 0.000106985,
  9.173e-05, 7.90172e-05, 6.84172e-05, 5.95574e-05, 5.21183e-05,
  4.58348e-05, 4.05127e-05, 3.59987e-05, 3.21583e-05, 2.88718e-05,
  2.60322e-05, 2.35687e-05, 2.14263e-05, 1.95489e-05
};

static double tem[121] = {
  285.14, 279.34, 273.91, 268.3, 263.24, 256.55, 250.2, 242.82, 236.17,
  229.87, 225.04, 221.19, 218.85, 217.19, 216.2, 215.68, 215.42, 215.55,
  215.92, 216.4, 216.93, 217.45, 218, 218.68, 219.39, 220.25, 221.3,
  222.41, 223.88, 225.42, 227.2, 229.52, 231.89, 234.51, 236.85, 239.42,
  241.94, 244.57, 247.36, 250.32, 253.34, 255.82, 258.27, 260.39,
  262.03, 263.45, 264.2, 264.78, 264.67, 264.38, 263.24, 262.03, 260.02,
  258.09, 255.63, 253.28, 250.43, 247.81, 245.26, 242.77, 240.38,
  237.94, 235.79, 233.53, 231.5, 229.53, 227.6, 225.62, 223.77, 222.06,
  220.33, 218.69, 217.18, 215.64, 214.13, 212.52, 210.86, 209.25,
  207.49, 205.81, 204.11, 202.22, 200.32, 198.39, 195.92, 193.46,
  190.94, 188.31, 185.82, 183.57, 181.43, 179.74, 178.64, 178.1, 178.25,
  178.7, 179.41, 180.67, 182.31, 184.18, 186.6, 189.53, 192.66, 196.54,
  201.13, 205.93, 211.73, 217.86, 225, 233.53, 242.57, 252.14, 261.48,
  272.97, 285.26, 299.12, 312.2, 324.17, 338.34, 352.56, 365.28
};

static double c2h2[121] = {
  1.352e-09, 2.83e-10, 1.269e-10, 6.926e-11, 4.346e-11, 2.909e-11,
  2.014e-11, 1.363e-11, 8.71e-12, 5.237e-12, 2.718e-12, 1.375e-12,
  5.786e-13, 2.16e-13, 7.317e-14, 2.551e-14, 1.055e-14, 4.758e-15,
  2.056e-15, 7.703e-16, 2.82e-16, 1.035e-16, 4.382e-17, 1.946e-17,
  9.638e-18, 5.2e-18, 2.811e-18, 1.494e-18, 7.925e-19, 4.213e-19,
  1.998e-19, 8.78e-20, 3.877e-20, 1.728e-20, 7.743e-21, 3.536e-21,
  1.623e-21, 7.508e-22, 3.508e-22, 1.65e-22, 7.837e-23, 3.733e-23,
  1.808e-23, 8.77e-24, 4.285e-24, 2.095e-24, 1.032e-24, 5.082e-25,
  2.506e-25, 1.236e-25, 6.088e-26, 2.996e-26, 1.465e-26, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
};

static double c2h6[121] = {
  2.667e-09, 2.02e-09, 1.658e-09, 1.404e-09, 1.234e-09, 1.109e-09,
  1.012e-09, 9.262e-10, 8.472e-10, 7.71e-10, 6.932e-10, 6.216e-10,
  5.503e-10, 4.87e-10, 4.342e-10, 3.861e-10, 3.347e-10, 2.772e-10,
  2.209e-10, 1.672e-10, 1.197e-10, 8.536e-11, 5.783e-11, 3.846e-11,
  2.495e-11, 1.592e-11, 1.017e-11, 6.327e-12, 3.895e-12, 2.403e-12,
  1.416e-12, 8.101e-13, 4.649e-13, 2.686e-13, 1.557e-13, 9.14e-14,
  5.386e-14, 3.19e-14, 1.903e-14, 1.14e-14, 6.875e-15, 4.154e-15,
  2.538e-15, 1.553e-15, 9.548e-16, 5.872e-16, 3.63e-16, 2.244e-16,
  1.388e-16, 8.587e-17, 5.308e-17, 3.279e-17, 2.017e-17, 1.238e-17,
  7.542e-18, 4.585e-18, 2.776e-18, 1.671e-18, 9.985e-19, 5.937e-19,
```

```
  3.518e-19, 2.07e-19, 1.215e-19, 7.06e-20, 4.097e-20, 2.37e-20,
  1.363e-20, 7.802e-21, 4.441e-21, 2.523e-21, 1.424e-21, 8.015e-22,
  4.497e-22, 2.505e-22, 1.391e-22, 7.691e-23, 4.238e-23, 2.331e-23,
  1.274e-23, 6.929e-24, 3.752e-24, 2.02e-24, 1.083e-24, 5.774e-25,
  3.041e-25, 1.593e-25, 8.308e-26, 4.299e-26, 2.195e-26, 1.112e-26,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0
};

static double ccl4[121] = {
  1.075e-10, 1.075e-10, 1.075e-10, 1.075e-10, 1.075e-10, 1.075e-10,
  1.075e-10, 1.075e-10, 1.075e-10, 1.06e-10, 1.024e-10, 9.69e-11,
  8.93e-11, 8.078e-11, 7.213e-11, 6.307e-11, 5.383e-11, 4.49e-11,
  3.609e-11, 2.705e-11, 1.935e-11, 1.385e-11, 8.35e-12, 5.485e-12,
  3.853e-12, 2.22e-12, 5.875e-13, 3.445e-13, 1.015e-13, 6.075e-14,
  4.383e-14, 2.692e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14,
  1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14,
  1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14,
  1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14,
  1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14,
  1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14,
  1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14,
  1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14,
  1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14,
  1e-14, 1e-14, 1e-14
};

static double ch4[121] = {
  1.864e-06, 1.835e-06, 1.819e-06, 1.805e-06, 1.796e-06, 1.788e-06,
  1.782e-06, 1.776e-06, 1.769e-06, 1.761e-06, 1.749e-06, 1.734e-06,
  1.716e-06, 1.692e-06, 1.654e-06, 1.61e-06, 1.567e-06, 1.502e-06,
  1.433e-06, 1.371e-06, 1.323e-06, 1.277e-06, 1.232e-06, 1.188e-06,
  1.147e-06, 1.108e-06, 1.07e-06, 1.027e-06, 9.854e-07, 9.416e-07,
  8.933e-07, 8.478e-07, 7.988e-07, 7.515e-07, 7.07e-07, 6.64e-07,
  6.239e-07, 5.864e-07, 5.512e-07, 5.184e-07, 4.87e-07, 4.571e-07,
  4.296e-07, 4.04e-07, 3.802e-07, 3.578e-07, 3.383e-07, 3.203e-07,
  3.032e-07, 2.889e-07, 2.76e-07, 2.635e-07, 2.519e-07, 2.409e-07,
  2.302e-07, 2.219e-07, 2.144e-07, 2.071e-07, 1.999e-07, 1.93e-07,
  1.862e-07, 1.795e-07, 1.731e-07, 1.668e-07, 1.607e-07, 1.548e-07,
  1.49e-07, 1.434e-07, 1.38e-07, 1.328e-07, 1.277e-07, 1.227e-07,
  1.18e-07, 1.134e-07, 1.089e-07, 1.046e-07, 1.004e-07, 9.635e-08,
  9.245e-08, 8.867e-08, 8.502e-08, 8.15e-08, 7.809e-08, 7.48e-08,
  7.159e-08, 6.849e-08, 6.55e-08, 6.262e-08, 5.98e-08, 5.708e-08,
  5.448e-08, 5.194e-08, 4.951e-08, 4.72e-08, 4.5e-08, 4.291e-08,
  4.093e-08, 3.905e-08, 3.729e-08, 3.563e-08, 3.408e-08, 3.265e-08,
  3.128e-08, 2.996e-08, 2.87e-08, 2.76e-08, 2.657e-08, 2.558e-08,
  2.467e-08, 2.385e-08, 2.307e-08, 2.234e-08, 2.168e-08, 2.108e-08,
  2.05e-08, 1.998e-08, 1.947e-08, 1.902e-08, 1.86e-08, 1.819e-08,
  1.782e-08
};

static double clo[121] = {
  7.419e-15, 1.061e-14, 1.518e-14, 2.195e-14, 3.175e-14, 4.666e-14,
  6.872e-14, 1.03e-13, 1.553e-13, 2.375e-13, 3.664e-13, 5.684e-13,
  8.915e-13, 1.402e-12, 2.269e-12, 4.125e-12, 7.501e-12, 1.257e-11,
  2.048e-11, 3.338e-11, 5.44e-11, 8.846e-11, 1.008e-10, 1.082e-10,
  1.157e-10, 1.232e-10, 1.312e-10, 1.539e-10, 1.822e-10, 2.118e-10,
  2.387e-10, 2.687e-10, 2.875e-10, 3.031e-10, 3.23e-10, 3.648e-10,
  4.117e-10, 4.477e-10, 4.633e-10, 4.794e-10, 4.95e-10, 5.104e-10,
  5.259e-10, 5.062e-10, 4.742e-10, 4.443e-10, 4.051e-10, 3.659e-10,
  3.305e-10, 2.911e-10, 2.54e-10, 2.215e-10, 1.927e-10, 1.675e-10,
  1.452e-10, 1.259e-10, 1.09e-10, 9.416e-11, 8.119e-11, 6.991e-11,
  6.015e-11, 5.163e-11, 4.43e-11, 3.789e-11, 3.24e-11, 2.769e-11,
  2.361e-11, 2.011e-11, 1.71e-11, 1.453e-11, 1.233e-11, 1.045e-11,
  8.851e-12, 7.48e-12, 6.316e-12, 5.326e-12, 4.487e-12, 3.778e-12,
  3.176e-12, 2.665e-12, 2.234e-12, 1.87e-12, 1.563e-12, 1.304e-12,
  1.085e-12, 9.007e-13, 7.468e-13, 6.179e-13, 5.092e-13, 4.188e-13,
  3.442e-13, 2.816e-13, 2.304e-13, 1.885e-13, 1.542e-13, 1.263e-13,
  1.035e-13, 8.5e-14, 7.004e-14, 5.783e-14, 4.795e-14, 4.007e-14,
  3.345e-14, 2.792e-14, 2.33e-14, 1.978e-14, 1.686e-14, 1.438e-14,
  1.234e-14, 1.07e-14, 9.312e-15, 8.131e-15, 7.164e-15, 6.367e-15,
  5.67e-15, 5.088e-15, 4.565e-15, 4.138e-15, 3.769e-15, 3.432e-15,
  3.148e-15
};

static double clono2[121] = {
  1.011e-13, 1.515e-13, 2.272e-13, 3.446e-13, 5.231e-13, 8.085e-13,
  1.253e-12, 1.979e-12, 3.149e-12, 5.092e-12, 8.312e-12, 1.366e-11,
  2.272e-11, 3.791e-11, 6.209e-11, 9.101e-11, 1.334e-10, 1.951e-10,
  2.853e-10, 3.94e-10, 4.771e-10, 5.771e-10, 6.675e-10, 7.665e-10,
  8.504e-10, 8.924e-10, 9.363e-10, 8.923e-10, 8.411e-10, 7.646e-10,
  6.525e-10, 5.576e-10, 4.398e-10, 3.403e-10, 2.612e-10, 1.915e-10,
  1.407e-10, 1.028e-10, 7.455e-11, 5.42e-11, 3.708e-11, 2.438e-11,
  1.618e-11, 1.075e-11, 7.17e-12, 4.784e-12, 3.205e-12, 2.147e-12,
  1.44e-12, 9.654e-13, 6.469e-13, 4.332e-13, 2.891e-13, 1.926e-13,
  1.274e-13, 8.422e-14, 5.547e-14, 3.636e-14, 2.368e-14, 1.536e-14,
  9.937e-15, 6.39e-15, 4.101e-15, 2.61e-15, 1.659e-15, 1.052e-15,
```

```
    6.638e-16, 4.172e-16, 2.61e-16, 1.63e-16, 1.013e-16, 6.275e-17,
    3.879e-17, 2.383e-17, 1.461e-17, 8.918e-18, 5.43e-18, 3.301e-18,
    1.997e-18, 1.203e-18, 7.216e-19, 4.311e-19, 2.564e-19, 1.519e-19,
    8.911e-20, 5.203e-20, 3.026e-20, 1.748e-20, 9.99e-21, 5.673e-21,
    3.215e-21, 1.799e-21, 1.006e-21, 5.628e-22, 3.146e-22, 1.766e-22,
    9.94e-23, 5.614e-23, 3.206e-23, 1.841e-23, 1.071e-23, 6.366e-24,
    3.776e-24, 2.238e-24, 1.326e-24, 8.253e-25, 5.201e-25, 3.279e-25,
    2.108e-25, 1.395e-25, 9.326e-26, 6.299e-26, 4.365e-26, 3.104e-26,
    2.219e-26, 1.621e-26, 1.185e-26, 8.92e-27, 6.804e-27, 5.191e-27,
    4.041e-27
  };

  static double co[121] = {
    1.907e-07, 1.553e-07, 1.362e-07, 1.216e-07, 1.114e-07, 1.036e-07,
    9.737e-08, 9.152e-08, 8.559e-08, 7.966e-08, 7.277e-08, 6.615e-08,
    5.884e-08, 5.22e-08, 4.699e-08, 4.284e-08, 3.776e-08, 3.274e-08,
    2.845e-08, 2.479e-08, 2.246e-08, 2.054e-08, 1.991e-08, 1.951e-08,
    1.94e-08, 2.009e-08, 2.1e-08, 2.201e-08, 2.322e-08, 2.45e-08,
    2.602e-08, 2.73e-08, 2.867e-08, 2.998e-08, 3.135e-08, 3.255e-08,
    3.352e-08, 3.426e-08, 3.484e-08, 3.53e-08, 3.593e-08, 3.671e-08,
    3.759e-08, 3.945e-08, 4.192e-08, 4.49e-08, 5.03e-08, 5.703e-08,
    6.538e-08, 7.878e-08, 9.644e-08, 1.196e-07, 1.498e-07, 1.904e-07,
    2.422e-07, 3.055e-07, 3.804e-07, 4.747e-07, 5.899e-07, 7.272e-07,
    8.91e-07, 1.071e-06, 1.296e-06, 1.546e-06, 1.823e-06, 2.135e-06,
    2.44e-06, 2.714e-06, 2.967e-06, 3.189e-06, 3.391e-06, 3.58e-06,
    3.773e-06, 4.022e-06, 4.346e-06, 4.749e-06, 5.199e-06, 5.668e-06,
    6.157e-06, 6.688e-06, 7.254e-06, 7.867e-06, 8.539e-06, 9.26e-06,
    1.009e-05, 1.119e-05, 1.228e-05, 1.365e-05, 1.506e-05, 1.641e-05,
    1.784e-05, 1.952e-05, 2.132e-05, 2.323e-05, 2.531e-05, 2.754e-05,
    3.047e-05, 3.459e-05, 3.922e-05, 4.439e-05, 4.825e-05, 5.077e-05,
    5.34e-05, 5.618e-05, 5.909e-05, 6.207e-05, 6.519e-05, 6.845e-05,
    6.819e-05, 6.726e-05, 6.622e-05, 6.512e-05, 6.671e-05, 6.862e-05,
    7.048e-05, 7.264e-05, 7.3e-05, 7.3e-05, 7.3e-05, 7.3e-05, 7.3e-05
  };

  static double cof2[121] = {
    7.5e-14, 1.055e-13, 1.485e-13, 2.111e-13, 3.001e-13, 4.333e-13,
    6.269e-13, 9.221e-13, 1.364e-12, 2.046e-12, 3.093e-12, 4.703e-12,
    7.225e-12, 1.113e-11, 1.66e-11, 2.088e-11, 2.626e-11, 3.433e-11,
    4.549e-11, 5.886e-11, 7.21e-11, 8.824e-11, 1.015e-10, 1.155e-10,
    1.288e-10, 1.388e-10, 1.497e-10, 1.554e-10, 1.606e-10, 1.639e-10,
    1.64e-10, 1.64e-10, 1.596e-10, 1.542e-10, 1.482e-10, 1.382e-10,
    1.289e-10, 1.198e-10, 1.109e-10, 1.026e-10, 9.484e-11, 8.75e-11,
    8.086e-11, 7.49e-11, 6.948e-11, 6.446e-11, 5.961e-11, 5.505e-11,
    5.085e-11, 4.586e-11, 4.1e-11, 3.665e-11, 3.235e-11, 2.842e-11,
    2.491e-11, 2.11e-11, 1.769e-11, 1.479e-11, 1.197e-11, 9.631e-12,
    7.74e-12, 6.201e-12, 4.963e-12, 3.956e-12, 3.151e-12, 2.507e-12,
    1.99e-12, 1.576e-12, 1.245e-12, 9.83e-13, 7.742e-13, 6.088e-13,
    4.782e-13, 3.745e-13, 2.929e-13, 2.286e-13, 1.782e-13, 1.388e-13,
    1.079e-13, 8.362e-14, 6.471e-14, 4.996e-14, 3.85e-14, 2.96e-14,
    2.265e-14, 1.729e-14, 1.317e-14, 9.998e-15, 7.549e-15, 5.683e-15,
    4.273e-15, 3.193e-15, 2.385e-15, 1.782e-15, 1.331e-15, 9.957e-16,
    7.461e-16, 5.601e-16, 4.228e-16, 3.201e-16, 2.438e-16, 1.878e-16,
    1.445e-16, 1.111e-16, 8.544e-17, 6.734e-17, 5.341e-17, 4.237e-17,
    3.394e-17, 2.759e-17, 2.254e-17, 1.851e-17, 1.54e-17, 1.297e-17,
    1.096e-17, 9.365e-18, 8e-18, 6.938e-18, 6.056e-18, 5.287e-18,
    4.662e-18
  };

  static double f11[121] = {
    2.65e-10, 2.65e-10, 2.65e-10, 2.65e-10, 2.65e-10, 2.65e-10, 2.65e-10,
    2.65e-10, 2.65e-10, 2.65e-10, 2.65e-10, 2.65e-10, 2.635e-10, 2.536e-10,
    2.44e-10, 2.348e-10, 2.258e-10, 2.153e-10, 2.046e-10, 1.929e-10,
    1.782e-10, 1.648e-10, 1.463e-10, 1.291e-10, 1.1e-10, 8.874e-11,
    7.165e-11, 5.201e-11, 3.744e-11, 2.577e-11, 1.64e-11, 1.048e-11,
    5.993e-12, 3.345e-12, 1.839e-12, 9.264e-13, 4.688e-13, 2.329e-13,
    1.129e-13, 5.505e-14, 2.825e-14, 1.492e-14, 7.997e-15, 5.384e-15,
    3.988e-15, 2.955e-15, 2.196e-15, 1.632e-15, 1.214e-15, 9.025e-16,
    6.708e-16, 4.984e-16, 3.693e-16, 2.733e-16, 2.013e-16, 1.481e-16,
    1.087e-16, 7.945e-17, 5.782e-17, 4.195e-17, 3.038e-17, 2.19e-17,
    1.577e-17, 1.128e-17, 8.063e-18, 5.753e-18, 4.09e-18, 2.899e-18,
    2.048e-18, 1.444e-18, 1.015e-18, 7.12e-19, 4.985e-19, 3.474e-19,
    2.417e-19, 1.677e-19, 1.161e-19, 8.029e-20, 5.533e-20, 3.799e-20,
    2.602e-20, 1.776e-20, 1.209e-20, 8.202e-21, 5.522e-21, 3.707e-21,
    2.48e-21, 1.652e-21, 1.091e-21, 7.174e-22, 4.709e-22, 3.063e-22,
    1.991e-22, 1.294e-22, 8.412e-23, 5.483e-23, 3.581e-23, 2.345e-23,
    1.548e-23, 1.027e-23, 6.869e-24, 4.673e-24, 3.173e-24, 2.153e-24,
    1.461e-24, 1.028e-24, 7.302e-25, 5.188e-25, 3.739e-25, 2.753e-25,
    2.043e-25, 1.528e-25, 1.164e-25, 9.041e-26, 7.051e-26, 5.587e-26,
    4.428e-26, 3.588e-26, 2.936e-26, 2.402e-26, 1.995e-26
  };

  static double f12[121] = {
    5.45e-10, 5.45e-10, 5.45e-10, 5.45e-10, 5.45e-10, 5.45e-10, 5.45e-10,
    5.45e-10, 5.45e-10, 5.45e-10, 5.45e-10, 5.45e-10, 5.429e-10, 5.291e-10,
    5.155e-10, 5.022e-10, 4.893e-10, 4.772e-10, 4.655e-10, 4.497e-10,
    4.249e-10, 4.015e-10, 3.632e-10, 3.261e-10, 2.858e-10, 2.408e-10,
```

```
  2.03e-10, 1.685e-10, 1.4e-10, 1.163e-10, 9.65e-11, 8.02e-11, 6.705e-11,
  5.624e-11, 4.764e-11, 4.249e-11, 3.792e-11, 3.315e-11, 2.819e-11,
  2.4e-11, 1.999e-11, 1.64e-11, 1.352e-11, 1.14e-11, 9.714e-12,
  8.28e-12, 7.176e-12, 6.251e-12, 5.446e-12, 4.72e-12, 4.081e-12,
  3.528e-12, 3.08e-12, 2.699e-12, 2.359e-12, 2.111e-12, 1.901e-12,
  1.709e-12, 1.534e-12, 1.376e-12, 1.233e-12, 1.103e-12, 9.869e-13,
  8.808e-13, 7.859e-13, 7.008e-13, 6.241e-13, 5.553e-13, 4.935e-13,
  4.383e-13, 3.889e-13, 3.447e-13, 3.054e-13, 2.702e-13, 2.389e-13,
  2.11e-13, 1.862e-13, 1.643e-13, 1.448e-13, 1.274e-13, 1.121e-13,
  9.844e-14, 8.638e-14, 7.572e-14, 6.62e-14, 5.782e-14, 5.045e-14,
  4.394e-14, 3.817e-14, 3.311e-14, 2.87e-14, 2.48e-14, 2.142e-14,
  1.851e-14, 1.599e-14, 1.383e-14, 1.196e-14, 1.036e-14, 9e-15,
  7.828e-15, 6.829e-15, 5.992e-15, 5.254e-15, 4.606e-15, 4.037e-15,
  3.583e-15, 3.19e-15, 2.841e-15, 2.542e-15, 2.291e-15, 2.07e-15,
  1.875e-15, 1.71e-15, 1.57e-15, 1.442e-15, 1.333e-15, 1.232e-15,
  1.147e-15, 1.071e-15, 1.001e-15, 9.396e-16
};

static double f14[121] = {
  9e-11, 9e-11, 9e-11, 9e-11, 9e-11, 9e-11, 9e-11, 9e-11, 9e-11, 9e-11,
  9e-11, 9e-11, 9e-11, 9e-11, 9e-11, 8.91e-11, 8.73e-11, 8.46e-11,
  8.19e-11, 7.92e-11, 7.74e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11,
  7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11,
  7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11,
  7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11,
  7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11,
  7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11,
  7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11,
  7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11,
  7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11,
  7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11,
  7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11,
  7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11,
  7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11,
  7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11,
  7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11
};

static double f22[121] = {
  1.4e-10, 1.4e-10, 1.4e-10, 1.4e-10, 1.4e-10, 1.4e-10, 1.4e-10,
  1.4e-10, 1.4e-10, 1.4e-10, 1.372e-10, 1.317e-10, 1.235e-10, 1.153e-10,
  1.075e-10, 1.002e-10, 9.332e-11, 8.738e-11, 8.194e-11, 7.7e-11,
  7.165e-11, 6.753e-11, 6.341e-11, 5.971e-11, 5.6e-11, 5.229e-11,
  4.859e-11, 4.488e-11, 4.118e-11, 3.83e-11, 3.568e-11, 3.308e-11,
  3.047e-11, 2.82e-11, 2.594e-11, 2.409e-11, 2.237e-11, 2.065e-11,
  1.894e-11, 1.771e-11, 1.647e-11, 1.532e-11, 1.416e-11, 1.332e-11,
  1.246e-11, 1.161e-11, 1.087e-11, 1.017e-11, 9.471e-12, 8.853e-12,
  8.235e-12, 7.741e-12, 7.247e-12, 6.836e-12, 6.506e-12, 6.176e-12,
  5.913e-12, 5.65e-12, 5.419e-12, 5.221e-12, 5.024e-12, 4.859e-12,
  4.694e-12, 4.546e-12, 4.414e-12, 4.282e-12, 4.15e-12, 4.019e-12,
  3.903e-12, 3.805e-12, 3.706e-12, 3.607e-12, 3.508e-12, 3.41e-12,
  3.31e-12, 3.212e-12, 3.129e-12, 3.047e-12, 2.964e-12, 2.882e-12,
  2.8e-12, 2.734e-12, 2.668e-12, 2.602e-12, 2.537e-12, 2.471e-12,
  2.421e-12, 2.372e-12, 2.322e-12, 2.273e-12, 2.224e-12, 2.182e-12,
  2.141e-12, 2.1e-12, 2.059e-12, 2.018e-12, 1.977e-12, 1.935e-12,
  1.894e-12, 1.853e-12, 1.812e-12, 1.77e-12, 1.73e-12, 1.688e-12,
  1.647e-12, 1.606e-12, 1.565e-12, 1.524e-12, 1.483e-12, 1.441e-12,
  1.4e-12, 1.359e-12, 1.317e-12, 1.276e-12, 1.235e-12, 1.194e-12,
  1.153e-12, 1.112e-12, 1.071e-12, 1.029e-12, 9.883e-13
};

static double h2o[121] = {
  0.01166, 0.008269, 0.005742, 0.003845, 0.00277, 0.001897, 0.001272,
  0.000827, 0.000539, 0.0003469, 0.0001579, 3.134e-05, 1.341e-05,
  6.764e-06, 4.498e-06, 3.703e-06, 3.724e-06, 3.899e-06, 4.002e-06,
  4.122e-06, 4.277e-06, 4.438e-06, 4.558e-06, 4.673e-06, 4.763e-06,
  4.809e-06, 4.856e-06, 4.936e-06, 5.021e-06, 5.114e-06, 5.222e-06,
  5.331e-06, 5.414e-06, 5.488e-06, 5.563e-06, 5.633e-06, 5.704e-06,
  5.767e-06, 5.819e-06, 5.872e-06, 5.914e-06, 5.949e-06, 5.984e-06,
  6.015e-06, 6.044e-06, 6.073e-06, 6.104e-06, 6.136e-06, 6.167e-06,
  6.189e-06, 6.208e-06, 6.226e-06, 6.212e-06, 6.185e-06, 6.158e-06,
  6.114e-06, 6.066e-06, 6.018e-06, 5.877e-06, 5.728e-06, 5.582e-06,
  5.437e-06, 5.296e-06, 5.156e-06, 5.02e-06, 4.886e-06, 4.754e-06,
  4.625e-06, 4.498e-06, 4.374e-06, 4.242e-06, 4.096e-06, 3.955e-06,
  3.817e-06, 3.683e-06, 3.491e-06, 3.204e-06, 2.94e-06, 2.696e-06,
  2.47e-06, 2.252e-06, 2.019e-06, 1.808e-06, 1.618e-06, 1.445e-06,
  1.285e-06, 1.105e-06, 9.489e-07, 8.121e-07, 6.938e-07, 5.924e-07,
  5.04e-07, 4.288e-07, 3.648e-07, 3.103e-07, 2.642e-07, 2.252e-07,
  1.921e-07, 1.643e-07, 1.408e-07, 1.211e-07, 1.048e-07, 9.063e-08,
  7.835e-08, 6.774e-08, 5.936e-08, 5.221e-08, 4.592e-08, 4.061e-08,
  3.62e-08, 3.236e-08, 2.902e-08, 2.62e-08, 2.383e-08, 2.171e-08,
  1.989e-08, 1.823e-08, 1.684e-08, 1.562e-08, 1.449e-08, 1.351e-08
};

static double h2o2[121] = {
  1.779e-10, 7.938e-10, 8.953e-10, 8.032e-10, 6.564e-10, 5.159e-10,
  4.003e-10, 3.026e-10, 2.222e-10, 1.58e-10, 1.044e-10, 6.605e-11,
```

```
  3.413e-11, 1.453e-11, 1.062e-11, 1.009e-11, 9.597e-12, 1.175e-11,
  1.572e-11, 2.091e-11, 2.746e-11, 3.603e-11, 4.791e-11, 6.387e-11,
  8.239e-11, 1.007e-10, 1.23e-10, 1.363e-10, 1.489e-10, 1.585e-10,
  1.608e-10, 1.632e-10, 1.576e-10, 1.502e-10, 1.423e-10, 1.302e-10,
  1.192e-10, 1.085e-10, 9.795e-11, 8.854e-11, 8.057e-11, 7.36e-11,
  6.736e-11, 6.362e-11, 6.087e-11, 5.825e-11, 5.623e-11, 5.443e-11,
  5.27e-11, 5.098e-11, 4.931e-11, 4.769e-11, 4.611e-11, 4.458e-11,
  4.308e-11, 4.102e-11, 3.887e-11, 3.682e-11, 3.521e-11, 3.369e-11,
  3.224e-11, 3.082e-11, 2.946e-11, 2.814e-11, 2.687e-11, 2.566e-11,
  2.449e-11, 2.336e-11, 2.227e-11, 2.123e-11, 2.023e-11, 1.927e-11,
  1.835e-11, 1.746e-11, 1.661e-11, 1.58e-11, 1.502e-11, 1.428e-11,
  1.357e-11, 1.289e-11, 1.224e-11, 1.161e-11, 1.102e-11, 1.045e-11,
  9.895e-12, 9.369e-12, 8.866e-12, 8.386e-12, 7.922e-12, 7.479e-12,
  7.06e-12, 6.656e-12, 6.274e-12, 5.914e-12, 5.575e-12, 5.257e-12,
  4.959e-12, 4.679e-12, 4.42e-12, 4.178e-12, 3.954e-12, 3.75e-12,
  3.557e-12, 3.372e-12, 3.198e-12, 3.047e-12, 2.908e-12, 2.775e-12,
  2.653e-12, 2.544e-12, 2.442e-12, 2.346e-12, 2.26e-12, 2.183e-12,
  2.11e-12, 2.044e-12, 1.98e-12, 1.924e-12, 1.871e-12, 1.821e-12,
  1.775e-12
};

static double hcn[121] = {
  5.5e-10, 5.5e-10, 5.5e-10, 5.5e-10, 5.5e-10, 5.5e-10, 5.5e-10,
  5.5e-10, 5.5e-10, 5.5e-10, 5.5e-10, 5.498e-10, 5.495e-10, 5.493e-10,
  5.49e-10, 5.488e-10, 4.717e-10, 3.946e-10, 3.174e-10, 2.4e-10,
  1.626e-10, 1.619e-10, 1.612e-10, 1.602e-10, 1.593e-10, 1.582e-10,
  1.572e-10, 1.56e-10, 1.549e-10, 1.539e-10, 1.53e-10, 1.519e-10,
  1.506e-10, 1.487e-10, 1.467e-10, 1.449e-10, 1.43e-10, 1.413e-10,
  1.397e-10, 1.382e-10, 1.368e-10, 1.354e-10, 1.337e-10, 1.315e-10,
  1.292e-10, 1.267e-10, 1.241e-10, 1.215e-10, 1.19e-10, 1.165e-10,
  1.141e-10, 1.118e-10, 1.096e-10, 1.072e-10, 1.047e-10, 1.021e-10,
  9.968e-11, 9.739e-11, 9.539e-11, 9.339e-11, 9.135e-11, 8.898e-11,
  8.664e-11, 8.439e-11, 8.249e-11, 8.075e-11, 7.904e-11, 7.735e-11,
  7.565e-11, 7.399e-11, 7.245e-11, 7.109e-11, 6.982e-11, 6.863e-11,
  6.755e-11, 6.657e-11, 6.587e-11, 6.527e-11, 6.476e-11, 6.428e-11,
  6.382e-11, 6.343e-11, 6.307e-11, 6.272e-11, 6.238e-11, 6.205e-11,
  6.17e-11, 6.137e-11, 6.102e-11, 6.072e-11, 6.046e-11, 6.03e-11,
  6.018e-11, 6.01e-11, 6.001e-11, 5.992e-11, 5.984e-11, 5.975e-11,
  5.967e-11, 5.958e-11, 5.95e-11, 5.941e-11, 5.933e-11, 5.925e-11,
  5.916e-11, 5.908e-11, 5.899e-11, 5.891e-11, 5.883e-11, 5.874e-11,
  5.866e-11, 5.858e-11, 5.85e-11, 5.841e-11, 5.833e-11, 5.825e-11,
  5.817e-11, 5.808e-11, 5.8e-11, 5.792e-11, 5.784e-11
};

static double hno3[121] = {
  1.809e-10, 7.234e-10, 5.899e-10, 4.342e-10, 3.277e-10, 2.661e-10,
  2.35e-10, 2.267e-10, 2.389e-10, 2.651e-10, 3.255e-10, 4.099e-10,
  5.42e-10, 6.978e-10, 8.807e-10, 1.112e-09, 1.405e-09, 2.04e-09,
  3.111e-09, 4.5e-09, 5.762e-09, 7.37e-09, 7.852e-09, 8.109e-09,
  8.067e-09, 7.554e-09, 7.076e-09, 6.268e-09, 5.524e-09, 4.749e-09,
  3.909e-09, 3.223e-09, 2.517e-09, 1.942e-09, 1.493e-09, 1.122e-09,
  8.449e-10, 6.361e-10, 4.787e-10, 3.611e-10, 2.804e-10, 2.215e-10,
  1.758e-10, 1.441e-10, 1.197e-10, 9.953e-11, 8.505e-11, 7.334e-11,
  6.325e-11, 5.625e-11, 5.058e-11, 4.548e-11, 4.122e-11, 3.748e-11,
  3.402e-11, 3.088e-11, 2.8e-11, 2.536e-11, 2.293e-11, 2.072e-11,
  1.871e-11, 1.687e-11, 1.52e-11, 1.368e-11, 1.23e-11, 1.105e-11,
  9.922e-12, 8.898e-12, 7.972e-12, 7.139e-12, 6.385e-12, 5.708e-12,
  5.099e-12, 4.549e-12, 4.056e-12, 3.613e-12, 3.216e-12, 2.862e-12,
  2.544e-12, 2.259e-12, 2.004e-12, 1.776e-12, 1.572e-12, 1.391e-12,
  1.227e-12, 1.082e-12, 9.528e-13, 8.379e-13, 7.349e-13, 6.436e-13,
  5.634e-13, 4.917e-13, 4.291e-13, 3.745e-13, 3.267e-13, 2.854e-13,
  2.494e-13, 2.181e-13, 1.913e-13, 1.68e-13, 1.479e-13, 1.31e-13,
  1.159e-13, 1.025e-13, 9.067e-14, 8.113e-14, 7.281e-14, 6.535e-14,
  5.892e-14, 5.348e-14, 4.867e-14, 4.439e-14, 4.073e-14, 3.76e-14,
  3.476e-14, 3.229e-14, 3e-14, 2.807e-14, 2.635e-14, 2.473e-14,
  2.332e-14
};

static double hno4[121] = {
  6.118e-12, 3.594e-12, 2.807e-12, 3.04e-12, 4.458e-12, 7.986e-12,
  1.509e-11, 2.661e-11, 3.738e-11, 4.652e-11, 4.429e-11, 3.992e-11,
  3.347e-11, 3.005e-11, 3.173e-11, 4.055e-11, 5.812e-11, 8.489e-11,
  1.19e-10, 1.482e-10, 1.766e-10, 2.103e-10, 2.35e-10, 2.598e-10,
  2.801e-10, 2.899e-10, 3e-10, 2.817e-10, 2.617e-10, 2.332e-10,
  1.933e-10, 1.605e-10, 1.232e-10, 9.285e-11, 6.941e-11, 4.951e-11,
  3.539e-11, 2.402e-11, 1.522e-11, 9.676e-12, 6.056e-12, 3.745e-12,
  2.34e-12, 1.463e-12, 9.186e-13, 5.769e-13, 3.322e-13, 1.853e-13,
  1.035e-13, 7.173e-14, 5.382e-14, 4.036e-14, 3.401e-14, 2.997e-14,
  2.635e-14, 2.316e-14, 2.034e-14, 1.783e-14, 1.56e-14, 1.363e-14,
  1.19e-14, 1.037e-14, 9.032e-15, 7.846e-15, 6.813e-15, 5.912e-15,
  5.121e-15, 4.431e-15, 3.829e-15, 3.306e-15, 2.851e-15, 2.456e-15,
  2.114e-15, 1.816e-15, 1.559e-15, 1.337e-15, 1.146e-15, 9.811e-16,
  8.389e-16, 7.162e-16, 6.109e-16, 5.203e-16, 4.425e-16, 3.76e-16,
  3.184e-16, 2.692e-16, 2.274e-16, 1.917e-16, 1.61e-16, 1.35e-16,
  1.131e-16, 9.437e-17, 7.874e-17, 6.57e-17, 5.481e-17, 4.579e-17,
  3.828e-17, 3.204e-17, 2.691e-17, 2.264e-17, 1.912e-17, 1.626e-17,
  1.382e-17, 1.174e-17, 9.972e-18, 8.603e-18, 7.45e-18, 6.453e-18,
```

```
  5.623e-18, 4.944e-18, 4.361e-18, 3.859e-18, 3.443e-18, 3.096e-18,
  2.788e-18, 2.528e-18, 2.293e-18, 2.099e-18, 1.929e-18, 1.773e-18,
  1.64e-18
};

static double hocl[121] = {
  1.056e-12, 1.194e-12, 1.35e-12, 1.531e-12, 1.737e-12, 1.982e-12,
  2.263e-12, 2.599e-12, 2.991e-12, 3.459e-12, 4.012e-12, 4.662e-12,
  5.438e-12, 6.35e-12, 7.425e-12, 8.686e-12, 1.016e-11, 1.188e-11,
  1.389e-11, 1.659e-11, 2.087e-11, 2.621e-11, 3.265e-11, 4.064e-11,
  4.859e-11, 5.441e-11, 6.09e-11, 6.373e-11, 6.611e-11, 6.94e-11,
  7.44e-11, 7.97e-11, 8.775e-11, 9.722e-11, 1.064e-10, 1.089e-10,
  1.114e-10, 1.106e-10, 1.053e-10, 1.004e-10, 9.006e-11, 7.778e-11,
  6.739e-11, 5.636e-11, 4.655e-11, 3.845e-11, 3.042e-11, 2.368e-11,
  1.845e-11, 1.442e-11, 1.127e-11, 8.814e-12, 6.544e-12, 4.763e-12,
  3.449e-12, 2.612e-12, 1.999e-12, 1.526e-12, 1.16e-12, 8.793e-13,
  6.655e-13, 5.017e-13, 3.778e-13, 2.829e-13, 2.117e-13, 1.582e-13,
  1.178e-13, 8.755e-14, 6.486e-14, 4.799e-14, 3.54e-14, 2.606e-14,
  1.916e-14, 1.403e-14, 1.026e-14, 7.48e-15, 5.446e-15, 3.961e-15,
  2.872e-15, 2.076e-15, 1.498e-15, 1.077e-15, 7.726e-16, 5.528e-16,
  3.929e-16, 2.785e-16, 1.969e-16, 1.386e-16, 9.69e-17, 6.747e-17,
  4.692e-17, 3.236e-17, 2.232e-17, 1.539e-17, 1.061e-17, 7.332e-18,
  5.076e-18, 3.522e-18, 2.461e-18, 1.726e-18, 1.22e-18, 8.75e-19,
  6.264e-19, 4.482e-19, 3.207e-19, 2.368e-19, 1.762e-19, 1.312e-19,
  9.891e-20, 7.595e-20, 5.87e-20, 4.567e-20, 3.612e-20, 2.904e-20,
  2.343e-20, 1.917e-20, 1.568e-20, 1.308e-20, 1.1e-20, 9.25e-21,
  7.881e-21
};

static double n2o[121] = {
  3.17e-07, 3.17e-07, 3.17e-07, 3.17e-07, 3.17e-07, 3.17e-07, 3.17e-07,
  3.17e-07, 3.17e-07, 3.17e-07, 3.124e-07, 3.077e-07, 3.03e-07,
  2.984e-07, 2.938e-07, 2.892e-07, 2.847e-07, 2.779e-07, 2.705e-07,
  2.631e-07, 2.557e-07, 2.484e-07, 2.345e-07, 2.201e-07, 2.01e-07,
  1.754e-07, 1.532e-07, 1.329e-07, 1.154e-07, 1.003e-07, 8.735e-08,
  7.617e-08, 6.512e-08, 5.547e-08, 4.709e-08, 3.915e-08, 3.259e-08,
  2.738e-08, 2.327e-08, 1.98e-08, 1.711e-08, 1.493e-08, 1.306e-08,
  1.165e-08, 1.049e-08, 9.439e-09, 8.375e-09, 7.391e-09, 6.525e-09,
  5.759e-09, 5.083e-09, 4.485e-09, 3.953e-09, 3.601e-09, 3.27e-09,
  2.975e-09, 2.757e-09, 2.556e-09, 2.37e-09, 2.195e-09, 2.032e-09,
  1.912e-09, 1.79e-09, 1.679e-09, 1.572e-09, 1.482e-09, 1.402e-09,
  1.326e-09, 1.254e-09, 1.187e-09, 1.127e-09, 1.071e-09, 1.02e-09,
  9.673e-10, 9.193e-10, 8.752e-10, 8.379e-10, 8.017e-10, 7.66e-10,
  7.319e-10, 7.004e-10, 6.721e-10, 6.459e-10, 6.199e-10, 5.942e-10,
  5.703e-10, 5.488e-10, 5.283e-10, 5.082e-10, 4.877e-10, 4.696e-10,
  4.52e-10, 4.355e-10, 4.198e-10, 4.039e-10, 3.888e-10, 3.754e-10,
  3.624e-10, 3.499e-10, 3.381e-10, 3.267e-10, 3.163e-10, 3.058e-10,
  2.959e-10, 2.864e-10, 2.77e-10, 2.686e-10, 2.604e-10, 2.534e-10,
  2.462e-10, 2.386e-10, 2.318e-10, 2.247e-10, 2.189e-10, 2.133e-10,
  2.071e-10, 2.014e-10, 1.955e-10, 1.908e-10, 1.86e-10, 1.817e-10
};

static double n2o5[121] = {
  1.231e-11, 3.035e-12, 1.702e-12, 9.877e-13, 8.081e-13, 9.039e-13,
  1.169e-12, 1.474e-12, 1.651e-12, 1.795e-12, 1.998e-12, 2.543e-12,
  4.398e-12, 7.698e-12, 1.28e-11, 2.131e-11, 3.548e-11, 5.894e-11,
  7.645e-11, 1.089e-10, 1.391e-10, 1.886e-10, 2.386e-10, 2.986e-10,
  3.487e-10, 3.994e-10, 4.5e-10, 4.6e-10, 4.591e-10, 4.1e-10, 3.488e-10,
  2.846e-10, 2.287e-10, 1.696e-10, 1.011e-10, 6.428e-11, 4.324e-11,
  2.225e-11, 6.214e-12, 3.608e-12, 8.793e-13, 4.491e-13, 1.04e-13,
  6.1e-14, 3.436e-14, 6.671e-15, 1.171e-15, 5.848e-16, 1.212e-16,
  1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16,
  1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16,
  1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16,
  1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16,
  1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16,
  1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16,
  1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16,
  1e-16, 1e-16
};

static double nh3[121] = {
  1e-10, 1e-10, 1e-10, 1e-10, 1e-10, 1e-10, 1e-10, 1e-10, 1e-10, 1e-10,
  1e-10, 1e-10, 1e-10, 1e-10, 9.444e-11, 8.488e-11, 7.241e-11, 5.785e-11,
  4.178e-11, 3.018e-11, 2.18e-11, 1.574e-11, 1.137e-11, 8.211e-12,
  5.973e-12, 4.327e-12, 3.118e-12, 2.234e-12, 1.573e-12, 1.04e-12,
  6.762e-13, 4.202e-13, 2.406e-13, 1.335e-13, 6.938e-14, 3.105e-14,
  1.609e-14, 1.033e-14, 6.432e-15, 4.031e-15, 2.555e-15, 1.656e-15,
  1.115e-15, 7.904e-16, 5.63e-16, 4.048e-16, 2.876e-16, 2.004e-16,
  1.356e-16, 9.237e-17, 6.235e-17, 4.223e-17, 3.009e-17, 2.328e-17,
  2.002e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17,
  1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17,
  1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17,
  1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17,
  1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17,
  1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17,
  1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17,
```

```
    1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17,
    1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17,
    1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17,
    1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17,
    1.914e-17
  };

  static double no[121] = {
    2.586e-10, 4.143e-11, 1.566e-11, 9.591e-12, 8.088e-12, 8.462e-12,
    1.013e-11, 1.328e-11, 1.855e-11, 2.678e-11, 3.926e-11, 5.464e-11,
    7.012e-11, 8.912e-11, 1.127e-10, 1.347e-10, 1.498e-10, 1.544e-10,
    1.602e-10, 1.824e-10, 2.078e-10, 2.366e-10, 2.691e-10, 5.141e-10,
    8.259e-10, 1.254e-09, 1.849e-09, 2.473e-09, 3.294e-09, 4.16e-09,
    5.095e-09, 6.11e-09, 6.93e-09, 7.888e-09, 8.903e-09, 9.713e-09,
    1.052e-08, 1.115e-08, 1.173e-08, 1.21e-08, 1.228e-08, 1.239e-08,
    1.231e-08, 1.213e-08, 1.192e-08, 1.138e-08, 1.085e-08, 1.008e-08,
    9.224e-09, 8.389e-09, 7.262e-09, 6.278e-09, 5.335e-09, 4.388e-09,
    3.589e-09, 2.761e-09, 2.129e-09, 1.633e-09, 1.243e-09, 9.681e-10,
    8.355e-10, 7.665e-10, 7.442e-10, 8.584e-10, 9.732e-10, 1.063e-09,
    1.163e-09, 1.286e-09, 1.472e-09, 1.707e-09, 2.032e-09, 2.474e-09,
    2.977e-09, 3.506e-09, 4.102e-09, 5.013e-09, 6.493e-09, 8.414e-09,
    1.077e-08, 1.367e-08, 1.777e-08, 2.625e-08, 3.926e-08, 5.545e-08,
    7.195e-08, 9.464e-08, 1.404e-07, 2.183e-07, 3.329e-07, 4.535e-07,
    6.158e-07, 8.187e-07, 1.075e-06, 1.422e-06, 1.979e-06, 2.71e-06,
    3.58e-06, 4.573e-06, 5.951e-06, 7.999e-06, 1.072e-05, 1.372e-05,
    1.697e-05, 2.112e-05, 2.643e-05, 3.288e-05, 3.994e-05, 4.794e-05,
    5.606e-05, 6.383e-05, 7.286e-05, 8.156e-05, 8.883e-05, 9.469e-05,
    9.848e-05, 0.0001023, 0.0001066, 0.0001115, 0.0001145, 0.0001142,
    0.0001133
  };

  static double no2[121] = {
    3.036e-09, 2.945e-10, 9.982e-11, 5.069e-11, 3.485e-11, 2.982e-11,
    2.947e-11, 3.164e-11, 3.714e-11, 4.586e-11, 6.164e-11, 8.041e-11,
    9.982e-11, 1.283e-10, 1.73e-10, 2.56e-10, 3.909e-10, 5.959e-10,
    9.081e-10, 1.384e-09, 1.788e-09, 2.189e-09, 2.686e-09, 3.091e-09,
    3.49e-09, 3.796e-09, 4.2e-09, 5.103e-09, 6.005e-09, 6.3e-09, 6.706e-09,
    7.07e-09, 7.434e-09, 7.663e-09, 7.788e-09, 7.8e-09, 7.597e-09,
    7.482e-09, 7.227e-09, 6.403e-09, 5.585e-09, 4.606e-09, 3.703e-09,
    2.984e-09, 2.183e-09, 1.48e-09, 8.441e-10, 5.994e-10, 3.799e-10,
    2.751e-10, 1.927e-10, 1.507e-10, 1.102e-10, 6.971e-11, 5.839e-11,
    3.904e-11, 3.087e-11, 2.176e-11, 1.464e-11, 1.209e-11, 8.497e-12,
    6.477e-12, 4.371e-12, 2.914e-12, 2.424e-12, 1.753e-12, 1.35e-12,
    9.417e-13, 6.622e-13, 5.148e-13, 3.841e-13, 3.446e-13, 3.01e-13,
    2.551e-13, 2.151e-13, 1.829e-13, 1.64e-13, 1.475e-13, 1.352e-13,
    1.155e-13, 9.963e-14, 9.771e-14, 9.577e-14, 9.384e-14, 9.186e-14,
    9e-14, 9e-14, 9e-14, 9e-14, 9e-14, 9e-14, 9e-14, 9e-14, 9e-14, 9e-14,
    9e-14, 9e-14, 9e-14, 9e-14, 9e-14, 9e-14, 9e-14, 9e-14, 9e-14, 9e-14,
    9e-14, 9e-14, 9e-14, 9e-14, 9e-14, 9e-14, 9e-14, 9e-14, 9e-14, 9e-14,
    9e-14, 9e-14, 9e-14, 9e-14, 9e-14, 9e-14
  };

  static double o3[121] = {
    2.218e-08, 3.394e-08, 3.869e-08, 4.219e-08, 4.501e-08, 4.778e-08,
    5.067e-08, 5.402e-08, 5.872e-08, 6.521e-08, 7.709e-08, 9.461e-08,
    1.269e-07, 1.853e-07, 2.723e-07, 3.964e-07, 5.773e-07, 8.2e-07,
    1.155e-06, 1.59e-06, 2.076e-06, 2.706e-06, 3.249e-06, 3.848e-06,
    4.459e-06, 4.986e-06, 5.573e-06, 5.958e-06, 6.328e-06, 6.661e-06,
    6.9e-06, 7.146e-06, 7.276e-06, 7.374e-06, 7.447e-06, 7.383e-06,
    7.321e-06, 7.161e-06, 6.879e-06, 6.611e-06, 6.216e-06, 5.765e-06,
    5.355e-06, 4.905e-06, 4.471e-06, 4.075e-06, 3.728e-06, 3.413e-06,
    3.125e-06, 2.856e-06, 2.607e-06, 2.379e-06, 2.17e-06, 1.978e-06,
    1.8e-06, 1.646e-06, 1.506e-06, 1.376e-06, 1.233e-06, 1.102e-06,
    9.839e-07, 8.771e-07, 7.814e-07, 6.947e-07, 6.102e-07, 5.228e-07,
    4.509e-07, 3.922e-07, 3.501e-07, 3.183e-07, 2.909e-07, 2.686e-07,
    2.476e-07, 2.284e-07, 2.109e-07, 2.003e-07, 2.013e-07, 2.022e-07,
    2.032e-07, 2.042e-07, 2.097e-07, 2.361e-07, 2.656e-07, 2.989e-07,
    3.37e-07, 3.826e-07, 4.489e-07, 5.26e-07, 6.189e-07, 7.312e-07,
    8.496e-07, 8.444e-07, 8.392e-07, 8.339e-07, 8.286e-07, 8.234e-07,
    8.181e-07, 8.129e-07, 8.077e-07, 8.026e-07, 6.918e-07, 5.176e-07,
    3.865e-07, 2.885e-07, 2.156e-07, 1.619e-07, 1.219e-07, 9.161e-08,
    6.972e-08, 5.399e-08, 3.498e-08, 2.111e-08, 1.322e-08, 8.482e-09,
    5.527e-09, 3.423e-09, 2.071e-09, 1.314e-09, 8.529e-10, 5.503e-10,
    3.665e-10
  };

  static double ocs[121] = {
    6e-10, 6e-10, 6e-10, 6e-10, 6e-10, 6e-10, 6e-10, 6e-10, 6e-10, 5.997e-10,
    5.989e-10, 5.881e-10, 5.765e-10, 5.433e-10, 5.074e-10, 4.567e-10,
    4.067e-10, 3.601e-10, 3.093e-10, 2.619e-10, 2.232e-10, 1.805e-10,
    1.46e-10, 1.187e-10, 8.03e-11, 5.435e-11, 3.686e-11, 2.217e-11,
    1.341e-11, 8.756e-12, 4.511e-12, 2.37e-12, 1.264e-12, 8.28e-13,
    5.263e-13, 3.209e-13, 1.717e-13, 9.068e-14, 4.709e-14, 2.389e-14,
    1.236e-14, 1.127e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14,
    1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14,
    1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14,
    1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14,
```

```
    1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14,
    1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14,
    1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14,
    1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14,
    1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14,
    1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14,
    1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14,
    1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14,
    1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14,
    1.091e-14, 1.091e-14, 1.091e-14
  };

  static double sf6[121] = {
    4.103e-12, 4.103e-12, 4.103e-12, 4.103e-12, 4.103e-12, 4.103e-12,
    4.103e-12, 4.103e-12, 4.103e-12, 4.087e-12, 4.064e-12, 4.023e-12,
    3.988e-12, 3.941e-12, 3.884e-12, 3.755e-12, 3.622e-12, 3.484e-12,
    3.32e-12, 3.144e-12, 2.978e-12, 2.811e-12, 2.653e-12, 2.489e-12,
    2.332e-12, 2.199e-12, 2.089e-12, 2.013e-12, 1.953e-12, 1.898e-12,
    1.859e-12, 1.826e-12, 1.798e-12, 1.776e-12, 1.757e-12, 1.742e-12,
    1.728e-12, 1.717e-12, 1.707e-12, 1.698e-12, 1.691e-12, 1.685e-12,
    1.679e-12, 1.675e-12, 1.671e-12, 1.668e-12, 1.665e-12, 1.663e-12,
    1.661e-12, 1.659e-12, 1.658e-12, 1.657e-12, 1.656e-12, 1.655e-12,
    1.654e-12, 1.653e-12, 1.653e-12, 1.652e-12, 1.652e-12, 1.652e-12,
    1.651e-12, 1.651e-12, 1.651e-12, 1.651e-12, 1.651e-12, 1.651e-12,
    1.651e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12,
    1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12,
    1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12,
    1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12,
    1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12,
    1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12,
    1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12,
    1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12
  };

  static double so2[121] = {
    1e-10, 1e-10, 1e-10, 1e-10, 1e-10, 1e-10, 1e-10, 1e-10, 1e-10, 1e-10,
    1e-10, 1e-10, 9.867e-11, 9.537e-11, 9e-11, 8.404e-11, 7.799e-11,
    7.205e-11, 6.616e-11, 6.036e-11, 5.475e-11, 5.007e-11, 4.638e-11,
    4.346e-11, 4.055e-11, 3.763e-11, 3.471e-11, 3.186e-11, 2.905e-11,
    2.631e-11, 2.358e-11, 2.415e-11, 2.949e-11, 3.952e-11, 5.155e-11,
    6.76e-11, 8.741e-11, 1.099e-10, 1.278e-10, 1.414e-10, 1.512e-10,
    1.607e-10, 1.699e-10, 1.774e-10, 1.832e-10, 1.871e-10, 1.907e-10,
    1.943e-10, 1.974e-10, 1.993e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10,
    2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10,
    2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10,
    2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10,
    2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10,
    2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10,
    2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10,
    2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10
  };

  static int ig_co2 = -999;

  double co2, *q[NG] = { NULL };

  int ig, ip, iw, iz;

  /* Find emitter index of CO2... */
  if (ig_co2 == -999)
    ig_co2 = find_emitter(ctl, "CO2");

  /* Identify variable... */
  for (ig = 0; ig < ctl->ng; ig++) {
    q[ig] = NULL;
    if (strcasecmp(ctl->emitter[ig], "C2H2") == 0)
      q[ig] = c2h2;
    if (strcasecmp(ctl->emitter[ig], "C2H6") == 0)
      q[ig] = c2h6;
    if (strcasecmp(ctl->emitter[ig], "CCl4") == 0)
      q[ig] = ccl4;
    if (strcasecmp(ctl->emitter[ig], "CH4") == 0)
      q[ig] = ch4;
    if (strcasecmp(ctl->emitter[ig], "ClO") == 0)
      q[ig] = clo;
    if (strcasecmp(ctl->emitter[ig], "ClONO2") == 0)
      q[ig] = clono2;
    if (strcasecmp(ctl->emitter[ig], "CO") == 0)
      q[ig] = co;
    if (strcasecmp(ctl->emitter[ig], "COF2") == 0)
      q[ig] = cof2;
    if (strcasecmp(ctl->emitter[ig], "F11") == 0)
      q[ig] = f11;
    if (strcasecmp(ctl->emitter[ig], "F12") == 0)
      q[ig] = f12;
    if (strcasecmp(ctl->emitter[ig], "F14") == 0)
      q[ig] = f14;
```

```
    if (strcasecmp(ctl->emitter[ig], "F22") == 0)
      q[ig] = f22;
    if (strcasecmp(ctl->emitter[ig], "H2O") == 0)
      q[ig] = h2o;
    if (strcasecmp(ctl->emitter[ig], "H2O2") == 0)
      q[ig] = h2o2;
    if (strcasecmp(ctl->emitter[ig], "HCN") == 0)
      q[ig] = hcn;
    if (strcasecmp(ctl->emitter[ig], "HNO3") == 0)
      q[ig] = hno3;
    if (strcasecmp(ctl->emitter[ig], "HNO4") == 0)
      q[ig] = hno4;
    if (strcasecmp(ctl->emitter[ig], "HOCl") == 0)
      q[ig] = hocl;
    if (strcasecmp(ctl->emitter[ig], "N2O") == 0)
      q[ig] = n2o;
    if (strcasecmp(ctl->emitter[ig], "N2O5") == 0)
      q[ig] = n2o5;
    if (strcasecmp(ctl->emitter[ig], "NH3") == 0)
      q[ig] = nh3;
    if (strcasecmp(ctl->emitter[ig], "NO") == 0)
      q[ig] = no;
    if (strcasecmp(ctl->emitter[ig], "NO2") == 0)
      q[ig] = no2;
    if (strcasecmp(ctl->emitter[ig], "O3") == 0)
      q[ig] = o3;
    if (strcasecmp(ctl->emitter[ig], "OCS") == 0)
      q[ig] = ocs;
    if (strcasecmp(ctl->emitter[ig], "SF6") == 0)
      q[ig] = sf6;
    if (strcasecmp(ctl->emitter[ig], "SO2") == 0)
      q[ig] = so2;
  }

  /* Loop over atmospheric data points... */
  for (ip = 0; ip < atm->np; ip++) {

    /* Get altitude index... */
    iz = locate(z, 121, atm->z[ip]);

    /* Interpolate pressure... */
    atm->p[ip] = EXP(z[iz], pre[iz], z[iz + 1], pre[iz + 1], atm->z[ip]);

    /* Interpolate temperature... */
    atm->t[ip] = LIN(z[iz], tem[iz], z[iz + 1], tem[iz + 1], atm->z[ip]);

    /* Interpolate trace gases... */
    for (ig = 0; ig < ctl->ng; ig++)
      if (q[ig] != NULL)
        atm->q[ig][ip] =
          LIN(z[iz], q[ig][iz], z[iz + 1], q[ig][iz + 1], atm->z[ip]);
      else
        atm->q[ig][ip] = 0;

    /* Set CO2... */
    if (ig_co2 >= 0) {
      co2 =
        371.789948e-6 + 2.026214e-6 * (atm->time[ip] - 63158400.) / 31557600.;
      atm->q[ig_co2][ip] = co2;
    }

    /* Set extinction to zero... */
    for (iw = 0; iw < ctl->nw; iw++)
      atm->k[iw][ip] = 0;
  }
}
```

Here is the call graph for this function:

5.31.2.6 double ctmco2 (double *nu,* double *p,* double *t,* double *u*)

Compute carbon dioxide continuum (optical depth).

Definition at line 874 of file jurassic.c.

```
              {

  static double co2296[2001] = { 9.3388e-5, 9.7711e-5, 1.0224e-4, 1.0697e-4,
    1.1193e-4, 1.1712e-4, 1.2255e-4, 1.2824e-4, 1.3419e-4, 1.4043e-4,
```

```
1.4695e-4, 1.5378e-4, 1.6094e-4, 1.6842e-4, 1.7626e-4, 1.8447e-4,
1.9307e-4, 2.0207e-4, 2.1149e-4, 2.2136e-4, 2.3169e-4, 2.4251e-4,
2.5384e-4, 2.657e-4, 2.7813e-4, 2.9114e-4, 3.0477e-4, 3.1904e-4,
3.3399e-4, 3.4965e-4, 3.6604e-4, 3.8322e-4, 4.0121e-4, 4.2006e-4,
4.398e-4, 4.6047e-4, 4.8214e-4, 5.0483e-4, 5.286e-4, 5.535e-4,
5.7959e-4, 6.0693e-4, 6.3557e-4, 6.6558e-4, 6.9702e-4, 7.2996e-4,
7.6449e-4, 8.0066e-4, 8.3856e-4, 8.7829e-4, 9.1991e-4, 9.6354e-4,
.0010093, .0010572, .0011074, .00116, .0012152, .001273,
.0013336, .0013972, .0014638, .0015336, .0016068, .0016835,
.001764, .0018483, .0019367, .0020295, .0021267, .0022286,
.0023355, .0024476, .0025652, .0026885, .0028178, .0029534,
.0030956, .0032448, .0034012, .0035654, .0037375, .0039181,
.0041076, .0043063, .0045148, .0047336, .0049632, .005204,
.0054567, .0057219, .0060002, .0062923, .0065988, .0069204,
.007258, .0076123, .0079842, .0083746, .0087844, .0092146,
.0096663, .01014, .010638, .011161, .01171, .012286, .012891,
.013527, .014194, .014895, .015631, .016404, .017217, .01807,
.018966, .019908, .020897, .021936, .023028, .024176, .025382,
.026649, .027981, .02938, .030851, .032397, .034023, .035732,
.037528, .039416, .041402, .04349, .045685, .047994, .050422,
.052975, .055661, .058486, .061458, .064584, .067873, .071334,
.074975, .078807, .082839, .087082, .091549, .096249, .1012,
.10641, .11189, .11767, .12375, .13015, .13689, .14399, .15147,
.15935, .16765, .17639, .18561, .19531, .20554, .21632, .22769,
.23967, .25229, .2656, .27964, .29443, .31004, .3265, .34386,
.36218, .3815, .40188, .42339, .44609, .47004, .49533, .52202,
.5502, .57995, .61137, .64455, .6796, .71663, .75574, .79707,
.84075, .88691, .9357, .98728, 1.0418, 1.0995, 1.1605, 1.225,
1.2932, 1.3654, 1.4418, 1.5227, 1.6083, 1.6989, 1.7948, 1.8964,
2.004, 2.118, 2.2388, 2.3668, 2.5025, 2.6463, 2.7988, 2.9606,
3.1321, 3.314, 3.5071, 3.712, 3.9296, 4.1605, 4.4058, 4.6663,
4.9431, 5.2374, 5.5501, 5.8818, 6.2353, 6.6114, 7.0115, 7.4372,
7.8905, 8.3731, 8.8871, 9.4349, 10.019, 10.641, 11.305, 12.013,
12.769, 13.576, 14.437, 15.358, 16.342, 17.39, 18.513, 19.716,
21.003, 22.379, 23.854, 25.436, 27.126, 28.942, 30.89, 32.973,
35.219, 37.634, 40.224, 43.021, 46.037, 49.29, 52.803, 56.447,
60.418, 64.792, 69.526, 74.637, 80.182, 86.193, 92.713, 99.786,
107.47, 115.84, 124.94, 134.86, 145.69, 157.49, 170.3, 184.39,
199.83, 216.4, 234.55, 254.72, 276.82, 299.85, 326.16, 354.99,
386.51, 416.68, 449.89, 490.12, 534.35, 578.25, 632.26, 692.61,
756.43, 834.75, 924.11, 1016.9, 996.96, 1102.7, 1219.2, 1351.9,
1494.3, 1654.1, 1826.5, 2027.9, 2249., 2453.8, 2714.4, 2999.4,
3209.5, 3509., 3840.4, 3907.5, 4190.7, 4533.5, 4648.3, 5059.1,
5561.6, 6191.4, 6820.8, 7905.9, 9362.2, 2431.3, 2211.3, 2046.8,
2023.8, 1985.9, 1905.9, 1491.1, 1369.8, 1262.2, 1200.7, 887.74,
820.25, 885.23, 887.21, 816.73, 1126.9, 1216.2, 1272.4, 1579.5,
1634.2, 1656.3, 1657.9, 1789.5, 1670.8, 1509.5, 8474.6, 7489.2,
6793.6, 6117., 5574.1, 5141.2, 5084.6, 4745.1, 4413.2, 4102.8,
4024.7, 3715., 3398.6, 3100.8, 2900.4, 2629.2, 2374., 2144.7,
1955.8, 1760.8, 1591.2, 1435.2, 1296.2, 1174., 1065.1, 967.76,
999.48, 897.45, 809.23, 732.77, 670.26, 611.93, 560.11, 518.77,
476.84, 438.8, 408.48, 380.21, 349.24, 322.71, 296.65, 272.85,
251.96, 232.04, 213.88, 197.69, 182.41, 168.41, 155.79, 144.05,
133.31, 123.48, 114.5, 106.21, 98.591, 91.612, 85.156, 79.204,
73.719, 68.666, 63.975, 59.637, 56.35, 52.545, 49.042, 45.788,
42.78, 39.992, 37.441, 35.037, 32.8, 30.744, 28.801, 26.986,
25.297, 23.731, 22.258, 20.883, 19.603, 18.403, 17.295, 16.249,
15.271, 14.356, 13.501, 12.701, 11.954, 11.254, 10.6, 9.9864,
9.4118, 8.8745, 8.3714, 7.8997, 7.4578, 7.0446, 6.6573, 6.2949,
5.9577, 5.6395, 5.3419, 5.063, 4.8037, 4.5608, 4.3452, 4.1364,
3.9413, 3.7394, 3.562, 3.3932, 3.2325, 3.0789, 2.9318, 2.7898,
2.6537, 2.5225, 2.3958, 2.2305, 2.1215, 2.0245, 1.9427, 1.8795,
1.8336, 1.7604, 1.7016, 1.6419, 1.5282, 1.4611, 1.3443, 1.27,
1.1675, 1.0824, 1.0534, .99833, .95854, .92981, .90887, .89346,
.88113, .87068, .86102, .85096, .88262, .86151, .83565, .80518,
.77045, .73736, .74744, .74954, .75773, .82267, .83493, .89402,
.89725, .93426, .95564, .94045, .94174, .93404, .92035, .90456,
.88621, .86673, .78117, .7515, .72056, .68822, .65658, .62764,
.55984, .55598, .57407, .60963, .63763, .66198, .61132, .60972,
.52496, .50649, .41872, .3964, .32422, .27276, .24048, .23772,
.2286, .22711, .23999, .32038, .34371, .36621, .38561, .39953,
.40636, .44913, .42716, .3919, .35477, .33935, .3351, .39746,
.40993, .49398, .49956, .56157, .54742, .57295, .57386, .55417,
.50745, .471, .43446, .39102, .34993, .31269, .27888, .24912,
.22291, .19994, .17972, .16197, .14633, .13252, .12029, .10942,
.099745, .091118, .083404, .076494, .070292, .064716, .059697,
.055173, .051093, .047411, .044089, .041092, .038392, .035965,
.033789, .031846, .030122, .028607, .02729, .026169, .025209,
.024405, .023766, .023288, .022925, .022716, .022681, .022685,
.022768, .023133, .023325, .023486, .024004, .024126, .024083,
.023785, .024023, .023029, .021649, .021108, .019454, .017809,
.017292, .016635, .017037, .018068, .018977, .018756, .017847,
.016557, .016142, .014459, .012869, .012381, .010875, .0098701,
.009285, .0091698, .0091701, .0096145, .010553, .01106, .012613,
.014362, .015017, .016507, .017741, .01768, .017784, .0171,
.016357, .016172, .017257, .018978, .020935, .021741, .023567,
.025183, .025589, .026732, .027648, .028278, .028215, .02856,
```

```
    .029015, .029062, .028851, .028497, .027825, .027801, .026523,
    .02487, .022967, .022168, .020194, .018605, .017903, .018439,
    .019697, .020311, .020855, .020057, .018608, .016738, .015963,
    .013844, .011801, .011134, .0097573, .0086007, .0086226,
    .0083721, .0090978, .0097616, .0098426, .011317, .012853, .01447,
    .014657, .015771, .016351, .016079, .014829, .013431, .013185,
    .013207, .01448, .016176, .017971, .018265, .019526, .020455,
    .019797, .019802, .0194, .018176, .017505, .016197, .015339,
    .014401, .013213, .012203, .011186, .010236, .0093288, .0084854,
    .0076837, .0069375, .0062614, .0056628, .0051153, .0046015,
    .0041501, .003752, .0033996, .0030865, .0028077, .0025586,
    .0023355, .0021353, .0019553, .0017931, .0016466, .0015141,
    .0013941, .0012852, .0011862, .0010962, .0010142, 9.3935e-4,
    8.71e-4, 8.0851e-4, 7.5132e-4, 6.9894e-4, 6.5093e-4, 6.0689e-4,
    5.6647e-4, 5.2935e-4, 4.9525e-4, 4.6391e-4, 4.3509e-4, 4.086e-4,
    3.8424e-4, 3.6185e-4, 3.4126e-4, 3.2235e-4, 3.0498e-4, 2.8904e-4,
    2.7444e-4, 2.6106e-4, 2.4883e-4, 2.3766e-4, 2.275e-4, 2.1827e-4,
    2.0992e-4, 2.0239e-4, 1.9563e-4, 1.896e-4, 1.8427e-4, 1.796e-4,
    1.7555e-4, 1.7209e-4, 1.692e-4, 1.6687e-4, 1.6505e-4, 1.6375e-4,
    1.6294e-4, 1.6261e-4, 1.6274e-4, 1.6334e-4, 1.6438e-4, 1.6587e-4,
    1.678e-4, 1.7017e-4, 1.7297e-4, 1.762e-4, 1.7988e-4, 1.8399e-4,
    1.8855e-4, 1.9355e-4, 1.9902e-4, 2.0494e-4, 2.1134e-4, 2.1823e-4,
    2.2561e-4, 2.335e-4, 2.4192e-4, 2.5088e-4, 2.604e-4, 2.705e-4,
    2.8119e-4, 2.9251e-4, 3.0447e-4, 3.171e-4, 3.3042e-4, 3.4447e-4,
    3.5927e-4, 3.7486e-4, 3.9127e-4, 4.0854e-4, 4.267e-4, 4.4579e-4,
    4.6586e-4, 4.8696e-4, 5.0912e-4, 5.324e-4, 5.5685e-4, 5.8253e-4,
    6.0949e-4, 6.378e-4, 6.6753e-4, 6.9873e-4, 7.3149e-4, 7.6588e-4,
    8.0198e-4, 8.3987e-4, 8.7964e-4, 9.2139e-4, 9.6522e-4, .0010112,
    .0010595, .0011102, .0011634, .0012193, .001278, .0013396,
    .0014043, .0014722, .0015436, .0016185, .0016972, .0017799,
    .0018668, .001958, .0020539, .0021547, .0022606, .0023719,
    .002489, .002612, .0027414, .0028775, .0030206, .0031712,
    .0033295, .0034962, .0036716, .0038563, .0040506, .0042553,
    .0044709, .004698, .0049373, .0051894, .0054552, .0057354,
    .006031, .0063427, .0066717, .0070188, .0073854, .0077726,
    .0081816, .0086138, .0090709, .0095543, .010066, .010607,
    .011181, .011789, .012433, .013116, .013842, .014613, .015432,
    .016304, .017233, .018224, .019281, .020394, .021574, .022836,
    .024181, .025594, .027088, .028707, .030401, .032245, .034219,
    .036262, .038539, .040987, .043578, .04641, .04949, .052726,
    .056326, .0602, .064093, .068521, .073278, .077734, .083064,
    .088731, .093885, .1003, .1072, .11365, .12187, .13078, .13989,
    .15095, .16299, .17634, .19116, .20628, .22419, .24386, .26587,
    .28811, .31399, .34321, .36606, .39675, .42742, .44243, .47197,
    .49993, .49027, .51147, .52803, .48931, .49729, .5026, .43854,
    .441, .44766, .43414, .46151, .50029, .55247, .43855, .32115,
    .32607, .3431, .36119, .38029, .41179, .43996, .47144, .51853,
    .55362, .59122, .66338, .69877, .74001, .82923, .86907, .90361,
    1.0025, 1.031, 1.0559, 1.104, 1.1178, 1.1341, 1.1547, 1.351,
    1.4772, 1.4812, 1.4907, 1.512, 1.5442, 1.5853, 1.6358, 1.6963,
    1.7674, 1.8474, 1.9353, 2.0335, 2.143, 2.2592, 2.3853, 2.5217,
    2.6686, 2.8273, 2.9998, 3.183, 3.3868, 3.6109, 3.8564, 4.1159,
    4.4079, 4.7278, 5.0497, 5.3695, 5.758, 6.0834, 6.4976, 6.9312,
    7.38, 7.5746, 7.9833, 8.3791, 8.3956, 8.7501, 9.1067, 9.072,
    9.4649, 9.9112, 10.402, 10.829, 11.605, 12.54, 12.713, 10.443,
    10.825, 11.375, 11.955, 12.623, 13.326, 14.101, 15.041, 15.547,
    16.461, 17.439, 18.716, 19.84, 21.036, 22.642, 23.901, 25.244,
    27.03, 28.411, 29.871, 31.403, 33.147, 34.744, 36.456, 39.239,
    43.605, 45.162, 47.004, 49.093, 51.391, 53.946, 56.673, 59.629,
    63.167, 66.576, 70.254, 74.222, 78.477, 83.034, 87.914, 93.18,
    98.77, 104.74, 111.15, 117.95, 125.23, 133.01, 141.33, 150.21,
    159.71, 169.89, 180.93, 192.54, 204.99, 218.34, 232.65, 248.,
    264.47, 282.14, 301.13, 321.53, 343.48, 367.08, 392.5, 419.88,
    449.4, 481.26, 515.64, 552.79, 592.99, 636.48, 683.61, 734.65,
    789.99, 850.02, 915.14, 985.81, 1062.5, 1147.1, 1237.8, 1336.4,
    1443.2, 1558.9, 1684.2, 1819.2, 1965.2, 2122.6, 2291.7, 2470.8,
    2665.7, 2874.9, 3099.4, 3337.9, 3541., 3813.3, 4111.9, 4439.3,
    4798.9, 5196., 5639.2, 6087.5, 6657.7, 7306.7, 8040.7, 8845.5,
    9702.2, 10670., 11739., 12842., 14141., 15498., 17068., 18729.,
    20557., 22559., 25248., 27664., 30207., 32915., 35611., 38081.,
    40715., 43191., 41651., 42750., 43785., 44353., 44366., 44189.,
    43618., 42862., 41878., 35133., 35215., 36383., 39420., 44055.,
    44155., 45850., 46853., 39197., 38274., 29942., 28553., 21792.,
    21228., 17106., 14955., 18181., 19557., 21427., 23728., 26301.,
    28584., 30775., 32536., 33867., 40089., 39204., 37329., 34452.,
    31373., 33921., 34800., 36043., 44415., 45162., 52181., 50895.,
    54140., 50840., 50468., 48302., 44915., 40910., 36754., 32755.,
    29093., 25860., 22962., 20448., 18247., 16326., 14645., 13165.,
    11861., 10708., 9686.9, 8779.7, 7971.9, 7250.8, 6605.7, 6027.2,
    5507.3, 5039.1, 4616.6, 4234.8, 3889., 3575.4, 3290.5, 3031.3,
    2795.2, 2579.9, 2383.1, 2203.3, 2038.6, 1887.6, 1749.1, 1621.9,
    1505., 1397.4, 1298.3, 1207., 1122.8, 1045., 973.1, 906.64,
    845.16, 788.22, 735.48, 686.57, 641.21, 599.1, 559.99, 523.64,
    489.85, 458.42, 429.16, 401.92, 376.54, 352.88, 330.82, 310.24,
    291.03, 273.09, 256.34, 240.69, 226.05, 212.37, 199.57, 187.59,
    176.37, 165.87, 156.03, 146.82, 138.17, 130.07, 122.47, 115.34,
    108.65, 102.37, 96.473, 90.934, 85.73, 80.84, 76.243, 71.922,
```

```
67.858, 64.034, 60.438, 57.052, 53.866, 50.866, 48.04, 45.379,
42.872, 40.51, 38.285, 36.188, 34.211, 32.347, 30.588, 28.929,
27.362, 25.884, 24.489, 23.171, 21.929, 20.755, 19.646, 18.599,
17.61, 16.677, 15.795, 14.961, 14.174, 13.43, 12.725, 12.06,
11.431, 10.834, 10.27, 9.7361, 9.2302, 8.7518, 8.2997, 7.8724,
7.4674, 7.0848, 6.7226, 6.3794, 6.054, 5.745, 5.4525, 5.1752,
4.9121, 4.6625, 4.4259, 4.2015, 3.9888, 3.7872, 3.5961, 3.4149,
3.2431, 3.0802, 2.9257, 2.7792, 2.6402, 2.5084, 2.3834, 2.2648,
2.1522, 2.0455, 1.9441, 1.848, 1.7567, 1.6701, 1.5878, 1.5097,
1.4356, 1.3651, 1.2981, 1.2345, 1.174, 1.1167, 1.062, 1.0101,
.96087, .91414, .86986, .82781, .78777, .74971, .71339, .67882,
.64604, .61473, .58507, .55676, .52987, .5044, .48014, .45715,
.43527, .41453, .3948, .37609, .35831, .34142, .32524, .30995,
.29536, .28142, .26807, .25527, .24311, .23166, .22077, .21053,
.20081, .19143, .18261, .17407, .16603, .15833, .15089, .14385,
.13707, .13065, .12449, .11865, .11306, .10774, .10266, .097818,
.093203, .088815, .084641, .080671, .076892, .073296, .069873,
.066613, .06351, .060555, .05774, .055058, .052504, .050071,
.047752, .045543, .043438, .041432, .039521, .037699, .035962,
.034307, .032729, .031225, .029791, .028423, .02712, .025877,
.024692, .023563, .022485, .021458, .020478, .019543, .018652,
.017802, .016992, .016219, .015481, .014778, .014107, .013467,
.012856, .012274, .011718, .011188, .010682, .0102, .0097393,
.0093001, .008881, .0084812, .0080997, .0077358, .0073885,
.0070571, .0067409, .0064393, .0061514, .0058768, .0056147,
.0053647, .0051262, .0048987, .0046816, .0044745, .0042769,
.0040884, .0039088, .0037373, .0035739, .003418, .0032693,
.0031277, .0029926, .0028639, .0027413, .0026245, .0025133,
.0024074, .0023066, .0022108, .0021196, .002033, .0019507,
.0018726, .0017985, .0017282, .0016617, .0015988, .0015394,
.0014834, .0014306, .0013811, .0013346, .0012911, .0012506,
.0012131, .0011784, .0011465, .0011175, .0010912, .0010678,
.0010472, .0010295, .0010147, .001003, 9.9428e-4, 9.8883e-4,
9.8673e-4, 9.8821e-4, 9.9343e-4, .0010027, .0010164, .0010348,
.0010586, .0010882, .0011245, .0011685, .0012145, .0012666,
.0013095, .0013688, .0014048, .0014663, .0015309, .0015499,
.0016144, .0016312, .001705, .0017892, .0018499, .0019715,
.0021102, .0022442, .0024284, .0025893, .0027703, .0029445,
.0031193, .003346, .0034552, .0036906, .0037584, .0040084,
.0041934, .0044587, .0047093, .0049759, .0053421, .0055134,
.0059048, .0058663, .0061036, .0063259, .0059657, .0060653,
.0060972, .0055539, .0055653, .0055772, .005331, .0054953,
.0055919, .0058684, .006183, .0066675, .0069808, .0075142,
.0078536, .0084282, .0089454, .0094625, .0093703, .0095857,
.0099283, .010063, .010521, .0097778, .0098175, .010379, .010447,
.0105, .010617, .010706, .01078, .011177, .011212, .011304,
.011446, .011603, .011816, .012165, .012545, .013069, .013539,
.01411, .014776, .016103, .017016, .017994, .018978, .01998,
.021799, .022745, .023681, .024627, .025562, .026992, .027958,
.029013, .030154, .031402, .03228, .033651, .035272, .037088,
.039021, .041213, .043597, .045977, .04877, .051809, .054943,
.058064, .061528, .06537, .069309, .071928, .075752, .079589,
.083352, .084096, .087497, .090817, .091198, .094966, .099045,
.10429, .10867, .11518, .12269, .13126, .14087, .15161, .16388,
.16423, .1759, .18721, .19994, .21275, .22513, .23041, .24231,
.25299, .25396, .26396, .27696, .27929, .2908, .30595, .31433,
.3282, .3429, .35944, .37467, .39277, .41245, .43326, .45649,
.48152, .51897, .54686, .57877, .61263, .64962, .68983, .73945,
.78619, .83537, .89622, .95002, 1.0067, 1.0742, 1.1355, 1.2007,
1.2738, 1.347, 1.4254, 1.5094, 1.6009, 1.6976, 1.8019, 1.9148,
2.0357, 2.166, 2.3066, 2.4579, 2.6208, 2.7966, 2.986, 3.188,
3.4081, 3.6456, 3.9, 4.1747, 4.4712, 4.7931, 5.1359, 5.5097,
5.9117, 6.3435, 6.8003, 7.3001, 7.8385, 8.3945, 9.011, 9.6869,
10.392, 11.18, 12.036, 12.938, 13.944, 14.881, 16.029, 17.255,
18.574, 19.945, 21.38, 22.9, 24.477, 26.128, 27.87, 29.037,
30.988, 33.145, 35.506, 37.76, 40.885, 44.487, 48.505, 52.911,
57.56, 61.964, 67.217, 72.26, 78.343, 85.08, 91.867, 99.435,
107.68, 116.97, 127.12, 138.32, 150.26, 163.04, 174.81, 189.26,
205.61, 224.68, 240.98, 261.88, 285.1, 307.58, 334.35, 363.53,
394.68, 427.85, 458.85, 489.25, 472.87, 486.93, 496.27, 501.52,
501.57, 497.14, 488.09, 476.32, 393.76, 388.51, 393.42, 414.45,
455.12, 514.62, 520.38, 547.42, 562.6, 487.47, 480.83, 391.06,
376.92, 303.7, 295.91, 256.03, 236.73, 280.38, 310.71, 335.53,
367.88, 401.94, 435.52, 469.13, 497.94, 588.82, 597.94, 597.2,
588.28, 571.2, 555.75, 603.56, 638.15, 680.75, 801.72, 848.01,
962.15, 990.06, 1068.1, 1076.2, 1115.3, 1134.2, 1136.6, 1119.1,
1108.9, 1090.6, 1068.7, 1041.9, 1005.4, 967.98, 927.08, 780.1,
751.41, 733.12, 742.65, 785.56, 855.16, 852.45, 878.1, 784.59,
777.81, 765.13, 622.93, 498.09, 474.89, 386.9, 378.48, 336.17,
322.04, 329.57, 350.5, 383.38, 420.02, 462.39, 499.71, 531.98,
654.99, 653.43, 639.99, 605.16, 554.16, 504.42, 540.64, 552.33,
679.46, 699.51, 713.91, 832.17, 919.91, 884.96, 907.57, 846.56,
818.56, 768.93, 706.71, 642.17, 575.95, 515.38, 459.07, 409.02,
364.61, 325.46, 291.1, 260.89, 234.39, 211.01, 190.38, 172.11,
155.91, 141.49, 128.63, 117.13, 106.84, 97.584, 89.262, 81.756,
74.975, 68.842, 63.28, 58.232, 53.641, 49.46, 45.649, 42.168,
38.991, 36.078, 33.409, 30.96, 28.71, 26.642, 24.737, 22.985,
```

```
  21.37, 19.882, 18.512, 17.242, 16.073, 14.987, 13.984, 13.05,
  12.186, 11.384, 10.637, 9.9436, 9.2988, 8.6991, 8.141, 7.6215,
  7.1378, 6.6872, 6.2671, 5.8754, 5.51, 5.1691, 4.851, 4.5539,
  4.2764, 4.0169, 3.7742, 3.5472, 3.3348, 3.1359, 2.9495, 2.7749,
  2.6113, 2.4578, 2.3139, 2.1789, 2.0523, 1.9334, 1.8219, 1.7171,
  1.6188, 1.5263, 1.4395, 1.3579, 1.2812, 1.209, 1.1411, 1.0773,
  1.0171, .96048, .90713, .85684, .80959, .76495, .72282, .68309,
  .64563, .61035, .57707, .54573, .51622, .48834, .46199, .43709,
  .41359, .39129, .37034, .35064, .33198, .31442, .29784, .28218,
  .26732, .25337, .24017, .22774, .21601, .20479, .19426
};

static double co2260[2001] = { 5.7971e-5, 6.0733e-5, 6.3628e-5, 6.6662e-5,
  6.9843e-5, 7.3176e-5, 7.6671e-5, 8.0334e-5, 8.4175e-5, 8.8201e-5,
  9.2421e-5, 9.6846e-5, 1.0149e-4, 1.0635e-4, 1.1145e-4, 1.1679e-4,
  1.224e-4, 1.2828e-4, 1.3444e-4, 1.409e-4, 1.4768e-4, 1.5479e-4,
  1.6224e-4, 1.7006e-4, 1.7826e-4, 1.8685e-4, 1.9587e-4, 2.0532e-4,
  2.1524e-4, 2.2565e-4, 2.3656e-4, 2.48e-4, 2.6001e-4, 2.7261e-4,
  2.8582e-4, 2.9968e-4, 3.1422e-4, 3.2948e-4, 3.4548e-4, 3.6228e-4,
  3.799e-4, 3.9838e-4, 4.1778e-4, 4.3814e-4, 4.595e-4, 4.8191e-4,
  5.0543e-4, 5.3012e-4, 5.5603e-4, 5.8321e-4, 6.1175e-4, 6.417e-4,
  6.7314e-4, 7.0614e-4, 7.4078e-4, 7.7714e-4, 8.1531e-4, 8.5538e-4,
  8.9745e-4, 9.4162e-4, 9.8798e-4, .0010367, .0010878, .0011415,
  .0011978, .001257, .0013191, .0013844, .001453, .0015249,
  .0016006, .00168, .0017634, .001851, .001943, .0020397, .0021412,
  .0022479, .00236, .0024778, .0026015, .0027316, .0028682,
  .0030117, .0031626, .0033211, .0034877, .0036628, .0038469,
  .0040403, .0042436, .0044574, .004682, .0049182, .0051665,
  .0054276, .0057021, .0059907, .0062942, .0066133, .0069489,
  .0073018, .0076729, .0080632, .0084738, .0089056, .0093599,
  .0098377, .01034, .010869, .011426, .012011, .012627, .013276,
  .013958, .014676, .015431, .016226, .017063, .017944, .018872,
  .019848, .020876, .021958, .023098, .024298, .025561, .026892,
  .028293, .029769, .031323, .032961, .034686, .036503, .038418,
  .040435, .042561, .044801, .047161, .049649, .052271, .055035,
  .057948, .061019, .064256, .06767, .07127, .075066, .079069,
  .083291, .087744, .092441, .097396, .10262, .10814, .11396,
  .1201, .12658, .13342, .14064, .14826, .1563, .1648, .17376,
  .18323, .19324, .2038, .21496, .22674, .23919, .25234, .26624,
  .28093, .29646, .31287, .33021, .34855, .36794, .38844, .41012,
  .43305, .45731, .48297, .51011, .53884, .56924, .60141, .63547,
  .67152, .70969, .75012, .79292, .83826, .8863, .93718, .99111,
  1.0482, 1.1088, 1.173, 1.2411, 1.3133, 1.3898, 1.471, 1.5571,
  1.6485, 1.7455, 1.8485, 1.9577, 2.0737, 2.197, 2.3278, 2.4668,
  2.6145, 2.7715, 2.9383, 3.1156, 3.3042, 3.5047, 3.7181, 3.9451,
  4.1866, 4.4437, 4.7174, 5.0089, 5.3192, 5.65, 6.0025, 6.3782,
  6.7787, 7.206, 7.6617, 8.1479, 8.6669, 9.221, 9.8128, 10.445,
  11.12, 11.843, 12.615, 13.441, 14.325, 15.271, 16.283, 17.367,
  18.529, 19.776, 21.111, 22.544, 24.082, 25.731, 27.504, 29.409,
  31.452, 33.654, 36.024, 38.573, 41.323, 44.29, 47.492, 50.951,
  54.608, 58.588, 62.929, 67.629, 72.712, 78.226, 84.207, 90.699,
  97.749, 105.42, 113.77, 122.86, 132.78, 143.61, 155.44, 168.33,
  182.48, 198.01, 214.87, 233.39, 253.86, 276.34, 300.3, 327.28,
  356.89, 389.48, 422.29, 458.99, 501.39, 548.13, 595.62, 652.74,
  716.54, 784.57, 866.78, 960.59, 1062.8, 1072.5, 1189.5, 1319.4,
  1467.6, 1630.2, 1813.7, 2016.9, 2253., 2515.3, 2773.5, 3092.8,
  3444.4, 3720.4, 4104.3, 4527.5, 4645.9, 5021.7, 5462.2, 5597.,
  6110.6, 6732.5, 7513.8, 8270.6, 9640.6, 11487., 2796.1, 2680.1,
  2441.6, 2404.2, 2334.8, 2215.2, 1642.5, 1477.9, 1328.1, 1223.5,
  843.34, 766.96, 831.65, 834.84, 774.85, 1156.3, 1275.6, 1366.1,
  1795.6, 1885., 1936.5, 1953.4, 2154.4, 2002.7, 1789.8, 10381.,
  9040., 8216.5, 7384.7, 6721.9, 6187.7, 6143.8, 5703.9, 5276.6,
  4873.1, 4736., 4325.3, 3927., 3554.1, 3286.1, 2950.1, 2642.4,
  2368.7, 2138.9, 1914., 1719.6, 1543.9, 1388.6, 1252.1, 1132.2,
  1024.1, 1025.4, 920.58, 829.59, 750.54, 685.01, 624.25, 570.14,
  525.81, 481.85, 441.95, 408.71, 377.23, 345.86, 318.51, 292.26,
  268.34, 247.04, 227.14, 209.02, 192.69, 177.59, 163.78, 151.26,
  139.73, 129.19, 119.53, 110.7, 102.57, 95.109, 88.264, 81.948,
  76.13, 70.768, 65.827, 61.251, 57.022, 53.495, 49.824, 46.443,
  43.307, 40.405, 37.716, 35.241, 32.923, 30.77, 28.78, 26.915,
  25.177, 23.56, 22.059, 20.654, 19.345, 18.126, 16.988, 15.93,
  14.939, 14.014, 13.149, 12.343, 11.589, 10.884, 10.225, 9.6093,
  9.0327, 8.4934, 7.9889, 7.5166, 7.0744, 6.6604, 6.2727, 5.9098,
  5.5701, 5.2529, 4.955, 4.676, 4.4148, 4.171, 3.9426, 3.7332,
  3.5347, 3.3493, 3.1677, 3.0025, 2.8466, 2.6994, 2.5601, 2.4277,
  2.3016, 2.1814, 2.0664, 1.9564, 1.8279, 1.7311, 1.6427, 1.5645,
  1.4982, 1.443, 1.374, 1.3146, 1.2562, 1.17, 1.1105, 1.0272,
  .96863, .89718, .83654, .80226, .75908, .72431, .69573, .67174,
  .65126, .63315, .61693, .60182, .58715, .59554, .57649, .55526,
  .53177, .50622, .48176, .4813, .47642, .47492, .50273, .50293,
  .52687, .52239, .53419, .53814, .52626, .52211, .51492, .50622,
  .49746, .48841, .4792, .43534, .41999, .40349, .38586, .36799,
  .35108, .31089, .30803, .3171, .33599, .35041, .36149, .32924,
  .32462, .27309, .25961, .20922, .19504, .15683, .13098, .11588,
  .11478, .11204, .11363, .12135, .16423, .17785, .19094, .20236,
  .21084, .2154, .24108, .22848, .20871, .18797, .17963, .17834,
  .21552, .22284, .26945, .27052, .30108, .28977, .29772, .29224,
```

```
.27658, .24956, .22777, .20654, .18392, .16338, .1452, .12916,
.1152, .10304, .092437, .083163, .075031, .067878, .061564,
.055976, .051018, .046609, .042679, .03917, .036032, .033223,
.030706, .02845, .026428, .024617, .022998, .021554, .02027,
.019136, .018141, .017278, .016541, .015926, .015432, .015058,
.014807, .014666, .014635, .014728, .014947, .01527, .015728,
.016345, .017026, .017798, .018839, .019752, .020636, .021886,
.022695, .02327, .023478, .024292, .023544, .022222, .021932,
.020052, .018143, .017722, .017031, .017782, .01938, .020734,
.020476, .019255, .017477, .016878, .014617, .012489, .011765,
.0099077, .0086446, .0079446, .0078644, .0079763, .008671,
.01001, .0108, .012933, .015349, .016341, .018484, .020254,
.020254, .020478, .019591, .018595, .018385, .019913, .022254,
.024847, .025809, .028053, .029924, .030212, .031367, .03222,
.032739, .032537, .03286, .033344, .033507, .033499, .033339,
.032809, .033041, .031723, .029837, .027511, .026603, .024032,
.021914, .020948, .021701, .023425, .024259, .024987, .023818,
.021768, .019223, .018144, .015282, .012604, .01163, .0097907,
.008336, .0082473, .0079582, .0088077, .009779, .010129, .012145,
.014378, .016761, .01726, .018997, .019998, .019809, .01819,
.016358, .016099, .01617, .017939, .020223, .022521, .02277,
.024279, .025247, .024222, .023989, .023224, .021493, .020362,
.018596, .017309, .015975, .014466, .013171, .011921, .01078,
.0097229, .0087612, .0078729, .0070682, .0063494, .0057156,
.0051459, .0046273, .0041712, .0037686, .0034119, .003095,
.0028126, .0025603, .0023342, .0021314, .0019489, .0017845,
.001636, .0015017, .00138, .0012697, .0011694, .0010782,
9.9507e-4, 9.1931e-4, 8.5013e-4, 7.869e-4, 7.2907e-4, 6.7611e-4,
6.2758e-4, 5.8308e-4, 5.4223e-4, 5.0473e-4, 4.7027e-4, 4.3859e-4,
4.0946e-4, 3.8265e-4, 3.5798e-4, 3.3526e-4, 3.1436e-4, 2.9511e-4,
2.7739e-4, 2.6109e-4, 2.4609e-4, 2.3229e-4, 2.1961e-4, 2.0797e-4,
1.9729e-4, 1.875e-4, 1.7855e-4, 1.7038e-4, 1.6294e-4, 1.5619e-4,
1.5007e-4, 1.4456e-4, 1.3961e-4, 1.3521e-4, 1.3131e-4, 1.2789e-4,
1.2494e-4, 1.2242e-4, 1.2032e-4, 1.1863e-4, 1.1733e-4, 1.1641e-4,
1.1585e-4, 1.1565e-4, 1.158e-4, 1.1629e-4, 1.1712e-4, 1.1827e-4,
1.1976e-4, 1.2158e-4, 1.2373e-4, 1.262e-4, 1.2901e-4, 1.3214e-4,
1.3562e-4, 1.3944e-4, 1.4361e-4, 1.4814e-4, 1.5303e-4, 1.5829e-4,
1.6394e-4, 1.6999e-4, 1.7644e-4, 1.8332e-4, 1.9063e-4, 1.984e-4,
2.0663e-4, 2.1536e-4, 2.246e-4, 2.3436e-4, 2.4468e-4, 2.5558e-4,
2.6708e-4, 2.7921e-4, 2.92e-4, 3.0548e-4, 3.1968e-4, 3.3464e-4,
3.5039e-4, 3.6698e-4, 3.8443e-4, 4.0281e-4, 4.2214e-4, 4.4248e-4,
4.6389e-4, 4.864e-4, 5.1009e-4, 5.3501e-4, 5.6123e-4, 5.888e-4,
6.1781e-4, 6.4833e-4, 6.8043e-4, 7.142e-4, 7.4973e-4, 7.8711e-4,
8.2644e-4, 8.6783e-4, 9.1137e-4, 9.5721e-4, .0010054, .0010562,
.0011096, .0011659, .0012251, .0012875, .0013532, .0014224,
.0014953, .001572, .0016529, .0017381, .0018279, .0019226,
.0020224, .0021277, .0022386, .0023557, .0024792, .0026095,
.002747, .0028921, .0030453, .0032071, .003378, .0035586,
.0037494, .003951, .0041642, .0043897, .0046282, .0048805,
.0051476, .0054304, .00573, .0060473, .0063837, .0067404,
.0071188, .0075203, .0079466, .0083994, .0088806, .0093922,
.0099366, .010516, .011134, .011792, .012494, .013244, .014046,
.014898, .015808, .016781, .017822, .018929, .020108, .02138,
.022729, .02419, .02576, .027412, .029233, .031198, .033301,
.035594, .038092, .040767, .04372, .046918, .050246, .053974,
.058009, .061976, .066586, .071537, .076209, .081856, .087998,
.093821, .10113, .10913, .11731, .12724, .13821, .15025, .1639,
.17807, .19472, .21356, .23496, .25758, .28387, .31389, .34104,
.37469, .40989, .43309, .46845, .5042, .5023, .52981, .55275,
.51075, .51976, .52457, .44779, .44721, .4503, .4243, .45244,
.49491, .55399, .39021, .24802, .2501, .2618, .27475, .28879,
.31317, .33643, .36257, .4018, .43275, .46525, .53333, .56599,
.60557, .70142, .74194, .77736, .88567, .91182, .93294, .98407,
.98772, .99176, .9995, 1.2405, 1.3602, 1.338, 1.3255, 1.3267,
1.3404, 1.3634, 1.3967, 1.4407, 1.4961, 1.5603, 1.6328, 1.7153,
1.8094, 1.9091, 2.018, 2.1367, 2.264, 2.4035, 2.5562, 2.7179,
2.9017, 3.1052, 3.3304, 3.5731, 3.8488, 4.1553, 4.4769, 4.7818,
5.1711, 5.5204, 5.9516, 6.4097, 6.8899, 7.1118, 7.5469, 7.9735,
7.9511, 8.3014, 8.6418, 8.4757, 8.8256, 9.2294, 9.6923, 10.033,
10.842, 11.851, 11.78, 8.8435, 9.1381, 9.5956, 10.076, 10.629,
11.22, 11.883, 12.69, 13.163, 13.974, 14.846, 16.027, 17.053,
18.148, 19.715, 20.907, 22.163, 23.956, 25.235, 26.566, 27.94,
29.576, 30.956, 32.432, 35.337, 39.911, 41.128, 42.625, 44.386,
46.369, 48.619, 51.031, 53.674, 56.825, 59.921, 63.286, 66.929,
70.859, 75.081, 79.618, 84.513, 89.739, 95.335, 101.35, 107.76,
114.63, 121.98, 129.87, 138.3, 147.34, 157.04, 167.56, 178.67,
190.61, 203.43, 217.19, 231.99, 247.88, 264.98, 283.37, 303.17,
324.49, 347.47, 372.25, 398.98, 427.85, 459.06, 492.8, 529.31,
568.89, 611.79, 658.35, 708.91, 763.87, 823.65, 888.72, 959.58,
1036.8, 1121.8, 1213.9, 1314.3, 1423.8, 1543., 1672.8, 1813.4,
1966.1, 2131.4, 2309.5, 2499.3, 2705., 2925.7, 3161.6, 3411.3,
3611.5, 3889.2, 4191.1, 4519.3, 4877.9, 5272.9, 5712.9, 6142.7,
6719.6, 7385., 8145., 8977.7, 9831.9, 10827., 11934., 13063.,
14434., 15878., 17591., 19435., 21510., 23835., 26835., 29740.,
32878., 36305., 39830., 43273., 46931., 50499., 49586., 51598.,
53429., 54619., 55081., 55102., 54485., 53487., 52042., 42689.,
42607., 44020., 47994., 54169., 53916., 55808., 56642., 46049.,
```

```
44243., 32929., 30658., 21963., 20835., 15962., 13679., 17652.,
19680., 22388., 25625., 29184., 32520., 35720., 38414., 40523.,
49228., 48173., 45678., 41768., 37600., 41313., 42654., 44465.,
55736., 56630., 65409., 63308., 66572., 61845., 60379., 56777.,
51920., 46601., 41367., 36529., 32219., 28470., 25192., 22362.,
19907., 17772., 15907., 14273., 12835., 11567., 10445., 9450.2,
8565.1, 7776., 7070.8, 6439.2, 5872.3, 5362.4, 4903., 4488.3,
4113.4, 3773.8, 3465.8, 3186.1, 2931.7, 2700.1, 2488.8, 2296.,
2119.8, 1958.6, 1810.9, 1675.6, 1551.4, 1437.3, 1332.4, 1236.,
1147.2, 1065.3, 989.86, 920.22, 855.91, 796.48, 741.53, 690.69,
643.62, 600.02, 559.6, 522.13, 487.35, 455.06, 425.08, 397.21,
371.3, 347.2, 324.78, 303.9, 284.46, 266.34, 249.45, 233.7,
219.01, 205.3, 192.5, 180.55, 169.38, 158.95, 149.2, 140.07,
131.54, 123.56, 116.09, 109.09, 102.54, 96.405, 90.655, 85.266,
80.213, 75.475, 71.031, 66.861, 62.948, 59.275, 55.827, 52.587,
49.544, 46.686, 43.998, 41.473, 39.099, 36.867, 34.768, 32.795,
30.939, 29.192, 27.546, 25.998, 24.539, 23.164, 21.869, 20.65,
19.501, 18.419, 17.399, 16.438, 15.532, 14.678, 13.874, 13.115,
12.4, 11.726, 11.088, 10.488, 9.921, 9.3846, 8.8784, 8.3996,
7.9469, 7.5197, 7.1174, 6.738, 6.379, 6.0409, 5.7213, 5.419,
5.1327, 4.8611, 4.6046, 4.3617, 4.1316, 3.9138, 3.7077, 3.5125,
3.3281, 3.1536, 2.9885, 2.8323, 2.6846, 2.5447, 2.4124, 2.2871,
2.1686, 2.0564, 1.9501, 1.8495, 1.7543, 1.6641, 1.5787, 1.4978,
1.4212, 1.3486, 1.2799, 1.2147, 1.1529, 1.0943, 1.0388, .98602,
.93596, .8886, .84352, .80078, .76029, .722, .68585, .65161,
.61901, .58808, .55854, .53044, .5039, .47853, .45459, .43173,
.41008, .38965, .37021, .35186, .33444, .31797, .30234, .28758,
.2736, .26036, .24764, .2357, .22431, .21342, .20295, .19288,
.18334, .17444, .166, .15815, .15072, .14348, .13674, .13015,
.12399, .11807, .11231, .10689, .10164, .096696, .091955,
.087476, .083183, .079113, .075229, .071536, .068026, .064698,
.06154, .058544, .055699, .052997, .050431, .047993, .045676,
.043475, .041382, .039392, .037501, .035702, .033991, .032364,
.030817, .029345, .027945, .026613, .025345, .024139, .022991,
.021899, .02086, .019871, .018929, .018033, .01718, .016368,
.015595, .014859, .014158, .013491, .012856, .012251, .011675,
.011126, .010604, .010107, .0096331, .009182, .0087523, .0083431,
.0079533, .0075821, .0072284, .0068915, .0065706, .0062649,
.0059737, .0056963, .005432, .0051802, .0049404, .0047118,
.0044941, .0042867, .0040891, .0039009, .0037216, .0035507,
.003388, .0032329, .0030852, .0029445, .0028105, .0026829,
.0025613, .0024455, .0023353, .0022303, .0021304, .0020353,
.0019448, .0018587, .0017767, .0016988, .0016247, .0015543,
.0014874, .0014238, .0013635, .0013062, .0012519, .0012005,
.0011517, .0011057, .0010621, .001021, 9.8233e-4, 9.4589e-4,
9.1167e-4, 8.7961e-4, 8.4964e-4, 8.2173e-4, 7.9582e-4, 7.7189e-4,
7.499e-4, 7.2983e-4, 7.1167e-4, 6.9542e-4, 6.8108e-4, 6.6866e-4,
6.5819e-4, 6.4971e-4, 6.4328e-4, 6.3895e-4, 6.3681e-4, 6.3697e-4,
6.3956e-4, 6.4472e-4, 6.5266e-4, 6.6359e-4, 6.778e-4, 6.9563e-4,
7.1749e-4, 7.4392e-4, 7.7556e-4, 8.1028e-4, 8.4994e-4, 8.8709e-4,
9.3413e-4, 9.6953e-4, .0010202, .0010738, .0010976, .0011507,
.0011686, .0012264, .001291, .0013346, .0014246, .0015293,
.0016359, .0017824, .0019255, .0020854, .002247, .0024148,
.0026199, .0027523, .0029704, .0030702, .0033047, .0035013,
.0037576, .0040275, .0043089, .0046927, .0049307, .0053486,
.0053809, .0056699, .0059325, .0055488, .005634, .0056392,
.004946, .0048855, .0048208, .0044386, .0045498, .0046377,
.0048939, .0052396, .0057324, .0060859, .0066906, .0071148,
.0077224, .0082687, .008769, .0084471, .008572, .0087729,
.008775, .0090742, .0080704, .0080288, .0085747, .0086087,
.0086408, .0088752, .0089381, .0089757, .0093532, .0092824,
.0092566, .0092645, .0092735, .009342, .0095806, .0097991,
.010213, .010611, .011129, .011756, .013237, .01412, .015034,
.015936, .01682, .018597, .019315, .019995, .020658, .021289,
.022363, .022996, .023716, .024512, .025434, .026067, .027118,
.028396, .029865, .031442, .033253, .03525, .037296, .039701,
.042356, .045154, .048059, .051294, .054893, .058636, .061407,
.065172, .068974, .072676, .073379, .076547, .079556, .079134,
.082308, .085739, .090192, .09359, .099599, .10669, .11496,
.1244, .13512, .14752, .14494, .15647, .1668, .17863, .19029,
.20124, .20254, .21179, .21982, .21625, .22364, .23405, .23382,
.2434, .25708, .26406, .27621, .28909, .30395, .31717, .33271,
.3496, .36765, .38774, .40949, .446, .46985, .49846, .5287, .562,
.59841, .64598, .68834, .7327, .78978, .8373, .88708, .94744,
1.0006, 1.0574, 1.1215, 1.1856, 1.2546, 1.3292, 1.4107, 1.4974,
1.5913, 1.6931, 1.8028, 1.9212, 2.0492, 2.1874, 2.3365, 2.4978,
2.6718, 2.8588, 3.062, 3.2818, 3.5188, 3.7752, 4.0527, 4.3542,
4.6782, 5.0312, 5.4123, 5.8246, 6.2639, 6.7435, 7.2636, 7.8064,
8.4091, 9.0696, 9.7677, 10.548, 11.4, 12.309, 13.324, 14.284,
15.445, 16.687, 18.019, 19.403, 20.847, 22.366, 23.925, 25.537,
27.213, 28.069, 29.864, 31.829, 33.988, 35.856, 38.829, 42.321,
46.319, 50.606, 55.126, 59.126, 64.162, 68.708, 74.615, 81.176,
87.739, 95.494, 103.83, 113.38, 123.99, 135.8, 148.7, 162.58,
176.32, 192.6, 211.47, 232.7, 252.64, 277.41, 305.38, 333.44,
366.42, 402.66, 442.14, 484.53, 526.42, 568.15, 558.78, 582.6,
600.98, 613.94, 619.44, 618.24, 609.84, 595.96, 484.86, 475.59,
478.49, 501.56, 552.19, 628.44, 630.39, 658.92, 671.96, 562.7,
```

```
    545.88, 423.43, 400.14, 306.59, 294.13, 246.8, 226.51, 278.21,
    314.39, 347.22, 389.13, 433.16, 477.48, 521.67, 560.54, 683.6,
    696.37, 695.91, 683.1, 658.24, 634.89, 698.85, 742.87, 796.66,
    954.49, 1009.5, 1150.5, 1179.1, 1267.9, 1272.4, 1312.7, 1330.4,
    1331.6, 1315.8, 1308.3, 1293.3, 1274.6, 1249.5, 1213.2, 1172.1,
    1124.4, 930.33, 893.36, 871.27, 883.54, 940.76, 1036., 1025.6,
    1053.1, 914.51, 894.15, 865.03, 670.63, 508.41, 475.15, 370.85,
    361.06, 319.38, 312.75, 331.87, 367.13, 415., 467.94, 525.49,
    578.41, 624.66, 794.82, 796.97, 780.29, 736.49, 670.18, 603.75,
    659.67, 679.8, 857.12, 884.05, 900.65, 1046.1, 1141.9, 1083.,
    1089.2, 1e3, 947.08, 872.31, 787.91, 704.75, 624.93, 553.68,
    489.91, 434.21, 385.64, 343.3, 306.42, 274.18, 245.94, 221.11,
    199.23, 179.88, 162.73, 147.48, 133.88, 121.73, 110.86, 101.1,
    92.323, 84.417, 77.281, 70.831, 64.991, 59.694, 54.884, 50.509,
    46.526, 42.893, 39.58, 36.549, 33.776, 31.236, 28.907, 26.77,
    24.805, 23., 21.339, 19.81, 18.404, 17.105, 15.909, 14.801,
    13.778, 12.83, 11.954, 11.142, 10.389, 9.691, 9.0434, 8.4423,
    7.8842, 7.3657, 6.8838, 6.4357, 6.0189, 5.6308, 5.2696, 4.9332,
    4.6198, 4.3277, 4.0553, 3.8012, 3.5639, 3.3424, 3.1355, 2.9422,
    2.7614, 2.5924, 2.4343, 2.2864, 2.148, 2.0184, 1.8971, 1.7835,
    1.677, 1.5773, 1.4838, 1.3961, 1.3139, 1.2369, 1.1645, 1.0966,
    1.0329, .97309, .91686, .86406, .81439, .76767, .72381, .68252,
    .64359, .60695, .57247, .54008, .50957, .48092, .45401, .42862,
    .40465, .38202, .36072, .34052, .3216, .30386, .28711, .27135,
    .25651, .24252, .2293, .21689, .20517, .19416, .18381, .17396,
    .16469
  };

  static double co2230[2001] = { 2.743e-5, 2.8815e-5, 3.027e-5, 3.1798e-5,
    3.3405e-5, 3.5094e-5, 3.6869e-5, 3.8734e-5, 4.0694e-5, 4.2754e-5,
    4.492e-5, 4.7196e-5, 4.9588e-5, 5.2103e-5, 5.4747e-5, 5.7525e-5,
    6.0446e-5, 6.3516e-5, 6.6744e-5, 7.0137e-5, 7.3704e-5, 7.7455e-5,
    8.1397e-5, 8.5543e-5, 8.9901e-5, 9.4484e-5, 9.9302e-5, 1.0437e-4,
    1.097e-4, 1.153e-4, 1.2119e-4, 1.2738e-4, 1.3389e-4, 1.4074e-4,
    1.4795e-4, 1.5552e-4, 1.6349e-4, 1.7187e-4, 1.8068e-4, 1.8995e-4,
    1.997e-4, 2.0996e-4, 2.2075e-4, 2.321e-4, 2.4403e-4, 2.5659e-4,
    2.698e-4, 2.837e-4, 2.9832e-4, 3.137e-4, 3.2988e-4, 3.4691e-4,
    3.6483e-4, 3.8368e-4, 4.0351e-4, 4.2439e-4, 4.4635e-4, 4.6947e-4,
    4.9379e-4, 5.1939e-4, 5.4633e-4, 5.7468e-4, 6.0452e-4, 6.3593e-4,
    6.69e-4, 7.038e-4, 7.4043e-4, 7.79e-4, 8.1959e-4, 8.6233e-4,
    9.0732e-4, 9.5469e-4, .0010046, .0010571, .0011124, .0011706,
    .0012319, .0012964, .0013644, .001436, .0015114, .0015908,
    .0016745, .0017625, .0018553, .0019531, .002056, .0021645,
    .0022788, .0023992, .002526, .0026596, .0028004, .0029488,
    .0031052, .0032699, .0034436, .0036265, .0038194, .0040227,
    .0042369, .0044628, .0047008, .0049518, .0052164, .0054953,
    .0057894, .0060995, .0064265, .0067713, .007135, .0075184,
    .0079228, .0083494, .0087993, .0092738, .0097745, .010303,
    .01086, .011448, .012068, .012722, .013413, .014142, .014911,
    .015723, .01658, .017484, .018439, .019447, .020511, .021635,
    .022821, .024074, .025397, .026794, .02827, .029829, .031475,
    .033215, .035052, .036994, .039045, .041213, .043504, .045926,
    .048485, .05119, .05405, .057074, .060271, .063651, .067225,
    .071006, .075004, .079233, .083708, .088441, .093449, .098749,
    .10436, .11029, .11657, .12322, .13026, .13772, .14561, .15397,
    .16282, .1722, .18214, .19266, .20381, .21563, .22816, .24143,
    .2555, .27043, .28625, .30303, .32082, .3397, .35972, .38097,
    .40352, .42746, .45286, .47983, .50847, .53888, .57119, .6055,
    .64196, .6807, .72187, .76564, .81217, .86165, .91427, .97025,
    1.0298, 1.0932, 1.1606, 1.2324, 1.3088, 1.3902, 1.477, 1.5693,
    1.6678, 1.7727, 1.8845, 2.0038, 2.131, 2.2666, 2.4114, 2.5659,
    2.7309, 2.907, 3.0951, 3.2961, 3.5109, 3.7405, 3.986, 4.2485,
    4.5293, 4.8299, 5.1516, 5.4961, 5.8651, 6.2605, 6.6842, 7.1385,
    7.6256, 8.1481, 8.7089, 9.3109, 9.9573, 10.652, 11.398, 12.2,
    13.063, 13.992, 14.99, 16.064, 17.222, 18.469, 19.813, 21.263,
    22.828, 24.516, 26.34, 28.31, 30.437, 32.738, 35.226, 37.914,
    40.824, 43.974, 47.377, 51.061, 55.011, 59.299, 63.961, 69.013,
    74.492, 80.444, 86.919, 93.836, 101.23, 109.25, 117.98, 127.47,
    137.81, 149.07, 161.35, 174.75, 189.42, 205.49, 223.02, 242.26,
    263.45, 286.75, 311.94, 340.01, 370.86, 404.92, 440.44, 480.27,
    525.17, 574.71, 626.22, 686.8, 754.38, 827.07, 913.38, 1011.7,
    1121.5, 1161.6, 1289.5, 1432.2, 1595.4, 1777., 1983.3, 2216.1,
    2485.7, 2788.3, 3101.5, 3481., 3902.1, 4257.1, 4740., 5272.8,
    5457.9, 5946.2, 6505.3, 6668.4, 7302.4, 8061.6, 9015.8, 9908.3,
    11613., 13956., 3249.6, 3243., 2901.5, 2841.3, 2729.6, 2558.2,
    1797.8, 1583.2, 1386., 1233.5, 787.74, 701.46, 761.66, 767.21,
    722.83, 1180.6, 1332.1, 1461.6, 2032.9, 2166., 2255.9, 2294.7,
    2587.2, 2396.5, 2122.4, 12553., 10784., 9832.5, 8827.3, 8029.1,
    7377.9, 7347.1, 6783.8, 6239.1, 5721.1, 5503., 4975.1, 4477.8,
    4021.3, 3676.8, 3275.3, 2914.9, 2597.4, 2328.2, 2075.4, 1857.6,
    1663.6, 1493.3, 1343.8, 1213.3, 1095.6, 1066.5, 958.91, 865.15,
    783.31, 714.35, 650.77, 593.98, 546.2, 499.9, 457.87, 421.75,
    387.61, 355.25, 326.62, 299.7, 275.21, 253.17, 232.83, 214.31,
    197.5, 182.08, 167.98, 155.12, 143.32, 132.5, 122.58, 113.48,
    105.11, 97.415, 90.182, 83.463, 77.281, 71.587, 66.341, 61.493,
    57.014, 53.062, 49.21, 45.663, 42.38, 39.348, 36.547, 33.967,
    31.573, 29.357, 27.314, 25.415, 23.658, 22.03, 20.524, 19.125,
```

```
17.829, 16.627, 15.511, 14.476, 13.514, 12.618, 11.786, 11.013,
10.294, 9.6246, 9.0018, 8.4218, 7.8816, 7.3783, 6.9092, 6.4719,
6.0641, 5.6838, 5.3289, 4.998, 4.6893, 4.4014, 4.1325, 3.8813,
3.6469, 3.4283, 3.2241, 3.035, 2.8576, 2.6922, 2.5348, 2.3896,
2.2535, 2.1258, 2.0059, 1.8929, 1.7862, 1.6854, 1.5898, 1.4992,
1.4017, 1.3218, 1.2479, 1.1809, 1.1215, 1.0693, 1.0116, .96016,
.9105, .84859, .80105, .74381, .69982, .65127, .60899, .57843,
.54592, .51792, .49336, .47155, .45201, .43426, .41807, .40303,
.38876, .3863, .37098, .35492, .33801, .32032, .30341, .29874,
.29193, .28689, .29584, .29155, .29826, .29195, .29287, .2904,
.28199, .27709, .27162, .26622, .26133, .25676, .25235, .23137,
.22365, .21519, .20597, .19636, .18699, .16485, .16262, .16643,
.17542, .18198, .18631, .16759, .16338, .13505, .1267, .10053,
.092554, .074093, .062159, .055523, .054849, .05401, .05528,
.058982, .07952, .08647, .093244, .099285, .10393, .10661,
.12072, .11417, .10396, .093265, .089137, .088909, .10902,
.11277, .13625, .13565, .14907, .14167, .1428, .13744, .12768,
.11382, .10244, .091686, .08109, .071739, .063616, .056579,
.050504, .045251, .040689, .036715, .033237, .030181, .027488,
.025107, .022998, .021125, .01946, .017979, .016661, .015489,
.014448, .013526, .012712, .011998, .011375, .010839, .010384,
.010007, .0097053, .0094783, .0093257, .0092489, .0092504,
.0093346, .0095077, .0097676, .01012, .01058, .011157, .011844,
.012672, .013665, .014766, .015999, .017509, .018972, .020444,
.022311, .023742, .0249, .025599, .026981, .026462, .025143,
.025066, .022814, .020458, .020026, .019142, .020189, .022371,
.024163, .023728, .02199, .019506, .018591, .015576, .012784,
.011744, .0094777, .0079148, .0070652, .006986, .0071758,
.008086, .0098025, .01087, .013609, .016764, .018137, .021061,
.023498, .023576, .023965, .022828, .021519, .021283, .023364,
.026457, .029782, .030856, .033486, .035515, .035543, .036558,
.037198, .037472, .037045, .037284, .03777, .038085, .038366,
.038526, .038282, .038915, .037697, .035667, .032941, .031959,
.028692, .025918, .024596, .025592, .027873, .028935, .02984,
.028148, .025305, .021912, .020454, .016732, .013357, .01205,
.009731, .0079881, .0077704, .0074387, .0083895, .0096776,
.010326, .01293, .015955, .019247, .020145, .02267, .024231,
.024184, .022131, .019784, .01955, .01971, .022119, .025116,
.027978, .028107, .029808, .030701, .029164, .028551, .027286,
.024946, .023259, .020982, .019221, .017471, .015643, .014074,
.01261, .011301, .010116, .0090582, .0081036, .0072542, .0065034,
.0058436, .0052571, .0047321, .0042697, .0038607, .0034977,
.0031747, .0028864, .0026284, .002397, .002189, .0020017,
.0018326, .0016798, .0015414, .0014159, .0013019, .0011983,
.0011039, .0010177, 9.391e-4, 8.6717e-4, 8.0131e-4, 7.4093e-4,
6.8553e-4, 6.3464e-4, 5.8787e-4, 5.4487e-4, 5.0533e-4, 4.69e-4,
4.3556e-4, 4.0474e-4, 3.7629e-4, 3.5e-4, 3.2569e-4, 3.032e-4,
2.8239e-4, 2.6314e-4, 2.4535e-4, 2.2891e-4, 2.1374e-4, 1.9975e-4,
1.8685e-4, 1.7498e-4, 1.6406e-4, 1.5401e-4, 1.4479e-4, 1.3633e-4,
1.2858e-4, 1.2148e-4, 1.1499e-4, 1.0907e-4, 1.0369e-4, 9.8791e-5,
9.4359e-5, 9.0359e-5, 8.6766e-5, 8.3555e-5, 8.0703e-5, 7.8192e-5,
7.6003e-5, 7.4119e-5, 7.2528e-5, 7.1216e-5, 7.0171e-5, 6.9385e-5,
6.8848e-5, 6.8554e-5, 6.8496e-5, 6.8669e-5, 6.9069e-5, 6.9694e-5,
7.054e-5, 7.1608e-5, 7.2896e-5, 7.4406e-5, 7.6139e-5, 7.8097e-5,
8.0283e-5, 8.2702e-5, 8.5357e-5, 8.8255e-5, 9.1402e-5, 9.4806e-5,
9.8473e-5, 1.0241e-4, 1.0664e-4, 1.1115e-4, 1.1598e-4, 1.2112e-4,
1.2659e-4, 1.3241e-4, 1.3859e-4, 1.4515e-4, 1.521e-4, 1.5947e-4,
1.6728e-4, 1.7555e-4, 1.8429e-4, 1.9355e-4, 2.0334e-4, 2.1369e-4,
2.2463e-4, 2.3619e-4, 2.4841e-4, 2.6132e-4, 2.7497e-4, 2.8938e-4,
3.0462e-4, 3.2071e-4, 3.3771e-4, 3.5567e-4, 3.7465e-4, 3.947e-4,
4.1588e-4, 4.3828e-4, 4.6194e-4, 4.8695e-4, 5.1338e-4, 5.4133e-4,
5.7087e-4, 6.0211e-4, 6.3515e-4, 6.701e-4, 7.0706e-4, 7.4617e-4,
7.8756e-4, 8.3136e-4, 8.7772e-4, 9.2681e-4, 9.788e-4, .0010339,
.0010922, .001154, .0012195, .0012889, .0013626, .0014407,
.0015235, .0016114, .0017048, .0018038, .001909, .0020207,
.0021395, .0022657, .0023998, .0025426, .0026944, .002856,
.0030281, .0032114, .0034068, .003615, .0038371, .004074,
.004327, .0045971, .0048857, .0051942, .0055239, .0058766,
.0062538, .0066573, .0070891, .007551, .0080455, .0085747,
.0091412, .0097481, .010397, .011092, .011837, .012638, .013495,
.014415, .01541, .016475, .017621, .018857, .020175, .02162,
.023185, .024876, .02672, .028732, .030916, .033319, .035939,
.038736, .041847, .04524, .048715, .052678, .056977, .061203,
.066184, .07164, .076952, .083477, .090674, .098049, .10697,
.1169, .1277, .14011, .15323, .1684, .18601, .20626, .22831,
.25417, .28407, .31405, .34957, .38823, .41923, .46026, .50409,
.51227, .54805, .57976, .53818, .55056, .557, .46741, .46403,
.4636, .42265, .45166, .49852, .56663, .34306, .17779, .17697,
.18346, .19129, .20014, .21778, .23604, .25649, .28676, .31238,
.33856, .39998, .4288, .46568, .56654, .60786, .64473, .76466,
.7897, .80778, .86443, .85736, .84798, .84157, 1.1385, 1.2446,
1.1923, 1.1552, 1.1338, 1.1266, 1.1292, 1.1431, 1.1683, 1.2059,
1.2521, 1.3069, 1.3712, 1.4471, 1.5275, 1.6165, 1.7145, 1.8189,
1.9359, 2.065, 2.2007, 2.3591, 2.5362, 2.7346, 2.9515, 3.2021,
3.4851, 3.7935, 4.0694, 4.4463, 4.807, 5.2443, 5.7178, 6.2231,
6.4796, 6.9461, 7.4099, 7.3652, 7.7182, 8.048, 7.7373, 8.0363,
8.3855, 8.8044, 9.0257, 9.8574, 10.948, 10.563, 6.8979, 7.0744,
```

```
7.4121, 7.7663, 8.1768, 8.6243, 9.1437, 9.7847, 10.182, 10.849,
11.572, 12.602, 13.482, 14.431, 15.907, 16.983, 18.11, 19.884,
21.02, 22.18, 23.355, 24.848, 25.954, 27.13, 30.186, 34.893,
35.682, 36.755, 38.111, 39.703, 41.58, 43.606, 45.868, 48.573,
51.298, 54.291, 57.559, 61.116, 64.964, 69.124, 73.628, 78.471,
83.683, 89.307, 95.341, 101.84, 108.83, 116.36, 124.46, 133.18,
142.57, 152.79, 163.69, 175.43, 188.11, 201.79, 216.55, 232.51,
249.74, 268.38, 288.54, 310.35, 333.97, 359.55, 387.26, 417.3,
449.88, 485.2, 523.54, 565.14, 610.28, 659.31, 712.56, 770.43,
833.36, 901.82, 976.36, 1057.6, 1146.8, 1243.8, 1350., 1466.3,
1593.6, 1732.7, 1884.1, 2049.1, 2228.2, 2421.9, 2629.4, 2853.7,
3094.4, 3351.1, 3622.3, 3829.8, 4123.1, 4438.3, 4777.2, 5144.1,
5545.4, 5990.5, 6404.5, 6996.8, 7687.6, 8482.9, 9349.4, 10203.,
11223., 12358., 13493., 14916., 16416., 18236., 20222., 22501.,
25102., 28358., 31707., 35404., 39538., 43911., 48391., 53193.,
58028., 58082., 61276., 64193., 66294., 67480., 67921., 67423.,
66254., 64341., 51737., 51420., 53072., 58145., 66195., 65358.,
67377., 67869., 53509., 50553., 35737., 32425., 21704., 19974.,
14457., 12142., 16798., 19489., 23049., 27270., 31910., 36457.,
40877., 44748., 47876., 59793., 58626., 55454., 50337., 44893.,
50228., 52216., 54747., 69541., 70455., 81014., 77694., 80533.,
73953., 70927., 65539., 59002., 52281., 45953., 40292., 35360.,
31124., 27478., 24346., 21647., 19308., 17271., 15491., 13927.,
12550., 11331., 10250., 9288.8, 8431.4, 7664.9, 6978.3, 6361.8,
5807.4, 5307.7, 4856.8, 4449., 4079.8, 3744.9, 3440.8, 3164.2,
2912.3, 2682.7, 2473., 2281.4, 2106., 1945.3, 1797.9, 1662.5,
1538.1, 1423.6, 1318.1, 1221., 1131.5, 1049., 972.99, 902.87,
838.01, 777.95, 722.2, 670.44, 622.35, 577.68, 536.21, 497.76,
462.12, 429.13, 398.61, 370.39, 344.29, 320.16, 297.85, 277.2,
258.08, 240.38, 223.97, 208.77, 194.66, 181.58, 169.43, 158.15,
147.67, 137.92, 128.86, 120.44, 112.6, 105.3, 98.499, 92.166,
86.264, 80.763, 75.632, 70.846, 66.381, 62.213, 58.321, 54.685,
51.288, 48.114, 45.145, 42.368, 39.772, 37.341, 35.065, 32.937,
30.943, 29.077, 27.33, 25.693, 24.158, 22.717, 21.367, 20.099,
18.909, 17.792, 16.744, 15.761, 14.838, 13.971, 13.157, 12.393,
11.676, 11.003, 10.369, 9.775, 9.2165, 8.6902, 8.1963, 7.7314,
7.2923, 6.8794, 6.4898, 6.122, 5.7764, 5.4525, 5.1484, 4.8611,
4.5918, 4.3379, 4.0982, 3.8716, 3.6567, 3.4545, 3.2634, 3.0828,
2.9122, 2.7512, 2.5993, 2.4561, 2.3211, 2.1938, 2.0737, 1.9603,
1.8534, 1.7525, 1.6572, 1.5673, 1.4824, 1.4022, 1.3265, 1.2551,
1.1876, 1.1239, 1.0637, 1.0069, .9532, .90248, .85454, .80921,
.76631, .72569, .6872, .65072, .61635, .5836, .55261, .52336,
.49581, .46998, .44559, .42236, .40036, .37929, .35924, .34043,
.32238, .30547, .28931, .27405, .25975, .24616, .23341, .22133,
.20997, .19924, .18917, .17967, .17075, .16211, .15411, .14646,
.13912, .13201, .12509, .11857, .11261, .10698, .10186, .097039,
.092236, .087844, .083443, .07938, .075452, .071564, .067931,
.064389, .061078, .057901, .054921, .052061, .049364, .046789,
.04435, .042044, .039866, .037808, .035863, .034023, .032282,
.030634, .029073, .027595, .026194, .024866, .023608, .022415,
.021283, .02021, .019193, .018228, .017312, .016443, .015619,
.014837, .014094, .01339, .012721, .012086, .011483, .010911,
.010368, .009852, .0093623, .0088972, .0084556, .0080362,
.0076379, .0072596, .0069003, .006559, .0062349, .0059269,
.0056344, .0053565, .0050925, .0048417, .0046034, .004377,
.0041618, .0039575, .0037633, .0035788, .0034034, .0032368,
.0030785, .002928, .0027851, .0026492, .0025201, .0023975,
.0022809, .0021701, .0020649, .0019649, .0018699, .0017796,
.0016938, .0016122, .0015348, .0014612, .0013913, .001325,
.0012619, .0012021, .0011452, .0010913, .0010401, 9.9149e-4,
9.454e-4, 9.0169e-4, 8.6024e-4, 8.2097e-4, 7.8377e-4, 7.4854e-4,
7.1522e-4, 6.8371e-4, 6.5393e-4, 6.2582e-4, 5.9932e-4, 5.7435e-4,
5.5087e-4, 5.2882e-4, 5.0814e-4, 4.8881e-4, 4.7076e-4, 4.5398e-4,
4.3843e-4, 4.2407e-4, 4.109e-4, 3.9888e-4, 3.88e-4, 3.7826e-4,
3.6963e-4, 3.6213e-4, 3.5575e-4, 3.505e-4, 3.464e-4, 3.4346e-4,
3.4173e-4, 3.4125e-4, 3.4206e-4, 3.4424e-4, 3.4787e-4, 3.5303e-4,
3.5986e-4, 3.6847e-4, 3.7903e-4, 3.9174e-4, 4.0681e-4, 4.2455e-4,
4.4527e-4, 4.6942e-4, 4.9637e-4, 5.2698e-4, 5.5808e-4, 5.9514e-4,
6.2757e-4, 6.689e-4, 7.1298e-4, 7.3955e-4, 7.8403e-4, 8.0449e-4,
8.5131e-4, 9.0256e-4, 9.3692e-4, .0010051, .0010846, .0011678,
.001282, .0014016, .0015355, .0016764, .0018272, .0020055,
.0021455, .0023421, .0024615, .0026786, .0028787, .0031259,
.0034046, .0036985, .0040917, .0043902, .0048349, .0049531,
.0052989, .0056148, .0052452, .0053357, .005333, .0045069,
.0043851, .004253, .003738, .0038084, .0039013, .0041505,
.0045372, .0050569, .0054507, .0061267, .0066122, .0072449,
.0078012, .0082651, .0076538, .0076573, .0076806, .0075227,
.0076269, .0063758, .006254, .0067749, .0067909, .0068231,
.0072143, .0072762, .0072954, .007679, .0075107, .0073658,
.0072441, .0071074, .0070378, .007176, .0072472, .0075844,
.0079291, .008412, .0090165, .010688, .011535, .012375, .013166,
.013895, .015567, .016011, .016392, .016737, .017043, .017731,
.018031, .018419, .018877, .019474, .019868, .020604, .021538,
.022653, .023869, .025288, .026879, .028547, .030524, .03274,
.035132, .03769, .040567, .043793, .047188, .049962, .053542,
.057205, .060776, .061489, .064419, .067124, .065945, .068487,
.071209, .074783, .077039, .082444, .08902, .09692, .10617,
```

```
    .11687, .12952, .12362, .13498, .14412, .15492, .16519, .1744,
    .17096, .17714, .18208, .17363, .17813, .18564, .18295, .19045,
    .20252, .20815, .21844, .22929, .24229, .25321, .26588, .2797,
    .29465, .31136, .32961, .36529, .38486, .41027, .43694, .4667,
    .49943, .54542, .58348, .62303, .67633, .71755, .76054, .81371,
    .85934, .90841, .96438, 1.0207, 1.0821, 1.1491, 1.2226, 1.3018,
    1.388, 1.4818, 1.5835, 1.6939, 1.8137, 1.9435, 2.0843, 2.237,
    2.4026, 2.5818, 2.7767, 2.9885, 3.2182, 3.4679, 3.7391, 4.0349,
    4.3554, 4.7053, 5.0849, 5.4986, 5.9436, 6.4294, 6.9598, 7.5203,
    8.143, 8.8253, 9.5568, 10.371, 11.267, 12.233, 13.31, 14.357,
    15.598, 16.93, 18.358, 19.849, 21.408, 23.04, 24.706, 26.409,
    28.153, 28.795, 30.549, 32.43, 34.49, 36.027, 38.955, 42.465,
    46.565, 50.875, 55.378, 59.002, 63.882, 67.949, 73.693, 80.095,
    86.403, 94.264, 102.65, 112.37, 123.3, 135.54, 149.14, 163.83,
    179.17, 196.89, 217.91, 240.94, 264.13, 292.39, 324.83, 358.21,
    397.16, 440.5, 488.6, 541.04, 595.3, 650.43, 652.03, 688.74,
    719.47, 743.54, 757.68, 762.35, 756.43, 741.42, 595.43, 580.97,
    580.83, 605.68, 667.88, 764.49, 759.93, 789.12, 798.17, 645.66,
    615.65, 455.05, 421.09, 306.45, 289.14, 235.7, 215.52, 274.57,
    316.53, 357.73, 409.89, 465.06, 521.84, 579.02, 630.64, 794.46,
    813., 813.56, 796.25, 761.57, 727.97, 812.14, 866.75, 932.5,
    1132.8, 1194.8, 1362.2, 1387.2, 1482.3, 1479.7, 1517.9, 1533.1,
    1534.2, 1523.3, 1522.5, 1515.5, 1505.2, 1486.5, 1454., 1412.,
    1358.8, 1107.8, 1060.9, 1033.5, 1048.2, 1122.4, 1248.9, 1227.1,
    1255.4, 1058.9, 1020.7, 970.59, 715.24, 512.56, 468.47, 349.3,
    338.26, 299.22, 301.26, 332.38, 382.08, 445.49, 515.87, 590.85,
    662.3, 726.05, 955.59, 964.11, 945.17, 891.48, 807.11, 720.9,
    803.36, 834.46, 1073.9, 1107.1, 1123.6, 1296., 1393.7, 1303.1,
    1284.3, 1161.8, 1078.8, 976.13, 868.72, 767.4, 674.72, 593.73,
    523.12, 462.24, 409.75, 364.34, 325., 290.73, 260.76, 234.46,
    211.28, 190.78, 172.61, 156.44, 142.01, 129.12, 117.57, 107.2,
    97.877, 89.47, 81.882, 75.021, 68.807, 63.171, 58.052, 53.396,
    49.155, 45.288, 41.759, 38.531, 35.576, 32.868, 30.384, 28.102,
    26.003, 24.071, 22.293, 20.655, 19.147, 17.756, 16.476, 15.292,
    14.198, 13.183, 12.241, 11.367, 10.554, 9.7989, 9.0978, 8.4475,
    7.845, 7.2868, 6.7704, 6.2927, 5.8508, 5.4421, 5.064, 4.714,
    4.3902, 4.0902, 3.8121, 3.5543, 3.315, 3.093, 2.8869, 2.6953,
    2.5172, 2.3517, 2.1977, 2.0544, 1.9211, 1.7969, 1.6812, 1.5735,
    1.4731, 1.3794, 1.2921, 1.2107, 1.1346, 1.0637, .99744, .93554,
    .87771, .82368, .77313, .72587, .6816, .64014, .60134, .565,
    .53086, .49883, .46881, .44074, .4144, .38979, .36679, .34513,
    .32474, .30552, .28751, .27045, .25458, .23976, .22584, .21278,
    .20051, .18899, .17815, .16801, .15846, .14954, .14117, .13328,
    .12584
  };

  double xw, dw, ew, cw296, cw260, cw230, dt230, dt260, dt296, ctw, ctmpth;

  int iw;

  /* Get CO2 continuum absorption... */
  xw = nu / 2 + 1;
  if (xw >= 1 && xw < 2001) {
    iw = (int) xw;
    dw = xw - iw;
    ew = 1 - dw;
    cw296 = ew * co2296[iw - 1] + dw * co2296[iw];
    cw260 = ew * co2260[iw - 1] + dw * co2260[iw];
    cw230 = ew * co2230[iw - 1] + dw * co2230[iw];
    dt230 = t - 230;
    dt260 = t - 260;
    dt296 = t - 296;
    ctw = dt260 * 5.050505e-4 * dt296 * cw230 - dt230 * 9.259259e-4
      * dt296 * cw260 + dt230 * 4.208754e-4 * dt260 * cw296;
    ctmpth = u / GSL_CONST_NUM_AVOGADRO / 1000 * p / P0 * ctw;
  } else
    ctmpth = 0;
  return ctmpth;
}
```

5.31.2.7 double ctmh2o (double *nu*, double *p*, double *t*, double *q*, double *u*)

Compute water vapor continuum (optical depth).

Definition at line 1737 of file jurassic.c.

```
                {

  static double h2o296[2001] = { .17, .1695, .172, .168, .1687, .1624, .1606,
    .1508, .1447, .1344, .1214, .1133, .1009, .09217, .08297, .06989,
```

```
.06513, .05469, .05056, .04417, .03779, .03484, .02994, .0272,
.02325, .02063, .01818, .01592, .01405, .01251, .0108, .009647,
.008424, .007519, .006555, .00588, .005136, .004511, .003989,
.003509, .003114, .00274, .002446, .002144, .001895, .001676,
.001486, .001312, .001164, .001031, 9.129e-4, 8.106e-4, 7.213e-4,
6.4e-4, 5.687e-4, 5.063e-4, 4.511e-4, 4.029e-4, 3.596e-4,
3.22e-4, 2.889e-4, 2.597e-4, 2.337e-4, 2.108e-4, 1.907e-4,
1.728e-4, 1.57e-4, 1.43e-4, 1.305e-4, 1.195e-4, 1.097e-4,
1.009e-4, 9.307e-5, 8.604e-5, 7.971e-5, 7.407e-5, 6.896e-5,
6.433e-5, 6.013e-5, 5.631e-5, 5.283e-5, 4.963e-5, 4.669e-5,
4.398e-5, 4.148e-5, 3.917e-5, 3.702e-5, 3.502e-5, 3.316e-5,
3.142e-5, 2.978e-5, 2.825e-5, 2.681e-5, 2.546e-5, 2.419e-5,
2.299e-5, 2.186e-5, 2.079e-5, 1.979e-5, 1.884e-5, 1.795e-5,
1.711e-5, 1.633e-5, 1.559e-5, 1.49e-5, 1.426e-5, 1.367e-5,
1.312e-5, 1.263e-5, 1.218e-5, 1.178e-5, 1.143e-5, 1.112e-5,
1.088e-5, 1.07e-5, 1.057e-5, 1.05e-5, 1.051e-5, 1.059e-5,
1.076e-5, 1.1e-5, 1.133e-5, 1.18e-5, 1.237e-5, 1.308e-5,
1.393e-5, 1.483e-5, 1.614e-5, 1.758e-5, 1.93e-5, 2.123e-5,
2.346e-5, 2.647e-5, 2.93e-5, 3.279e-5, 3.745e-5, 4.152e-5,
4.813e-5, 5.477e-5, 6.203e-5, 7.331e-5, 8.056e-5, 9.882e-5,
1.05e-4, 1.21e-4, 1.341e-4, 1.572e-4, 1.698e-4, 1.968e-4,
2.175e-4, 2.431e-4, 2.735e-4, 2.867e-4, 3.19e-4, 3.371e-4,
3.554e-4, 3.726e-4, 3.837e-4, 3.878e-4, 3.864e-4, 3.858e-4,
3.841e-4, 3.852e-4, 3.815e-4, 3.762e-4, 3.618e-4, 3.579e-4,
3.45e-4, 3.202e-4, 3.018e-4, 2.785e-4, 2.602e-4, 2.416e-4,
2.097e-4, 1.939e-4, 1.689e-4, 1.498e-4, 1.308e-4, 1.17e-4,
1.011e-4, 9.237e-5, 7.909e-5, 7.006e-5, 6.112e-5, 5.401e-5,
4.914e-5, 4.266e-5, 3.963e-5, 3.316e-5, 3.037e-5, 2.598e-5,
2.294e-5, 2.066e-5, 1.813e-5, 1.583e-5, 1.423e-5, 1.247e-5,
1.116e-5, 9.76e-6, 8.596e-6, 7.72e-6, 6.825e-6, 6.108e-6,
5.366e-6, 4.733e-6, 4.229e-6, 3.731e-6, 3.346e-6, 2.972e-6,
2.628e-6, 2.356e-6, 2.102e-6, 1.878e-6, 1.678e-6, 1.507e-6,
1.348e-6, 1.21e-6, 1.089e-6, 9.806e-7, 8.857e-7, 8.004e-7,
7.261e-7, 6.599e-7, 6.005e-7, 5.479e-7, 5.011e-7, 4.595e-7,
4.219e-7, 3.885e-7, 3.583e-7, 3.314e-7, 3.071e-7, 2.852e-7,
2.654e-7, 2.474e-7, 2.311e-7, 2.162e-7, 2.026e-7, 1.902e-7,
1.788e-7, 1.683e-7, 1.587e-7, 1.497e-7, 1.415e-7, 1.338e-7,
1.266e-7, 1.2e-7, 1.138e-7, 1.08e-7, 1.027e-7, 9.764e-8,
9.296e-8, 8.862e-8, 8.458e-8, 8.087e-8, 7.744e-8, 7.429e-8,
7.145e-8, 6.893e-8, 6.664e-8, 6.468e-8, 6.322e-8, 6.162e-8,
6.07e-8, 5.992e-8, 5.913e-8, 5.841e-8, 5.796e-8, 5.757e-8,
5.746e-8, 5.731e-8, 5.679e-8, 5.577e-8, 5.671e-8, 5.656e-8,
5.594e-8, 5.593e-8, 5.602e-8, 5.62e-8, 5.693e-8, 5.725e-8,
5.858e-8, 6.037e-8, 6.249e-8, 6.535e-8, 6.899e-8, 7.356e-8,
7.918e-8, 8.618e-8, 9.385e-8, 1.039e-7, 1.158e-7, 1.29e-7,
1.437e-7, 1.65e-7, 1.871e-7, 2.121e-7, 2.427e-7, 2.773e-7,
3.247e-7, 3.677e-7, 4.037e-7, 4.776e-7, 5.101e-7, 6.214e-7,
6.936e-7, 7.581e-7, 8.486e-7, 9.355e-7, 9.942e-7, 1.063e-6,
1.123e-6, 1.191e-6, 1.215e-6, 1.247e-6, 1.26e-6, 1.271e-6,
1.284e-6, 1.317e-6, 1.323e-6, 1.349e-6, 1.353e-6, 1.362e-6,
1.344e-6, 1.329e-6, 1.336e-6, 1.327e-6, 1.325e-6, 1.359e-6,
1.374e-6, 1.415e-6, 1.462e-6, 1.526e-6, 1.619e-6, 1.735e-6,
1.863e-6, 2.034e-6, 2.265e-6, 2.482e-6, 2.756e-6, 3.103e-6,
3.466e-6, 3.832e-6, 4.378e-6, 4.913e-6, 5.651e-6, 6.311e-6,
7.169e-6, 8.057e-6, 9.253e-6, 1.047e-5, 1.212e-5, 1.36e-5,
1.569e-5, 1.776e-5, 2.02e-5, 2.281e-5, 2.683e-5, 2.994e-5,
3.488e-5, 3.896e-5, 4.499e-5, 5.175e-5, 6.035e-5, 6.34e-5,
7.281e-5, 7.923e-5, 8.348e-5, 9.631e-5, 1.044e-4, 1.102e-4,
1.176e-4, 1.244e-4, 1.283e-4, 1.326e-4, 1.4e-4, 1.395e-4,
1.387e-4, 1.363e-4, 1.314e-4, 1.241e-4, 1.228e-4, 1.148e-4,
1.086e-4, 1.018e-4, 8.89e-5, 8.316e-5, 7.292e-5, 6.452e-5,
5.625e-5, 5.045e-5, 4.38e-5, 3.762e-5, 3.29e-5, 2.836e-5,
2.485e-5, 2.168e-5, 1.895e-5, 1.659e-5, 1.453e-5, 1.282e-5,
1.132e-5, 1.001e-5, 8.836e-6, 7.804e-6, 6.922e-6, 6.116e-6,
5.429e-6, 4.824e-6, 4.278e-6, 3.788e-6, 3.371e-6, 2.985e-6,
2.649e-6, 2.357e-6, 2.09e-6, 1.858e-6, 1.647e-6, 1.462e-6,
1.299e-6, 1.155e-6, 1.028e-6, 9.142e-7, 8.132e-7, 7.246e-7,
6.451e-7, 5.764e-7, 5.151e-7, 4.603e-7, 4.121e-7, 3.694e-7,
3.318e-7, 2.985e-7, 2.69e-7, 2.428e-7, 2.197e-7, 1.992e-7,
1.81e-7, 1.649e-7, 1.506e-7, 1.378e-7, 1.265e-7, 1.163e-7,
1.073e-7, 9.918e-8, 9.191e-8, 8.538e-8, 7.949e-8, 7.419e-8,
6.94e-8, 6.508e-8, 6.114e-8, 5.761e-8, 5.437e-8, 5.146e-8,
4.89e-8, 4.636e-8, 4.406e-8, 4.201e-8, 4.015e-8, 3.84e-8,
3.661e-8, 3.51e-8, 3.377e-8, 3.242e-8, 3.13e-8, 3.015e-8,
2.918e-8, 2.83e-8, 2.758e-8, 2.707e-8, 2.656e-8, 2.619e-8,
2.609e-8, 2.615e-8, 2.63e-8, 2.675e-8, 2.745e-8, 2.842e-8,
2.966e-8, 3.125e-8, 3.318e-8, 3.565e-8, 3.85e-8, 4.191e-8,
4.59e-8, 5.059e-8, 5.607e-8, 6.239e-8, 6.958e-8, 7.796e-8,
8.773e-8, 9.88e-8, 1.114e-7, 1.258e-7, 1.422e-7, 1.61e-7,
1.822e-7, 2.06e-7, 2.337e-7, 2.645e-7, 2.996e-7, 3.393e-7,
3.843e-7, 4.363e-7, 4.935e-7, 5.607e-7, 6.363e-7, 7.242e-7,
8.23e-7, 9.411e-7, 1.071e-6, 1.232e-6, 1.402e-6, 1.6e-6, 1.82e-6,
2.128e-6, 2.386e-6, 2.781e-6, 3.242e-6, 3.653e-6, 4.323e-6,
4.747e-6, 5.321e-6, 5.919e-6, 6.681e-6, 7.101e-6, 7.983e-6,
8.342e-6, 8.741e-6, 9.431e-6, 9.952e-6, 1.026e-5, 1.055e-5,
1.095e-5, 1.095e-5, 1.087e-5, 1.056e-5, 1.026e-5, 9.715e-6,
9.252e-6, 8.452e-6, 7.958e-6, 7.268e-6, 6.295e-6, 6.003e-6, 5e-6,
```

```
4.591e-6, 3.983e-6, 3.479e-6, 3.058e-6, 2.667e-6, 2.293e-6,
1.995e-6, 1.747e-6, 1.517e-6, 1.335e-6, 1.165e-6, 1.028e-6,
9.007e-7, 7.956e-7, 7.015e-7, 6.192e-7, 5.491e-7, 4.859e-7,
4.297e-7, 3.799e-7, 3.38e-7, 3.002e-7, 2.659e-7, 2.366e-7,
2.103e-7, 1.861e-7, 1.655e-7, 1.469e-7, 1.309e-7, 1.162e-7,
1.032e-7, 9.198e-8, 8.181e-8, 7.294e-8, 6.516e-8, 5.787e-8,
5.163e-8, 4.612e-8, 4.119e-8, 3.695e-8, 3.308e-8, 2.976e-8,
2.67e-8, 2.407e-8, 2.171e-8, 1.965e-8, 1.78e-8, 1.617e-8,
1.47e-8, 1.341e-8, 1.227e-8, 1.125e-8, 1.033e-8, 9.524e-9,
8.797e-9, 8.162e-9, 7.565e-9, 7.04e-9, 6.56e-9, 6.129e-9,
5.733e-9, 5.376e-9, 5.043e-9, 4.75e-9, 4.466e-9, 4.211e-9,
3.977e-9, 3.759e-9, 3.558e-9, 3.373e-9, 3.201e-9, 3.043e-9,
2.895e-9, 2.76e-9, 2.635e-9, 2.518e-9, 2.411e-9, 2.314e-9,
2.23e-9, 2.151e-9, 2.087e-9, 2.035e-9, 1.988e-9, 1.946e-9,
1.927e-9, 1.916e-9, 1.916e-9, 1.933e-9, 1.966e-9, 2.018e-9,
2.09e-9, 2.182e-9, 2.299e-9, 2.442e-9, 2.623e-9, 2.832e-9,
3.079e-9, 3.368e-9, 3.714e-9, 4.104e-9, 4.567e-9, 5.091e-9,
5.701e-9, 6.398e-9, 7.194e-9, 8.127e-9, 9.141e-9, 1.035e-8,
1.177e-8, 1.338e-8, 1.508e-8, 1.711e-8, 1.955e-8, 2.216e-8,
2.534e-8, 2.871e-8, 3.291e-8, 3.711e-8, 4.285e-8, 4.868e-8,
5.509e-8, 6.276e-8, 7.262e-8, 8.252e-8, 9.4e-8, 1.064e-7,
1.247e-7, 1.411e-7, 1.626e-7, 1.827e-7, 2.044e-7, 2.284e-7,
2.452e-7, 2.854e-7, 3.026e-7, 3.278e-7, 3.474e-7, 3.693e-7,
3.93e-7, 4.104e-7, 4.22e-7, 4.439e-7, 4.545e-7, 4.778e-7,
4.812e-7, 5.018e-7, 4.899e-7, 5.075e-7, 5.073e-7, 5.171e-7,
5.131e-7, 5.25e-7, 5.617e-7, 5.846e-7, 6.239e-7, 6.696e-7,
7.398e-7, 8.073e-7, 9.15e-7, 1.009e-6, 1.116e-6, 1.264e-6,
1.439e-6, 1.644e-6, 1.856e-6, 2.147e-6, 2.317e-6, 2.713e-6,
2.882e-6, 2.99e-6, 3.489e-6, 3.581e-6, 4.033e-6, 4.26e-6,
4.543e-6, 4.84e-6, 4.826e-6, 5.013e-6, 5.252e-6, 5.277e-6,
5.306e-6, 5.236e-6, 5.123e-6, 5.171e-6, 4.843e-6, 4.615e-6,
4.385e-6, 3.97e-6, 3.693e-6, 3.231e-6, 2.915e-6, 2.495e-6,
2.144e-6, 1.91e-6, 1.639e-6, 1.417e-6, 1.226e-6, 1.065e-6,
9.29e-7, 8.142e-7, 7.161e-7, 6.318e-7, 5.581e-7, 4.943e-7,
4.376e-7, 3.884e-7, 3.449e-7, 3.06e-7, 2.712e-7, 2.412e-7,
2.139e-7, 1.903e-7, 1.689e-7, 1.499e-7, 1.331e-7, 1.183e-7,
1.05e-7, 9.362e-8, 8.306e-8, 7.403e-8, 6.578e-8, 5.853e-8,
5.216e-8, 4.632e-8, 4.127e-8, 3.678e-8, 3.279e-8, 2.923e-8,
2.612e-8, 2.339e-8, 2.094e-8, 1.877e-8, 1.686e-8, 1.516e-8,
1.366e-8, 1.234e-8, 1.114e-8, 1.012e-8, 9.182e-9, 8.362e-9,
7.634e-9, 6.981e-9, 6.406e-9, 5.888e-9, 5.428e-9, 5.021e-9,
4.65e-9, 4.326e-9, 4.033e-9, 3.77e-9, 3.536e-9, 3.327e-9,
3.141e-9, 2.974e-9, 2.825e-9, 2.697e-9, 2.584e-9, 2.488e-9,
2.406e-9, 2.34e-9, 2.292e-9, 2.259e-9, 2.244e-9, 2.243e-9,
2.272e-9, 2.31e-9, 2.378e-9, 2.454e-9, 2.618e-9, 2.672e-9,
2.831e-9, 3.05e-9, 3.225e-9, 3.425e-9, 3.677e-9, 3.968e-9,
4.221e-9, 4.639e-9, 4.96e-9, 5.359e-9, 5.649e-9, 6.23e-9,
6.716e-9, 7.218e-9, 7.746e-9, 7.988e-9, 8.627e-9, 8.999e-9,
9.442e-9, 9.82e-9, 1.015e-8, 1.06e-8, 1.079e-8, 1.109e-8,
1.137e-8, 1.186e-8, 1.18e-8, 1.187e-8, 1.194e-8, 1.192e-8,
1.224e-8, 1.245e-8, 1.246e-8, 1.318e-8, 1.377e-8, 1.471e-8,
1.582e-8, 1.713e-8, 1.853e-8, 2.063e-8, 2.27e-8, 2.567e-8,
2.891e-8, 3.264e-8, 3.744e-8, 4.286e-8, 4.915e-8, 5.623e-8,
6.336e-8, 7.293e-8, 8.309e-8, 9.319e-8, 1.091e-7, 1.243e-7,
1.348e-7, 1.449e-7, 1.62e-7, 1.846e-7, 1.937e-7, 2.04e-7,
2.179e-7, 2.298e-7, 2.433e-7, 2.439e-7, 2.464e-7, 2.611e-7,
2.617e-7, 2.582e-7, 2.453e-7, 2.401e-7, 2.349e-7, 2.203e-7,
2.066e-7, 1.939e-7, 1.78e-7, 1.558e-7, 1.391e-7, 1.203e-7,
1.048e-7, 9.464e-8, 8.306e-8, 7.239e-8, 6.317e-8, 5.52e-8,
4.847e-8, 4.282e-8, 3.796e-8, 3.377e-8, 2.996e-8, 2.678e-8,
2.4e-8, 2.134e-8, 1.904e-8, 1.705e-8, 1.523e-8, 1.35e-8,
1.204e-8, 1.07e-8, 9.408e-9, 8.476e-9, 7.47e-9, 6.679e-9,
5.929e-9, 5.267e-9, 4.711e-9, 4.172e-9, 3.761e-9, 3.288e-9,
2.929e-9, 2.609e-9, 2.315e-9, 2.042e-9, 1.844e-9, 1.64e-9,
1.47e-9, 1.31e-9, 1.176e-9, 1.049e-9, 9.377e-10, 8.462e-10,
7.616e-10, 6.854e-10, 6.191e-10, 5.596e-10, 5.078e-10, 4.611e-10,
4.197e-10, 3.83e-10, 3.505e-10, 3.215e-10, 2.956e-10, 2.726e-10,
2.521e-10, 2.338e-10, 2.173e-10, 2.026e-10, 1.895e-10, 1.777e-10,
1.672e-10, 1.579e-10, 1.496e-10, 1.423e-10, 1.358e-10, 1.302e-10,
1.254e-10, 1.216e-10, 1.187e-10, 1.163e-10, 1.147e-10, 1.145e-10,
1.15e-10, 1.17e-10, 1.192e-10, 1.25e-10, 1.298e-10, 1.345e-10,
1.405e-10, 1.538e-10, 1.648e-10, 1.721e-10, 1.872e-10, 1.968e-10,
2.089e-10, 2.172e-10, 2.317e-10, 2.389e-10, 2.503e-10, 2.585e-10,
2.686e-10, 2.8e-10, 2.895e-10, 3.019e-10, 3.037e-10, 3.076e-10,
3.146e-10, 3.198e-10, 3.332e-10, 3.397e-10, 3.54e-10, 3.667e-10,
3.895e-10, 4.071e-10, 4.565e-10, 4.983e-10, 5.439e-10, 5.968e-10,
6.676e-10, 7.456e-10, 8.405e-10, 9.478e-10, 1.064e-9, 1.218e-9,
1.386e-9, 1.581e-9, 1.787e-9, 2.032e-9, 2.347e-9, 2.677e-9,
3.008e-9, 3.544e-9, 4.056e-9, 4.687e-9, 5.331e-9, 6.227e-9,
6.854e-9, 8.139e-9, 8.945e-9, 9.865e-9, 1.125e-8, 1.178e-8,
1.364e-8, 1.436e-8, 1.54e-8, 1.672e-8, 1.793e-8, 1.906e-8,
2.036e-8, 2.144e-8, 2.292e-8, 2.371e-8, 2.493e-8, 2.606e-8,
2.706e-8, 2.866e-8, 3.036e-8, 3.136e-8, 3.405e-8, 3.665e-8,
3.837e-8, 4.229e-8, 4.748e-8, 5.32e-8, 5.763e-8, 6.677e-8,
7.216e-8, 7.716e-8, 8.958e-8, 9.419e-8, 1.036e-7, 1.108e-7,
1.189e-7, 1.246e-7, 1.348e-7, 1.31e-7, 1.361e-7, 1.364e-7,
1.363e-7, 1.343e-7, 1.293e-7, 1.254e-7, 1.235e-7, 1.158e-7,
```

```
1.107e-7, 9.961e-8, 9.011e-8, 7.91e-8, 6.916e-8, 6.338e-8,
5.564e-8, 4.827e-8, 4.198e-8, 3.695e-8, 3.276e-8, 2.929e-8,
2.633e-8, 2.391e-8, 2.192e-8, 2.021e-8, 1.89e-8, 1.772e-8,
1.667e-8, 1.603e-8, 1.547e-8, 1.537e-8, 1.492e-8, 1.515e-8,
1.479e-8, 1.45e-8, 1.513e-8, 1.495e-8, 1.529e-8, 1.565e-8,
1.564e-8, 1.553e-8, 1.569e-8, 1.584e-8, 1.57e-8, 1.538e-8,
1.513e-8, 1.472e-8, 1.425e-8, 1.349e-8, 1.328e-8, 1.249e-8,
1.17e-8, 1.077e-8, 9.514e-9, 8.614e-9, 7.46e-9, 6.621e-9,
5.775e-9, 5.006e-9, 4.308e-9, 3.747e-9, 3.24e-9, 2.84e-9,
2.481e-9, 2.184e-9, 1.923e-9, 1.71e-9, 1.504e-9, 1.334e-9,
1.187e-9, 1.053e-9, 9.367e-10, 8.306e-10, 7.419e-10, 6.63e-10,
5.918e-10, 5.277e-10, 4.717e-10, 4.222e-10, 3.783e-10, 3.39e-10,
3.036e-10, 2.729e-10, 2.455e-10, 2.211e-10, 1.995e-10, 1.804e-10,
1.635e-10, 1.485e-10, 1.355e-10, 1.24e-10, 1.139e-10, 1.051e-10,
9.757e-11, 9.114e-11, 8.577e-11, 8.139e-11, 7.792e-11, 7.52e-11,
7.39e-11, 7.311e-11, 7.277e-11, 7.482e-11, 7.698e-11, 8.162e-11,
8.517e-11, 8.968e-11, 9.905e-11, 1.075e-10, 1.187e-10, 1.291e-10,
1.426e-10, 1.573e-10, 1.734e-10, 1.905e-10, 2.097e-10, 2.28e-10,
2.473e-10, 2.718e-10, 2.922e-10, 3.128e-10, 3.361e-10, 3.641e-10,
3.91e-10, 4.196e-10, 4.501e-10, 4.932e-10, 5.258e-10, 5.755e-10,
6.253e-10, 6.664e-10, 7.344e-10, 7.985e-10, 8.877e-10, 1.005e-9,
1.118e-9, 1.251e-9, 1.428e-9, 1.61e-9, 1.888e-9, 2.077e-9,
2.331e-9, 2.751e-9, 3.061e-9, 3.522e-9, 3.805e-9, 4.181e-9,
4.575e-9, 5.167e-9, 5.634e-9, 6.007e-9, 6.501e-9, 6.829e-9,
7.211e-9, 7.262e-9, 7.696e-9, 7.832e-9, 7.799e-9, 7.651e-9,
7.304e-9, 7.15e-9, 6.977e-9, 6.603e-9, 6.209e-9, 5.69e-9,
5.432e-9, 4.764e-9, 4.189e-9, 3.64e-9, 3.203e-9, 2.848e-9,
2.51e-9, 2.194e-9, 1.946e-9, 1.75e-9, 1.567e-9, 1.426e-9,
1.302e-9, 1.197e-9, 1.109e-9, 1.035e-9, 9.719e-10, 9.207e-10,
8.957e-10, 8.578e-10, 8.262e-10, 8.117e-10, 7.987e-10, 7.875e-10,
7.741e-10, 7.762e-10, 7.537e-10, 7.424e-10, 7.474e-10, 7.294e-10,
7.216e-10, 7.233e-10, 7.075e-10, 6.892e-10, 6.618e-10, 6.314e-10,
6.208e-10, 5.689e-10, 5.55e-10, 4.984e-10, 4.6e-10, 4.078e-10,
3.879e-10, 3.459e-10, 2.982e-10, 2.626e-10, 2.329e-10, 1.988e-10,
1.735e-10, 1.487e-10, 1.297e-10, 1.133e-10, 9.943e-11, 8.736e-11,
7.726e-11, 6.836e-11, 6.053e-11, 5.384e-11, 4.789e-11, 4.267e-11,
3.804e-11, 3.398e-11, 3.034e-11, 2.71e-11, 2.425e-11, 2.173e-11,
1.95e-11, 1.752e-11, 1.574e-11, 1.418e-11, 1.278e-11, 1.154e-11,
1.044e-11, 9.463e-12, 8.602e-12, 7.841e-12, 7.171e-12, 6.584e-12,
6.073e-12, 5.631e-12, 5.254e-12, 4.937e-12, 4.679e-12, 4.476e-12,
4.328e-12, 4.233e-12, 4.194e-12, 4.211e-12, 4.286e-12, 4.424e-12,
4.628e-12, 4.906e-12, 5.262e-12, 5.708e-12, 6.254e-12, 6.914e-12,
7.714e-12, 8.677e-12, 9.747e-12, 1.101e-11, 1.256e-11, 1.409e-11,
1.597e-11, 1.807e-11, 2.034e-11, 2.316e-11, 2.622e-11, 2.962e-11,
3.369e-11, 3.819e-11, 4.329e-11, 4.932e-11, 5.589e-11, 6.364e-11,
7.284e-11, 8.236e-11, 9.447e-11, 1.078e-10, 1.229e-10, 1.417e-10,
1.614e-10, 1.843e-10, 2.107e-10, 2.406e-10, 2.728e-10, 3.195e-10,
3.595e-10, 4.153e-10, 4.736e-10, 5.41e-10, 6.088e-10, 6.769e-10,
7.691e-10, 8.545e-10, 9.621e-10, 1.047e-9, 1.161e-9, 1.296e-9,
1.424e-9, 1.576e-9, 1.739e-9, 1.893e-9, 2.08e-9, 2.336e-9,
2.604e-9, 2.76e-9, 3.001e-9, 3.365e-9, 3.55e-9, 3.895e-9,
4.183e-9, 4.614e-9, 4.846e-9, 5.068e-9, 5.427e-9, 5.541e-9,
5.864e-9, 5.997e-9, 5.997e-9, 6.061e-9, 5.944e-9, 5.855e-9,
5.661e-9, 5.523e-9, 5.374e-9, 4.94e-9, 4.688e-9, 4.17e-9,
3.913e-9, 3.423e-9, 2.997e-9, 2.598e-9, 2.253e-9, 1.946e-9,
1.71e-9, 1.507e-9, 1.336e-9, 1.19e-9, 1.068e-9, 9.623e-10,
8.772e-10, 8.007e-10, 7.42e-10, 6.884e-10, 6.483e-10, 6.162e-10,
5.922e-10, 5.688e-10, 5.654e-10, 5.637e-10, 5.701e-10, 5.781e-10,
5.874e-10, 6.268e-10, 6.357e-10, 6.525e-10, 7.137e-10, 7.441e-10,
8.024e-10, 8.485e-10, 9.143e-10, 9.536e-10, 9.717e-10, 1.018e-9,
1.042e-9, 1.054e-9, 1.092e-9, 1.079e-9, 1.064e-9, 1.043e-9,
1.02e-9, 9.687e-10, 9.273e-10, 9.208e-10, 9.068e-10, 7.687e-10,
7.385e-10, 6.595e-10, 5.87e-10, 5.144e-10, 4.417e-10, 3.804e-10,
3.301e-10, 2.866e-10, 2.509e-10, 2.202e-10, 1.947e-10, 1.719e-10,
1.525e-10, 1.361e-10, 1.21e-10, 1.084e-10, 9.8e-11, 8.801e-11,
7.954e-11, 7.124e-11, 6.335e-11, 5.76e-11, 5.132e-11, 4.601e-11,
4.096e-11, 3.657e-11, 3.25e-11, 2.909e-11, 2.587e-11, 2.297e-11,
2.05e-11, 1.828e-11, 1.632e-11, 1.462e-11, 1.314e-11, 1.185e-11,
1.073e-11, 9.76e-12, 8.922e-12, 8.206e-12, 7.602e-12, 7.1e-12,
6.694e-12, 6.378e-12, 6.149e-12, 6.004e-12, 5.941e-12, 5.962e-12,
6.069e-12, 6.265e-12, 6.551e-12, 6.935e-12, 7.457e-12, 8.074e-12,
8.811e-12, 9.852e-12, 1.086e-11, 1.207e-11, 1.361e-11, 1.553e-11,
1.737e-11, 1.93e-11, 2.175e-11, 2.41e-11, 2.706e-11, 3.023e-11,
3.313e-11, 3.657e-11, 4.118e-11, 4.569e-11, 5.025e-11, 5.66e-11,
6.231e-11, 6.881e-11, 7.996e-11, 8.526e-11, 9.694e-11, 1.106e-10,
1.222e-10, 1.355e-10, 1.525e-10, 1.775e-10, 1.924e-10, 2.181e-10,
2.379e-10, 2.662e-10, 2.907e-10, 3.154e-10, 3.366e-10, 3.579e-10,
3.858e-10, 4.046e-10, 4.196e-10, 4.166e-10, 4.457e-10, 4.466e-10,
4.404e-10, 4.337e-10, 4.15e-10, 4.083e-10, 3.91e-10, 3.723e-10,
3.514e-10, 3.303e-10, 2.847e-10, 2.546e-10, 2.23e-10, 1.994e-10,
1.733e-10, 1.488e-10, 1.297e-10, 1.144e-10, 1.004e-10, 8.741e-11,
7.928e-11, 7.034e-11, 6.323e-11, 5.754e-11, 5.25e-11, 4.85e-11,
4.502e-11, 4.286e-11, 4.028e-11, 3.899e-11, 3.824e-11, 3.761e-11,
3.804e-11, 3.839e-11, 3.845e-11, 4.244e-11, 4.382e-11, 4.582e-11,
4.847e-11, 5.209e-11, 5.384e-11, 5.887e-11, 6.371e-11, 6.737e-11,
7.168e-11, 7.415e-11, 7.827e-11, 8.037e-11, 8.12e-11, 8.071e-11,
8.008e-11, 7.851e-11, 7.544e-11, 7.377e-11, 7.173e-11, 6.801e-11,
```

```
  6.267e-11, 5.727e-11, 5.288e-11, 4.853e-11, 4.082e-11, 3.645e-11,
  3.136e-11, 2.672e-11, 2.304e-11, 1.986e-11, 1.725e-11, 1.503e-11,
  1.315e-11, 1.153e-11, 1.014e-11, 8.942e-12, 7.901e-12, 6.993e-12,
  6.199e-12, 5.502e-12, 4.89e-12, 4.351e-12, 3.878e-12, 3.461e-12,
  3.094e-12, 2.771e-12, 2.488e-12, 2.241e-12, 2.025e-12, 1.838e-12,
  1.677e-12, 1.541e-12, 1.427e-12, 1.335e-12, 1.262e-12, 1.209e-12,
  1.176e-12, 1.161e-12, 1.165e-12, 1.189e-12, 1.234e-12, 1.3e-12,
  1.389e-12, 1.503e-12, 1.644e-12, 1.814e-12, 2.017e-12, 2.255e-12,
  2.534e-12, 2.858e-12, 3.231e-12, 3.661e-12, 4.153e-12, 4.717e-12,
  5.36e-12, 6.094e-12, 6.93e-12, 7.882e-12, 8.966e-12, 1.02e-11,
  1.162e-11, 1.324e-11, 1.51e-11, 1.72e-11, 1.965e-11, 2.237e-11,
  2.56e-11, 2.927e-11, 3.371e-11, 3.842e-11, 4.429e-11, 5.139e-11,
  5.798e-11, 6.697e-11, 7.626e-11, 8.647e-11, 1.022e-10, 1.136e-10,
  1.3e-10, 1.481e-10, 1.672e-10, 1.871e-10, 2.126e-10, 2.357e-10,
  2.583e-10, 2.997e-10, 3.289e-10, 3.702e-10, 4.012e-10, 4.319e-10,
  4.527e-10, 5.001e-10, 5.448e-10, 5.611e-10, 5.76e-10, 5.965e-10,
  6.079e-10, 6.207e-10, 6.276e-10, 6.222e-10, 6.137e-10, 6e-10,
  5.814e-10, 5.393e-10, 5.35e-10, 4.947e-10, 4.629e-10, 4.117e-10,
  3.712e-10, 3.372e-10, 2.923e-10, 2.55e-10, 2.232e-10, 1.929e-10,
  1.679e-10, 1.46e-10, 1.289e-10, 1.13e-10, 9.953e-11, 8.763e-11,
  7.76e-11, 6.9e-11, 6.16e-11, 5.525e-11, 4.958e-11, 4.489e-11,
  4.072e-11, 3.728e-11, 3.438e-11, 3.205e-11, 3.006e-11, 2.848e-11,
  2.766e-11, 2.688e-11, 2.664e-11, 2.67e-11, 2.696e-11, 2.786e-11,
  2.861e-11, 3.009e-11, 3.178e-11, 3.389e-11, 3.587e-11, 3.819e-11,
  4.054e-11, 4.417e-11, 4.703e-11, 5.137e-11, 5.46e-11, 6.055e-11,
  6.333e-11, 6.773e-11, 7.219e-11, 7.717e-11, 8.131e-11, 8.491e-11,
  8.574e-11, 9.01e-11, 9.017e-11, 8.999e-11, 8.959e-11, 8.838e-11,
  8.579e-11, 8.162e-11, 8.098e-11, 7.472e-11, 7.108e-11, 6.559e-11,
  5.994e-11, 5.172e-11, 4.424e-11, 3.951e-11, 3.34e-11, 2.902e-11,
  2.541e-11, 2.215e-11, 1.945e-11, 1.716e-11, 1.503e-11, 1.339e-11,
  1.185e-11, 1.05e-11, 9.336e-12, 8.307e-12, 7.312e-12, 6.55e-12,
  5.836e-12, 5.178e-12, 4.6e-12, 4.086e-12, 3.639e-12, 3.247e-12,
  2.904e-12, 2.604e-12, 2.341e-12, 2.112e-12, 1.914e-12, 1.744e-12,
  1.598e-12, 1.476e-12, 1.374e-12, 1.293e-12, 1.23e-12, 1.185e-12,
  1.158e-12, 1.147e-12, 1.154e-12, 1.177e-12, 1.219e-12, 1.28e-12,
  1.36e-12, 1.463e-12, 1.591e-12, 1.75e-12, 1.94e-12, 2.156e-12,
  2.43e-12, 2.748e-12, 3.052e-12, 3.533e-12, 3.967e-12, 4.471e-12,
  5.041e-12, 5.86e-12, 6.664e-12, 7.522e-12, 8.342e-12, 9.412e-12,
  1.072e-11, 1.213e-11, 1.343e-11, 1.496e-11, 1.664e-11, 1.822e-11,
  2.029e-11, 2.233e-11, 2.457e-11, 2.709e-11, 2.928e-11, 3.115e-11,
  3.356e-11, 3.592e-11, 3.818e-11, 3.936e-11, 4.061e-11, 4.149e-11,
  4.299e-11, 4.223e-11, 4.251e-11, 4.287e-11, 4.177e-11, 4.094e-11,
  3.942e-11, 3.772e-11, 3.614e-11, 3.394e-11, 3.222e-11, 2.791e-11,
  2.665e-11, 2.309e-11, 2.032e-11, 1.74e-11, 1.535e-11, 1.323e-11,
  1.151e-11, 9.803e-12, 8.65e-12, 7.54e-12, 6.619e-12, 5.832e-12,
  5.113e-12, 4.503e-12, 3.975e-12, 3.52e-12, 3.112e-12, 2.797e-12,
  2.5e-12, 2.24e-12, 2.013e-12, 1.819e-12, 1.653e-12, 1.513e-12,
  1.395e-12, 1.299e-12, 1.225e-12, 1.168e-12, 1.124e-12, 1.148e-12,
  1.107e-12, 1.128e-12, 1.169e-12, 1.233e-12, 1.307e-12, 1.359e-12,
  1.543e-12, 1.686e-12, 1.794e-12, 2.028e-12, 2.21e-12, 2.441e-12,
  2.653e-12, 2.828e-12, 3.093e-12, 3.28e-12, 3.551e-12, 3.677e-12,
  3.803e-12, 3.844e-12, 4.068e-12, 4.093e-12, 4.002e-12, 3.904e-12,
  3.624e-12, 3.633e-12, 3.622e-12, 3.443e-12, 3.184e-12, 2.934e-12,
  2.476e-12, 2.212e-12, 1.867e-12, 1.594e-12, 1.37e-12, 1.192e-12,
  1.045e-12, 9.211e-13, 8.17e-13, 7.29e-13, 6.55e-13, 5.929e-13,
  5.415e-13, 4.995e-13, 4.661e-13, 4.406e-13, 4.225e-13, 4.116e-13,
  4.075e-13, 4.102e-13, 4.198e-13, 4.365e-13, 4.606e-13, 4.925e-13,
  5.326e-13, 5.818e-13, 6.407e-13, 7.104e-13, 7.92e-13, 8.868e-13,
  9.964e-13, 1.123e-12, 1.268e-12, 1.434e-12, 1.626e-12, 1.848e-12,
  2.107e-12, 2.422e-12, 2.772e-12, 3.145e-12, 3.704e-12, 4.27e-12,
  4.721e-12, 5.361e-12, 6.083e-12, 7.095e-12, 7.968e-12, 9.228e-12,
  1.048e-11, 1.187e-11, 1.336e-11, 1.577e-11, 1.772e-11, 2.017e-11,
  2.25e-11, 2.63e-11, 2.911e-11, 3.356e-11, 3.82e-11, 4.173e-11,
  4.811e-11, 5.254e-11, 5.839e-11, 6.187e-11, 6.805e-11, 7.118e-11,
  7.369e-11, 7.664e-11, 7.794e-11, 7.947e-11, 8.036e-11, 7.954e-11,
  7.849e-11, 7.518e-11, 7.462e-11, 6.926e-11, 6.531e-11, 6.197e-11,
  5.421e-11, 4.777e-11, 4.111e-11, 3.679e-11, 3.166e-11, 2.786e-11,
  2.436e-11, 2.144e-11, 1.859e-11, 1.628e-11, 1.414e-11, 1.237e-11,
  1.093e-11, 9.558e-12
};

static double h2o260[2001] = { .2752, .2732, .2749, .2676, .2667, .2545,
  .2497, .2327, .2218, .2036, .1825, .1694, .1497, .1353, .121,
  .1014, .09405, .07848, .07195, .06246, .05306, .04853, .04138,
  .03735, .03171, .02785, .02431, .02111, .01845, .0164, .01405,
  .01255, .01098, .009797, .008646, .007779, .006898, .006099,
  .005453, .004909, .004413, .003959, .003581, .003199, .002871,
  .002583, .00233, .002086, .001874, .001684, .001512, .001361,
  .001225, .0011, 9.89e-4, 8.916e-4, 8.039e-4, 7.256e-4, 6.545e-4,
  5.918e-4, 5.359e-4, 4.867e-4, 4.426e-4, 4.033e-4, 3.682e-4,
  3.366e-4, 3.085e-4, 2.833e-4, 2.605e-4, 2.403e-4, 2.221e-4,
  2.055e-4, 1.908e-4, 1.774e-4, 1.653e-4, 1.544e-4, 1.443e-4,
  1.351e-4, 1.267e-4, 1.19e-4, 1.119e-4, 1.053e-4, 9.922e-5,
  9.355e-5, 8.831e-5, 8.339e-5, 7.878e-5, 7.449e-5, 7.043e-5,
  6.664e-5, 6.307e-5, 5.969e-5, 5.654e-5, 5.357e-5, 5.075e-5,
  4.81e-5, 4.56e-5, 4.322e-5, 4.102e-5, 3.892e-5, 3.696e-5,
  3.511e-5, 3.339e-5, 3.177e-5, 3.026e-5, 2.886e-5, 2.756e-5,
```

```
2.636e-5, 2.527e-5, 2.427e-5, 2.337e-5, 2.257e-5, 2.185e-5,
2.127e-5, 2.08e-5, 2.041e-5, 2.013e-5, 2e-5, 1.997e-5, 2.009e-5,
2.031e-5, 2.068e-5, 2.124e-5, 2.189e-5, 2.267e-5, 2.364e-5,
2.463e-5, 2.618e-5, 2.774e-5, 2.937e-5, 3.144e-5, 3.359e-5,
3.695e-5, 4.002e-5, 4.374e-5, 4.947e-5, 5.431e-5, 6.281e-5,
7.169e-5, 8.157e-5, 9.728e-5, 1.079e-4, 1.337e-4, 1.442e-4,
1.683e-4, 1.879e-4, 2.223e-4, 2.425e-4, 2.838e-4, 3.143e-4,
3.527e-4, 4.012e-4, 4.237e-4, 4.747e-4, 5.057e-4, 5.409e-4,
5.734e-4, 5.944e-4, 6.077e-4, 6.175e-4, 6.238e-4, 6.226e-4,
6.248e-4, 6.192e-4, 6.098e-4, 5.818e-4, 5.709e-4, 5.465e-4,
5.043e-4, 4.699e-4, 4.294e-4, 3.984e-4, 3.672e-4, 3.152e-4,
2.883e-4, 2.503e-4, 2.211e-4, 1.92e-4, 1.714e-4, 1.485e-4,
1.358e-4, 1.156e-4, 1.021e-4, 8.887e-5, 7.842e-5, 7.12e-5,
6.186e-5, 5.73e-5, 4.792e-5, 4.364e-5, 3.72e-5, 3.28e-5,
2.946e-5, 2.591e-5, 2.261e-5, 2.048e-5, 1.813e-5, 1.63e-5,
1.447e-5, 1.282e-5, 1.167e-5, 1.041e-5, 9.449e-6, 8.51e-6,
7.596e-6, 6.961e-6, 6.272e-6, 5.728e-6, 5.198e-6, 4.667e-6,
4.288e-6, 3.897e-6, 3.551e-6, 3.235e-6, 2.952e-6, 2.688e-6,
2.449e-6, 2.241e-6, 2.05e-6, 1.879e-6, 1.722e-6, 1.582e-6,
1.456e-6, 1.339e-6, 1.236e-6, 1.144e-6, 1.06e-6, 9.83e-7,
9.149e-7, 8.535e-7, 7.973e-7, 7.466e-7, 6.999e-7, 6.574e-7,
6.18e-7, 5.821e-7, 5.487e-7, 5.18e-7, 4.896e-7, 4.631e-7,
4.386e-7, 4.16e-7, 3.945e-7, 3.748e-7, 3.562e-7, 3.385e-7,
3.222e-7, 3.068e-7, 2.922e-7, 2.788e-7, 2.659e-7, 2.539e-7,
2.425e-7, 2.318e-7, 2.219e-7, 2.127e-7, 2.039e-7, 1.958e-7,
1.885e-7, 1.818e-7, 1.758e-7, 1.711e-7, 1.662e-7, 1.63e-7,
1.605e-7, 1.58e-7, 1.559e-7, 1.545e-7, 1.532e-7, 1.522e-7,
1.51e-7, 1.495e-7, 1.465e-7, 1.483e-7, 1.469e-7, 1.448e-7,
1.444e-7, 1.436e-7, 1.426e-7, 1.431e-7, 1.425e-7, 1.445e-7,
1.477e-7, 1.515e-7, 1.567e-7, 1.634e-7, 1.712e-7, 1.802e-7,
1.914e-7, 2.024e-7, 2.159e-7, 2.295e-7, 2.461e-7, 2.621e-7,
2.868e-7, 3.102e-7, 3.394e-7, 3.784e-7, 4.223e-7, 4.864e-7,
5.501e-7, 6.039e-7, 7.193e-7, 7.728e-7, 9.514e-7, 1.073e-6,
1.18e-6, 1.333e-6, 1.472e-6, 1.566e-6, 1.677e-6, 1.784e-6,
1.904e-6, 1.953e-6, 2.02e-6, 2.074e-6, 2.128e-6, 2.162e-6,
2.219e-6, 2.221e-6, 2.249e-6, 2.239e-6, 2.235e-6, 2.185e-6,
2.141e-6, 2.124e-6, 2.09e-6, 2.068e-6, 2.1e-6, 2.104e-6,
2.142e-6, 2.181e-6, 2.257e-6, 2.362e-6, 2.5e-6, 2.664e-6,
2.884e-6, 3.189e-6, 3.48e-6, 3.847e-6, 4.313e-6, 4.79e-6,
5.25e-6, 5.989e-6, 6.692e-6, 7.668e-6, 8.52e-6, 9.606e-6,
1.073e-5, 1.225e-5, 1.377e-5, 1.582e-5, 1.761e-5, 2.029e-5,
2.284e-5, 2.602e-5, 2.94e-5, 3.483e-5, 3.928e-5, 4.618e-5,
5.24e-5, 6.132e-5, 7.183e-5, 8.521e-5, 9.111e-5, 1.07e-4,
1.184e-4, 1.264e-4, 1.475e-4, 1.612e-4, 1.704e-4, 1.818e-4,
1.924e-4, 1.994e-4, 2.061e-4, 2.18e-4, 2.187e-4, 2.2e-4,
2.196e-4, 2.131e-4, 2.015e-4, 1.988e-4, 1.847e-4, 1.729e-4,
1.597e-4, 1.373e-4, 1.262e-4, 1.087e-4, 9.439e-5, 8.061e-5,
7.093e-5, 6.049e-5, 5.12e-5, 4.435e-5, 3.817e-5, 3.34e-5,
2.927e-5, 2.573e-5, 2.291e-5, 2.04e-5, 1.827e-5, 1.636e-5,
1.463e-5, 1.309e-5, 1.17e-5, 1.047e-5, 9.315e-6, 8.328e-6,
7.458e-6, 6.665e-6, 5.94e-6, 5.316e-6, 4.752e-6, 4.252e-6,
3.825e-6, 3.421e-6, 3.064e-6, 2.746e-6, 2.465e-6, 2.216e-6,
1.99e-6, 1.79e-6, 1.609e-6, 1.449e-6, 1.306e-6, 1.177e-6,
1.063e-6, 9.607e-7, 8.672e-7, 7.855e-7, 7.118e-7, 6.46e-7,
5.871e-7, 5.34e-7, 4.868e-7, 4.447e-7, 4.068e-7, 3.729e-7,
3.423e-7, 3.151e-7, 2.905e-7, 2.686e-7, 2.484e-7, 2.306e-7,
2.142e-7, 1.995e-7, 1.86e-7, 1.738e-7, 1.626e-7, 1.522e-7,
1.427e-7, 1.338e-7, 1.258e-7, 1.183e-7, 1.116e-7, 1.056e-7,
9.972e-8, 9.46e-8, 9.007e-8, 8.592e-8, 8.195e-8, 7.816e-8,
7.483e-8, 7.193e-8, 6.892e-8, 6.642e-8, 6.386e-8, 6.154e-8,
5.949e-8, 5.764e-8, 5.622e-8, 5.479e-8, 5.364e-8, 5.301e-8,
5.267e-8, 5.263e-8, 5.313e-8, 5.41e-8, 5.55e-8, 5.745e-8,
6.003e-8, 6.311e-8, 6.713e-8, 7.173e-8, 7.724e-8, 8.368e-8,
9.121e-8, 9.986e-8, 1.097e-7, 1.209e-7, 1.338e-7, 1.486e-7,
1.651e-7, 1.837e-7, 2.048e-7, 2.289e-7, 2.557e-7, 2.857e-7,
3.195e-7, 3.587e-7, 4.015e-7, 4.497e-7, 5.049e-7, 5.665e-7,
6.366e-7, 7.121e-7, 7.996e-7, 8.946e-7, 1.002e-6, 1.117e-6,
1.262e-6, 1.416e-6, 1.611e-6, 1.807e-6, 2.056e-6, 2.351e-6,
2.769e-6, 3.138e-6, 3.699e-6, 4.386e-6, 5.041e-6, 6.074e-6,
6.812e-6, 7.79e-6, 8.855e-6, 1.014e-5, 1.095e-5, 1.245e-5,
1.316e-5, 1.39e-5, 1.504e-5, 1.583e-5, 1.617e-5, 1.652e-5,
1.713e-5, 1.724e-5, 1.715e-5, 1.668e-5, 1.629e-5, 1.552e-5,
1.478e-5, 1.34e-5, 1.245e-5, 1.121e-5, 9.575e-6, 8.956e-6,
7.345e-6, 6.597e-6, 5.612e-6, 4.818e-6, 4.165e-6, 3.579e-6,
3.041e-6, 2.623e-6, 2.29e-6, 1.984e-6, 1.748e-6, 1.534e-6,
1.369e-6, 1.219e-6, 1.092e-6, 9.8e-7, 8.762e-7, 7.896e-7,
7.104e-7, 6.364e-7, 5.691e-7, 5.107e-7, 4.575e-7, 4.09e-7,
3.667e-7, 3.287e-7, 2.931e-7, 2.633e-7, 2.356e-7, 2.111e-7,
1.895e-7, 1.697e-7, 1.525e-7, 1.369e-7, 1.233e-7, 1.114e-7,
9.988e-8, 9.004e-8, 8.149e-8, 7.352e-8, 6.662e-8, 6.03e-8,
5.479e-8, 4.974e-8, 4.532e-8, 4.129e-8, 3.781e-8, 3.462e-8,
3.176e-8, 2.919e-8, 2.687e-8, 2.481e-8, 2.292e-8, 2.119e-8,
1.967e-8, 1.828e-8, 1.706e-8, 1.589e-8, 1.487e-8, 1.393e-8,
1.307e-8, 1.228e-8, 1.156e-8, 1.089e-8, 1.028e-8, 9.696e-9,
9.159e-9, 8.658e-9, 8.187e-9, 7.746e-9, 7.34e-9, 6.953e-9,
6.594e-9, 6.259e-9, 5.948e-9, 5.66e-9, 5.386e-9, 5.135e-9,
4.903e-9, 4.703e-9, 4.515e-9, 4.362e-9, 4.233e-9, 4.117e-9,
```

```
4.017e-9, 3.962e-9, 3.924e-9, 3.905e-9, 3.922e-9, 3.967e-9,
4.046e-9, 4.165e-9, 4.32e-9, 4.522e-9, 4.769e-9, 5.083e-9,
5.443e-9, 5.872e-9, 6.366e-9, 6.949e-9, 7.601e-9, 8.371e-9,
9.22e-9, 1.02e-8, 1.129e-8, 1.251e-8, 1.393e-8, 1.542e-8,
1.72e-8, 1.926e-8, 2.152e-8, 2.392e-8, 2.678e-8, 3.028e-8,
3.39e-8, 3.836e-8, 4.309e-8, 4.9e-8, 5.481e-8, 6.252e-8,
7.039e-8, 7.883e-8, 8.849e-8, 1.012e-7, 1.142e-7, 1.3e-7,
1.475e-7, 1.732e-7, 1.978e-7, 2.304e-7, 2.631e-7, 2.988e-7,
3.392e-7, 3.69e-7, 4.355e-7, 4.672e-7, 5.11e-7, 5.461e-7,
5.828e-7, 6.233e-7, 6.509e-7, 6.672e-7, 6.969e-7, 7.104e-7,
7.439e-7, 7.463e-7, 7.708e-7, 7.466e-7, 7.668e-7, 7.549e-7,
7.586e-7, 7.384e-7, 7.439e-7, 7.785e-7, 7.915e-7, 8.31e-7,
8.745e-7, 9.558e-7, 1.038e-6, 1.173e-6, 1.304e-6, 1.452e-6,
1.671e-6, 1.931e-6, 2.239e-6, 2.578e-6, 3.032e-6, 3.334e-6,
3.98e-6, 4.3e-6, 4.518e-6, 5.321e-6, 5.508e-6, 6.211e-6, 6.59e-6,
7.046e-6, 7.555e-6, 7.558e-6, 7.875e-6, 8.319e-6, 8.433e-6,
8.59e-6, 8.503e-6, 8.304e-6, 8.336e-6, 7.739e-6, 7.301e-6,
6.827e-6, 6.078e-6, 5.551e-6, 4.762e-6, 4.224e-6, 3.538e-6,
2.984e-6, 2.619e-6, 2.227e-6, 1.923e-6, 1.669e-6, 1.462e-6,
1.294e-6, 1.155e-6, 1.033e-6, 9.231e-7, 8.238e-7, 7.36e-7,
6.564e-7, 5.869e-7, 5.236e-7, 4.673e-7, 4.174e-7, 3.736e-7,
3.33e-7, 2.976e-7, 2.657e-7, 2.367e-7, 2.106e-7, 1.877e-7,
1.671e-7, 1.494e-7, 1.332e-7, 1.192e-7, 1.065e-7, 9.558e-8,
8.586e-8, 7.717e-8, 6.958e-8, 6.278e-8, 5.666e-8, 5.121e-8,
4.647e-8, 4.213e-8, 3.815e-8, 3.459e-8, 3.146e-8, 2.862e-8,
2.604e-8, 2.375e-8, 2.162e-8, 1.981e-8, 1.817e-8, 1.67e-8,
1.537e-8, 1.417e-8, 1.31e-8, 1.215e-8, 1.128e-8, 1.05e-8,
9.793e-9, 9.158e-9, 8.586e-9, 8.068e-9, 7.595e-9, 7.166e-9,
6.778e-9, 6.427e-9, 6.108e-9, 5.826e-9, 5.571e-9, 5.347e-9,
5.144e-9, 4.968e-9, 4.822e-9, 4.692e-9, 4.589e-9, 4.506e-9,
4.467e-9, 4.44e-9, 4.466e-9, 4.515e-9, 4.718e-9, 4.729e-9,
4.937e-9, 5.249e-9, 5.466e-9, 5.713e-9, 6.03e-9, 6.436e-9,
6.741e-9, 7.33e-9, 7.787e-9, 8.414e-9, 8.908e-9, 9.868e-9,
1.069e-8, 1.158e-8, 1.253e-8, 1.3e-8, 1.409e-8, 1.47e-8,
1.548e-8, 1.612e-8, 1.666e-8, 1.736e-8, 1.763e-8, 1.812e-8,
1.852e-8, 1.923e-8, 1.897e-8, 1.893e-8, 1.888e-8, 1.868e-8,
1.895e-8, 1.899e-8, 1.876e-8, 1.96e-8, 2.02e-8, 2.121e-8,
2.239e-8, 2.379e-8, 2.526e-8, 2.766e-8, 2.994e-8, 3.332e-8,
3.703e-8, 4.158e-8, 4.774e-8, 5.499e-8, 6.355e-8, 7.349e-8,
8.414e-8, 9.846e-8, 1.143e-7, 1.307e-7, 1.562e-7, 1.817e-7,
2.011e-7, 2.192e-7, 2.485e-7, 2.867e-7, 3.035e-7, 3.223e-7,
3.443e-7, 3.617e-7, 3.793e-7, 3.793e-7, 3.839e-7, 4.081e-7,
4.117e-7, 4.085e-7, 3.92e-7, 3.851e-7, 3.754e-7, 3.49e-7,
3.229e-7, 2.978e-7, 2.691e-7, 2.312e-7, 2.029e-7, 1.721e-7,
1.472e-7, 1.308e-7, 1.132e-7, 9.736e-8, 8.458e-8, 7.402e-8,
6.534e-8, 5.811e-8, 5.235e-8, 4.762e-8, 4.293e-8, 3.896e-8,
3.526e-8, 3.165e-8, 2.833e-8, 2.551e-8, 2.288e-8, 2.036e-8,
1.82e-8, 1.626e-8, 1.438e-8, 1.299e-8, 1.149e-8, 1.03e-8,
9.148e-9, 8.122e-9, 7.264e-9, 6.425e-9, 5.777e-9, 5.06e-9,
4.502e-9, 4.013e-9, 3.567e-9, 3.145e-9, 2.864e-9, 2.553e-9,
2.311e-9, 2.087e-9, 1.886e-9, 1.716e-9, 1.556e-9, 1.432e-9,
1.311e-9, 1.202e-9, 1.104e-9, 1.013e-9, 9.293e-10, 8.493e-10,
7.79e-10, 7.185e-10, 6.642e-10, 6.141e-10, 5.684e-10, 5.346e-10,
5.032e-10, 4.725e-10, 4.439e-10, 4.176e-10, 3.93e-10, 3.714e-10,
3.515e-10, 3.332e-10, 3.167e-10, 3.02e-10, 2.887e-10, 2.769e-10,
2.665e-10, 2.578e-10, 2.503e-10, 2.436e-10, 2.377e-10, 2.342e-10,
2.305e-10, 2.296e-10, 2.278e-10, 2.321e-10, 2.355e-10, 2.402e-10,
2.478e-10, 2.67e-10, 2.848e-10, 2.982e-10, 3.263e-10, 3.438e-10,
3.649e-10, 3.829e-10, 4.115e-10, 4.264e-10, 4.473e-10, 4.63e-10,
4.808e-10, 4.995e-10, 5.142e-10, 5.313e-10, 5.318e-10, 5.358e-10,
5.452e-10, 5.507e-10, 5.698e-10, 5.782e-10, 5.983e-10, 6.164e-10,
6.532e-10, 6.811e-10, 7.624e-10, 8.302e-10, 9.067e-10, 9.937e-10,
1.104e-9, 1.221e-9, 1.361e-9, 1.516e-9, 1.675e-9, 1.883e-9,
2.101e-9, 2.349e-9, 2.614e-9, 2.92e-9, 3.305e-9, 3.724e-9,
4.142e-9, 4.887e-9, 5.614e-9, 6.506e-9, 7.463e-9, 8.817e-9,
9.849e-9, 1.187e-8, 1.321e-8, 1.474e-8, 1.698e-8, 1.794e-8,
2.09e-8, 2.211e-8, 2.362e-8, 2.556e-8, 2.729e-8, 2.88e-8,
3.046e-8, 3.167e-8, 3.367e-8, 3.457e-8, 3.59e-8, 3.711e-8,
3.826e-8, 4.001e-8, 4.211e-8, 4.315e-8, 4.661e-8, 5.01e-8,
5.249e-8, 5.84e-8, 6.628e-8, 7.512e-8, 8.253e-8, 9.722e-8,
1.067e-7, 1.153e-7, 1.347e-7, 1.428e-7, 1.577e-7, 1.694e-7,
1.833e-7, 1.938e-7, 2.108e-7, 2.059e-7, 2.157e-7, 2.185e-7,
2.208e-7, 2.182e-7, 2.093e-7, 2.014e-7, 1.962e-7, 1.819e-7,
1.713e-7, 1.51e-7, 1.34e-7, 1.154e-7, 9.89e-8, 8.88e-8, 7.673e-8,
6.599e-8, 5.73e-8, 5.081e-8, 4.567e-8, 4.147e-8, 3.773e-8,
3.46e-8, 3.194e-8, 2.953e-8, 2.759e-8, 2.594e-8, 2.442e-8,
2.355e-8, 2.283e-8, 2.279e-8, 2.231e-8, 2.279e-8, 2.239e-8,
2.21e-8, 2.309e-8, 2.293e-8, 2.352e-8, 2.415e-8, 2.43e-8,
2.426e-8, 2.465e-8, 2.5e-8, 2.496e-8, 2.465e-8, 2.445e-8,
2.383e-8, 2.299e-8, 2.165e-8, 2.113e-8, 1.968e-8, 1.819e-8,
1.644e-8, 1.427e-8, 1.27e-8, 1.082e-8, 9.428e-9, 8.091e-9,
6.958e-9, 5.988e-9, 5.246e-9, 4.601e-9, 4.098e-9, 3.664e-9,
3.287e-9, 2.942e-9, 2.656e-9, 2.364e-9, 2.118e-9, 1.903e-9,
1.703e-9, 1.525e-9, 1.365e-9, 1.229e-9, 1.107e-9, 9.96e-10,
8.945e-10, 8.08e-10, 7.308e-10, 6.616e-10, 5.994e-10, 5.422e-10,
4.929e-10, 4.478e-10, 4.07e-10, 3.707e-10, 3.379e-10, 3.087e-10,
2.823e-10, 2.592e-10, 2.385e-10, 2.201e-10, 2.038e-10, 1.897e-10,
```

```
1.774e-10, 1.667e-10, 1.577e-10, 1.502e-10, 1.437e-10, 1.394e-10,
1.358e-10, 1.324e-10, 1.329e-10, 1.324e-10, 1.36e-10, 1.39e-10,
1.424e-10, 1.544e-10, 1.651e-10, 1.817e-10, 1.984e-10, 2.195e-10,
2.438e-10, 2.7e-10, 2.991e-10, 3.322e-10, 3.632e-10, 3.957e-10,
4.36e-10, 4.701e-10, 5.03e-10, 5.381e-10, 5.793e-10, 6.19e-10,
6.596e-10, 7.004e-10, 7.561e-10, 7.934e-10, 8.552e-10, 9.142e-10,
9.57e-10, 1.027e-9, 1.097e-9, 1.193e-9, 1.334e-9, 1.47e-9,
1.636e-9, 1.871e-9, 2.122e-9, 2.519e-9, 2.806e-9, 3.203e-9,
3.846e-9, 4.362e-9, 5.114e-9, 5.643e-9, 6.305e-9, 6.981e-9,
7.983e-9, 8.783e-9, 9.419e-9, 1.017e-8, 1.063e-8, 1.121e-8,
1.13e-8, 1.201e-8, 1.225e-8, 1.232e-8, 1.223e-8, 1.177e-8,
1.151e-8, 1.116e-8, 1.047e-8, 9.698e-9, 8.734e-9, 8.202e-9,
7.041e-9, 6.074e-9, 5.172e-9, 4.468e-9, 3.913e-9, 3.414e-9,
2.975e-9, 2.65e-9, 2.406e-9, 2.173e-9, 2.009e-9, 1.861e-9,
1.727e-9, 1.612e-9, 1.514e-9, 1.43e-9, 1.362e-9, 1.333e-9,
1.288e-9, 1.249e-9, 1.238e-9, 1.228e-9, 1.217e-9, 1.202e-9,
1.209e-9, 1.177e-9, 1.157e-9, 1.165e-9, 1.142e-9, 1.131e-9,
1.138e-9, 1.117e-9, 1.1e-9, 1.069e-9, 1.023e-9, 1.005e-9,
9.159e-10, 8.863e-10, 7.865e-10, 7.153e-10, 6.247e-10, 5.846e-10,
5.133e-10, 4.36e-10, 3.789e-10, 3.335e-10, 2.833e-10, 2.483e-10,
2.155e-10, 1.918e-10, 1.709e-10, 1.529e-10, 1.374e-10, 1.235e-10,
1.108e-10, 9.933e-11, 8.932e-11, 8.022e-11, 7.224e-11, 6.52e-11,
5.896e-11, 5.328e-11, 4.813e-11, 4.365e-11, 3.961e-11, 3.594e-11,
3.266e-11, 2.967e-11, 2.701e-11, 2.464e-11, 2.248e-11, 2.054e-11,
1.878e-11, 1.721e-11, 1.579e-11, 1.453e-11, 1.341e-11, 1.241e-11,
1.154e-11, 1.078e-11, 1.014e-11, 9.601e-12, 9.167e-12, 8.838e-12,
8.614e-12, 8.493e-12, 8.481e-12, 8.581e-12, 8.795e-12, 9.131e-12,
9.601e-12, 1.021e-11, 1.097e-11, 1.191e-11, 1.303e-11, 1.439e-11,
1.601e-11, 1.778e-11, 1.984e-11, 2.234e-11, 2.474e-11, 2.766e-11,
3.085e-11, 3.415e-11, 3.821e-11, 4.261e-11, 4.748e-11, 5.323e-11,
5.935e-11, 6.619e-11, 7.418e-11, 8.294e-11, 9.26e-11, 1.039e-10,
1.156e-10, 1.297e-10, 1.46e-10, 1.641e-10, 1.858e-10, 2.1e-10,
2.383e-10, 2.724e-10, 3.116e-10, 3.538e-10, 4.173e-10, 4.727e-10,
5.503e-10, 6.337e-10, 7.32e-10, 8.298e-10, 9.328e-10, 1.059e-9,
1.176e-9, 1.328e-9, 1.445e-9, 1.593e-9, 1.77e-9, 1.954e-9,
2.175e-9, 2.405e-9, 2.622e-9, 2.906e-9, 3.294e-9, 3.713e-9,
3.98e-9, 4.384e-9, 4.987e-9, 5.311e-9, 5.874e-9, 6.337e-9,
7.027e-9, 7.39e-9, 7.769e-9, 8.374e-9, 8.605e-9, 9.165e-9,
9.415e-9, 9.511e-9, 9.704e-9, 9.588e-9, 9.45e-9, 9.086e-9,
8.798e-9, 8.469e-9, 7.697e-9, 7.168e-9, 6.255e-9, 5.772e-9,
4.97e-9, 4.271e-9, 3.653e-9, 3.154e-9, 2.742e-9, 2.435e-9,
2.166e-9, 1.936e-9, 1.731e-9, 1.556e-9, 1.399e-9, 1.272e-9,
1.157e-9, 1.066e-9, 9.844e-10, 9.258e-10, 8.787e-10, 8.421e-10,
8.083e-10, 8.046e-10, 8.067e-10, 8.181e-10, 8.325e-10, 8.517e-10,
9.151e-10, 9.351e-10, 9.677e-10, 1.071e-9, 1.126e-9, 1.219e-9,
1.297e-9, 1.408e-9, 1.476e-9, 1.517e-9, 1.6e-9, 1.649e-9,
1.678e-9, 1.746e-9, 1.742e-9, 1.728e-9, 1.699e-9, 1.655e-9,
1.561e-9, 1.48e-9, 1.451e-9, 1.411e-9, 1.171e-9, 1.106e-9,
9.714e-10, 8.523e-10, 7.346e-10, 6.241e-10, 5.371e-10, 4.704e-10,
4.144e-10, 3.683e-10, 3.292e-10, 2.942e-10, 2.62e-10, 2.341e-10,
2.104e-10, 1.884e-10, 1.7e-10, 1.546e-10, 1.394e-10, 1.265e-10,
1.14e-10, 1.019e-10, 9.279e-11, 8.283e-11, 7.458e-11, 6.668e-11,
5.976e-11, 5.33e-11, 4.794e-11, 4.289e-11, 3.841e-11, 3.467e-11,
3.13e-11, 2.832e-11, 2.582e-11, 2.356e-11, 2.152e-11, 1.97e-11,
1.808e-11, 1.664e-11, 1.539e-11, 1.434e-11, 1.344e-11, 1.269e-11,
1.209e-11, 1.162e-11, 1.129e-11, 1.108e-11, 1.099e-11, 1.103e-11,
1.119e-11, 1.148e-11, 1.193e-11, 1.252e-11, 1.329e-11, 1.421e-11,
1.555e-11, 1.685e-11, 1.839e-11, 2.054e-11, 2.317e-11, 2.571e-11,
2.839e-11, 3.171e-11, 3.49e-11, 3.886e-11, 4.287e-11, 4.645e-11,
5.047e-11, 5.592e-11, 6.109e-11, 6.628e-11, 7.381e-11, 8.088e-11,
8.966e-11, 1.045e-10, 1.12e-10, 1.287e-10, 1.486e-10, 1.662e-10,
1.866e-10, 2.133e-10, 2.524e-10, 2.776e-10, 3.204e-10, 3.559e-10,
4.028e-10, 4.448e-10, 4.882e-10, 5.244e-10, 5.605e-10, 6.018e-10,
6.328e-10, 6.579e-10, 6.541e-10, 7.024e-10, 7.074e-10, 7.068e-10,
7.009e-10, 6.698e-10, 6.545e-10, 6.209e-10, 5.834e-10, 5.412e-10,
5.001e-10, 4.231e-10, 3.727e-10, 3.211e-10, 2.833e-10, 2.447e-10,
2.097e-10, 1.843e-10, 1.639e-10, 1.449e-10, 1.27e-10, 1.161e-10,
1.033e-10, 9.282e-11, 8.407e-11, 7.639e-11, 7.023e-11, 6.474e-11,
6.142e-11, 5.76e-11, 5.568e-11, 5.472e-11, 5.39e-11, 5.455e-11,
5.54e-11, 5.587e-11, 6.23e-11, 6.49e-11, 6.868e-11, 7.382e-11,
8.022e-11, 8.372e-11, 9.243e-11, 1.004e-10, 1.062e-10, 1.13e-10,
1.176e-10, 1.244e-10, 1.279e-10, 1.298e-10, 1.302e-10, 1.312e-10,
1.295e-10, 1.244e-10, 1.211e-10, 1.167e-10, 1.098e-10, 9.927e-11,
8.854e-11, 8.011e-11, 7.182e-11, 5.923e-11, 5.212e-11, 4.453e-11,
3.832e-11, 3.371e-11, 2.987e-11, 2.651e-11, 2.354e-11, 2.093e-11,
1.863e-11, 1.662e-11, 1.486e-11, 1.331e-11, 1.193e-11, 1.071e-11,
9.628e-12, 8.66e-12, 7.801e-12, 7.031e-12, 6.347e-12, 5.733e-12,
5.182e-12, 4.695e-12, 4.26e-12, 3.874e-12, 3.533e-12, 3.235e-12,
2.979e-12, 2.76e-12, 2.579e-12, 2.432e-12, 2.321e-12, 2.246e-12,
2.205e-12, 2.196e-12, 2.223e-12, 2.288e-12, 2.387e-12, 2.525e-12,
2.704e-12, 2.925e-12, 3.191e-12, 3.508e-12, 3.876e-12, 4.303e-12,
4.793e-12, 5.347e-12, 5.978e-12, 6.682e-12, 7.467e-12, 8.34e-12,
9.293e-12, 1.035e-11, 1.152e-11, 1.285e-11, 1.428e-11, 1.586e-11,
1.764e-11, 1.972e-11, 2.214e-11, 2.478e-11, 2.776e-11, 3.151e-11,
3.591e-11, 4.103e-11, 4.66e-11, 5.395e-11, 6.306e-11, 7.172e-11,
8.358e-11, 9.67e-11, 1.11e-10, 1.325e-10, 1.494e-10, 1.736e-10,
2.007e-10, 2.296e-10, 2.608e-10, 3.004e-10, 3.361e-10, 3.727e-10,
```

```
    4.373e-10, 4.838e-10, 5.483e-10, 6.006e-10, 6.535e-10, 6.899e-10,
    7.687e-10, 8.444e-10, 8.798e-10, 9.135e-10, 9.532e-10, 9.757e-10,
    9.968e-10, 1.006e-9, 9.949e-10, 9.789e-10, 9.564e-10, 9.215e-10,
    8.51e-10, 8.394e-10, 7.707e-10, 7.152e-10, 6.274e-10, 5.598e-10,
    5.028e-10, 4.3e-10, 3.71e-10, 3.245e-10, 2.809e-10, 2.461e-10,
    2.154e-10, 1.91e-10, 1.685e-10, 1.487e-10, 1.313e-10, 1.163e-10,
    1.031e-10, 9.172e-11, 8.221e-11, 7.382e-11, 6.693e-11, 6.079e-11,
    5.581e-11, 5.167e-11, 4.811e-11, 4.506e-11, 4.255e-11, 4.083e-11,
    3.949e-11, 3.881e-11, 3.861e-11, 3.858e-11, 3.951e-11, 4.045e-11,
    4.24e-11, 4.487e-11, 4.806e-11, 5.133e-11, 5.518e-11, 5.919e-11,
    6.533e-11, 7.031e-11, 7.762e-11, 8.305e-11, 9.252e-11, 9.727e-11,
    1.045e-10, 1.117e-10, 1.2e-10, 1.275e-10, 1.341e-10, 1.362e-10,
    1.438e-10, 1.45e-10, 1.455e-10, 1.455e-10, 1.434e-10, 1.381e-10,
    1.301e-10, 1.276e-10, 1.163e-10, 1.089e-10, 9.911e-11, 8.943e-11,
    7.618e-11, 6.424e-11, 5.717e-11, 4.866e-11, 4.257e-11, 3.773e-11,
    3.331e-11, 2.958e-11, 2.629e-11, 2.316e-11, 2.073e-11, 1.841e-11,
    1.635e-11, 1.464e-11, 1.31e-11, 1.16e-11, 1.047e-11, 9.408e-12,
    8.414e-12, 7.521e-12, 6.705e-12, 5.993e-12, 5.371e-12, 4.815e-12,
    4.338e-12, 3.921e-12, 3.567e-12, 3.265e-12, 3.01e-12, 2.795e-12,
    2.613e-12, 2.464e-12, 2.346e-12, 2.256e-12, 2.195e-12, 2.165e-12,
    2.166e-12, 2.198e-12, 2.262e-12, 2.364e-12, 2.502e-12, 2.682e-12,
    2.908e-12, 3.187e-12, 3.533e-12, 3.946e-12, 4.418e-12, 5.013e-12,
    5.708e-12, 6.379e-12, 7.43e-12, 8.39e-12, 9.51e-12, 1.078e-11,
    1.259e-11, 1.438e-11, 1.63e-11, 1.814e-11, 2.055e-11, 2.348e-11,
    2.664e-11, 2.956e-11, 3.3e-11, 3.677e-11, 4.032e-11, 4.494e-11,
    4.951e-11, 5.452e-11, 6.014e-11, 6.5e-11, 6.915e-11, 7.45e-11,
    7.971e-11, 8.468e-11, 8.726e-11, 8.995e-11, 9.182e-11, 9.509e-11,
    9.333e-11, 9.386e-11, 9.457e-11, 9.21e-11, 9.019e-11, 8.68e-11,
    8.298e-11, 7.947e-11, 7.46e-11, 7.082e-11, 6.132e-11, 5.855e-11,
    5.073e-11, 4.464e-11, 3.825e-11, 3.375e-11, 2.911e-11, 2.535e-11,
    2.16e-11, 1.907e-11, 1.665e-11, 1.463e-11, 1.291e-11, 1.133e-11,
    9.997e-12, 8.836e-12, 7.839e-12, 6.943e-12, 6.254e-12, 5.6e-12,
    5.029e-12, 4.529e-12, 4.102e-12, 3.737e-12, 3.428e-12, 3.169e-12,
    2.959e-12, 2.798e-12, 2.675e-12, 2.582e-12, 2.644e-12, 2.557e-12,
    2.614e-12, 2.717e-12, 2.874e-12, 3.056e-12, 3.187e-12, 3.631e-12,
    3.979e-12, 4.248e-12, 4.817e-12, 5.266e-12, 5.836e-12, 6.365e-12,
    6.807e-12, 7.47e-12, 7.951e-12, 8.636e-12, 8.972e-12, 9.314e-12,
    9.445e-12, 1.003e-11, 1.013e-11, 9.937e-12, 9.729e-12, 9.064e-12,
    9.119e-12, 9.124e-12, 8.704e-12, 8.078e-12, 7.47e-12, 6.329e-12,
    5.674e-12, 4.808e-12, 4.119e-12, 3.554e-12, 3.103e-12, 2.731e-12,
    2.415e-12, 2.15e-12, 1.926e-12, 1.737e-12, 1.578e-12, 1.447e-12,
    1.34e-12, 1.255e-12, 1.191e-12, 1.146e-12, 1.121e-12, 1.114e-12,
    1.126e-12, 1.156e-12, 1.207e-12, 1.278e-12, 1.372e-12, 1.49e-12,
    1.633e-12, 1.805e-12, 2.01e-12, 2.249e-12, 2.528e-12, 2.852e-12,
    3.228e-12, 3.658e-12, 4.153e-12, 4.728e-12, 5.394e-12, 6.176e-12,
    7.126e-12, 8.188e-12, 9.328e-12, 1.103e-11, 1.276e-11, 1.417e-11,
    1.615e-11, 1.84e-11, 2.155e-11, 2.429e-11, 2.826e-11, 3.222e-11,
    3.664e-11, 4.14e-11, 4.906e-11, 5.536e-11, 6.327e-11, 7.088e-11,
    8.316e-11, 9.242e-11, 1.07e-10, 1.223e-10, 1.341e-10, 1.553e-10,
    1.703e-10, 1.9e-10, 2.022e-10, 2.233e-10, 2.345e-10, 2.438e-10,
    2.546e-10, 2.599e-10, 2.661e-10, 2.703e-10, 2.686e-10, 2.662e-10,
    2.56e-10, 2.552e-10, 2.378e-10, 2.252e-10, 2.146e-10, 1.885e-10,
    1.668e-10, 1.441e-10, 1.295e-10, 1.119e-10, 9.893e-11, 8.687e-11,
    7.678e-11, 6.685e-11, 5.879e-11, 5.127e-11, 4.505e-11, 3.997e-11,
    3.511e-11
  };

  static double h2ofrn[2001] = { .01095, .01126, .01205, .01322, .0143,
    .01506, .01548, .01534, .01486, .01373, .01262, .01134, .01001,
    .008702, .007475, .006481, .00548, .0046, .003833, .00311,
    .002543, .002049, .00168, .001374, .001046, 8.193e-4, 6.267e-4,
    4.968e-4, 3.924e-4, 2.983e-4, 2.477e-4, 1.997e-4, 1.596e-4,
    1.331e-4, 1.061e-4, 8.942e-5, 7.168e-5, 5.887e-5, 4.848e-5,
    3.817e-5, 3.17e-5, 2.579e-5, 2.162e-5, 1.768e-5, 1.49e-5,
    1.231e-5, 1.013e-5, 8.555e-6, 7.328e-6, 6.148e-6, 5.207e-6,
    4.387e-6, 3.741e-6, 3.22e-6, 2.753e-6, 2.346e-6, 1.985e-6,
    1.716e-6, 1.475e-6, 1.286e-6, 1.122e-6, 9.661e-7, 8.284e-7,
    7.057e-7, 6.119e-7, 5.29e-7, 4.571e-7, 3.948e-7, 3.432e-7,
    2.983e-7, 2.589e-7, 2.265e-7, 1.976e-7, 1.704e-7, 1.456e-7,
    1.26e-7, 1.101e-7, 9.648e-8, 8.415e-8, 7.34e-8, 6.441e-8,
    5.643e-8, 4.94e-8, 4.276e-8, 3.703e-8, 3.227e-8, 2.825e-8,
    2.478e-8, 2.174e-8, 1.898e-8, 1.664e-8, 1.458e-8, 1.278e-8,
    1.126e-8, 9.891e-9, 8.709e-9, 7.652e-9, 6.759e-9, 5.975e-9,
    5.31e-9, 4.728e-9, 4.214e-9, 3.792e-9, 3.463e-9, 3.226e-9,
    2.992e-9, 2.813e-9, 2.749e-9, 2.809e-9, 2.913e-9, 3.037e-9,
    3.413e-9, 3.738e-9, 4.189e-9, 4.808e-9, 5.978e-9, 7.088e-9,
    8.071e-9, 9.61e-9, 1.21e-8, 1.5e-8, 1.764e-8, 2.221e-8, 2.898e-8,
    3.948e-8, 5.068e-8, 6.227e-8, 7.898e-8, 1.033e-7, 1.437e-7,
    1.889e-7, 2.589e-7, 3.59e-7, 4.971e-7, 7.156e-7, 9.983e-7,
    1.381e-6, 1.929e-6, 2.591e-6, 3.453e-6, 4.57e-6, 5.93e-6,
    7.552e-6, 9.556e-6, 1.183e-5, 1.425e-5, 1.681e-5, 1.978e-5,
    2.335e-5, 2.668e-5, 3.022e-5, 3.371e-5, 3.715e-5, 3.967e-5,
    4.06e-5, 4.01e-5, 3.809e-5, 3.491e-5, 3.155e-5, 2.848e-5,
    2.678e-5, 2.66e-5, 2.811e-5, 3.071e-5, 3.294e-5, 3.459e-5,
    3.569e-5, 3.56e-5, 3.434e-5, 3.186e-5, 2.916e-5, 2.622e-5,
    2.275e-5, 1.918e-5, 1.62e-5, 1.373e-5, 1.182e-5, 1.006e-5,
    8.556e-6, 7.26e-6, 6.107e-6, 5.034e-6, 4.211e-6, 3.426e-6,
```

```
2.865e-6, 2.446e-6, 1.998e-6, 1.628e-6, 1.242e-6, 1.005e-6,
7.853e-7, 6.21e-7, 5.071e-7, 4.156e-7, 3.548e-7, 2.825e-7,
2.261e-7, 1.916e-7, 1.51e-7, 1.279e-7, 1.059e-7, 9.14e-8,
7.707e-8, 6.17e-8, 5.311e-8, 4.263e-8, 3.518e-8, 2.961e-8,
2.457e-8, 2.119e-8, 1.712e-8, 1.439e-8, 1.201e-8, 1.003e-8,
8.564e-9, 7.199e-9, 6.184e-9, 5.206e-9, 4.376e-9, 3.708e-9,
3.157e-9, 2.725e-9, 2.361e-9, 2.074e-9, 1.797e-9, 1.562e-9,
1.364e-9, 1.196e-9, 1.042e-9, 8.862e-10, 7.648e-10, 6.544e-10,
5.609e-10, 4.791e-10, 4.108e-10, 3.531e-10, 3.038e-10, 2.618e-10,
2.268e-10, 1.969e-10, 1.715e-10, 1.496e-10, 1.308e-10, 1.147e-10,
1.008e-10, 8.894e-11, 7.885e-11, 7.031e-11, 6.355e-11, 5.854e-11,
5.534e-11, 5.466e-11, 5.725e-11, 6.447e-11, 7.943e-11, 1.038e-10,
1.437e-10, 2.04e-10, 2.901e-10, 4.051e-10, 5.556e-10, 7.314e-10,
9.291e-10, 1.134e-9, 1.321e-9, 1.482e-9, 1.596e-9, 1.669e-9,
1.715e-9, 1.762e-9, 1.817e-9, 1.828e-9, 1.848e-9, 1.873e-9,
1.902e-9, 1.894e-9, 1.864e-9, 1.841e-9, 1.797e-9, 1.704e-9,
1.559e-9, 1.382e-9, 1.187e-9, 1.001e-9, 8.468e-10, 7.265e-10,
6.521e-10, 6.381e-10, 6.66e-10, 7.637e-10, 9.705e-10, 1.368e-9,
1.856e-9, 2.656e-9, 3.954e-9, 5.96e-9, 8.72e-9, 1.247e-8,
1.781e-8, 2.491e-8, 3.311e-8, 4.272e-8, 5.205e-8, 6.268e-8,
7.337e-8, 8.277e-8, 9.185e-8, 1.004e-7, 1.091e-7, 1.159e-7,
1.188e-7, 1.175e-7, 1.124e-7, 1.033e-7, 9.381e-8, 8.501e-8,
7.956e-8, 7.894e-8, 8.331e-8, 9.102e-8, 9.836e-8, 1.035e-7,
1.064e-7, 1.06e-7, 1.032e-7, 9.808e-8, 9.139e-8, 8.442e-8,
7.641e-8, 6.881e-8, 6.161e-8, 5.404e-8, 4.804e-8, 4.446e-8,
4.328e-8, 4.259e-8, 4.421e-8, 4.673e-8, 4.985e-8, 5.335e-8,
5.796e-8, 6.542e-8, 7.714e-8, 8.827e-8, 1.04e-7, 1.238e-7,
1.499e-7, 1.829e-7, 2.222e-7, 2.689e-7, 3.303e-7, 3.981e-7,
4.84e-7, 5.91e-7, 7.363e-7, 9.087e-7, 1.139e-6, 1.455e-6,
1.866e-6, 2.44e-6, 3.115e-6, 3.941e-6, 4.891e-6, 5.992e-6,
7.111e-6, 8.296e-6, 9.21e-6, 9.987e-6, 1.044e-5, 1.073e-5,
1.092e-5, 1.106e-5, 1.138e-5, 1.171e-5, 1.186e-5, 1.186e-5,
1.179e-5, 1.166e-5, 1.151e-5, 1.16e-5, 1.197e-5, 1.241e-5,
1.268e-5, 1.26e-5, 1.184e-5, 1.063e-5, 9.204e-6, 7.584e-6,
6.053e-6, 4.482e-6, 3.252e-6, 2.337e-6, 1.662e-6, 1.18e-6,
8.15e-7, 5.95e-7, 4.354e-7, 3.302e-7, 2.494e-7, 1.93e-7,
1.545e-7, 1.25e-7, 1.039e-7, 8.602e-8, 7.127e-8, 5.897e-8,
4.838e-8, 4.018e-8, 3.28e-8, 2.72e-8, 2.307e-8, 1.972e-8,
1.654e-8, 1.421e-8, 1.174e-8, 1.004e-8, 8.739e-9, 7.358e-9,
6.242e-9, 5.303e-9, 4.567e-9, 3.94e-9, 3.375e-9, 2.864e-9,
2.422e-9, 2.057e-9, 1.75e-9, 1.505e-9, 1.294e-9, 1.101e-9,
9.401e-10, 8.018e-10, 6.903e-10, 5.965e-10, 5.087e-10, 4.364e-10,
3.759e-10, 3.247e-10, 2.809e-10, 2.438e-10, 2.123e-10, 1.853e-10,
1.622e-10, 1.426e-10, 1.26e-10, 1.125e-10, 1.022e-10, 9.582e-11,
9.388e-11, 9.801e-11, 1.08e-10, 1.276e-10, 1.551e-10, 1.903e-10,
2.291e-10, 2.724e-10, 3.117e-10, 3.4e-10, 3.562e-10, 3.625e-10,
3.619e-10, 3.429e-10, 3.221e-10, 2.943e-10, 2.645e-10, 2.338e-10,
2.062e-10, 1.901e-10, 1.814e-10, 1.827e-10, 1.906e-10, 1.984e-10,
2.04e-10, 2.068e-10, 2.075e-10, 2.018e-10, 1.959e-10, 1.897e-10,
1.852e-10, 1.791e-10, 1.696e-10, 1.634e-10, 1.598e-10, 1.561e-10,
1.518e-10, 1.443e-10, 1.377e-10, 1.346e-10, 1.342e-10, 1.375e-10,
1.525e-10, 1.767e-10, 2.108e-10, 2.524e-10, 2.981e-10, 3.477e-10,
4.262e-10, 5.326e-10, 6.646e-10, 8.321e-10, 1.069e-9, 1.386e-9,
1.743e-9, 2.216e-9, 2.808e-9, 3.585e-9, 4.552e-9, 5.907e-9,
7.611e-9, 9.774e-9, 1.255e-8, 1.666e-8, 2.279e-8, 3.221e-8,
4.531e-8, 6.4e-8, 9.187e-8, 1.295e-7, 1.825e-7, 2.431e-7,
3.181e-7, 4.009e-7, 4.941e-7, 5.88e-7, 6.623e-7, 7.155e-7,
7.451e-7, 7.594e-7, 7.541e-7, 7.467e-7, 7.527e-7, 7.935e-7,
8.461e-7, 8.954e-7, 9.364e-7, 9.843e-7, 1.024e-6, 1.05e-6,
1.059e-6, 1.074e-6, 1.072e-6, 1.043e-6, 9.789e-7, 8.803e-7,
7.662e-7, 6.378e-7, 5.133e-7, 3.958e-7, 2.914e-7, 2.144e-7,
1.57e-7, 1.14e-7, 8.47e-8, 6.2e-8, 4.657e-8, 3.559e-8, 2.813e-8,
2.222e-8, 1.769e-8, 1.391e-8, 1.125e-8, 9.186e-9, 7.704e-9,
6.447e-9, 5.381e-9, 4.442e-9, 3.669e-9, 3.057e-9, 2.564e-9,
2.153e-9, 1.784e-9, 1.499e-9, 1.281e-9, 1.082e-9, 9.304e-10,
8.169e-10, 6.856e-10, 5.866e-10, 5.043e-10, 4.336e-10, 3.731e-10,
3.175e-10, 2.745e-10, 2.374e-10, 2.007e-10, 1.737e-10, 1.508e-10,
1.302e-10, 1.13e-10, 9.672e-11, 8.375e-11, 7.265e-11, 6.244e-11,
5.343e-11, 4.654e-11, 3.975e-11, 3.488e-11, 3.097e-11, 2.834e-11,
2.649e-11, 2.519e-11, 2.462e-11, 2.443e-11, 2.44e-11, 2.398e-11,
2.306e-11, 2.183e-11, 2.021e-11, 1.821e-11, 1.599e-11, 1.403e-11,
1.196e-11, 1.023e-11, 8.728e-12, 7.606e-12, 6.941e-12, 6.545e-12,
6.484e-12, 6.6e-12, 6.718e-12, 6.785e-12, 6.746e-12, 6.724e-12,
6.764e-12, 6.995e-12, 7.144e-12, 7.32e-12, 7.33e-12, 7.208e-12,
6.789e-12, 6.09e-12, 5.337e-12, 4.62e-12, 4.037e-12, 3.574e-12,
3.311e-12, 3.346e-12, 3.566e-12, 3.836e-12, 4.076e-12, 4.351e-12,
4.691e-12, 5.114e-12, 5.427e-12, 6.167e-12, 7.436e-12, 8.842e-12,
1.038e-11, 1.249e-11, 1.54e-11, 1.915e-11, 2.48e-11, 3.256e-11,
4.339e-11, 5.611e-11, 7.519e-11, 1.037e-10, 1.409e-10, 1.883e-10,
2.503e-10, 3.38e-10, 4.468e-10, 5.801e-10, 7.335e-10, 8.98e-10,
1.11e-9, 1.363e-9, 1.677e-9, 2.104e-9, 2.681e-9, 3.531e-9,
4.621e-9, 6.106e-9, 8.154e-9, 1.046e-8, 1.312e-8, 1.607e-8,
1.948e-8, 2.266e-8, 2.495e-8, 2.655e-8, 2.739e-8, 2.739e-8,
2.662e-8, 2.589e-8, 2.59e-8, 2.664e-8, 2.833e-8, 3.023e-8,
3.305e-8, 3.558e-8, 3.793e-8, 3.961e-8, 4.056e-8, 4.102e-8,
4.025e-8, 3.917e-8, 3.706e-8, 3.493e-8, 3.249e-8, 3.096e-8,
3.011e-8, 3.111e-8, 3.395e-8, 3.958e-8, 4.875e-8, 6.066e-8,
```

```
7.915e-8, 1.011e-7, 1.3e-7, 1.622e-7, 2.003e-7, 2.448e-7,
2.863e-7, 3.317e-7, 3.655e-7, 3.96e-7, 4.098e-7, 4.168e-7,
4.198e-7, 4.207e-7, 4.289e-7, 4.384e-7, 4.471e-7, 4.524e-7,
4.574e-7, 4.633e-7, 4.785e-7, 5.028e-7, 5.371e-7, 5.727e-7,
5.955e-7, 5.998e-7, 5.669e-7, 5.082e-7, 4.397e-7, 3.596e-7,
2.814e-7, 2.074e-7, 1.486e-7, 1.057e-7, 7.25e-8, 4.946e-8,
3.43e-8, 2.447e-8, 1.793e-8, 1.375e-8, 1.096e-8, 9.091e-9,
7.709e-9, 6.631e-9, 5.714e-9, 4.886e-9, 4.205e-9, 3.575e-9,
3.07e-9, 2.631e-9, 2.284e-9, 2.002e-9, 1.745e-9, 1.509e-9,
1.284e-9, 1.084e-9, 9.163e-10, 7.663e-10, 6.346e-10, 5.283e-10,
4.354e-10, 3.59e-10, 2.982e-10, 2.455e-10, 2.033e-10, 1.696e-10,
1.432e-10, 1.211e-10, 1.02e-10, 8.702e-11, 7.38e-11, 6.293e-11,
5.343e-11, 4.532e-11, 3.907e-11, 3.365e-11, 2.945e-11, 2.558e-11,
2.192e-11, 1.895e-11, 1.636e-11, 1.42e-11, 1.228e-11, 1.063e-11,
9.348e-12, 8.2e-12, 7.231e-12, 6.43e-12, 5.702e-12, 5.052e-12,
4.469e-12, 4e-12, 3.679e-12, 3.387e-12, 3.197e-12, 3.158e-12,
3.327e-12, 3.675e-12, 4.292e-12, 5.437e-12, 7.197e-12, 1.008e-11,
1.437e-11, 2.035e-11, 2.905e-11, 4.062e-11, 5.528e-11, 7.177e-11,
9.064e-11, 1.109e-10, 1.297e-10, 1.473e-10, 1.652e-10, 1.851e-10,
2.079e-10, 2.313e-10, 2.619e-10, 2.958e-10, 3.352e-10, 3.796e-10,
4.295e-10, 4.923e-10, 5.49e-10, 5.998e-10, 6.388e-10, 6.645e-10,
6.712e-10, 6.549e-10, 6.38e-10, 6.255e-10, 6.253e-10, 6.459e-10,
6.977e-10, 7.59e-10, 8.242e-10, 8.92e-10, 9.403e-10, 9.701e-10,
9.483e-10, 9.135e-10, 8.617e-10, 7.921e-10, 7.168e-10, 6.382e-10,
5.677e-10, 5.045e-10, 4.572e-10, 4.312e-10, 4.145e-10, 4.192e-10,
4.541e-10, 5.368e-10, 6.771e-10, 8.962e-10, 1.21e-9, 1.659e-9,
2.33e-9, 3.249e-9, 4.495e-9, 5.923e-9, 7.642e-9, 9.607e-9,
1.178e-8, 1.399e-8, 1.584e-8, 1.73e-8, 1.816e-8, 1.87e-8,
1.868e-8, 1.87e-8, 1.884e-8, 1.99e-8, 2.15e-8, 2.258e-8,
2.364e-8, 2.473e-8, 2.602e-8, 2.689e-8, 2.731e-8, 2.816e-8,
2.859e-8, 2.839e-8, 2.703e-8, 2.451e-8, 2.149e-8, 1.787e-8,
1.449e-8, 1.111e-8, 8.282e-9, 6.121e-9, 4.494e-9, 3.367e-9,
2.487e-9, 1.885e-9, 1.503e-9, 1.249e-9, 1.074e-9, 9.427e-10,
8.439e-10, 7.563e-10, 6.772e-10, 6.002e-10, 5.254e-10, 4.588e-10,
3.977e-10, 3.449e-10, 3.003e-10, 2.624e-10, 2.335e-10, 2.04e-10,
1.771e-10, 1.534e-10, 1.296e-10, 1.097e-10, 9.173e-11, 7.73e-11,
6.547e-11, 5.191e-11, 4.198e-11, 3.361e-11, 2.732e-11, 2.244e-11,
1.791e-11, 1.509e-11, 1.243e-11, 1.035e-11, 8.969e-12, 7.394e-12,
6.323e-12, 5.282e-12, 4.543e-12, 3.752e-12, 3.14e-12, 2.6e-12,
2.194e-12, 1.825e-12, 1.511e-12, 1.245e-12, 1.024e-12, 8.539e-13,
7.227e-13, 6.102e-13, 5.189e-13, 4.43e-13, 3.774e-13, 3.236e-13,
2.8e-13, 2.444e-13, 2.156e-13, 1.932e-13, 1.775e-13, 1.695e-13,
1.672e-13, 1.704e-13, 1.825e-13, 2.087e-13, 2.614e-13, 3.377e-13,
4.817e-13, 6.989e-13, 1.062e-12, 1.562e-12, 2.288e-12, 3.295e-12,
4.55e-12, 5.965e-12, 7.546e-12, 9.395e-12, 1.103e-11, 1.228e-11,
1.318e-11, 1.38e-11, 1.421e-11, 1.39e-11, 1.358e-11, 1.336e-11,
1.342e-11, 1.356e-11, 1.424e-11, 1.552e-11, 1.73e-11, 1.951e-11,
2.128e-11, 2.249e-11, 2.277e-11, 2.226e-11, 2.111e-11, 1.922e-11,
1.775e-11, 1.661e-11, 1.547e-11, 1.446e-11, 1.323e-11, 1.21e-11,
1.054e-11, 9.283e-12, 8.671e-12, 8.67e-12, 9.429e-12, 1.062e-11,
1.255e-11, 1.506e-11, 1.818e-11, 2.26e-11, 2.831e-11, 3.723e-11,
5.092e-11, 6.968e-11, 9.826e-11, 1.349e-10, 1.87e-10, 2.58e-10,
3.43e-10, 4.424e-10, 5.521e-10, 6.812e-10, 8.064e-10, 9.109e-10,
9.839e-10, 1.028e-9, 1.044e-9, 1.029e-9, 1.005e-9, 1.002e-9,
1.038e-9, 1.122e-9, 1.233e-9, 1.372e-9, 1.524e-9, 1.665e-9,
1.804e-9, 1.908e-9, 2.015e-9, 2.117e-9, 2.219e-9, 2.336e-9,
2.531e-9, 2.805e-9, 3.189e-9, 3.617e-9, 4.208e-9, 4.911e-9,
5.619e-9, 6.469e-9, 7.188e-9, 7.957e-9, 8.503e-9, 9.028e-9,
9.571e-9, 9.99e-9, 1.055e-8, 1.102e-8, 1.132e-8, 1.141e-8,
1.145e-8, 1.145e-8, 1.176e-8, 1.224e-8, 1.304e-8, 1.388e-8,
1.445e-8, 1.453e-8, 1.368e-8, 1.22e-8, 1.042e-8, 8.404e-9,
6.403e-9, 4.643e-9, 3.325e-9, 2.335e-9, 1.638e-9, 1.19e-9,
9.161e-10, 7.412e-10, 6.226e-10, 5.516e-10, 5.068e-10, 4.831e-10,
4.856e-10, 5.162e-10, 5.785e-10, 6.539e-10, 7.485e-10, 8.565e-10,
9.534e-10, 1.052e-9, 1.115e-9, 1.173e-9, 1.203e-9, 1.224e-9,
1.243e-9, 1.248e-9, 1.261e-9, 1.265e-9, 1.25e-9, 1.217e-9,
1.176e-9, 1.145e-9, 1.153e-9, 1.199e-9, 1.278e-9, 1.366e-9,
1.426e-9, 1.444e-9, 1.365e-9, 1.224e-9, 1.051e-9, 8.539e-10,
6.564e-10, 4.751e-10, 3.404e-10, 2.377e-10, 1.631e-10, 1.114e-10,
7.87e-11, 5.793e-11, 4.284e-11, 3.3e-11, 2.62e-11, 2.152e-11,
1.777e-11, 1.496e-11, 1.242e-11, 1.037e-11, 8.725e-12, 7.004e-12,
5.718e-12, 4.769e-12, 3.952e-12, 3.336e-12, 2.712e-12, 2.213e-12,
1.803e-12, 1.492e-12, 1.236e-12, 1.006e-12, 8.384e-13, 7.063e-13,
5.879e-13, 4.93e-13, 4.171e-13, 3.569e-13, 3.083e-13, 2.688e-13,
2.333e-13, 2.035e-13, 1.82e-13, 1.682e-13, 1.635e-13, 1.674e-13,
1.769e-13, 2.022e-13, 2.485e-13, 3.127e-13, 4.25e-13, 5.928e-13,
8.514e-13, 1.236e-12, 1.701e-12, 2.392e-12, 3.231e-12, 4.35e-12,
5.559e-12, 6.915e-12, 8.519e-12, 1.013e-11, 1.146e-11, 1.24e-11,
1.305e-11, 1.333e-11, 1.318e-11, 1.263e-11, 1.238e-11, 1.244e-11,
1.305e-11, 1.432e-11, 1.623e-11, 1.846e-11, 2.09e-11, 2.328e-11,
2.526e-11, 2.637e-11, 2.702e-11, 2.794e-11, 2.889e-11, 2.989e-11,
3.231e-11, 3.68e-11, 4.375e-11, 5.504e-11, 7.159e-11, 9.502e-11,
1.279e-10, 1.645e-10, 2.098e-10, 2.618e-10, 3.189e-10, 3.79e-10,
4.303e-10, 4.753e-10, 5.027e-10, 5.221e-10, 5.293e-10, 5.346e-10,
5.467e-10, 5.796e-10, 6.2e-10, 6.454e-10, 6.705e-10, 6.925e-10,
7.233e-10, 7.35e-10, 7.538e-10, 7.861e-10, 8.077e-10, 8.132e-10,
7.749e-10, 7.036e-10, 6.143e-10, 5.093e-10, 4.089e-10, 3.092e-10,
```

```
2.299e-10, 1.705e-10, 1.277e-10, 9.723e-11, 7.533e-11, 6.126e-11,
5.154e-11, 4.428e-11, 3.913e-11, 3.521e-11, 3.297e-11, 3.275e-11,
3.46e-11, 3.798e-11, 4.251e-11, 4.745e-11, 5.232e-11, 5.606e-11,
5.82e-11, 5.88e-11, 5.79e-11, 5.661e-11, 5.491e-11, 5.366e-11,
5.341e-11, 5.353e-11, 5.336e-11, 5.293e-11, 5.248e-11, 5.235e-11,
5.208e-11, 5.322e-11, 5.521e-11, 5.725e-11, 5.827e-11, 5.685e-11,
5.245e-11, 4.612e-11, 3.884e-11, 3.129e-11, 2.404e-11, 1.732e-11,
1.223e-11, 8.574e-12, 5.888e-12, 3.986e-12, 2.732e-12, 1.948e-12,
1.414e-12, 1.061e-12, 8.298e-13, 6.612e-13, 5.413e-13, 4.472e-13,
3.772e-13, 3.181e-13, 2.645e-13, 2.171e-13, 1.778e-13, 1.464e-13,
1.183e-13, 9.637e-14, 7.991e-14, 6.668e-14, 5.57e-14, 4.663e-14,
3.848e-14, 3.233e-14, 2.706e-14, 2.284e-14, 1.944e-14, 1.664e-14,
1.43e-14, 1.233e-14, 1.066e-14, 9.234e-15, 8.023e-15, 6.993e-15,
6.119e-15, 5.384e-15, 4.774e-15, 4.283e-15, 3.916e-15, 3.695e-15,
3.682e-15, 4.004e-15, 4.912e-15, 6.853e-15, 1.056e-14, 1.712e-14,
2.804e-14, 4.516e-14, 7.113e-14, 1.084e-13, 1.426e-13, 1.734e-13,
1.978e-13, 2.194e-13, 2.388e-13, 2.489e-13, 2.626e-13, 2.865e-13,
3.105e-13, 3.387e-13, 3.652e-13, 3.984e-13, 4.398e-13, 4.906e-13,
5.55e-13, 6.517e-13, 7.813e-13, 9.272e-13, 1.164e-12, 1.434e-12,
1.849e-12, 2.524e-12, 3.328e-12, 4.523e-12, 6.108e-12, 8.207e-12,
1.122e-11, 1.477e-11, 1.9e-11, 2.412e-11, 2.984e-11, 3.68e-11,
4.353e-11, 4.963e-11, 5.478e-11, 5.903e-11, 6.233e-11, 6.483e-11,
6.904e-11, 7.569e-11, 8.719e-11, 1.048e-10, 1.278e-10, 1.557e-10,
1.869e-10, 2.218e-10, 2.61e-10, 2.975e-10, 3.371e-10, 3.746e-10,
4.065e-10, 4.336e-10, 4.503e-10, 4.701e-10, 4.8e-10, 4.917e-10,
5.038e-10, 5.128e-10, 5.143e-10, 5.071e-10, 5.019e-10, 5.025e-10,
5.183e-10, 5.496e-10, 5.877e-10, 6.235e-10, 6.42e-10, 6.234e-10,
5.698e-10, 4.916e-10, 4.022e-10, 3.126e-10, 2.282e-10, 1.639e-10,
1.142e-10, 7.919e-11, 5.69e-11, 4.313e-11, 3.413e-11, 2.807e-11,
2.41e-11, 2.166e-11, 2.024e-11, 1.946e-11, 1.929e-11, 1.963e-11,
2.035e-11, 2.162e-11, 2.305e-11, 2.493e-11, 2.748e-11, 3.048e-11,
3.413e-11, 3.754e-11, 4.155e-11, 4.635e-11, 5.11e-11, 5.734e-11,
6.338e-11, 6.99e-11, 7.611e-11, 8.125e-11, 8.654e-11, 8.951e-11,
9.182e-11, 9.31e-11, 9.273e-11, 9.094e-11, 8.849e-11, 8.662e-11,
8.67e-11, 8.972e-11, 9.566e-11, 1.025e-10, 1.083e-10, 1.111e-10,
1.074e-10, 9.771e-11, 8.468e-11, 6.958e-11, 5.47e-11, 4.04e-11,
2.94e-11, 2.075e-11, 1.442e-11, 1.01e-11, 7.281e-12, 5.409e-12,
4.138e-12, 3.304e-12, 2.784e-12, 2.473e-12, 2.273e-12, 2.186e-12,
2.118e-12, 2.066e-12, 1.958e-12, 1.818e-12, 1.675e-12, 1.509e-12,
1.349e-12, 1.171e-12, 9.838e-13, 8.213e-13, 6.765e-13, 5.378e-13,
4.161e-13, 3.119e-13, 2.279e-13, 1.637e-13, 1.152e-13, 8.112e-14,
5.919e-14, 4.47e-14, 3.492e-14, 2.811e-14, 2.319e-14, 1.948e-14,
1.66e-14, 1.432e-14, 1.251e-14, 1.109e-14, 1.006e-14, 9.45e-15,
9.384e-15, 1.012e-14, 1.216e-14, 1.636e-14, 2.305e-14, 3.488e-14,
5.572e-14, 8.479e-14, 1.265e-13, 1.905e-13, 2.73e-13, 3.809e-13,
4.955e-13, 6.303e-13, 7.861e-13, 9.427e-13, 1.097e-12, 1.212e-12,
1.328e-12, 1.415e-12, 1.463e-12, 1.495e-12, 1.571e-12, 1.731e-12,
1.981e-12, 2.387e-12, 2.93e-12, 3.642e-12, 4.584e-12, 5.822e-12,
7.278e-12, 9.193e-12, 1.135e-11, 1.382e-11, 1.662e-11, 1.958e-11,
2.286e-11, 2.559e-11, 2.805e-11, 2.988e-11, 3.106e-11, 3.182e-11,
3.2e-11, 3.258e-11, 3.362e-11, 3.558e-11, 3.688e-11, 3.8e-11,
3.929e-11, 4.062e-11, 4.186e-11, 4.293e-11, 4.48e-11, 4.643e-11,
4.704e-11, 4.571e-11, 4.206e-11, 3.715e-11, 3.131e-11, 2.541e-11,
1.978e-11, 1.508e-11, 1.146e-11, 8.7e-12, 6.603e-12, 5.162e-12,
4.157e-12, 3.408e-12, 2.829e-12, 2.405e-12, 2.071e-12, 1.826e-12,
1.648e-12, 1.542e-12, 1.489e-12, 1.485e-12, 1.493e-12, 1.545e-12,
1.637e-12, 1.814e-12, 2.061e-12, 2.312e-12, 2.651e-12, 3.03e-12,
3.46e-12, 3.901e-12, 4.306e-12, 4.721e-12, 5.008e-12, 5.281e-12,
5.541e-12, 5.791e-12, 6.115e-12, 6.442e-12, 6.68e-12, 6.791e-12,
6.831e-12, 6.839e-12, 6.946e-12, 7.128e-12, 7.537e-12, 8.036e-12,
8.392e-12, 8.526e-12, 8.11e-12, 7.325e-12, 6.329e-12, 5.183e-12,
4.081e-12, 2.985e-12, 2.141e-12, 1.492e-12, 1.015e-12, 6.684e-13,
4.414e-13, 2.987e-13, 2.038e-13, 1.391e-13, 9.86e-14, 7.24e-14,
5.493e-14, 4.288e-14, 3.427e-14, 2.787e-14, 2.296e-14, 1.909e-14,
1.598e-14, 1.344e-14, 1.135e-14, 9.616e-15, 8.169e-15, 6.957e-15,
5.938e-15, 5.08e-15, 4.353e-15, 3.738e-15, 3.217e-15, 2.773e-15,
2.397e-15, 2.077e-15, 1.805e-15, 1.575e-15, 1.382e-15, 1.221e-15,
1.09e-15, 9.855e-16, 9.068e-16, 8.537e-16, 8.27e-16, 8.29e-16,
8.634e-16, 9.359e-16, 1.055e-15, 1.233e-15, 1.486e-15, 1.839e-15,
2.326e-15, 2.998e-15, 3.934e-15, 5.256e-15, 7.164e-15, 9.984e-15,
1.427e-14, 2.099e-14, 3.196e-14, 5.121e-14, 7.908e-14, 1.131e-13,
1.602e-13, 2.239e-13, 3.075e-13, 4.134e-13, 5.749e-13, 7.886e-13,
1.071e-12, 1.464e-12, 2.032e-12, 2.8e-12, 3.732e-12, 4.996e-12,
6.483e-12, 8.143e-12, 1.006e-11, 1.238e-11, 1.484e-11, 1.744e-11,
2.02e-11, 2.274e-11, 2.562e-11, 2.848e-11, 3.191e-11, 3.617e-11,
4.081e-11, 4.577e-11, 4.937e-11, 5.204e-11, 5.401e-11, 5.462e-11,
5.507e-11, 5.51e-11, 5.605e-11, 5.686e-11, 5.739e-11, 5.766e-11,
5.74e-11, 5.754e-11, 5.761e-11, 5.777e-11, 5.712e-11, 5.51e-11,
5.088e-11, 4.438e-11, 3.728e-11, 2.994e-11, 2.305e-11, 1.715e-11,
1.256e-11, 9.208e-12, 6.745e-12, 5.014e-12, 3.785e-12, 2.9e-12,
2.239e-12, 1.757e-12, 1.414e-12, 1.142e-12, 9.482e-13, 8.01e-13,
6.961e-13, 6.253e-13, 5.735e-13, 5.433e-13, 5.352e-13, 5.493e-13,
5.706e-13, 6.068e-13, 6.531e-13, 7.109e-13, 7.767e-13, 8.59e-13,
9.792e-13, 1.142e-12, 1.371e-12, 1.65e-12, 1.957e-12, 2.302e-12,
2.705e-12, 3.145e-12, 3.608e-12, 4.071e-12, 4.602e-12, 5.133e-12,
5.572e-12, 5.987e-12, 6.248e-12, 6.533e-12, 6.757e-12, 6.935e-12,
7.224e-12, 7.422e-12, 7.538e-12, 7.547e-12, 7.495e-12, 7.543e-12,
```

```
    7.725e-12, 8.139e-12, 8.627e-12, 9.146e-12, 9.443e-12, 9.318e-12,
    8.649e-12, 7.512e-12, 6.261e-12, 4.915e-12, 3.647e-12, 2.597e-12,
    1.785e-12, 1.242e-12, 8.66e-13, 6.207e-13, 4.61e-13, 3.444e-13,
    2.634e-13, 2.1e-13, 1.725e-13, 1.455e-13, 1.237e-13, 1.085e-13,
    9.513e-14, 7.978e-14, 6.603e-14, 5.288e-14, 4.084e-14, 2.952e-14,
    2.157e-14, 1.593e-14, 1.199e-14, 9.267e-15, 7.365e-15, 6.004e-15,
    4.995e-15, 4.218e-15, 3.601e-15, 3.101e-15, 2.692e-15, 2.36e-15,
    2.094e-15, 1.891e-15, 1.755e-15, 1.699e-15, 1.755e-15, 1.987e-15,
    2.506e-15, 3.506e-15, 5.289e-15, 8.311e-15, 1.325e-14, 2.129e-14,
    3.237e-14, 4.595e-14, 6.441e-14, 8.433e-14, 1.074e-13, 1.383e-13,
    1.762e-13, 2.281e-13, 2.831e-13, 3.523e-13, 4.38e-13, 5.304e-13,
    6.29e-13, 7.142e-13, 8.032e-13, 8.934e-13, 9.888e-13, 1.109e-12,
    1.261e-12, 1.462e-12, 1.74e-12, 2.099e-12, 2.535e-12, 3.008e-12,
    3.462e-12, 3.856e-12, 4.098e-12, 4.239e-12, 4.234e-12, 4.132e-12,
    3.986e-12, 3.866e-12, 3.829e-12, 3.742e-12, 3.705e-12, 3.694e-12,
    3.765e-12, 3.849e-12, 3.929e-12, 4.056e-12, 4.092e-12, 4.047e-12,
    3.792e-12, 3.407e-12, 2.953e-12, 2.429e-12, 1.931e-12, 1.46e-12,
    1.099e-12, 8.199e-13, 6.077e-13, 4.449e-13, 3.359e-13, 2.524e-13,
    1.881e-13, 1.391e-13, 1.02e-13, 7.544e-14, 5.555e-14, 4.22e-14,
    3.321e-14, 2.686e-14, 2.212e-14, 1.78e-14, 1.369e-14, 1.094e-14,
    9.13e-15, 8.101e-15, 7.828e-15, 8.393e-15, 1.012e-14, 1.259e-14,
    1.538e-14, 1.961e-14, 2.619e-14, 3.679e-14, 5.049e-14, 6.917e-14,
    8.88e-14, 1.115e-13, 1.373e-13, 1.619e-13, 1.878e-13, 2.111e-13,
    2.33e-13, 2.503e-13, 2.613e-13, 2.743e-13, 2.826e-13, 2.976e-13,
    3.162e-13, 3.36e-13, 3.491e-13, 3.541e-13, 3.595e-13, 3.608e-13,
    3.709e-13, 3.869e-13, 4.12e-13, 4.366e-13, 4.504e-13, 4.379e-13,
    3.955e-13, 3.385e-13, 2.741e-13, 2.089e-13, 1.427e-13, 9.294e-14,
    5.775e-14, 3.565e-14, 2.21e-14, 1.398e-14, 9.194e-15, 6.363e-15,
    4.644e-15, 3.55e-15, 2.808e-15, 2.274e-15, 1.871e-15, 1.557e-15,
    1.308e-15, 1.108e-15, 9.488e-16, 8.222e-16, 7.238e-16, 6.506e-16,
    6.008e-16, 5.742e-16, 5.724e-16, 5.991e-16, 6.625e-16, 7.775e-16,
    9.734e-16, 1.306e-15, 1.88e-15, 2.879e-15, 4.616e-15, 7.579e-15,
    1.248e-14, 2.03e-14, 3.244e-14, 5.171e-14, 7.394e-14, 9.676e-14,
    1.199e-13, 1.467e-13, 1.737e-13, 2.02e-13, 2.425e-13, 3.016e-13,
    3.7e-13, 4.617e-13, 5.949e-13, 7.473e-13, 9.378e-13, 1.191e-12,
    1.481e-12, 1.813e-12, 2.232e-12, 2.722e-12, 3.254e-12, 3.845e-12,
    4.458e-12, 5.048e-12, 5.511e-12, 5.898e-12, 6.204e-12, 6.293e-12,
    6.386e-12, 6.467e-12, 6.507e-12, 6.466e-12, 6.443e-12, 6.598e-12,
    6.873e-12, 7.3e-12, 7.816e-12, 8.368e-12, 8.643e-12, 8.466e-12,
    7.871e-12, 6.853e-12, 5.714e-12, 4.482e-12, 3.392e-12, 2.613e-12,
    2.008e-12, 1.562e-12, 1.228e-12, 9.888e-13, 7.646e-13, 5.769e-13,
    4.368e-13, 3.324e-13, 2.508e-13, 1.916e-13
  };

  static double xfcrev[15] =
    { 1.003, 1.009, 1.015, 1.023, 1.029, 1.033, 1.037,
    1.039, 1.04, 1.046, 1.036, 1.027, 1.01, 1.002, 1.
  };

  double a1, a2, a3, dw, ew, dx, xw, xx, vf2, vf6, cw260, cw296,
    sfac, fscal, cwfrn, ctmpth, ctwfrn, ctwslf;

  int iw, ix;

  /* Get H2O continuum absorption... */
  xw = nu / 10 + 1;
  if (xw >= 1 && xw < 2001) {
    iw = (int) xw;
    dw = xw - iw;
    ew = 1 - dw;
    cw296 = ew * h2o296[iw - 1] + dw * h2o296[iw];
    cw260 = ew * h2o260[iw - 1] + dw * h2o260[iw];
    cwfrn = ew * h2ofrn[iw - 1] + dw * h2ofrn[iw];
    if (nu <= 820 || nu >= 960) {
      sfac = 1;
    } else {
      xx = (nu - 820) / 10;
      ix = (int) xx;
      dx = xx - ix;
      sfac = (1 - dx) * xfcrev[ix] + dx * xfcrev[ix + 1];
    }
    ctwslf = sfac * cw296 * pow(cw260 / cw296, (296 - t) / (296 - 260));
    vf2 = gsl_pow_2(nu - 370);
    vf6 = gsl_pow_3(vf2);
    fscal = 36100 / (vf2 + vf6 * 1e-8 + 36100) * -.25 + 1;
    ctwfrn = cwfrn * fscal;
    a1 = nu * u * tanh(.7193876 / t * nu);
    a2 = 296 / t;
    a3 = p / P0 * (q * ctwslf + (1 - q) * ctwfrn) * 1e-20;
    ctmpth = a1 * a2 * a3;
  } else
    ctmpth = 0;
  return ctmpth;
}
```

5.31.2.8 double ctmn2 (double *nu*, double *p*, double *t*)

Compute nitrogen continuum (absorption coefficient).

Definition at line 2790 of file jurassic.c.

```
           {

  static double ba[98] = { 0., 4.45e-8, 5.22e-8, 6.46e-8, 7.75e-8, 9.03e-8,
    1.06e-7, 1.21e-7, 1.37e-7, 1.57e-7, 1.75e-7, 2.01e-7, 2.3e-7,
    2.59e-7, 2.95e-7, 3.26e-7, 3.66e-7, 4.05e-7, 4.47e-7, 4.92e-7,
    5.34e-7, 5.84e-7, 6.24e-7, 6.67e-7, 7.14e-7, 7.26e-7, 7.54e-7,
    7.84e-7, 8.09e-7, 8.42e-7, 8.62e-7, 8.87e-7, 9.11e-7, 9.36e-7,
    9.76e-7, 1.03e-6, 1.11e-6, 1.23e-6, 1.39e-6, 1.61e-6, 1.76e-6,
    1.94e-6, 1.97e-6, 1.87e-6, 1.75e-6, 1.56e-6, 1.42e-6, 1.35e-6,
    1.32e-6, 1.29e-6, 1.29e-6, 1.3e-6, 1.32e-6, 1.33e-6,
    1.34e-6, 1.35e-6, 1.33e-6, 1.31e-6, 1.29e-6, 1.24e-6, 1.2e-6,
    1.16e-6, 1.1e-6, 1.04e-6, 9.96e-7, 9.38e-7, 8.63e-7, 7.98e-7,
    7.26e-7, 6.55e-7, 5.94e-7, 5.35e-7, 4.74e-7, 4.24e-7, 3.77e-7,
    3.33e-7, 2.96e-7, 2.63e-7, 2.34e-7, 2.08e-7, 1.85e-7, 1.67e-7,
    1.47e-7, 1.32e-7, 1.2e-7, 1.09e-7, 9.85e-8, 9.08e-8, 8.18e-8,
    7.56e-8, 6.85e-8, 6.14e-8, 5.83e-8, 5.77e-8, 5e-8, 4.32e-8, 0.
  };

  static double betaa[98] = { 802., 802., 761., 722., 679., 646., 609., 562.,
    511., 472., 436., 406., 377., 355., 338., 319., 299., 278., 255.,
    233., 208., 184., 149., 107., 66., 25., -13., -49., -82., -104.,
    -119., -130., -139., -144., -146., -146., -147., -148., -150.,
    -153., -160., -169., -181., -189., -195., -200., -205., -209.,
    -211., -210., -210., -209., -205., -199., -190., -180., -168.,
    -157., -143., -126., -108., -89., -63., -32., 1., 35., 65., 95.,
    121., 141., 152., 161., 164., 164., 161., 155., 148., 143., 137.,
    133., 131., 133., 139., 150., 165., 187., 213., 248., 284., 321.,
    372., 449., 514., 569., 609., 642., 673., 673.
  };

  static double nua[98] = { 2120., 2125., 2130., 2135., 2140., 2145., 2150.,
    2155., 2160., 2165., 2170., 2175., 2180., 2185., 2190., 2195.,
    2200., 2205., 2210., 2215., 2220., 2225., 2230., 2235., 2240.,
    2245., 2250., 2255., 2260., 2265., 2270., 2275., 2280., 2285.,
    2290., 2295., 2300., 2305., 2310., 2315., 2320., 2325., 2330.,
    2335., 2340., 2345., 2350., 2355., 2360., 2365., 2370., 2375.,
    2380., 2385., 2390., 2395., 2400., 2405., 2410., 2415., 2420.,
    2425., 2430., 2435., 2440., 2445., 2450., 2455., 2460., 2465.,
    2470., 2475., 2480., 2485., 2490., 2495., 2500., 2505., 2510.,
    2515., 2520., 2525., 2530., 2535., 2540., 2545., 2550., 2555.,
    2560., 2565., 2570., 2575., 2580., 2585., 2590., 2595., 2600., 2605.
  };

  double b, beta, q_n2 = 0.79, t0 = 273, tr = 296;

  int idx;

  /* Check wavenumber range... */
  if (nu < nua[0] || nu > nua[97])
    return 0;

  /* Interpolate B and beta... */
  idx = locate(nua, 98, nu);
  b = LIN(nua[idx], ba[idx], nua[idx + 1], ba[idx + 1], nu);
  beta = LIN(nua[idx], betaa[idx], nua[idx + 1], betaa[idx + 1], nu);

  /* Compute absorption coefficient... */
  return 0.1 * gsl_pow_2(p / P0) * gsl_pow_2(t0 / t)
    * exp(beta * (1 / tr - 1 / t))
    * q_n2 * b * (q_n2 + (1 - q_n2) * (1.294 - 0.4545 * t / tr));
}
```

Here is the call graph for this function:

5.31.2.9 double ctmo2 (double *nu*, double *p*, double *t*)

Compute oxygen continuum (absorption coefficient).

Definition at line 2857 of file jurassic.c.

```
          {

  static double ba[90] = { 0., .061, .074, .084, .096, .12, .162, .208, .246,
    .285, .314, .38, .444, .5, .571, .673, .768, .853, .966, 1.097,
    1.214, 1.333, 1.466, 1.591, 1.693, 1.796, 1.922, 2.037, 2.154,
    2.264, 2.375, 2.508, 2.671, 2.847, 3.066, 3.417, 3.828, 4.204,
    4.453, 4.599, 4.528, 4.284, 3.955, 3.678, 3.477, 3.346, 3.29,
    3.251, 3.231, 3.226, 3.212, 3.192, 3.108, 3.033, 2.911, 2.798,
    2.646, 2.508, 2.322, 2.13, 1.928, 1.757, 1.588, 1.417, 1.253,
    1.109, .99, .888, .791, .678, .587, .524, .464, .403, .357, .32,
    .29, .267, .242, .215, .182, .16, .146, .128, .103, .087, .081,
    .071, .064, 0.
  };

  static double betaa[90] = { 467., 467., 400., 315., 379., 368., 475., 521.,
    531., 512., 442., 444., 430., 381., 335., 324., 296., 248., 215.,
    193., 158., 127., 101., 71., 31., -6., -26., -47., -63., -79.,
    -88., -88., -87., -90., -98., -99., -109., -134., -160., -167.,
    -164., -158., -153., -151., -156., -166., -168., -173., -170.,
    -161., -145., -126., -108., -84., -59., -29., 4., 41., 73., 97.,
    123., 159., 198., 220., 242., 256., 281., 311., 334., 319., 313.,
    321., 323., 310., 315., 320., 335., 361., 378., 373., 338., 319.,
    346., 322., 291., 290., 350., 371., 504., 504.
  };

  static double nua[90] = { 1360., 1365., 1370., 1375., 1380., 1385., 1390.,
    1395., 1400., 1405., 1410., 1415., 1420., 1425., 1430., 1435.,
    1440., 1445., 1450., 1455., 1460., 1465., 1470., 1475., 1480.,
    1485., 1490., 1495., 1500., 1505., 1510., 1515., 1520., 1525.,
    1530., 1535., 1540., 1545., 1550., 1555., 1560., 1565., 1570.,
    1575., 1580., 1585., 1590., 1595., 1600., 1605., 1610., 1615.,
    1620., 1625., 1630., 1635., 1640., 1645., 1650., 1655., 1660.,
    1665., 1670., 1675., 1680., 1685., 1690., 1695., 1700., 1705.,
    1710., 1715., 1720., 1725., 1730., 1735., 1740., 1745., 1750.,
    1755., 1760., 1765., 1770., 1775., 1780., 1785., 1790., 1795.,
    1800., 1805.
  };

  double b, beta, q_o2 = 0.21, t0 = 273, tr = 296;

  int idx;

  /* Check wavenumber range... */
  if (nu < nua[0] || nu > nua[89])
    return 0;

  /* Interpolate B and beta... */
  idx = locate(nua, 90, nu);
  b = LIN(nua[idx], ba[idx], nua[idx + 1], ba[idx + 1], nu);
  beta = LIN(nua[idx], betaa[idx], nua[idx + 1], betaa[idx + 1], nu);

  /* Compute absorption coefficient... */
  return 0.1 * gsl_pow_2(p / P0) * gsl_pow_2(t0 / t)
    * exp(beta * (1 / tr - 1 / t)) * q_o2 * b;
}
```

Here is the call graph for this function:

5.31.2.10 void copy_atm (ctl_t * *ctl*, atm_t * *atm_dest*, atm_t * *atm_src*, int *init*)

Copy and initialize atmospheric data.

Definition at line 2918 of file jurassic.c.

```
          {

  int ig, ip, iw;

  size_t s;

  /* Data size... */
  s = (size_t) atm_src->np * sizeof(double);

  /* Copy data... */
  atm_dest->np = atm_src->np;
  memcpy(atm_dest->time, atm_src->time, s);
  memcpy(atm_dest->z, atm_src->z, s);
  memcpy(atm_dest->lon, atm_src->lon, s);
  memcpy(atm_dest->lat, atm_src->lat, s);
```

```
  memcpy(atm_dest->p, atm_src->p, s);
  memcpy(atm_dest->t, atm_src->t, s);
  for (ig = 0; ig < ctl->ng; ig++)
    memcpy(atm_dest->q[ig], atm_src->q[ig], s);
  for (iw = 0; iw < ctl->nw; iw++)
    memcpy(atm_dest->k[iw], atm_src->k[iw], s);

  /* Initialize... */
  if (init)
    for (ip = 0; ip < atm_dest->np; ip++) {
      atm_dest->p[ip] = 0;
      atm_dest->t[ip] = 0;
      for (ig = 0; ig < ctl->ng; ig++)
        atm_dest->q[ig][ip] = 0;
      for (iw = 0; iw < ctl->nw; iw++)
        atm_dest->k[iw][ip] = 0;
    }
}
```

5.31.2.11 void copy_obs (ctl_t * *ctl,* obs_t * *obs_dest,* obs_t * *obs_src,* int *init*)

Copy and initialize observation data.

Definition at line 2958 of file jurassic.c.

```
              {

  int id, ir;

  size_t s;

  /* Data size... */
  s = (size_t) obs_src->nr * sizeof(double);

  /* Copy data... */
  obs_dest->nr = obs_src->nr;
  memcpy(obs_dest->time, obs_src->time, s);
  memcpy(obs_dest->obsz, obs_src->obsz, s);
  memcpy(obs_dest->obslon, obs_src->obslon, s);
  memcpy(obs_dest->obslat, obs_src->obslat, s);
  memcpy(obs_dest->vpz, obs_src->vpz, s);
  memcpy(obs_dest->vplon, obs_src->vplon, s);
  memcpy(obs_dest->vplat, obs_src->vplat, s);
  memcpy(obs_dest->tpz, obs_src->tpz, s);
  memcpy(obs_dest->tplon, obs_src->tplon, s);
  memcpy(obs_dest->tplat, obs_src->tplat, s);
  for (id = 0; id < ctl->nd; id++)
    memcpy(obs_dest->rad[id], obs_src->rad[id], s);
  for (id = 0; id < ctl->nd; id++)
    memcpy(obs_dest->tau[id], obs_src->tau[id], s);

  /* Initialize... */
  if (init)
    for (id = 0; id < ctl->nd; id++)
      for (ir = 0; ir < obs_dest->nr; ir++)
        if (gsl_finite(obs_dest->rad[id][ir])) {
          obs_dest->rad[id][ir] = 0;
          obs_dest->tau[id][ir] = 0;
        }
}
```

5.31.2.12 int find_emitter (ctl_t * *ctl,* const char * *emitter*)

Find index of an emitter.

Definition at line 3000 of file jurassic.c.

```
                           {

  int ig;

  for (ig = 0; ig < ctl->ng; ig++)
    if (strcasecmp(ctl->emitter[ig], emitter) == 0)
      return ig;

  return -1;
}
```

5.31.2.13 void formod (ctl_t * *ctl*, atm_t * *atm*, obs_t * *obs*)

Determine ray paths and compute radiative transfer.

Definition at line 3015 of file jurassic.c.

```
                 {

  int id, ir, *mask;

  /* Allocate... */
  ALLOC(mask, int,
        ND * NR);

  /* Save observation mask... */
  for (id = 0; id < ctl->nd; id++)
    for (ir = 0; ir < obs->nr; ir++)
      mask[id * NR + ir] = !gsl_finite(obs->rad[id][ir]);

  /* Hydrostatic equilibrium... */
  hydrostatic(ctl, atm);

  /* Claculate pencil beams... */
  for (ir = 0; ir < obs->nr; ir++)
    formod_pencil(ctl, atm, obs, ir);

  /* Apply field-of-view convolution... */
  formod_fov(ctl, obs);

  /* Convert radiance to brightness temperature... */
  if (ctl->write_bbt)
    for (id = 0; id < ctl->nd; id++)
      for (ir = 0; ir < obs->nr; ir++)
        obs->rad[id][ir] = brightness(obs->rad[id][ir], ctl->nu[id]);

  /* Apply observation mask... */
  for (id = 0; id < ctl->nd; id++)
    for (ir = 0; ir < obs->nr; ir++)
      if (mask[id * NR + ir])
        obs->rad[id][ir] = GSL_NAN;

  /* Free... */
  free(mask);
}
```

Here is the call graph for this function:

5.31.2.14 void formod_continua (ctl_t * *ctl*, los_t * *los*, int *ip*, double * *beta*)

Compute absorption coefficient of continua.

Definition at line 3059 of file jurassic.c.

```
                  {

  static int ig_co2 = -999, ig_h2o = -999;

  int id;

  /* Extinction... */
  for (id = 0; id < ctl->nd; id++)
    beta[id] = los->k[ctl->window[id]][ip];

  /* CO2 continuum... */
  if (ctl->ctm_co2) {
    if (ig_co2 == -999)
      ig_co2 = find_emitter(ctl, "CO2");
    if (ig_co2 >= 0)
      for (id = 0; id < ctl->nd; id++)
        beta[id] += ctmco2(ctl->nu[id], los->p[ip], los->t[ip],
                           los->u[ig_co2][ip]) / los->ds[ip];
  }

  /* H2O continuum... */
  if (ctl->ctm_h2o) {
```

```
    if (ig_h2o == -999)
      ig_h2o = find_emitter(ctl, "H2O");
    if (ig_h2o >= 0)
      for (id = 0; id < ctl->nd; id++)
        beta[id] += ctmh2o(ctl->nu[id], los->p[ip], los->t[ip],
                           los->q[ig_h2o][ip],
                           los->u[ig_h2o][ip]) / los->ds[ip];
  }

  /* N2 continuum... */
  if (ctl->ctm_n2)
    for (id = 0; id < ctl->nd; id++)
      beta[id] += ctmn2(ctl->nu[id], los->p[ip], los->t[ip]);

  /* O2 continuum... */
  if (ctl->ctm_o2)
    for (id = 0; id < ctl->nd; id++)
      beta[id] += ctmo2(ctl->nu[id], los->p[ip], los->t[ip]);
}
```

Here is the call graph for this function:

5.31.2.15 void formod_fov (ctl_t * *ctl,* obs_t * *obs*)

Apply field of view convolution.

Definition at line 3107 of file jurassic.c.

```
                {

  static double dz[NSHAPE], w[NSHAPE];

  static int init = 0, n;

  obs_t *obs2;

  double rad[ND][NR], tau[ND][NR], wsum, z[NR], zfov;

  int i, id, idx, ir, ir2, nz;

  /* Do not take into account FOV... */
  if (ctl->fov[0] == '-')
    return;

  /* Initialize FOV data... */
  if (!init) {
    init = 1;
    read_shape(ctl->fov, dz, w, &n);
  }

  /* Allocate... */
  ALLOC(obs2, obs_t, 1);

  /* Copy observation data... */
  copy_obs(ctl, obs2, obs, 0);

  /* Loop over ray paths... */
  for (ir = 0; ir < obs->nr; ir++) {

    /* Get radiance and transmittance profiles... */
    nz = 0;
    for (ir2 = GSL_MAX(ir - NFOV, 0); ir2 < GSL_MIN(ir + 1 + NFOV, obs->nr);
         ir2++)
      if (obs->time[ir2] == obs->time[ir]) {
        z[nz] = obs2->vpz[ir2];
        for (id = 0; id < ctl->nd; id++) {
          rad[id][nz] = obs2->rad[id][ir2];
          tau[id][nz] = obs2->tau[id][ir2];
        }
        nz++;
      }
    if (nz < 2)
      ERRMSG("Cannot apply FOV convolution!");

    /* Convolute profiles with FOV... */
    wsum = 0;
    for (id = 0; id < ctl->nd; id++) {
      obs->rad[id][ir] = 0;
      obs->tau[id][ir] = 0;
```

```
    }
    for (i = 0; i < n; i++) {
      zfov = obs->vpz[ir] + dz[i];
      idx = locate(z, nz, zfov);
      for (id = 0; id < ctl->nd; id++) {
        obs->rad[id][ir] += w[i]
          * LIN(z[idx], rad[id][idx], z[idx + 1], rad[id][idx + 1], zfov);
        obs->tau[id][ir] += w[i]
          * LIN(z[idx], tau[id][idx], z[idx + 1], tau[id][idx + 1], zfov);
      }
      wsum += w[i];
    }
    for (id = 0; id < ctl->nd; id++) {
      obs->rad[id][ir] /= wsum;
      obs->tau[id][ir] /= wsum;
    }
  }

  /* Free... */
  free(obs2);
}
```

Here is the call graph for this function:

5.31.2.16 void formod_pencil (ctl_t * *ctl,* atm_t * *atm,* obs_t * *obs,* int *ir*)

Compute radiative transfer for a pencil beam.

Definition at line 3184 of file jurassic.c.

```
          {

  static tbl_t *tbl;

  static int init = 0;

  los_t *los;

  double beta_ctm[ND], eps, src_planck[ND], tau_path[NG][ND], tau_gas[ND];

  int id, ip;

  /* Initialize look-up tables... */
  if (!init) {
    init = 1;
    ALLOC(tbl, tbl_t, 1);
    init_tbl(ctl, tbl);
  }

  /* Allocate... */
  ALLOC(los, los_t, 1);

  /* Initialize... */
  for (id = 0; id < ctl->nd; id++) {
    obs->rad[id][ir] = 0;
    obs->tau[id][ir] = 1;
  }

  /* Raytracing... */
  raytrace(ctl, atm, obs, los, ir);

  /* Loop over LOS points... */
  for (ip = 0; ip < los->np; ip++) {

    /* Get trace gas transmittance... */
    intpol_tbl(ctl, tbl, los, ip, tau_path, tau_gas);

    /* Get continuum absorption... */
    formod_continua(ctl, los, ip, beta_ctm);

    /* Compute Planck function... */
    formod_srcfunc(ctl, tbl, los->t[ip], src_planck);

    /* Loop over channels... */
    for (id = 0; id < ctl->nd; id++)
      if (tau_gas[id] > 0) {

        /* Get segment emissivity... */
        eps = 1 - tau_gas[id] * exp(-beta_ctm[id] * los->ds[ip]);
```

```

      /* Compute radiance... */
      obs->rad[id][ir] += src_planck[id] * eps * obs->tau[id][ir];

      /* Compute path transmittance... */
      obs->tau[id][ir] *= (1 - eps);
    }
  }

  /* Add surface... */
  if (los->tsurf > 0) {
    formod_srcfunc(ctl, tbl, los->tsurf, src_planck);
    for (id = 0; id < ctl->nd; id++)
      obs->rad[id][ir] += src_planck[id] * obs->tau[id][ir];
  }

  /* Free... */
  free(los);
}
```

Here is the call graph for this function:

5.31.2.17 void formod_srcfunc (ctl_t * *ctl*, tbl_t * *tbl*, double *t*, double * *src*)

Compute Planck source function.

Definition at line 3259 of file jurassic.c.

```
                 {

  int id, it;

  /* Determine index in temperature array... */
  it = locate(tbl->st, TBLNS, t);

  /* Interpolate Planck function value... */
  for (id = 0; id < ctl->nd; id++)
    src[id] = LIN(tbl->st[it], tbl->sr[id][it],
                  tbl->st[it + 1], tbl->sr[id][it + 1], t);
}
```

Here is the call graph for this function:

5.31.2.18 void geo2cart (double *z*, double *lon*, double *lat*, double * *x*)

Convert geolocation to Cartesian coordinates.

Definition at line 3278 of file jurassic.c.

```
               {

  double radius;

  radius = z + RE;
  x[0] = radius * cos(lat / 180 * M_PI) * cos(lon / 180 * M_PI);
  x[1] = radius * cos(lat / 180 * M_PI) * sin(lon / 180 * M_PI);
  x[2] = radius * sin(lat / 180 * M_PI);
}
```

5.31.2.19 double gravity (double *z*, double *lat*)

Determine gravity of Earth.

Definition at line 3294 of file jurassic.c.

```
                {

  /* Compute gravity according to 1967 Geodetic Reference System... */
  return 9.780318 * (1 + 0.0053024 * gsl_pow_2(sin(lat / 180 * M_PI))
                     - 0.0000058 * gsl_pow_2(sin(2 * lat / 180 * M_PI))) -
    3.086e-3 * z;
}
```

5.31.2.20 void hydrostatic (ctl_t * ctl, atm_t * atm)

Set hydrostatic equilibrium.

Definition at line 3306 of file jurassic.c.

```
                 {

  static int ig_h2o = -999;

  double dzmin = 1e99, e = 0, mean, mmair = 28.96456e-3, mmh2o =
    18.0153e-3, z;

  int i, ip, ipref = 0, ipts = 20;

  /* Check reference height... */
  if (ctl->hydz < 0)
    return;

  /* Determine emitter index of H2O... */
  if (ig_h2o == -999)
    ig_h2o = find_emitter(ctl, "H2O");

  /* Find air parcel next to reference height... */
  for (ip = 0; ip < atm->np; ip++)
    if (fabs(atm->z[ip] - ctl->hydz) < dzmin) {
      dzmin = fabs(atm->z[ip] - ctl->hydz);
      ipref = ip;
    }

  /* Upper part of profile... */
  for (ip = ipref + 1; ip < atm->np; ip++) {
    mean = 0;
    for (i = 0; i < ipts; i++) {
      z = LIN(0.0, atm->z[ip - 1], ipts - 1.0, atm->z[ip], (double) i);
      if (ig_h2o >= 0)
        e = LIN(0.0, atm->q[ig_h2o][ip - 1],
                ipts - 1.0, atm->q[ig_h2o][ip], (double) i);
      mean += (e * mmh2o + (1 - e) * mmair)
        * gravity(z, atm->lat[ipref]) / GSL_CONST_MKSA_MOLAR_GAS
        / LIN(0.0, atm->t[ip - 1], ipts - 1.0, atm->t[ip], (double) i) / ipts;
    }

    /* Compute p(z,T)... */
    atm->p[ip] =
      exp(log(atm->p[ip - 1]) - mean * 1000 * (atm->z[ip] - atm->z[ip - 1]));
  }

  /* Lower part of profile... */
  for (ip = ipref - 1; ip >= 0; ip--) {
    mean = 0;
    for (i = 0; i < ipts; i++) {
      z = LIN(0.0, atm->z[ip + 1], ipts - 1.0, atm->z[ip], (double) i);
      if (ig_h2o >= 0)
        e = LIN(0.0, atm->q[ig_h2o][ip + 1],
                ipts - 1.0, atm->q[ig_h2o][ip], (double) i);
      mean += (e * mmh2o + (1 - e) * mmair)
        * gravity(z, atm->lat[ipref]) / GSL_CONST_MKSA_MOLAR_GAS
        / LIN(0.0, atm->t[ip + 1], ipts - 1.0, atm->t[ip], (double) i) / ipts;
    }

    /* Compute p(z,T)... */
    atm->p[ip] =
      exp(log(atm->p[ip + 1]) - mean * 1000 * (atm->z[ip] - atm->z[ip + 1]));
  }
}
```

Here is the call graph for this function:

5.31.2.21 void idx2name (ctl_t * ctl, int idx, char * quantity)

Determine name of state vector quantity for given index.

Definition at line 3371 of file jurassic.c.

```
                  {

  int ig, iw;

  if (idx == IDXP)
    sprintf(quantity, "PRESSURE");

  if (idx == IDXT)
    sprintf(quantity, "TEMPERATURE");

  for (ig = 0; ig < ctl->ng; ig++)
    if (idx == IDXQ(ig))
      sprintf(quantity, "%s", ctl->emitter[ig]);

  for (iw = 0; iw < ctl->nw; iw++)
    if (idx == IDXK(iw))
      sprintf(quantity, "EXTINCT_WINDOW%d", iw);
}
```

5.31.2.22 void init_tbl (ctl_t * ctl, tbl_t * tbl)

Initialize look-up tables.

Definition at line 3395 of file jurassic.c.

```
                {

  FILE *in;

  char filename[LEN], line[LEN];

  double eps, eps_old, press, press_old, temp, temp_old, u, u_old,
    f[NSHAPE], fsum, nu[NSHAPE];

  int i, id, ig, ip, it, n;

  /* Loop over trace gases and channels... */
  for (ig = 0; ig < ctl->ng; ig++)
#pragma omp parallel for default(none) shared(ctl,tbl,ig) private(in,filename,line,eps,eps_old,press,
      press_old,temp,temp_old,u,u_old,id,ip,it)
    for (id = 0; id < ctl->nd; id++) {

      /* Initialize... */
      tbl->np[ig][id] = -1;
      eps_old = -999;
      press_old = -999;
      temp_old = -999;
      u_old = -999;

      /* Try to open file... */
      sprintf(filename, "%s_%.4f_%s.tab",
              ctl->tblbase, ctl->nu[id], ctl->emitter[ig]);
      if (!(in = fopen(filename, "r"))) {
        printf("Missing emissivity table: %s\n", filename);
        continue;
      }
      printf("Read emissivity table: %s\n", filename);

      /* Read data... */
      while (fgets(line, LEN, in)) {

        /* Parse line... */
        if (sscanf(line, "%lg %lg %lg %lg", &press, &temp, &u, &eps) != 4)
          continue;

        /* Determine pressure index... */
        if (press != press_old) {
          press_old = press;
          if ((++tbl->np[ig][id]) >= TBLNP)
            ERRMSG("Too many pressure levels!");
          tbl->nt[ig][id][tbl->np[ig][id]] = -1;
        }

        /* Determine temperature index... */
        if (temp != temp_old) {
          temp_old = temp;
          if ((++tbl->nt[ig][id][tbl->np[ig][id]]) >= TBLNT)
            ERRMSG("Too many temperatures!");
          tbl->nu[ig][id][tbl->np[ig][id]]
            [tbl->nt[ig][id][tbl->np[ig][id]]] = -1;
```

```
        }

        /* Determine column density index... */
        if ((eps > eps_old && u > u_old) || tbl->nu[ig][id][tbl->np[ig][id]]
            [tbl->nt[ig][id][tbl->np[ig][id]]] < 0) {
          eps_old = eps;
          u_old = u;
          if ((++tbl->nu[ig][id][tbl->np[ig][id]]
               [tbl->nt[ig][id][tbl->np[ig][id]]]) >= TBLNU) {
            tbl->nu[ig][id][tbl->np[ig][id]]
              [tbl->nt[ig][id][tbl->np[ig][id]]]--;
            continue;
          }
        }

        /* Store data... */
        tbl->p[ig][id][tbl->np[ig][id]] = press;
        tbl->t[ig][id][tbl->np[ig][id]][tbl->nt[ig][id][tbl->np[ig][id]]]
          = temp;
        tbl->u[ig][id][tbl->np[ig][id]][tbl->nt[ig][id][tbl->np[ig][id]]]
          [tbl->nu[ig][id][tbl->np[ig][id]]
           [tbl->nt[ig][id][tbl->np[ig][id]]]] = (float) u;
        tbl->eps[ig][id][tbl->np[ig][id]][tbl->nt[ig][id][tbl->np[ig][id]]]
          [tbl->nu[ig][id][tbl->np[ig][id]]
           [tbl->nt[ig][id][tbl->np[ig][id]]]] = (float) eps;
      }

      /* Increment counters... */
      tbl->np[ig][id]++;
      for (ip = 0; ip < tbl->np[ig][id]; ip++) {
        tbl->nt[ig][id][ip]++;
        for (it = 0; it < tbl->nt[ig][id][ip]; it++)
          tbl->nu[ig][id][ip][it]++;
      }

      /* Close file... */
      fclose(in);
    }

  /* Write info... */
  printf("Initialize source function table...\n");

  /* Loop over channels... */
#pragma omp parallel for default(none) shared(ctl,tbl,ig) private(filename,it,i,n,f,fsum,nu)
  for (id = 0; id < ctl->nd; id++) {

    /* Read filter function... */
    sprintf(filename, "%s_%.4f.filt", ctl->tblbase, ctl->nu[id]);
    read_shape(filename, nu, f, &n);

    /* Compute source function table... */
    for (it = 0; it < TBLNS; it++) {

      /* Set temperature... */
      tbl->st[it] = LIN(0.0, TMIN, TBLNS - 1.0, TMAX, (double) it);

      /* Integrate Planck function... */
      fsum = 0;
      tbl->sr[id][it] = 0;
      for (i = 0; i < n; i++) {
        fsum += f[i];
        tbl->sr[id][it] += f[i] * planck(tbl->st[it], nu[i]);
      }
      tbl->sr[id][it] /= fsum;
    }
  }
}
```

Here is the call graph for this function:

5.31.2.23 void intpol_atm (ctl_t * *ctl*, atm_t * *atm*, double *z*, double * *p*, double * *t*, double * *q*, double * *k*)

Interpolate atmospheric data.

Definition at line 3521 of file jurassic.c.

```
                {

  int ig, ip, iw;
```

```

  /* Get array index... */
  ip = locate(atm->z, atm->np, z);

  /* Interpolate... */
  *p = EXP(atm->z[ip], atm->p[ip], atm->z[ip + 1], atm->p[ip + 1], z);
  *t = LIN(atm->z[ip], atm->t[ip], atm->z[ip + 1], atm->t[ip + 1], z);
  for (ig = 0; ig < ctl->ng; ig++)
    q[ig] =
      LIN(atm->z[ip], atm->q[ig][ip], atm->z[ip + 1], atm->q[ig][ip + 1], z);
  for (iw = 0; iw < ctl->nw; iw++)
    k[iw] =
      LIN(atm->z[ip], atm->k[iw][ip], atm->z[ip + 1], atm->k[iw][ip + 1], z);
}
```

Here is the call graph for this function:

5.31.2.24 void intpol_tbl (ctl_t * *ctl,* tbl_t * *tbl,* los_t * *los,* int *ip,* double *tau_path[NG][ND],* double *tau_seg[ND]*)

Get transmittance from look-up tables.

Definition at line 3548 of file jurassic.c.

```
                        {

  double eps, eps00, eps01, eps10, eps11, u;

  int id, ig, ipr, it0, it1;

  /* Initialize... */
  if (ip <= 0)
    for (ig = 0; ig < ctl->ng; ig++)
      for (id = 0; id < ctl->nd; id++)
        tau_path[ig][id] = 1;

  /* Loop over channels... */
  for (id = 0; id < ctl->nd; id++) {

    /* Initialize... */
    tau_seg[id] = 1;

    /* Loop over emitters.... */
    for (ig = 0; ig < ctl->ng; ig++) {

      /* Check size of table (pressure)... */
      if (tbl->np[ig][id] < 2)
        eps = 0;

      /* Check transmittance... */
      else if (tau_path[ig][id] < 1e-9)
        eps = 1;

      /* Interpolate... */
      else {

        /* Determine pressure and temperature indices... */
        ipr = locate(tbl->p[ig][id], tbl->np[ig][id], los->p[ip]);
        it0 = locate(tbl->t[ig][id][ipr], tbl->nt[ig][id][ipr], los->t[ip]);
        it1 =
          locate(tbl->t[ig][id][ipr + 1], tbl->nt[ig][id][ipr + 1],
                 los->t[ip]);

        /* Check size of table (temperature and column density)... */
        if (tbl->nt[ig][id][ipr] < 2 || tbl->nt[ig][id][ipr + 1] < 2
            || tbl->nu[ig][id][ipr][it0] < 2
            || tbl->nu[ig][id][ipr][it0 + 1] < 2
            || tbl->nu[ig][id][ipr + 1][it1] < 2
            || tbl->nu[ig][id][ipr + 1][it1 + 1] < 2)
          eps = 0;

        else {

          /* Get emissivities of extended path... */
          u = intpol_tbl_u(tbl, ig, id, ipr, it0, 1 - tau_path[ig][id]);
          eps00 = intpol_tbl_eps(tbl, ig, id, ipr, it0, u + los->u[ig][ip]);

          u = intpol_tbl_u(tbl, ig, id, ipr, it0 + 1, 1 - tau_path[ig][id]);
          eps01 =
            intpol_tbl_eps(tbl, ig, id, ipr, it0 + 1, u + los->u[ig][ip]);
```

```
          u = intpol_tbl_u(tbl, ig, id, ipr + 1, it1, 1 - tau_path[ig][id]);
          eps10 =
            intpol_tbl_eps(tbl, ig, id, ipr + 1, it1, u + los->u[ig][ip]);

          u =
            intpol_tbl_u(tbl, ig, id, ipr + 1, it1 + 1, 1 - tau_path[ig][id]);
          eps11 =
            intpol_tbl_eps(tbl, ig, id, ipr + 1, it1 + 1, u + los->
      u[ig][ip]);

          /* Interpolate with respect to temperature... */
          eps00 = LIN(tbl->t[ig][id][ipr][it0], eps00,
                      tbl->t[ig][id][ipr][it0 + 1], eps01, los->t[ip]);
          eps11 = LIN(tbl->t[ig][id][ipr + 1][it1], eps10,
                      tbl->t[ig][id][ipr + 1][it1 + 1], eps11, los->t[ip]);

          /* Interpolate with respect to pressure... */
          eps00 = LIN(tbl->p[ig][id][ipr], eps00,
                      tbl->p[ig][id][ipr + 1], eps11, los->p[ip]);

          /* Check emssivity range... */
          eps00 = GSL_MAX(GSL_MIN(eps00, 1), 0);

          /* Determine segment emissivity... */
          eps = 1 - (1 - eps00) / tau_path[ig][id];
        }
      }

      /* Get transmittance of extended path... */
      tau_path[ig][id] *= (1 - eps);

      /* Get segment transmittance... */
      tau_seg[id] *= (1 - eps);
    }
  }
}
```

Here is the call graph for this function:

5.31.2.25 double intpol_tbl_eps (tbl_t * *tbl,* int *ig,* int *id,* int *ip,* int *it,* double *u*)

Interpolate emissivity from look-up tables.

Definition at line 3649 of file jurassic.c.

```
              {

  int idx;

  /* Lower boundary... */
  if (u < tbl->u[ig][id][ip][it][0])
    return LIN(0, 0, tbl->u[ig][id][ip][it][0], tbl->eps[ig][id][ip][it][0],
               u);

  /* Upper boundary... */
  else if (u > tbl->u[ig][id][ip][it][tbl->nu[ig][id][ip][it] - 1])
    return LIN(tbl->u[ig][id][ip][it][tbl->nu[ig][id][ip][it] - 1],
               tbl->eps[ig][id][ip][it][tbl->nu[ig][id][ip][it] - 1],
               1e30, 1, u);

  /* Interpolation... */
  else {

    /* Get index... */
    idx = locate_tbl(tbl->u[ig][id][ip][it], tbl->nu[ig][id][ip][it], u);

    /* Interpolate... */
    return
      LIN(tbl->u[ig][id][ip][it][idx], tbl->eps[ig][id][ip][it][idx],
          tbl->u[ig][id][ip][it][idx + 1], tbl->eps[ig][id][ip][it][idx + 1],
          u);
  }
}
```

Here is the call graph for this function:

5.31.2.26 double intpol_tbl_u (**tbl_t** ∗ *tbl,* int *ig,* int *id,* int *ip,* int *it,* double *eps*)

Interpolate column density from look-up tables.

Definition at line 3686 of file jurassic.c.

```
                {

  int idx;

  /* Lower boundary... */
  if (eps < tbl->eps[ig][id][ip][it][0])
    return LIN(0, 0, tbl->eps[ig][id][ip][it][0], tbl->u[ig][id][ip][it][0],
               eps);

  /* Upper boundary... */
  else if (eps > tbl->eps[ig][id][ip][it][tbl->nu[ig][id][ip][it] - 1])
    return LIN(tbl->eps[ig][id][ip][it][tbl->nu[ig][id][ip][it] - 1],
               tbl->u[ig][id][ip][it][tbl->nu[ig][id][ip][it] - 1],
               1, 1e30, eps);

  /* Interpolation... */
  else {

    /* Get index... */
    idx = locate_tbl(tbl->eps[ig][id][ip][it], tbl->nu[ig][id][ip][it], eps);

    /* Interpolate... */
    return
      LIN(tbl->eps[ig][id][ip][it][idx], tbl->u[ig][id][ip][it][idx],
          tbl->eps[ig][id][ip][it][idx + 1], tbl->u[ig][id][ip][it][idx + 1],
          eps);
  }
}
```

Here is the call graph for this function:

5.31.2.27 void jsec2time (double *jsec,* int ∗ *year,* int ∗ *mon,* int ∗ *day,* int ∗ *hour,* int ∗ *min,* int ∗ *sec,* double ∗ *remain*)

Convert seconds to date.

Definition at line 3723 of file jurassic.c.

```
                    {

  struct tm t0, *t1;

  time_t jsec0;

  t0.tm_year = 100;
  t0.tm_mon = 0;
  t0.tm_mday = 1;
  t0.tm_hour = 0;
  t0.tm_min = 0;
  t0.tm_sec = 0;

  jsec0 = (time_t) jsec + timegm(&t0);
  t1 = gmtime(&jsec0);

  *year = t1->tm_year + 1900;
  *mon = t1->tm_mon + 1;
  *day = t1->tm_mday;
  *hour = t1->tm_hour;
  *min = t1->tm_min;
  *sec = t1->tm_sec;
  *remain = jsec - floor(jsec);
}
```

5.31.2.28 void kernel (ctl_t * ctl, atm_t * atm, obs_t * obs, gsl_matrix * k)

Compute Jacobians.

Definition at line 3758 of file jurassic.c.

```
                    {

  atm_t *atm1;
  obs_t *obs1;

  gsl_vector *x0, *x1, *yy0, *yy1;

  int *iqa, j;

  double h;

  size_t i, n, m;

  /* Get sizes... */
  m = k->size1;
  n = k->size2;

  /* Allocate... */
  x0 = gsl_vector_alloc(n);
  yy0 = gsl_vector_alloc(m);
  ALLOC(iqa, int,
        N);

  /* Compute radiance for undisturbed atmospheric data... */
  formod(ctl, atm, obs);

  /* Compose vectors... */
  atm2x(ctl, atm, x0, iqa, NULL);
  obs2y(ctl, obs, yy0, NULL, NULL);

  /* Initialize kernel matrix... */
  gsl_matrix_set_zero(k);

  /* Loop over state vector elements... */
#pragma omp parallel for default(none) shared(ctl,atm,obs,k,x0,yy0,n,m,iqa) private(i, j, h, x1, yy1, atm1,
 obs1)
  for (j = 0; j < (int) n; j++) {

    /* Allocate... */
    x1 = gsl_vector_alloc(n);
    yy1 = gsl_vector_alloc(m);
    ALLOC(atm1, atm_t, 1);
    ALLOC(obs1, obs_t, 1);

    /* Set perturbation size... */
    if (iqa[j] == IDXP)
      h = GSL_MAX(fabs(0.01 * gsl_vector_get(x0, (size_t) j)), 1e-7);
    else if (iqa[j] == IDXT)
      h = 1;
    else if (iqa[j] >= IDXQ(0) && iqa[j] < IDXQ(ctl->ng))
      h = GSL_MAX(fabs(0.01 * gsl_vector_get(x0, (size_t) j)), 1e-15);
    else if (iqa[j] >= IDXK(0) && iqa[j] < IDXK(ctl->nw))
      h = 1e-4;
    else
      ERRMSG("Cannot set perturbation size!");

    /* Disturb state vector element... */
    gsl_vector_memcpy(x1, x0);
    gsl_vector_set(x1, (size_t) j, gsl_vector_get(x1, (size_t) j) + h);
    copy_atm(ctl, atm1, atm, 0);
    copy_obs(ctl, obs1, obs, 0);
    x2atm(ctl, x1, atm1);

    /* Compute radiance for disturbed atmospheric data... */
    formod(ctl, atm1, obs1);

    /* Compose measurement vector for disturbed radiance data... */
    obs2y(ctl, obs1, yy1, NULL, NULL);

    /* Compute derivatives... */
    for (i = 0; i < m; i++)
      gsl_matrix_set(k, i, (size_t) j,
                     (gsl_vector_get(yy1, i) - gsl_vector_get(yy0, i)) / h);

    /* Free... */
    gsl_vector_free(x1);
    gsl_vector_free(yy1);
```

```
    free(atm1);
    free(obs1);
  }

  /* Free... */
  gsl_vector_free(x0);
  gsl_vector_free(yy0);
  free(iqa);
}
```

Here is the call graph for this function:

5.31.2.29 int locate (double * *xx*, int *n*, double *x*)

Find array index.

Definition at line 3850 of file jurassic.c.

```
             {

  int i, ilo, ihi;

  ilo = 0;
  ihi = n - 1;
  i = (ihi + ilo) >> 1;

  if (xx[i] < xx[i + 1])
    while (ihi > ilo + 1) {
      i = (ihi + ilo) >> 1;
      if (xx[i] > x)
        ihi = i;
      else
        ilo = i;
  } else
    while (ihi > ilo + 1) {
      i = (ihi + ilo) >> 1;
      if (xx[i] <= x)
        ihi = i;
      else
        ilo = i;
    }

  return ilo;
}
```

5.31.2.30 int locate_tbl (float * *xx*, int *n*, double *x*)

Find array index in float array.

Definition at line 3882 of file jurassic.c.

```
             {

  int i, ilo, ihi;

  ilo = 0;
  ihi = n - 1;
  i = (ihi + ilo) >> 1;

  while (ihi > ilo + 1) {
    i = (ihi + ilo) >> 1;
    if (xx[i] > x)
      ihi = i;
    else
      ilo = i;
  }

  return ilo;
}
```

5.31.2.31 size_t obs2y (ctl_t * *ctl*, obs_t * *obs*, gsl_vector * *y*, int * *ida*, int * *ira*)

Compose measurement vector.

Definition at line 3906 of file jurassic.c.

```
              {

  int id, ir;

  size_t m = 0;

  /* Determine measurement vector... */
  for (ir = 0; ir < obs->nr; ir++)
    for (id = 0; id < ctl->nd; id++)
      if (gsl_finite(obs->rad[id][ir])) {
        if (y != NULL)
          gsl_vector_set(y, m, obs->rad[id][ir]);
        if (ida != NULL)
          ida[m] = id;
        if (ira != NULL)
          ira[m] = ir;
        m++;
      }

  return m;
}
```

5.31.2.32 double planck (double *t*, double *nu*)

Compute Planck function.

Definition at line 3935 of file jurassic.c.

```
               {

  return C1 * gsl_pow_3(nu) / gsl_expm1(C2 * nu / t);
}
```

5.31.2.33 void raytrace (ctl_t * *ctl*, atm_t * *atm*, obs_t * *obs*, los_t * *los*, int *ir*)

Do ray-tracing to determine LOS.

Definition at line 3944 of file jurassic.c.

```
            {

  double cosa, d, dmax, dmin = 0, ds, ex0[3], ex1[3], frac, h = 0.02, k[NW],
    lat, lon, n, naux, ng[3], norm, p, q[NG], t, x[3], xh[3],
    xobs[3], xvp[3], z = 1e99, zmax, zmin, zrefrac = 60;

  int i, ig, ip, iw, stop = 0;

  /* Initialize... */
  los->np = 0;
  los->tsurf = -999;
  obs->tpz[ir] = obs->vpz[ir];
  obs->tplon[ir] = obs->vplon[ir];
  obs->tplat[ir] = obs->vplat[ir];

  /* Get altitude range of atmospheric data... */
  gsl_stats_minmax(&zmin, &zmax, atm->z, 1, (size_t) atm->np);

  /* Check observer altitude... */
  if (obs->obsz[ir] < zmin)
    ERRMSG("Observer below surface!");

  /* Check view point altitude... */
  if (obs->vpz[ir] > zmax)
    return;
```

```
  /* Determine Cartesian coordinates for observer and view point... */
  geo2cart(obs->obsz[ir], obs->obslon[ir], obs->obslat[ir], xobs);
  geo2cart(obs->vpz[ir], obs->vplon[ir], obs->vplat[ir], xvp);

  /* Determine initial tangent vector... */
  for (i = 0; i < 3; i++)
    ex0[i] = xvp[i] - xobs[i];
  norm = NORM(ex0);
  for (i = 0; i < 3; i++)
    ex0[i] /= norm;

  /* Observer within atmosphere... */
  for (i = 0; i < 3; i++)
    x[i] = xobs[i];

  /* Observer above atmosphere (search entry point)... */
  if (obs->obsz[ir] > zmax) {
    dmax = norm;
    while (fabs(dmin - dmax) > 0.001) {
      d = (dmax + dmin) / 2;
      for (i = 0; i < 3; i++)
        x[i] = xobs[i] + d * ex0[i];
      cart2geo(x, &z, &lon, &lat);
      if (z <= zmax && z > zmax - 0.001)
        break;
      if (z < zmax - 0.0005)
        dmax = d;
      else
        dmin = d;
    }
  }

  /* Ray-tracing... */
  while (1) {

    /* Set step length... */
    ds = ctl->rayds;
    if (ctl->raydz > 0) {
      norm = NORM(x);
      for (i = 0; i < 3; i++)
        xh[i] = x[i] / norm;
      cosa = fabs(DOTP(ex0, xh));
      if (cosa != 0)
        ds = GSL_MIN(ctl->rayds, ctl->raydz / cosa);
    }

    /* Determine geolocation... */
    cart2geo(x, &z, &lon, &lat);

    /* Check if LOS hits the ground or has left atmosphere... */
    if (z < zmin || z > zmax) {
      stop = (z < zmin ? 2 : 1);
      frac =
        ((z <
          zmin ? zmin : zmax) - los->z[los->np - 1]) / (z - los->z[los->np -
                                                                  1]);
      geo2cart(los->z[los->np - 1], los->lon[los->np - 1],
               los->lat[los->np - 1], xh);
      for (i = 0; i < 3; i++)
        x[i] = xh[i] + frac * (x[i] - xh[i]);
      cart2geo(x, &z, &lon, &lat);
      los->ds[los->np - 1] = ds * frac;
      ds = 0;
    }

    /* Interpolate atmospheric data... */
    intpol_atm(ctl, atm, z, &p, &t, q, k);

    /* Save data... */
    los->lon[los->np] = lon;
    los->lat[los->np] = lat;
    los->z[los->np] = z;
    los->p[los->np] = p;
    los->t[los->np] = t;
    for (ig = 0; ig < ctl->ng; ig++)
      los->q[ig][los->np] = q[ig];
    for (iw = 0; iw < ctl->nw; iw++)
      los->k[iw][los->np] = k[iw];
    los->ds[los->np] = ds;

    /* Increment and check number of LOS points... */
    if ((++los->np) > NLOS)
      ERRMSG("Too many LOS points!");

    /* Check stop flag... */
    if (stop) {
```

```
      los->tsurf = (stop == 2 ? t : -999);
      break;
    }

    /* Determine refractivity... */
    if (ctl->refrac && z <= zrefrac)
      n = 1 + refractivity(p, t);
    else
      n = 1;

    /* Construct new tangent vector (first term)... */
    for (i = 0; i < 3; i++)
      ex1[i] = ex0[i] * n;

    /* Compute gradient of refractivity... */
    if (ctl->refrac && z <= zrefrac) {
      for (i = 0; i < 3; i++)
        xh[i] = x[i] + 0.5 * ds * ex0[i];
      cart2geo(xh, &z, &lon, &lat);
      intpol_atm(ctl, atm, z, &p, &t, q, k);
      n = refractivity(p, t);
      for (i = 0; i < 3; i++) {
        xh[i] += h;
        cart2geo(xh, &z, &lon, &lat);
        intpol_atm(ctl, atm, z, &p, &t, q, k);
        naux = refractivity(p, t);
        ng[i] = (naux - n) / h;
        xh[i] -= h;
      }
    } else
      for (i = 0; i < 3; i++)
        ng[i] = 0;

    /* Construct new tangent vector (second term)... */
    for (i = 0; i < 3; i++)
      ex1[i] += ds * ng[i];

    /* Normalize new tangent vector... */
    norm = NORM(ex1);
    for (i = 0; i < 3; i++)
      ex1[i] /= norm;

    /* Determine next point of LOS... */
    for (i = 0; i < 3; i++)
      x[i] += 0.5 * ds * (ex0[i] + ex1[i]);

    /* Copy tangent vector... */
    for (i = 0; i < 3; i++)
      ex0[i] = ex1[i];
  }

  /* Get tangent point (to be done before changing segment lengths!)... */
  tangent_point(los, &obs->tpz[ir], &obs->tplon[ir], &obs->tplat[ir]);

  /* Change segment lengths according to trapezoid rule... */
  for (ip = los->np - 1; ip >= 1; ip--)
    los->ds[ip] = 0.5 * (los->ds[ip - 1] + los->ds[ip]);
  los->ds[0] *= 0.5;

  /* Compute column density... */
  for (ip = 0; ip < los->np; ip++)
    for (ig = 0; ig < ctl->ng; ig++)
      los->u[ig][ip] = 10 * los->q[ig][ip] * los->p[ip]
        / (GSL_CONST_MKSA_BOLTZMANN * los->t[ip]) * los->ds[ip];
}
```

Here is the call graph for this function:

5.31.2.34 void read_atm (const char * *dirname*, const char * *filename*, ctl_t * *ctl*, atm_t * *atm*)

Read atmospheric data.

Definition at line 4129 of file jurassic.c.

```
                  {

  FILE *in;

```

```
  char file[LEN], line[LEN], *tok;

  int ig, iw;

  /* Init... */
  atm->np = 0;

  /* Set filename... */
  if (dirname != NULL)
    sprintf(file, "%s/%s", dirname, filename);
  else
    sprintf(file, "%s", filename);

  /* Write info... */
  printf("Read atmospheric data: %s\n", file);

  /* Open file... */
  if (!(in = fopen(file, "r")))
    ERRMSG("Cannot open file!");

  /* Read line... */
  while (fgets(line, LEN, in)) {

    /* Read data... */
    TOK(line, tok, "%lg", atm->time[atm->np]);
    TOK(NULL, tok, "%lg", atm->z[atm->np]);
    TOK(NULL, tok, "%lg", atm->lon[atm->np]);
    TOK(NULL, tok, "%lg", atm->lat[atm->np]);
    TOK(NULL, tok, "%lg", atm->p[atm->np]);
    TOK(NULL, tok, "%lg", atm->t[atm->np]);
    for (ig = 0; ig < ctl->ng; ig++)
      TOK(NULL, tok, "%lg", atm->q[ig][atm->np]);
    for (iw = 0; iw < ctl->nw; iw++)
      TOK(NULL, tok, "%lg", atm->k[iw][atm->np]);

    /* Increment data point counter... */
    if ((++atm->np) > NP)
      ERRMSG("Too many data points!");
  }

  /* Close file... */
  fclose(in);

  /* Check number of points... */
  if (atm->np < 1)
    ERRMSG("Could not read any data!");
}
```

5.31.2.35 void read_ctl (int *argc*, char ∗ *argv[ ]*, ctl_t ∗ *ctl*)

Read forward model control parameters.

Definition at line 4187 of file jurassic.c.

```
                 {

  int id, ig, iw;

  /* Write info... */
  printf("\nJuelich Rapid Spectral Simulation Code (JURASSIC)\n"
         "(executable: %s | compiled: %s, %s)\n\n",
         argv[0], __DATE__, __TIME__);

  /* Emitters... */
  ctl->ng = (int) scan_ctl(argc, argv, "NG", -1, "0", NULL);
  if (ctl->ng < 0 || ctl->ng > NG)
    ERRMSG("Set 0 <= NG <= MAX!");
  for (ig = 0; ig < ctl->ng; ig++)
    scan_ctl(argc, argv, "EMITTER", ig, "", ctl->emitter[ig]);

  /* Radiance channels... */
  ctl->nd = (int) scan_ctl(argc, argv, "ND", -1, "0", NULL);
  if (ctl->nd < 0 || ctl->nd > ND)
    ERRMSG("Set 0 <= ND <= MAX!");
  for (id = 0; id < ctl->nd; id++)
    ctl->nu[id] = scan_ctl(argc, argv, "NU", id, "", NULL);

  /* Spectral windows... */
  ctl->nw = (int) scan_ctl(argc, argv, "NW", -1, "1", NULL);
  if (ctl->nw < 0 || ctl->nw > NW)
```

```
    ERRMSG("Set 0 <= NW <= MAX!");
  for (id = 0; id < ctl->nd; id++)
    ctl->window[id] = (int) scan_ctl(argc, argv, "WINDOW", id, "0", NULL);

  /* Emissivity look-up tables... */
  scan_ctl(argc, argv, "TBLBASE", -1, "-", ctl->tblbase);

  /* Hydrostatic equilibrium... */
  ctl->hydz = scan_ctl(argc, argv, "HYDZ", -1, "-999", NULL);

  /* Continua... */
  ctl->ctm_co2 = (int) scan_ctl(argc, argv, "CTM_CO2", -1, "1", NULL);
  ctl->ctm_h2o = (int) scan_ctl(argc, argv, "CTM_H2O", -1, "1", NULL);
  ctl->ctm_n2 = (int) scan_ctl(argc, argv, "CTM_N2", -1, "1", NULL);
  ctl->ctm_o2 = (int) scan_ctl(argc, argv, "CTM_O2", -1, "1", NULL);

  /* Ray-tracing... */
  ctl->refrac = (int) scan_ctl(argc, argv, "REFRAC", -1, "1", NULL);
  ctl->rayds = scan_ctl(argc, argv, "RAYDS", -1, "10", NULL);
  ctl->raydz = scan_ctl(argc, argv, "RAYDZ", -1, "0.5", NULL);

  /* Field of view... */
  scan_ctl(argc, argv, "FOV", -1, "-", ctl->fov);

  /* Retrieval interface... */
  ctl->retp_zmin = scan_ctl(argc, argv, "RETP_ZMIN", -1, "-999", NULL);
  ctl->retp_zmax = scan_ctl(argc, argv, "RETP_ZMAX", -1, "-999", NULL);
  ctl->rett_zmin = scan_ctl(argc, argv, "RETT_ZMIN", -1, "-999", NULL);
  ctl->rett_zmax = scan_ctl(argc, argv, "RETT_ZMAX", -1, "-999", NULL);
  for (ig = 0; ig < ctl->ng; ig++) {
    ctl->retq_zmin[ig] = scan_ctl(argc, argv, "RETQ_ZMIN", ig, "-999", NULL);
    ctl->retq_zmax[ig] = scan_ctl(argc, argv, "RETQ_ZMAX", ig, "-999", NULL);
  }
  for (iw = 0; iw < ctl->nw; iw++) {
    ctl->retk_zmin[iw] = scan_ctl(argc, argv, "RETK_ZMIN", iw, "-999", NULL);
    ctl->retk_zmax[iw] = scan_ctl(argc, argv, "RETK_ZMAX", iw, "-999", NULL);
  }

  /* Output flags... */
  ctl->write_bbt = (int) scan_ctl(argc, argv, "WRITE_BBT", -1, "0", NULL);
  ctl->write_matrix =
    (int) scan_ctl(argc, argv, "WRITE_MATRIX", -1, "0", NULL);
}
```

Here is the call graph for this function:

5.31.2.36 void read_matrix (const char ∗ *dirname*, const char ∗ *filename*, gsl_matrix ∗ *matrix*)

Read matrix.

Definition at line 4262 of file jurassic.c.

```
                       {

  FILE *in;

  char dum[LEN], file[LEN], line[LEN];

  double value;

  int i, j;

  /* Set filename... */
  if (dirname != NULL)
    sprintf(file, "%s/%s", dirname, filename);
  else
    sprintf(file, "%s", filename);

  /* Write info... */
  printf("Read matrix: %s\n", file);

  /* Open file... */
  if (!(in = fopen(file, "r")))
    ERRMSG("Cannot open file!");

  /* Read data... */
  gsl_matrix_set_zero(matrix);
  while (fgets(line, LEN, in))
    if (sscanf(line, "%d %s %s %s %s %s %d %s %s %s %s %s %lg",
```

```
                &i, dum, dum, dum, dum, dum,
                &j, dum, dum, dum, dum, dum, &value) == 13)
        gsl_matrix_set(matrix, (size_t) i, (size_t) j, value);

    /* Close file... */
    fclose(in);
}
```

5.31.2.37 void read_obs (const char ∗ *dirname,* const char ∗ *filename,* ctl_t ∗ *ctl,* obs_t ∗ *obs*)

Read observation data.

Definition at line 4302 of file jurassic.c.

```
                {

    FILE *in;

    char file[LEN], line[LEN], *tok;

    int id;

    /* Init... */
    obs->nr = 0;

    /* Set filename... */
    if (dirname != NULL)
      sprintf(file, "%s/%s", dirname, filename);
    else
      sprintf(file, "%s", filename);

    /* Write info... */
    printf("Read observation data: %s\n", file);

    /* Open file... */
    if (!(in = fopen(file, "r")))
      ERRMSG("Cannot open file!");

    /* Read line... */
    while (fgets(line, LEN, in)) {

      /* Read data... */
      TOK(line, tok, "%lg", obs->time[obs->nr]);
      TOK(NULL, tok, "%lg", obs->obsz[obs->nr]);
      TOK(NULL, tok, "%lg", obs->obslon[obs->nr]);
      TOK(NULL, tok, "%lg", obs->obslat[obs->nr]);
      TOK(NULL, tok, "%lg", obs->vpz[obs->nr]);
      TOK(NULL, tok, "%lg", obs->vplon[obs->nr]);
      TOK(NULL, tok, "%lg", obs->vplat[obs->nr]);
      TOK(NULL, tok, "%lg", obs->tpz[obs->nr]);
      TOK(NULL, tok, "%lg", obs->tplon[obs->nr]);
      TOK(NULL, tok, "%lg", obs->tplat[obs->nr]);
      for (id = 0; id < ctl->nd; id++)
        TOK(NULL, tok, "%lg", obs->rad[id][obs->nr]);
      for (id = 0; id < ctl->nd; id++)
        TOK(NULL, tok, "%lg", obs->tau[id][obs->nr]);

      /* Increment counter... */
      if ((++obs->nr) > NR)
        ERRMSG("Too many rays!");
    }

    /* Close file... */
    fclose(in);

    /* Check number of points... */
    if (obs->nr < 1)
      ERRMSG("Could not read any data!");
}
```

5.31.2.38 void read_shape (const char ∗ *filename,* double ∗ *x,* double ∗ *y,* int ∗ *n*)

Read shape function.

Definition at line 4364 of file jurassic.c.

```
        {

  FILE *in;

  char line[LEN];

  /* Write info... */
  printf("Read shape function: %s\n", filename);

  /* Open file... */
  if (!(in = fopen(filename, "r")))
    ERRMSG("Cannot open file!");

  /* Read data... */
  *n = 0;
  while (fgets(line, LEN, in))
    if (sscanf(line, "%lg %lg", &x[*n], &y[*n]) == 2)
      if ((++(*n)) > NSHAPE)
        ERRMSG("Too many data points!");

  /* Check number of points... */
  if (*n < 1)
    ERRMSG("Could not read any data!");

  /* Close file... */
  fclose(in);
}
```

5.31.2.39 double refractivity (double *p*, double *t*)

Compute refractivity (return value is n - 1).

Definition at line 4398 of file jurassic.c.

```
          {

  /* Refractivity of air at 4 to 15 micron... */
  return 7.753e-05 * p / t;
}
```

5.31.2.40 double scan_ctl (int *argc*, char ∗ *argv[ ]*, const char ∗ *varname*, int *arridx*, const char ∗ *defvalue*, char ∗ *value*)

Search control parameter file for variable entry.

Definition at line 4408 of file jurassic.c.

```
             {

  FILE *in = NULL;

  char dummy[LEN], fullname1[LEN], fullname2[LEN], line[LEN],
    msg[LEN], rvarname[LEN], rval[LEN];

  int contain = 0, i;

  /* Open file... */
  if (argv[1][0] != '-')
    if (!(in = fopen(argv[1], "r")))
      ERRMSG("Cannot open file!");

  /* Set full variable name... */
  if (arridx >= 0) {
    sprintf(fullname1, "%s[%d]", varname, arridx);
    sprintf(fullname2, "%s[*]", varname);
  } else {
    sprintf(fullname1, "%s", varname);
    sprintf(fullname2, "%s", varname);
  }

  /* Read data... */
  if (in != NULL)
    while (fgets(line, LEN, in))
      if (sscanf(line, "%s %s %s", rvarname, dummy, rval) == 3)
        if (strcasecmp(rvarname, fullname1) == 0 ||
```

```
            strcasecmp(rvarname, fullname2) == 0) {
          contain = 1;
          break;
        }
  for (i = 1; i < argc - 1; i++)
    if (strcasecmp(argv[i], fullname1) == 0 ||
        strcasecmp(argv[i], fullname2) == 0) {
      sprintf(rval, "%s", argv[i + 1]);
      contain = 1;
      break;
    }

  /* Close file... */
  if (in != NULL)
    fclose(in);

  /* Check for missing variables... */
  if (!contain) {
    if (strlen(defvalue) > 0)
      sprintf(rval, "%s", defvalue);
    else {
      sprintf(msg, "Missing variable %s!\n", fullname1);
      ERRMSG(msg);
    }
  }

  /* Write info... */
  printf("%s = %s\n", fullname1, rval);

  /* Return values... */
  if (value != NULL)
    sprintf(value, "%s", rval);
  return atof(rval);
}
```

5.31.2.41 void tangent_point (los_t ∗ *los,* double ∗ *tpz,* double ∗ *tplon,* double ∗ *tplat*)

Find tangent point of a given LOS.

Definition at line 4479 of file jurassic.c.

```
                  {

  double a, b, c, dummy, v[3], v0[3], v2[3], x, x1, x2, yy0, yy1, yy2;

  size_t i, ip;

  /* Find minimum altitude... */
  ip = gsl_stats_min_index(los->z, 1, (size_t) los->np);

  /* Nadir or zenith... */
  if (ip <= 0 || ip >= (size_t) los->np - 1) {
    *tpz = los->z[los->np - 1];
    *tplon = los->lon[los->np - 1];
    *tplat = los->lat[los->np - 1];
  }

  /* Limb... */
  else {

    /* Determine interpolating polynomial y=a*x^2+b*x+c... */
    yy0 = los->z[ip - 1];
    yy1 = los->z[ip];
    yy2 = los->z[ip + 1];
    x1 = sqrt(gsl_pow_2(los->ds[ip]) - gsl_pow_2(yy1 - yy0));
    x2 = x1 + sqrt(gsl_pow_2(los->ds[ip + 1]) - gsl_pow_2(yy2 - yy1));
    a = 1 / (x1 - x2) * (-(yy0 - yy1) / x1 + (yy0 - yy2) / x2);
    b = -(yy0 - yy1) / x1 - a * x1;
    c = yy0;

    /* Get tangent point location... */
    x = -b / (2 * a);
    *tpz = a * x * x + b * x + c;
    geo2cart(los->z[ip - 1], los->lon[ip - 1], los->lat[ip - 1], v0);
    geo2cart(los->z[ip + 1], los->lon[ip + 1], los->lat[ip + 1], v2);
    for (i = 0; i < 3; i++)
      v[i] = LIN(0.0, v0[i], x2, v2[i], x);
    cart2geo(v, &dummy, tplon, tplat);
  }
}
```

Here is the call graph for this function:

5.31.2.42 void time2jsec (int *year,* int *mon,* int *day,* int *hour,* int *min,* int *sec,* double *remain,* double * *jsec*)

Convert date to seconds.

Definition at line 4525 of file jurassic.c.

```
                  {

  struct tm t0, t1;

  t0.tm_year = 100;
  t0.tm_mon = 0;
  t0.tm_mday = 1;
  t0.tm_hour = 0;
  t0.tm_min = 0;
  t0.tm_sec = 0;

  t1.tm_year = year - 1900;
  t1.tm_mon = mon - 1;
  t1.tm_mday = day;
  t1.tm_hour = hour;
  t1.tm_min = min;
  t1.tm_sec = sec;

  *jsec = (double) timegm(&t1) - (double) timegm(&t0) + remain;
}
```

5.31.2.43 void timer (const char * *name,* const char * *file,* const char * *func,* int *line,* int *mode*)

Measure wall-clock time.

Definition at line 4556 of file jurassic.c.

```
              {

  static double dt_w, w0[10];

  static int l0[10], nt;

  struct timeval tim;

  /* Start new timer... */
  if (mode == 1) {
    gettimeofday(&tim, NULL);
    w0[nt] = (double) tim.tv_sec + (double) tim.tv_usec / 1e6;
    l0[nt] = line;
    if ((++nt) >= 10)
      ERRMSG("Too many timers!");
  }

  /* Write elapsed time... */
  else {

    /* Check timer index... */
    if (nt - 1 < 0)
      ERRMSG("Coding error!");

    /* Get time differences... */
    gettimeofday(&tim, NULL);
    dt_w = (double) tim.tv_sec + (double) tim.tv_usec / 1e6 - w0[nt - 1];

    /* Write elapsed time... */
    printf("Timer '%s' (%s, %s, l%d-%d): %.3f sec\n",
           name, file, func, l0[nt - 1], line, dt_w);
  }

  /* Stop timer... */
  if (mode == 3)
    nt--;
}
```

5.31.2.44 void write_atm (const char * *dirname*, const char * *filename*, ctl_t * *ctl*, atm_t * *atm*)

Write atmospheric data.

Definition at line 4601 of file jurassic.c.

```
               {

  FILE *out;

  char file[LEN];

  int ig, ip, iw, n = 6;

  /* Set filename... */
  if (dirname != NULL)
    sprintf(file, "%s/%s", dirname, filename);
  else
    sprintf(file, "%s", filename);

  /* Write info... */
  printf("Write atmospheric data: %s\n", file);

  /* Create file... */
  if (!(out = fopen(file, "w")))
    ERRMSG("Cannot create file!");

  /* Write header... */
  fprintf(out,
          "# $1 = time (seconds since 2000-01-01T00:00Z)\n"
          "# $2 = altitude [km]\n"
          "# $3 = longitude [deg]\n"
          "# $4 = latitude [deg]\n"
          "# $5 = pressure [hPa]\n" "# $6 = temperature [K]\n");
  for (ig = 0; ig < ctl->ng; ig++)
    fprintf(out, "# $%d = %s volume mixing ratio\n", ++n, ctl->emitter[ig]);
  for (iw = 0; iw < ctl->nw; iw++)
    fprintf(out, "# $%d = window %d: extinction [1/km]\n", ++n, iw);

  /* Write data... */
  for (ip = 0; ip < atm->np; ip++) {
    if (ip == 0 || atm->lat[ip] != atm->lat[ip - 1]
        || atm->lon[ip] != atm->lon[ip - 1])
      fprintf(out, "\n");
    fprintf(out, "%.2f %g %g %g %g %g", atm->time[ip], atm->z[ip],
            atm->lon[ip], atm->lat[ip], atm->p[ip], atm->t[ip]);
    for (ig = 0; ig < ctl->ng; ig++)
      fprintf(out, " %g", atm->q[ig][ip]);
    for (iw = 0; iw < ctl->nw; iw++)
      fprintf(out, " %g", atm->k[iw][ip]);
    fprintf(out, "\n");
  }

  /* Close file... */
  fclose(out);
}
```

5.31.2.45 void write_matrix (const char * *dirname*, const char * *filename*, ctl_t * *ctl*, gsl_matrix * *matrix*, atm_t * *atm*, obs_t * *obs*, const char * *rowspace*, const char * *colspace*, const char * *sort*)

Write matrix.

Definition at line 4658 of file jurassic.c.

```
                    {

  FILE *out;

  char file[LEN], quantity[LEN];

  int *cida, *ciqa, *cipa, *cira, *rida, *riqa, *ripa, *rira;

  size_t i, j, nc, nr;

  /* Check output flag... */
  if (!ctl->write_matrix)
```

```
    return;

  /* Allocate... */
  ALLOC(cida, int, M);
  ALLOC(ciqa, int,
        N);
  ALLOC(cipa, int,
        N);
  ALLOC(cira, int,
        M);
  ALLOC(rida, int,
        M);
  ALLOC(riqa, int,
        N);
  ALLOC(ripa, int,
        N);
  ALLOC(rira, int,
        M);

  /* Set filename... */
  if (dirname != NULL)
    sprintf(file, "%s/%s", dirname, filename);
  else
    sprintf(file, "%s", filename);

  /* Write info... */
  printf("Write matrix: %s\n", file);

  /* Create file... */
  if (!(out = fopen(file, "w")))
    ERRMSG("Cannot create file!");

  /* Write header (row space)... */
  if (rowspace[0] == 'y') {

    fprintf(out,
            "# $1 = Row: index (measurement space)\n"
            "# $2 = Row: channel wavenumber [cm^-1]\n"
            "# $3 = Row: time (seconds since 2000-01-01T00:00Z)\n"
            "# $4 = Row: view point altitude [km]\n"
            "# $5 = Row: view point longitude [deg]\n"
            "# $6 = Row: view point latitude [deg]\n");

    /* Get number of rows... */
    nr = obs2y(ctl, obs, NULL, rida, rira);

  } else {

    fprintf(out,
            "# $1 = Row: index (state space)\n"
            "# $2 = Row: name of quantity\n"
            "# $3 = Row: time (seconds since 2000-01-01T00:00Z)\n"
            "# $4 = Row: altitude [km]\n"
            "# $5 = Row: longitude [deg]\n" "# $6 = Row: latitude [deg]\n");

    /* Get number of rows... */
    nr = atm2x(ctl, atm, NULL, riqa, ripa);
  }

  /* Write header (column space)... */
  if (colspace[0] == 'y') {

    fprintf(out,
            "# $7 = Col: index (measurement space)\n"
            "# $8 = Col: channel wavenumber [cm^-1]\n"
            "# $9 = Col: time (seconds since 2000-01-01T00:00Z)\n"
            "# $10 = Col: view point altitude [km]\n"
            "# $11 = Col: view point longitude [deg]\n"
            "# $12 = Col: view point latitude [deg]\n");

    /* Get number of columns... */
    nc = obs2y(ctl, obs, NULL, cida, cira);

  } else {

    fprintf(out,
            "# $7 = Col: index (state space)\n"
            "# $8 = Col: name of quantity\n"
            "# $9 = Col: time (seconds since 2000-01-01T00:00Z)\n"
            "# $10 = Col: altitude [km]\n"
            "# $11 = Col: longitude [deg]\n" "# $12 = Col: latitude [deg]\n");

    /* Get number of columns... */
    nc = atm2x(ctl, atm, NULL, ciqa, cipa);
  }

  /* Write header entry... */
```

```
  fprintf(out, "# $13 = Matrix element\n\n");

  /* Write matrix data... */
  i = j = 0;
  while (i < nr && j < nc) {

    /* Write info about the row... */
    if (rowspace[0] == 'y')
      fprintf(out, "%d %g %.2f %g %g %g",
              (int) i, ctl->nu[rida[i]],
              obs->time[rira[i]], obs->vpz[rira[i]],
              obs->vplon[rira[i]], obs->vplat[rira[i]]);
    else {
      idx2name(ctl, riqa[i], quantity);
      fprintf(out, "%d %s %.2f %g %g %g", (int) i, quantity,
              atm->time[ripa[i]], atm->z[ripa[i]],
              atm->lon[ripa[i]], atm->lat[ripa[i]]);
    }

    /* Write info about the column... */
    if (colspace[0] == 'y')
      fprintf(out, " %d %g %.2f %g %g %g",
              (int) j, ctl->nu[cida[j]],
              obs->time[cira[j]], obs->vpz[cira[j]],
              obs->vplon[cira[j]], obs->vplat[cira[j]]);
    else {
      idx2name(ctl, ciqa[j], quantity);
      fprintf(out, " %d %s %.2f %g %g %g", (int) j, quantity,
              atm->time[cipa[j]], atm->z[cipa[j]],
              atm->lon[cipa[j]], atm->lat[cipa[j]]);
    }

    /* Write matrix entry... */
    fprintf(out, " %g\n", gsl_matrix_get(matrix, i, j));

    /* Set matrix indices... */
    if (sort[0] == 'r') {
      j++;
      if (j >= nc) {
        j = 0;
        i++;
        fprintf(out, "\n");
      }
    } else {
      i++;
      if (i >= nr) {
        i = 0;
        j++;
        fprintf(out, "\n");
      }
    }
  }

  /* Close file... */
  fclose(out);

  /* Free... */
  free(cida);
  free(ciqa);
  free(cipa);
  free(cira);
  free(rida);
  free(riqa);
  free(ripa);
  free(rira);
}
```

Here is the call graph for this function:

5.31.2.46 void write_obs (const char * *dirname*, const char * *filename*, ctl_t * *ctl*, obs_t * *obs*)

Write observation data.

Definition at line 4835 of file jurassic.c.

```
                 {

  FILE *out;

```

```
  char file[LEN];

  int id, ir, n = 10;

  /* Set filename... */
  if (dirname != NULL)
    sprintf(file, "%s/%s", dirname, filename);
  else
    sprintf(file, "%s", filename);

  /* Write info... */
  printf("Write observation data: %s\n", file);

  /* Create file... */
  if (!(out = fopen(file, "w")))
    ERRMSG("Cannot create file!");

  /* Write header... */
  fprintf(out,
          "# $1 = time (seconds since 2000-01-01T00:00Z)\n"
          "# $2 = observer altitude [km]\n"
          "# $3 = observer longitude [deg]\n"
          "# $4 = observer latitude [deg]\n"
          "# $5 = view point altitude [km]\n"
          "# $6 = view point longitude [deg]\n"
          "# $7 = view point latitude [deg]\n"
          "# $8 = tangent point altitude [km]\n"
          "# $9 = tangent point longitude [deg]\n"
          "# $10 = tangent point latitude [deg]\n");
  for (id = 0; id < ctl->nd; id++)
    fprintf(out, "# $%d = channel %g: radiance [W/(m^2 sr cm^-1)]\n",
            ++n, ctl->nu[id]);
  for (id = 0; id < ctl->nd; id++)
    fprintf(out, "# $%d = channel %g: transmittance\n", ++n, ctl->nu[id]);

  /* Write data... */
  for (ir = 0; ir < obs->nr; ir++) {
    if (ir == 0 || obs->time[ir] != obs->time[ir - 1])
      fprintf(out, "\n");
    fprintf(out, "%.2f %g %g %g %g %g %g %g %g %g", obs->time[ir],
            obs->obsz[ir], obs->obslon[ir], obs->obslat[ir],
            obs->vpz[ir], obs->vplon[ir], obs->vplat[ir],
            obs->tpz[ir], obs->tplon[ir], obs->tplat[ir]);
    for (id = 0; id < ctl->nd; id++)
      fprintf(out, " %g", obs->rad[id][ir]);
    for (id = 0; id < ctl->nd; id++)
      fprintf(out, " %g", obs->tau[id][ir]);
    fprintf(out, "\n");
  }

  /* Close file... */
  fclose(out);
}
```

5.31.2.47 void x2atm (ctl_t * *ctl*, gsl_vector * *x*, atm_t * *atm*)

Decompose parameter vector or state vector.

Definition at line 4899 of file jurassic.c.

```
                 {

  int ig, iw;

  size_t n = 0;

  /* Set pressure... */
  x2atm_help(atm, ctl->retp_zmin, ctl->retp_zmax, atm->
p, x, &n);

  /* Set temperature... */
  x2atm_help(atm, ctl->rett_zmin, ctl->rett_zmax, atm->
t, x, &n);

  /* Set volume mixing ratio... */
  for (ig = 0; ig < ctl->ng; ig++)
    x2atm_help(atm, ctl->retq_zmin[ig], ctl->retq_zmax[ig],
               atm->q[ig], x, &n);

  /* Set extinction... */
  for (iw = 0; iw < ctl->nw; iw++)
    x2atm_help(atm, ctl->retk_zmin[iw], ctl->retk_zmax[iw],
               atm->k[iw], x, &n);
}
```

Here is the call graph for this function:

5.31.2.48 void x2atm_help (atm_t ∗ *atm*, double *zmin*, double *zmax*, double ∗ *value*, gsl_vector ∗ *x*, size_t ∗ *n*)

Extract elements from state vector.

Definition at line 4927 of file jurassic.c.

```
                 {

  int ip;

  /* Extract state vector elements... */
  for (ip = 0; ip < atm->np; ip++)
    if (atm->z[ip] >= zmin && atm->z[ip] <= zmax) {
      value[ip] = gsl_vector_get(x, *n);
      (*n)++;
    }
}
```

5.31.2.49 void y2obs (ctl_t ∗ *ctl*, gsl_vector ∗ *y*, obs_t ∗ *obs*)

Decompose measurement vector.

Definition at line 4947 of file jurassic.c.

```
                  {

  int id, ir;

  size_t m = 0;

  /* Decompose measurement vector... */
  for (ir = 0; ir < obs->nr; ir++)
    for (id = 0; id < ctl->nd; id++)
      if (gsl_finite(obs->rad[id][ir])) {
        obs->rad[id][ir] = gsl_vector_get(y, m);
        m++;
      }
}
```

5.32 jurassic.c

```
/*
  This file is part of JURASSIC.

  JURASSIC is free software: you can redistribute it and/or modify
  it under the terms of the GNU General Public License as published by
  the Free Software Foundation, either version 3 of the License, or
  (at your option) any later version.

  JURASSIC is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
  GNU General Public License for more details.

  You should have received a copy of the GNU General Public License
  along with JURASSIC. If not, see <http://www.gnu.org/licenses/>.

  Copright (C) 2003-2015 Forschungszentrum Juelich GmbH
*/

#include "jurassic.h"

/*****************************************************************************/

size_t atm2x(
  ctl_t * ctl,
  atm_t * atm,
  gsl_vector * x,
```

```
  int *iqa,
  int *ipa) {

  int ig, iw;

  size_t n = 0;

  /* Add pressure... */
  atm2x_help(atm, ctl->retp_zmin, ctl->retp_zmax,
             atm->p, IDXP, x, iqa, ipa, &n);

  /* Add temperature... */
  atm2x_help(atm, ctl->rett_zmin, ctl->rett_zmax,
             atm->t, IDXT, x, iqa, ipa, &n);

  /* Add volume mixing ratios... */
  for (ig = 0; ig < ctl->ng; ig++)
    atm2x_help(atm, ctl->retq_zmin[ig], ctl->retq_zmax[ig],
               atm->q[ig], IDXQ(ig), x, iqa, ipa, &n);

  /* Add extinction... */
  for (iw = 0; iw < ctl->nw; iw++)
    atm2x_help(atm, ctl->retk_zmin[iw], ctl->retk_zmax[iw],
               atm->k[iw], IDXK(iw), x, iqa, ipa, &n);

  return n;
}

/*****************************************************************************/

void atm2x_help(
  atm_t * atm,
  double zmin,
  double zmax,
  double *value,
  int val_iqa,
  gsl_vector * x,
  int *iqa,
  int *ipa,
  size_t * n) {

  int ip;

  /* Add elements to state vector... */
  for (ip = 0; ip < atm->np; ip++)
    if (atm->z[ip] >= zmin && atm->z[ip] <= zmax) {
      if (x != NULL)
        gsl_vector_set(x, *n, value[ip]);
      if (iqa != NULL)
        iqa[*n] = val_iqa;
      if (ipa != NULL)
        ipa[*n] = ip;
      (*n)++;
    }
}

/*****************************************************************************/

double brightness(
  double rad,
  double nu) {

  return C2 * nu / gsl_log1p(C1 * gsl_pow_3(nu) / rad);
}


/*****************************************************************************/

void cart2geo(
  double *x,
  double *z,
  double *lon,
  double *lat) {

  double radius;

  radius = NORM(x);
  *lat = asin(x[2] / radius) * 180 / M_PI;
  *lon = atan2(x[1], x[0]) * 180 / M_PI;
  *z = radius - RE;
}

/*****************************************************************************/

void climatology(
  ctl_t * ctl,
  atm_t * atm) {
```

```
static double z[121] = {
  0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19,
  20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37,
  38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55,
  56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73,
  74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91,
  92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107,
  108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 120
};

static double pre[121] = {
  1017, 901.083, 796.45, 702.227, 617.614, 541.644, 473.437, 412.288,
  357.603, 308.96, 265.994, 228.348, 195.619, 167.351, 143.039, 122.198,
  104.369, 89.141, 76.1528, 65.0804, 55.641, 47.591, 40.7233, 34.8637,
  29.8633, 25.5956, 21.9534, 18.8445, 16.1909, 13.9258, 11.9913,
  10.34, 8.92988, 7.72454, 6.6924, 5.80701, 5.04654, 4.39238, 3.82902,
  3.34337, 2.92413, 2.56128, 2.2464, 1.97258, 1.73384, 1.52519, 1.34242,
  1.18197, 1.04086, 0.916546, 0.806832, 0.709875, 0.624101, 0.548176,
  0.480974, 0.421507, 0.368904, 0.322408, 0.281386, 0.245249, 0.213465,
  0.185549, 0.161072, 0.139644, 0.120913, 0.104568, 0.0903249, 0.0779269,
  0.0671493, 0.0577962, 0.0496902, 0.0426736, 0.0366093, 0.0313743,
  0.0268598, 0.0229699, 0.0196206, 0.0167399, 0.0142646, 0.0121397,
  0.0103181, 0.00875775, 0.00742226, 0.00628076, 0.00530519, 0.00447183,
  0.00376124, 0.00315632, 0.00264248, 0.00220738, 0.00184003, 0.00153095,
  0.00127204, 0.00105608, 0.000876652, 0.00072798, 0.00060492,
  0.000503201, 0.000419226, 0.000349896, 0.000292659, 0.000245421,
  0.000206394, 0.000174125, 0.000147441, 0.000125333, 0.000106985,
  9.173e-05, 7.90172e-05, 6.84172e-05, 5.95574e-05, 5.21183e-05,
  4.58348e-05, 4.05127e-05, 3.59987e-05, 3.21583e-05, 2.88718e-05,
  2.60322e-05, 2.35687e-05, 2.14263e-05, 1.95489e-05
};

static double tem[121] = {
  285.14, 279.34, 273.91, 268.3, 263.24, 256.55, 250.2, 242.82, 236.17,
  229.87, 225.04, 221.19, 218.85, 217.19, 216.2, 215.68, 215.42, 215.55,
  215.92, 216.4, 216.93, 217.45, 218, 218.68, 219.39, 220.25, 221.3,
  222.41, 223.88, 225.42, 227.2, 229.52, 231.89, 234.51, 236.85, 239.42,
  241.94, 244.57, 247.36, 250.32, 253.34, 255.82, 258.27, 260.39,
  262.03, 263.45, 264.2, 264.78, 264.67, 264.38, 263.24, 262.03, 260.02,
  258.09, 255.63, 253.28, 250.43, 247.81, 245.26, 242.77, 240.38,
  237.94, 235.79, 233.53, 231.5, 229.53, 227.6, 225.62, 223.77, 222.06,
  220.33, 218.69, 217.18, 215.64, 214.13, 212.52, 210.86, 209.25,
  207.49, 205.81, 204.11, 202.22, 200.32, 198.39, 195.92, 193.46,
  190.94, 188.31, 185.82, 183.57, 181.43, 179.74, 178.64, 178.1, 178.25,
  178.7, 179.41, 180.67, 182.31, 184.18, 186.6, 189.53, 192.66, 196.54,
  201.13, 205.93, 211.73, 217.86, 225, 233.53, 242.57, 252.14, 261.48,
  272.97, 285.26, 299.12, 312.2, 324.17, 338.34, 352.56, 365.28
};

static double c2h2[121] = {
  1.352e-09, 2.83e-10, 1.269e-10, 6.926e-11, 4.346e-11, 2.909e-11,
  2.014e-11, 1.363e-11, 8.71e-12, 5.237e-12, 2.718e-12, 1.375e-12,
  5.786e-13, 2.16e-13, 7.317e-14, 2.551e-14, 1.055e-14, 4.758e-15,
  2.056e-15, 7.703e-16, 2.82e-16, 1.035e-16, 4.382e-17, 1.946e-17,
  9.638e-18, 5.2e-18, 2.811e-18, 1.494e-18, 7.925e-19, 4.213e-19,
  1.998e-19, 8.78e-20, 3.877e-20, 1.728e-20, 7.743e-21, 3.536e-21,
  1.623e-21, 7.508e-22, 3.508e-22, 1.65e-22, 7.837e-23, 3.733e-23,
  1.808e-23, 8.77e-24, 4.285e-24, 2.095e-24, 1.032e-24, 5.082e-25,
  2.506e-25, 1.236e-25, 6.088e-26, 2.996e-26, 1.465e-26, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
};

static double c2h6[121] = {
  2.667e-09, 2.02e-09, 1.658e-09, 1.404e-09, 1.234e-09, 1.109e-09,
  1.012e-09, 9.262e-10, 8.472e-10, 7.71e-10, 6.932e-10, 6.216e-10,
  5.503e-10, 4.87e-10, 4.342e-10, 3.861e-10, 3.347e-10, 2.772e-10,
  2.209e-10, 1.672e-10, 1.197e-10, 8.536e-11, 5.783e-11, 3.846e-11,
  2.495e-11, 1.592e-11, 1.017e-11, 6.327e-12, 3.895e-12, 2.403e-12,
  1.416e-12, 8.101e-13, 4.649e-13, 2.686e-13, 1.557e-13, 9.14e-14,
  5.386e-14, 3.19e-14, 1.903e-14, 1.14e-14, 6.875e-15, 4.154e-15,
  2.538e-15, 1.553e-15, 9.548e-16, 5.872e-16, 3.63e-16, 2.244e-16,
  1.388e-16, 8.587e-17, 5.308e-17, 3.279e-17, 2.017e-17, 1.238e-17,
  7.542e-18, 4.585e-18, 2.776e-18, 1.671e-18, 9.985e-19, 5.937e-19,
  3.518e-19, 2.07e-19, 1.215e-19, 7.06e-20, 4.097e-20, 2.37e-20,
  1.363e-20, 7.802e-21, 4.441e-21, 2.523e-21, 1.424e-21, 8.015e-22,
  4.497e-22, 2.505e-22, 1.391e-22, 7.691e-23, 4.238e-23, 2.331e-23,
  1.274e-23, 6.929e-24, 3.752e-24, 2.02e-24, 1.083e-24, 5.774e-25,
  3.041e-25, 1.593e-25, 8.308e-26, 4.299e-26, 2.195e-26, 1.112e-26,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0
};

static double ccl4[121] = {
  1.075e-10, 1.075e-10, 1.075e-10, 1.075e-10, 1.075e-10, 1.075e-10,
```

```
    1.075e-10, 1.075e-10, 1.075e-10, 1.06e-10, 1.024e-10, 9.69e-11,
    8.93e-11, 8.078e-11, 7.213e-11, 6.307e-11, 5.383e-11, 4.49e-11,
    3.609e-11, 2.705e-11, 1.935e-11, 1.385e-11, 8.35e-12, 5.485e-12,
    3.853e-12, 2.22e-12, 5.875e-13, 3.445e-13, 1.015e-13, 6.075e-14,
    4.383e-14, 2.692e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14,
    1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14,
    1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14,
    1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14,
    1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14,
    1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14,
    1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14,
    1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14,
    1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14,
    1e-14, 1e-14, 1e-14
  };

  static double ch4[121] = {
    1.864e-06, 1.835e-06, 1.819e-06, 1.805e-06, 1.796e-06, 1.788e-06,
    1.782e-06, 1.776e-06, 1.769e-06, 1.761e-06, 1.749e-06, 1.734e-06,
    1.716e-06, 1.692e-06, 1.654e-06, 1.61e-06, 1.567e-06, 1.502e-06,
    1.433e-06, 1.371e-06, 1.323e-06, 1.277e-06, 1.232e-06, 1.188e-06,
    1.147e-06, 1.108e-06, 1.07e-06, 1.027e-06, 9.854e-07, 9.416e-07,
    8.933e-07, 8.478e-07, 7.988e-07, 7.515e-07, 7.07e-07, 6.64e-07,
    6.239e-07, 5.864e-07, 5.512e-07, 5.184e-07, 4.87e-07, 4.571e-07,
    4.296e-07, 4.04e-07, 3.802e-07, 3.578e-07, 3.383e-07, 3.203e-07,
    3.032e-07, 2.889e-07, 2.76e-07, 2.635e-07, 2.519e-07, 2.409e-07,
    2.302e-07, 2.219e-07, 2.144e-07, 2.071e-07, 1.999e-07, 1.93e-07,
    1.862e-07, 1.795e-07, 1.731e-07, 1.668e-07, 1.607e-07, 1.548e-07,
    1.49e-07, 1.434e-07, 1.38e-07, 1.328e-07, 1.277e-07, 1.227e-07,
    1.18e-07, 1.134e-07, 1.089e-07, 1.046e-07, 1.004e-07, 9.635e-08,
    9.245e-08, 8.867e-08, 8.502e-08, 8.15e-08, 7.809e-08, 7.48e-08,
    7.159e-08, 6.849e-08, 6.55e-08, 6.262e-08, 5.98e-08, 5.708e-08,
    5.448e-08, 5.194e-08, 4.951e-08, 4.72e-08, 4.5e-08, 4.291e-08,
    4.093e-08, 3.905e-08, 3.729e-08, 3.563e-08, 3.408e-08, 3.265e-08,
    3.128e-08, 2.996e-08, 2.87e-08, 2.76e-08, 2.657e-08, 2.558e-08,
    2.467e-08, 2.385e-08, 2.307e-08, 2.234e-08, 2.168e-08, 2.108e-08,
    2.05e-08, 1.998e-08, 1.947e-08, 1.902e-08, 1.86e-08, 1.819e-08,
    1.782e-08
  };

  static double clo[121] = {
    7.419e-15, 1.061e-14, 1.518e-14, 2.195e-14, 3.175e-14, 4.666e-14,
    6.872e-14, 1.03e-13, 1.553e-13, 2.375e-13, 3.664e-13, 5.684e-13,
    8.915e-13, 1.402e-12, 2.269e-12, 4.125e-12, 7.501e-12, 1.257e-11,
    2.048e-11, 3.338e-11, 5.44e-11, 8.846e-11, 1.008e-10, 1.082e-10,
    1.157e-10, 1.232e-10, 1.312e-10, 1.539e-10, 1.822e-10, 2.118e-10,
    2.387e-10, 2.687e-10, 2.875e-10, 3.031e-10, 3.23e-10, 3.648e-10,
    4.117e-10, 4.477e-10, 4.633e-10, 4.794e-10, 4.95e-10, 5.104e-10,
    5.259e-10, 5.062e-10, 4.742e-10, 4.443e-10, 4.051e-10, 3.659e-10,
    3.305e-10, 2.911e-10, 2.54e-10, 2.215e-10, 1.927e-10, 1.675e-10,
    1.452e-10, 1.259e-10, 1.09e-10, 9.416e-11, 8.119e-11, 6.991e-11,
    6.015e-11, 5.163e-11, 4.43e-11, 3.789e-11, 3.24e-11, 2.769e-11,
    2.361e-11, 2.011e-11, 1.71e-11, 1.453e-11, 1.233e-11, 1.045e-11,
    8.851e-12, 7.48e-12, 6.316e-12, 5.326e-12, 4.487e-12, 3.778e-12,
    3.176e-12, 2.665e-12, 2.234e-12, 1.87e-12, 1.563e-12, 1.304e-12,
    1.085e-12, 9.007e-13, 7.468e-13, 6.179e-13, 5.092e-13, 4.188e-13,
    3.442e-13, 2.816e-13, 2.304e-13, 1.885e-13, 1.542e-13, 1.263e-13,
    1.035e-13, 8.5e-14, 7.004e-14, 5.783e-14, 4.795e-14, 4.007e-14,
    3.345e-14, 2.792e-14, 2.33e-14, 1.978e-14, 1.686e-14, 1.438e-14,
    1.234e-14, 1.07e-14, 9.312e-15, 8.131e-15, 7.164e-15, 6.367e-15,
    5.67e-15, 5.088e-15, 4.565e-15, 4.138e-15, 3.769e-15, 3.432e-15,
    3.148e-15
  };

  static double clono2[121] = {
    1.011e-13, 1.515e-13, 2.272e-13, 3.446e-13, 5.231e-13, 8.085e-13,
    1.253e-12, 1.979e-12, 3.149e-12, 5.092e-12, 8.312e-12, 1.366e-11,
    2.272e-11, 3.791e-11, 6.209e-11, 9.101e-11, 1.334e-10, 1.951e-10,
    2.853e-10, 3.94e-10, 4.771e-10, 5.771e-10, 6.675e-10, 7.665e-10,
    8.504e-10, 8.924e-10, 9.363e-10, 8.923e-10, 8.411e-10, 7.646e-10,
    6.525e-10, 5.576e-10, 4.398e-10, 3.403e-10, 2.612e-10, 1.915e-10,
    1.407e-10, 1.028e-10, 7.455e-11, 5.42e-11, 3.708e-11, 2.438e-11,
    1.618e-11, 1.075e-11, 7.17e-12, 4.784e-12, 3.205e-12, 2.147e-12,
    1.44e-12, 9.654e-13, 6.469e-13, 4.332e-13, 2.891e-13, 1.926e-13,
    1.274e-13, 8.422e-14, 5.547e-14, 3.636e-14, 2.368e-14, 1.536e-14,
    9.937e-15, 6.39e-15, 4.101e-15, 2.61e-15, 1.659e-15, 1.052e-15,
    6.638e-16, 4.172e-16, 2.61e-16, 1.63e-16, 1.013e-16, 6.275e-17,
    3.879e-17, 2.383e-17, 1.461e-17, 8.918e-18, 5.43e-18, 3.301e-18,
    1.997e-18, 1.203e-18, 7.216e-19, 4.311e-19, 2.564e-19, 1.519e-19,
    8.911e-20, 5.203e-20, 3.026e-20, 1.748e-20, 9.99e-21, 5.673e-21,
    3.215e-21, 1.799e-21, 1.006e-21, 5.628e-22, 3.146e-22, 1.766e-22,
    9.94e-23, 5.614e-23, 3.206e-23, 1.841e-23, 1.071e-23, 6.366e-24,
    3.776e-24, 2.238e-24, 1.326e-24, 8.253e-25, 5.201e-25, 3.279e-25,
    2.108e-25, 1.395e-25, 9.326e-26, 6.299e-26, 4.365e-26, 3.104e-26,
    2.219e-26, 1.621e-26, 1.185e-26, 8.92e-27, 6.804e-27, 5.191e-27,
    4.041e-27
  };
```

```
static double co[121] = {
  1.907e-07, 1.553e-07, 1.362e-07, 1.216e-07, 1.114e-07, 1.036e-07,
  9.737e-08, 9.152e-08, 8.559e-08, 7.966e-08, 7.277e-08, 6.615e-08,
  5.884e-08, 5.22e-08, 4.699e-08, 4.284e-08, 3.776e-08, 3.274e-08,
  2.845e-08, 2.479e-08, 2.246e-08, 2.054e-08, 1.991e-08, 1.951e-08,
  1.94e-08, 2.009e-08, 2.1e-08, 2.201e-08, 2.322e-08, 2.45e-08,
  2.602e-08, 2.73e-08, 2.867e-08, 2.998e-08, 3.135e-08, 3.255e-08,
  3.352e-08, 3.426e-08, 3.484e-08, 3.53e-08, 3.593e-08, 3.671e-08,
  3.759e-08, 3.945e-08, 4.192e-08, 4.49e-08, 5.03e-08, 5.703e-08,
  6.538e-08, 7.878e-08, 9.644e-08, 1.196e-07, 1.498e-07, 1.904e-07,
  2.422e-07, 3.055e-07, 3.804e-07, 4.747e-07, 5.899e-07, 7.272e-07,
  8.91e-07, 1.071e-06, 1.296e-06, 1.546e-06, 1.823e-06, 2.135e-06,
  2.44e-06, 2.714e-06, 2.967e-06, 3.189e-06, 3.391e-06, 3.58e-06,
  3.773e-06, 4.022e-06, 4.346e-06, 4.749e-06, 5.199e-06, 5.668e-06,
  6.157e-06, 6.688e-06, 7.254e-06, 7.867e-06, 8.539e-06, 9.26e-06,
  1.009e-05, 1.119e-05, 1.228e-05, 1.365e-05, 1.506e-05, 1.641e-05,
  1.784e-05, 1.952e-05, 2.132e-05, 2.323e-05, 2.531e-05, 2.754e-05,
  3.047e-05, 3.459e-05, 3.922e-05, 4.439e-05, 4.825e-05, 5.077e-05,
  5.34e-05, 5.618e-05, 5.909e-05, 6.207e-05, 6.519e-05, 6.845e-05,
  6.819e-05, 6.726e-05, 6.622e-05, 6.512e-05, 6.671e-05, 6.862e-05,
  7.048e-05, 7.264e-05, 7.3e-05, 7.3e-05, 7.3e-05, 7.3e-05, 7.3e-05
};

static double cof2[121] = {
  7.5e-14, 1.055e-13, 1.485e-13, 2.111e-13, 3.001e-13, 4.333e-13,
  6.269e-13, 9.221e-13, 1.364e-12, 2.046e-12, 3.093e-12, 4.703e-12,
  7.225e-12, 1.113e-11, 1.66e-11, 2.088e-11, 2.626e-11, 3.433e-11,
  4.549e-11, 5.886e-11, 7.21e-11, 8.824e-11, 1.015e-10, 1.155e-10,
  1.288e-10, 1.388e-10, 1.497e-10, 1.554e-10, 1.606e-10, 1.639e-10,
  1.64e-10, 1.64e-10, 1.596e-10, 1.542e-10, 1.482e-10, 1.382e-10,
  1.289e-10, 1.198e-10, 1.109e-10, 1.026e-10, 9.484e-11, 8.75e-11,
  8.086e-11, 7.49e-11, 6.948e-11, 6.446e-11, 5.961e-11, 5.505e-11,
  5.085e-11, 4.586e-11, 4.1e-11, 3.665e-11, 3.235e-11, 2.842e-11,
  2.491e-11, 2.11e-11, 1.769e-11, 1.479e-11, 1.197e-11, 9.631e-12,
  7.74e-12, 6.201e-12, 4.963e-12, 3.956e-12, 3.151e-12, 2.507e-12,
  1.99e-12, 1.576e-12, 1.245e-12, 9.83e-13, 7.742e-13, 6.088e-13,
  4.782e-13, 3.745e-13, 2.929e-13, 2.286e-13, 1.782e-13, 1.388e-13,
  1.079e-13, 8.362e-14, 6.471e-14, 4.996e-14, 3.85e-14, 2.96e-14,
  2.265e-14, 1.729e-14, 1.317e-14, 9.998e-15, 7.549e-15, 5.683e-15,
  4.273e-15, 3.193e-15, 2.385e-15, 1.782e-15, 1.331e-15, 9.957e-16,
  7.461e-16, 5.601e-16, 4.228e-16, 3.201e-16, 2.438e-16, 1.878e-16,
  1.445e-16, 1.111e-16, 8.544e-17, 6.734e-17, 5.341e-17, 4.237e-17,
  3.394e-17, 2.759e-17, 2.254e-17, 1.851e-17, 1.54e-17, 1.297e-17,
  1.096e-17, 9.365e-18, 8e-18, 6.938e-18, 6.056e-18, 5.287e-18,
  4.662e-18
};

static double f11[121] = {
  2.65e-10, 2.65e-10, 2.65e-10, 2.65e-10, 2.65e-10, 2.65e-10, 2.65e-10,
  2.65e-10, 2.65e-10, 2.65e-10, 2.65e-10, 2.65e-10, 2.635e-10, 2.536e-10,
  2.44e-10, 2.348e-10, 2.258e-10, 2.153e-10, 2.046e-10, 1.929e-10,
  1.782e-10, 1.648e-10, 1.463e-10, 1.291e-10, 1.1e-10, 8.874e-11,
  7.165e-11, 5.201e-11, 3.744e-11, 2.577e-11, 1.64e-11, 1.048e-11,
  5.993e-12, 3.345e-12, 1.839e-12, 9.264e-13, 4.688e-13, 2.329e-13,
  1.129e-13, 5.505e-14, 2.825e-14, 1.492e-14, 7.997e-15, 5.384e-15,
  3.988e-15, 2.955e-15, 2.196e-15, 1.632e-15, 1.214e-15, 9.025e-16,
  6.708e-16, 4.984e-16, 3.693e-16, 2.733e-16, 2.013e-16, 1.481e-16,
  1.087e-16, 7.945e-17, 5.782e-17, 4.195e-17, 3.038e-17, 2.19e-17,
  1.577e-17, 1.128e-17, 8.063e-18, 5.753e-18, 4.09e-18, 2.899e-18,
  2.048e-18, 1.444e-18, 1.015e-18, 7.12e-19, 4.985e-19, 3.474e-19,
  2.417e-19, 1.677e-19, 1.161e-19, 8.029e-20, 5.533e-20, 3.799e-20,
  2.602e-20, 1.776e-20, 1.209e-20, 8.202e-21, 5.522e-21, 3.707e-21,
  2.48e-21, 1.652e-21, 1.091e-21, 7.174e-22, 4.709e-22, 3.063e-22,
  1.991e-22, 1.294e-22, 8.412e-23, 5.483e-23, 3.581e-23, 2.345e-23,
  1.548e-23, 1.027e-23, 6.869e-24, 4.673e-24, 3.173e-24, 2.153e-24,
  1.461e-24, 1.028e-24, 7.302e-25, 5.188e-25, 3.739e-25, 2.753e-25,
  2.043e-25, 1.528e-25, 1.164e-25, 9.041e-26, 7.051e-26, 5.587e-26,
  4.428e-26, 3.588e-26, 2.936e-26, 2.402e-26, 1.995e-26
};

static double f12[121] = {
  5.45e-10, 5.45e-10, 5.45e-10, 5.45e-10, 5.45e-10, 5.45e-10, 5.45e-10,
  5.45e-10, 5.45e-10, 5.45e-10, 5.45e-10, 5.45e-10, 5.429e-10, 5.291e-10,
  5.155e-10, 5.022e-10, 4.893e-10, 4.772e-10, 4.655e-10, 4.497e-10,
  4.249e-10, 4.015e-10, 3.632e-10, 3.261e-10, 2.858e-10, 2.408e-10,
  2.03e-10, 1.685e-10, 1.4e-10, 1.163e-10, 9.65e-11, 8.02e-11, 6.705e-11,
  5.624e-11, 4.764e-11, 4.249e-11, 3.792e-11, 3.315e-11, 2.819e-11,
  2.4e-11, 1.999e-11, 1.64e-11, 1.352e-11, 1.14e-11, 9.714e-12,
  8.28e-12, 7.176e-12, 6.251e-12, 5.446e-12, 4.72e-12, 4.081e-12,
  3.528e-12, 3.08e-12, 2.699e-12, 2.359e-12, 2.111e-12, 1.901e-12,
  1.709e-12, 1.534e-12, 1.376e-12, 1.233e-12, 1.103e-12, 9.869e-13,
  8.808e-13, 7.859e-13, 7.008e-13, 6.241e-13, 5.553e-13, 4.935e-13,
  4.383e-13, 3.889e-13, 3.447e-13, 3.054e-13, 2.702e-13, 2.389e-13,
  2.11e-13, 1.862e-13, 1.643e-13, 1.448e-13, 1.274e-13, 1.121e-13,
  9.844e-14, 8.638e-14, 7.572e-14, 6.62e-14, 5.782e-14, 5.045e-14,
  4.394e-14, 3.817e-14, 3.311e-14, 2.87e-14, 2.48e-14, 2.142e-14,
```

```
  1.851e-14, 1.599e-14, 1.383e-14, 1.196e-14, 1.036e-14, 9e-15,
  7.828e-15, 6.829e-15, 5.992e-15, 5.254e-15, 4.606e-15, 4.037e-15,
  3.583e-15, 3.19e-15, 2.841e-15, 2.542e-15, 2.291e-15, 2.07e-15,
  1.875e-15, 1.71e-15, 1.57e-15, 1.442e-15, 1.333e-15, 1.232e-15,
  1.147e-15, 1.071e-15, 1.001e-15, 9.396e-16
};

static double f14[121] = {
  9e-11, 9e-11, 9e-11, 9e-11, 9e-11, 9e-11, 9e-11, 9e-11, 9e-11, 9e-11,
  9e-11, 9e-11, 9e-11, 9e-11, 9e-11, 8.91e-11, 8.73e-11, 8.46e-11,
  8.19e-11, 7.92e-11, 7.74e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11,
  7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11,
  7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11,
  7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11,
  7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11,
  7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11,
  7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11,
  7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11,
  7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11,
  7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11,
  7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11,
  7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11,
  7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11,
  7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11,
  7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11
};

static double f22[121] = {
  1.4e-10, 1.4e-10, 1.4e-10, 1.4e-10, 1.4e-10, 1.4e-10, 1.4e-10,
  1.4e-10, 1.4e-10, 1.4e-10, 1.372e-10, 1.317e-10, 1.235e-10, 1.153e-10,
  1.075e-10, 1.002e-10, 9.332e-11, 8.738e-11, 8.194e-11, 7.7e-11,
  7.165e-11, 6.753e-11, 6.341e-11, 5.971e-11, 5.6e-11, 5.229e-11,
  4.859e-11, 4.488e-11, 4.118e-11, 3.83e-11, 3.568e-11, 3.308e-11,
  3.047e-11, 2.82e-11, 2.594e-11, 2.409e-11, 2.237e-11, 2.065e-11,
  1.894e-11, 1.771e-11, 1.647e-11, 1.532e-11, 1.416e-11, 1.332e-11,
  1.246e-11, 1.161e-11, 1.087e-11, 1.017e-11, 9.471e-12, 8.853e-12,
  8.235e-12, 7.741e-12, 7.247e-12, 6.836e-12, 6.506e-12, 6.176e-12,
  5.913e-12, 5.65e-12, 5.419e-12, 5.221e-12, 5.024e-12, 4.859e-12,
  4.694e-12, 4.546e-12, 4.414e-12, 4.282e-12, 4.15e-12, 4.019e-12,
  3.903e-12, 3.805e-12, 3.706e-12, 3.607e-12, 3.508e-12, 3.41e-12,
  3.31e-12, 3.212e-12, 3.129e-12, 3.047e-12, 2.964e-12, 2.882e-12,
  2.8e-12, 2.734e-12, 2.668e-12, 2.602e-12, 2.537e-12, 2.471e-12,
  2.421e-12, 2.372e-12, 2.322e-12, 2.273e-12, 2.224e-12, 2.182e-12,
  2.141e-12, 2.1e-12, 2.059e-12, 2.018e-12, 1.977e-12, 1.935e-12,
  1.894e-12, 1.853e-12, 1.812e-12, 1.77e-12, 1.73e-12, 1.688e-12,
  1.647e-12, 1.606e-12, 1.565e-12, 1.524e-12, 1.483e-12, 1.441e-12,
  1.4e-12, 1.359e-12, 1.317e-12, 1.276e-12, 1.235e-12, 1.194e-12,
  1.153e-12, 1.112e-12, 1.071e-12, 1.029e-12, 9.883e-13
};

static double h2o[121] = {
  0.01166, 0.008269, 0.005742, 0.003845, 0.00277, 0.001897, 0.001272,
  0.000827, 0.000539, 0.0003469, 0.0001579, 3.134e-05, 1.341e-05,
  6.764e-06, 4.498e-06, 3.703e-06, 3.724e-06, 3.899e-06, 4.002e-06,
  4.122e-06, 4.277e-06, 4.438e-06, 4.558e-06, 4.673e-06, 4.763e-06,
  4.809e-06, 4.856e-06, 4.936e-06, 5.021e-06, 5.114e-06, 5.222e-06,
  5.331e-06, 5.414e-06, 5.488e-06, 5.563e-06, 5.633e-06, 5.704e-06,
  5.767e-06, 5.819e-06, 5.872e-06, 5.914e-06, 5.949e-06, 5.984e-06,
  6.015e-06, 6.044e-06, 6.073e-06, 6.104e-06, 6.136e-06, 6.167e-06,
  6.189e-06, 6.208e-06, 6.226e-06, 6.212e-06, 6.185e-06, 6.158e-06,
  6.114e-06, 6.066e-06, 6.018e-06, 5.877e-06, 5.728e-06, 5.582e-06,
  5.437e-06, 5.296e-06, 5.156e-06, 5.02e-06, 4.886e-06, 4.754e-06,
  4.625e-06, 4.498e-06, 4.374e-06, 4.242e-06, 4.096e-06, 3.955e-06,
  3.817e-06, 3.683e-06, 3.491e-06, 3.204e-06, 2.94e-06, 2.696e-06,
  2.47e-06, 2.252e-06, 2.019e-06, 1.808e-06, 1.618e-06, 1.445e-06,
  1.285e-06, 1.105e-06, 9.489e-07, 8.121e-07, 6.938e-07, 5.924e-07,
  5.04e-07, 4.288e-07, 3.648e-07, 3.103e-07, 2.642e-07, 2.252e-07,
  1.921e-07, 1.643e-07, 1.408e-07, 1.211e-07, 1.048e-07, 9.063e-08,
  7.835e-08, 6.774e-08, 5.936e-08, 5.221e-08, 4.592e-08, 4.061e-08,
  3.62e-08, 3.236e-08, 2.902e-08, 2.62e-08, 2.383e-08, 2.171e-08,
  1.989e-08, 1.823e-08, 1.684e-08, 1.562e-08, 1.449e-08, 1.351e-08
};

static double h2o2[121] = {
  1.779e-10, 7.938e-10, 8.953e-10, 8.032e-10, 6.564e-10, 5.159e-10,
  4.003e-10, 3.026e-10, 2.222e-10, 1.58e-10, 1.044e-10, 6.605e-11,
  3.413e-11, 1.453e-11, 1.062e-11, 1.009e-11, 9.597e-12, 1.175e-11,
  1.572e-11, 2.091e-11, 2.746e-11, 3.603e-11, 4.791e-11, 6.387e-11,
  8.239e-11, 1.007e-10, 1.23e-10, 1.363e-10, 1.489e-10, 1.585e-10,
  1.608e-10, 1.632e-10, 1.576e-10, 1.502e-10, 1.423e-10, 1.302e-10,
  1.192e-10, 1.085e-10, 9.795e-11, 8.854e-11, 8.057e-11, 7.36e-11,
  6.736e-11, 6.362e-11, 6.087e-11, 5.825e-11, 5.623e-11, 5.443e-11,
  5.27e-11, 5.098e-11, 4.931e-11, 4.769e-11, 4.611e-11, 4.458e-11,
  4.308e-11, 4.102e-11, 3.887e-11, 3.682e-11, 3.521e-11, 3.369e-11,
  3.224e-11, 3.082e-11, 2.946e-11, 2.814e-11, 2.687e-11, 2.566e-11,
  2.449e-11, 2.336e-11, 2.227e-11, 2.123e-11, 2.023e-11, 1.927e-11,
  1.835e-11, 1.746e-11, 1.661e-11, 1.58e-11, 1.502e-11, 1.428e-11,
```

```
  1.357e-11, 1.289e-11, 1.224e-11, 1.161e-11, 1.102e-11, 1.045e-11,
  9.895e-12, 9.369e-12, 8.866e-12, 8.386e-12, 7.922e-12, 7.479e-12,
  7.06e-12, 6.656e-12, 6.274e-12, 5.914e-12, 5.575e-12, 5.257e-12,
  4.959e-12, 4.679e-12, 4.42e-12, 4.178e-12, 3.954e-12, 3.75e-12,
  3.557e-12, 3.372e-12, 3.198e-12, 3.047e-12, 2.908e-12, 2.775e-12,
  2.653e-12, 2.544e-12, 2.442e-12, 2.346e-12, 2.26e-12, 2.183e-12,
  2.11e-12, 2.044e-12, 1.98e-12, 1.924e-12, 1.871e-12, 1.821e-12,
  1.775e-12
};

static double hcn[121] = {
  5.5e-10, 5.5e-10, 5.5e-10, 5.5e-10, 5.5e-10, 5.5e-10,
  5.5e-10, 5.5e-10, 5.5e-10, 5.5e-10, 5.498e-10, 5.495e-10, 5.493e-10,
  5.49e-10, 5.488e-10, 4.717e-10, 3.946e-10, 3.174e-10, 2.4e-10,
  1.626e-10, 1.619e-10, 1.612e-10, 1.602e-10, 1.593e-10, 1.582e-10,
  1.572e-10, 1.56e-10, 1.549e-10, 1.539e-10, 1.53e-10, 1.519e-10,
  1.506e-10, 1.487e-10, 1.467e-10, 1.449e-10, 1.43e-10, 1.413e-10,
  1.397e-10, 1.382e-10, 1.368e-10, 1.354e-10, 1.337e-10, 1.315e-10,
  1.292e-10, 1.267e-10, 1.241e-10, 1.215e-10, 1.19e-10, 1.165e-10,
  1.141e-10, 1.118e-10, 1.096e-10, 1.072e-10, 1.047e-10, 1.021e-10,
  9.968e-11, 9.739e-11, 9.539e-11, 9.339e-11, 9.135e-11, 8.898e-11,
  8.664e-11, 8.439e-11, 8.249e-11, 8.075e-11, 7.904e-11, 7.735e-11,
  7.565e-11, 7.399e-11, 7.245e-11, 7.109e-11, 6.982e-11, 6.863e-11,
  6.755e-11, 6.657e-11, 6.587e-11, 6.527e-11, 6.476e-11, 6.428e-11,
  6.382e-11, 6.343e-11, 6.307e-11, 6.272e-11, 6.238e-11, 6.205e-11,
  6.17e-11, 6.137e-11, 6.102e-11, 6.072e-11, 6.046e-11, 6.03e-11,
  6.018e-11, 6.01e-11, 6.001e-11, 5.992e-11, 5.984e-11, 5.975e-11,
  5.967e-11, 5.958e-11, 5.95e-11, 5.941e-11, 5.933e-11, 5.925e-11,
  5.916e-11, 5.908e-11, 5.899e-11, 5.891e-11, 5.883e-11, 5.874e-11,
  5.866e-11, 5.858e-11, 5.85e-11, 5.841e-11, 5.833e-11, 5.825e-11,
  5.817e-11, 5.808e-11, 5.8e-11, 5.792e-11, 5.784e-11
};

static double hno3[121] = {
  1.809e-10, 7.234e-10, 5.899e-10, 4.342e-10, 3.277e-10, 2.661e-10,
  2.35e-10, 2.267e-10, 2.389e-10, 2.651e-10, 3.255e-10, 4.099e-10,
  5.42e-10, 6.978e-10, 8.807e-10, 1.112e-09, 1.405e-09, 2.04e-09,
  3.111e-09, 4.5e-09, 5.762e-09, 7.37e-09, 7.852e-09, 8.109e-09,
  8.067e-09, 7.554e-09, 7.076e-09, 6.268e-09, 5.524e-09, 4.749e-09,
  3.909e-09, 3.223e-09, 2.517e-09, 1.942e-09, 1.493e-09, 1.122e-09,
  8.449e-10, 6.361e-10, 4.787e-10, 3.611e-10, 2.804e-10, 2.215e-10,
  1.758e-10, 1.441e-10, 1.197e-10, 9.953e-11, 8.505e-11, 7.334e-11,
  6.325e-11, 5.625e-11, 5.058e-11, 4.548e-11, 4.122e-11, 3.748e-11,
  3.402e-11, 3.088e-11, 2.8e-11, 2.536e-11, 2.293e-11, 2.072e-11,
  1.871e-11, 1.687e-11, 1.52e-11, 1.368e-11, 1.23e-11, 1.105e-11,
  9.922e-12, 8.898e-12, 7.972e-12, 7.139e-12, 6.385e-12, 5.708e-12,
  5.099e-12, 4.549e-12, 4.056e-12, 3.613e-12, 3.216e-12, 2.862e-12,
  2.544e-12, 2.259e-12, 2.004e-12, 1.776e-12, 1.572e-12, 1.391e-12,
  1.227e-12, 1.082e-12, 9.528e-13, 8.379e-13, 7.349e-13, 6.436e-13,
  5.634e-13, 4.917e-13, 4.291e-13, 3.745e-13, 3.267e-13, 2.854e-13,
  2.494e-13, 2.181e-13, 1.913e-13, 1.68e-13, 1.479e-13, 1.31e-13,
  1.159e-13, 1.025e-13, 9.067e-14, 8.113e-14, 7.281e-14, 6.535e-14,
  5.892e-14, 5.348e-14, 4.867e-14, 4.439e-14, 4.073e-14, 3.76e-14,
  3.476e-14, 3.229e-14, 3e-14, 2.807e-14, 2.635e-14, 2.473e-14,
  2.332e-14
};

static double hno4[121] = {
  6.118e-12, 3.594e-12, 2.807e-12, 3.04e-12, 4.458e-12, 7.986e-12,
  1.509e-11, 2.661e-11, 3.738e-11, 4.652e-11, 4.429e-11, 3.992e-11,
  3.347e-11, 3.005e-11, 3.173e-11, 4.055e-11, 5.812e-11, 8.489e-11,
  1.19e-10, 1.482e-10, 1.766e-10, 2.103e-10, 2.35e-10, 2.598e-10,
  2.801e-10, 2.899e-10, 3e-10, 2.817e-10, 2.617e-10, 2.332e-10,
  1.933e-10, 1.605e-10, 1.232e-10, 9.285e-11, 6.941e-11, 4.951e-11,
  3.539e-11, 2.402e-11, 1.522e-11, 9.676e-12, 6.056e-12, 3.745e-12,
  2.34e-12, 1.463e-12, 9.186e-13, 5.769e-13, 3.322e-13, 1.853e-13,
  1.035e-13, 7.173e-14, 5.382e-14, 4.036e-14, 3.401e-14, 2.997e-14,
  2.635e-14, 2.316e-14, 2.034e-14, 1.783e-14, 1.56e-14, 1.363e-14,
  1.19e-14, 1.037e-14, 9.032e-15, 7.846e-15, 6.813e-15, 5.912e-15,
  5.121e-15, 4.431e-15, 3.829e-15, 3.306e-15, 2.851e-15, 2.456e-15,
  2.114e-15, 1.816e-15, 1.559e-15, 1.337e-15, 1.146e-15, 9.811e-16,
  8.389e-16, 7.162e-16, 6.109e-16, 5.203e-16, 4.425e-16, 3.76e-16,
  3.184e-16, 2.692e-16, 2.274e-16, 1.917e-16, 1.61e-16, 1.35e-16,
  1.131e-16, 9.437e-17, 7.874e-17, 6.57e-17, 5.481e-17, 4.579e-17,
  3.828e-17, 3.204e-17, 2.691e-17, 2.264e-17, 1.912e-17, 1.626e-17,
  1.382e-17, 1.174e-17, 9.972e-18, 8.603e-18, 7.45e-18, 6.453e-18,
  5.623e-18, 4.944e-18, 4.361e-18, 3.859e-18, 3.443e-18, 3.096e-18,
  2.788e-18, 2.528e-18, 2.293e-18, 2.099e-18, 1.929e-18, 1.773e-18,
  1.64e-18
};

static double hocl[121] = {
  1.056e-12, 1.194e-12, 1.35e-12, 1.531e-12, 1.737e-12, 1.982e-12,
  2.263e-12, 2.599e-12, 2.991e-12, 3.459e-12, 4.012e-12, 4.662e-12,
  5.438e-12, 6.35e-12, 7.425e-12, 8.686e-12, 1.016e-11, 1.188e-11,
  1.389e-11, 1.659e-11, 2.087e-11, 2.621e-11, 3.265e-11, 4.064e-11,
  4.859e-11, 5.441e-11, 6.09e-11, 6.373e-11, 6.611e-11, 6.94e-11,
```

```
    7.44e-11, 7.97e-11, 8.775e-11, 9.722e-11, 1.064e-10, 1.089e-10,
    1.114e-10, 1.106e-10, 1.053e-10, 1.004e-10, 9.006e-11, 7.778e-11,
    6.739e-11, 5.636e-11, 4.655e-11, 3.845e-11, 3.042e-11, 2.368e-11,
    1.845e-11, 1.442e-11, 1.127e-11, 8.814e-12, 6.544e-12, 4.763e-12,
    3.449e-12, 2.612e-12, 1.999e-12, 1.526e-12, 1.16e-12, 8.793e-13,
    6.655e-13, 5.017e-13, 3.778e-13, 2.829e-13, 2.117e-13, 1.582e-13,
    1.178e-13, 8.755e-14, 6.486e-14, 4.799e-14, 3.54e-14, 2.606e-14,
    1.916e-14, 1.403e-14, 1.026e-14, 7.48e-15, 5.446e-15, 3.961e-15,
    2.872e-15, 2.076e-15, 1.498e-15, 1.077e-15, 7.726e-16, 5.528e-16,
    3.929e-16, 2.785e-16, 1.969e-16, 1.386e-16, 9.69e-17, 6.747e-17,
    4.692e-17, 3.236e-17, 2.232e-17, 1.539e-17, 1.061e-17, 7.332e-18,
    5.076e-18, 3.522e-18, 2.461e-18, 1.726e-18, 1.22e-18, 8.75e-19,
    6.264e-19, 4.482e-19, 3.207e-19, 2.368e-19, 1.762e-19, 1.312e-19,
    9.891e-20, 7.595e-20, 5.87e-20, 4.567e-20, 3.612e-20, 2.904e-20,
    2.343e-20, 1.917e-20, 1.568e-20, 1.308e-20, 1.1e-20, 9.25e-21,
    7.881e-21
  };

  static double n2o[121] = {
    3.17e-07, 3.17e-07, 3.17e-07, 3.17e-07, 3.17e-07, 3.17e-07, 3.17e-07,
    3.17e-07, 3.17e-07, 3.17e-07, 3.124e-07, 3.077e-07, 3.03e-07,
    2.984e-07, 2.938e-07, 2.892e-07, 2.847e-07, 2.779e-07, 2.705e-07,
    2.631e-07, 2.557e-07, 2.484e-07, 2.345e-07, 2.201e-07, 2.01e-07,
    1.754e-07, 1.532e-07, 1.329e-07, 1.154e-07, 1.003e-07, 8.735e-08,
    7.617e-08, 6.512e-08, 5.547e-08, 4.709e-08, 3.915e-08, 3.259e-08,
    2.738e-08, 2.327e-08, 1.98e-08, 1.711e-08, 1.493e-08, 1.306e-08,
    1.165e-08, 1.049e-08, 9.439e-09, 8.375e-09, 7.391e-09, 6.525e-09,
    5.759e-09, 5.083e-09, 4.485e-09, 3.953e-09, 3.601e-09, 3.27e-09,
    2.975e-09, 2.757e-09, 2.556e-09, 2.37e-09, 2.195e-09, 2.032e-09,
    1.912e-09, 1.79e-09, 1.679e-09, 1.572e-09, 1.482e-09, 1.402e-09,
    1.326e-09, 1.254e-09, 1.187e-09, 1.127e-09, 1.071e-09, 1.02e-09,
    9.673e-10, 9.193e-10, 8.752e-10, 8.379e-10, 8.017e-10, 7.66e-10,
    7.319e-10, 7.004e-10, 6.721e-10, 6.459e-10, 6.199e-10, 5.942e-10,
    5.703e-10, 5.488e-10, 5.283e-10, 5.082e-10, 4.877e-10, 4.696e-10,
    4.52e-10, 4.355e-10, 4.198e-10, 4.039e-10, 3.888e-10, 3.754e-10,
    3.624e-10, 3.499e-10, 3.381e-10, 3.267e-10, 3.163e-10, 3.058e-10,
    2.959e-10, 2.864e-10, 2.77e-10, 2.686e-10, 2.604e-10, 2.534e-10,
    2.462e-10, 2.386e-10, 2.318e-10, 2.247e-10, 2.189e-10, 2.133e-10,
    2.071e-10, 2.014e-10, 1.955e-10, 1.908e-10, 1.86e-10, 1.817e-10
  };

  static double n2o5[121] = {
    1.231e-11, 3.035e-12, 1.702e-12, 9.877e-13, 8.081e-13, 9.039e-13,
    1.169e-12, 1.474e-12, 1.651e-12, 1.795e-12, 1.998e-12, 2.543e-12,
    4.398e-12, 7.698e-12, 1.28e-11, 2.131e-11, 3.548e-11, 5.894e-11,
    7.645e-11, 1.089e-10, 1.391e-10, 1.886e-10, 2.386e-10, 2.986e-10,
    3.487e-10, 3.994e-10, 4.5e-10, 4.6e-10, 4.591e-10, 4.1e-10, 3.488e-10,
    2.846e-10, 2.287e-10, 1.696e-10, 1.011e-10, 6.428e-11, 4.324e-11,
    2.225e-11, 6.214e-12, 3.608e-12, 8.793e-13, 4.491e-13, 1.04e-13,
    6.1e-14, 3.436e-14, 6.671e-15, 1.171e-15, 5.848e-16, 1.212e-16,
    1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16,
    1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16,
    1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16,
    1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16,
    1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16,
    1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16,
    1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16,
    1e-16, 1e-16
  };

  static double nh3[121] = {
    1e-10, 1e-10, 1e-10, 1e-10, 1e-10, 1e-10, 1e-10, 1e-10, 1e-10, 1e-10,
    1e-10, 1e-10, 1e-10, 1e-10, 9.444e-11, 8.488e-11, 7.241e-11, 5.785e-11,
    4.178e-11, 3.018e-11, 2.18e-11, 1.574e-11, 1.137e-11, 8.211e-12,
    5.973e-12, 4.327e-12, 3.118e-12, 2.234e-12, 1.573e-12, 1.04e-12,
    6.762e-13, 4.202e-13, 2.406e-13, 1.335e-13, 6.938e-14, 3.105e-14,
    1.609e-14, 1.033e-14, 6.432e-15, 4.031e-15, 2.555e-15, 1.656e-15,
    1.115e-15, 7.904e-16, 5.63e-16, 4.048e-16, 2.876e-16, 2.004e-16,
    1.356e-16, 9.237e-17, 6.235e-17, 4.223e-17, 3.009e-17, 2.328e-17,
    2.002e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17,
    1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17,
    1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17,
    1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17,
    1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17,
    1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17,
    1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17,
    1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17,
    1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17,
    1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17,
    1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17,
    1.914e-17
  };

  static double no[121] = {
    2.586e-10, 4.143e-11, 1.566e-11, 9.591e-12, 8.088e-12, 8.462e-12,
    1.013e-11, 1.328e-11, 1.855e-11, 2.678e-11, 3.926e-11, 5.464e-11,
    7.012e-11, 8.912e-11, 1.127e-10, 1.347e-10, 1.498e-10, 1.544e-10,
```

```
  1.602e-10, 1.824e-10, 2.078e-10, 2.366e-10, 2.691e-10, 5.141e-10,
  8.259e-10, 1.254e-09, 1.849e-09, 2.473e-09, 3.294e-09, 4.16e-09,
  5.095e-09, 6.11e-09, 6.93e-09, 7.888e-09, 8.903e-09, 9.713e-09,
  1.052e-08, 1.115e-08, 1.173e-08, 1.21e-08, 1.228e-08, 1.239e-08,
  1.231e-08, 1.213e-08, 1.192e-08, 1.138e-08, 1.085e-08, 1.008e-08,
  9.224e-09, 8.389e-09, 7.262e-09, 6.278e-09, 5.335e-09, 4.388e-09,
  3.589e-09, 2.761e-09, 2.129e-09, 1.633e-09, 1.243e-09, 9.681e-10,
  8.355e-10, 7.665e-10, 7.442e-10, 8.584e-10, 9.732e-10, 1.063e-09,
  1.163e-09, 1.286e-09, 1.472e-09, 1.707e-09, 2.032e-09, 2.474e-09,
  2.977e-09, 3.506e-09, 4.102e-09, 5.013e-09, 6.493e-09, 8.414e-09,
  1.077e-08, 1.367e-08, 1.777e-08, 2.625e-08, 3.926e-08, 5.545e-08,
  7.195e-08, 9.464e-08, 1.404e-07, 2.183e-07, 3.329e-07, 4.535e-07,
  6.158e-07, 8.187e-07, 1.075e-06, 1.422e-06, 1.979e-06, 2.71e-06,
  3.58e-06, 4.573e-06, 5.951e-06, 7.999e-06, 1.072e-05, 1.372e-05,
  1.697e-05, 2.112e-05, 2.643e-05, 3.288e-05, 3.994e-05, 4.794e-05,
  5.606e-05, 6.383e-05, 7.286e-05, 8.156e-05, 8.883e-05, 9.469e-05,
  9.848e-05, 0.0001023, 0.0001066, 0.0001115, 0.0001145, 0.0001142,
  0.0001133
};

static double no2[121] = {
  3.036e-09, 2.945e-10, 9.982e-11, 5.069e-11, 3.485e-11, 2.982e-11,
  2.947e-11, 3.164e-11, 3.714e-11, 4.586e-11, 6.164e-11, 8.041e-11,
  9.982e-11, 1.283e-10, 1.73e-10, 2.56e-10, 3.909e-10, 5.959e-10,
  9.081e-10, 1.384e-09, 1.788e-09, 2.189e-09, 2.686e-09, 3.091e-09,
  3.49e-09, 3.796e-09, 4.2e-09, 5.103e-09, 6.005e-09, 6.3e-09, 6.706e-09,
  7.07e-09, 7.434e-09, 7.663e-09, 7.788e-09, 7.8e-09, 7.597e-09,
  7.482e-09, 7.227e-09, 6.403e-09, 5.585e-09, 4.606e-09, 3.703e-09,
  2.984e-09, 2.183e-09, 1.48e-09, 8.441e-10, 5.994e-10, 3.799e-10,
  2.751e-10, 1.927e-10, 1.507e-10, 1.102e-10, 6.971e-11, 5.839e-11,
  3.904e-11, 3.087e-11, 2.176e-11, 1.464e-11, 1.209e-11, 8.497e-12,
  6.477e-12, 4.371e-12, 2.914e-12, 2.424e-12, 1.753e-12, 1.35e-12,
  9.417e-13, 6.622e-13, 5.148e-13, 3.841e-13, 3.446e-13, 3.01e-13,
  2.551e-13, 2.151e-13, 1.829e-13, 1.64e-13, 1.475e-13, 1.352e-13,
  1.155e-13, 9.963e-14, 9.771e-14, 9.577e-14, 9.384e-14, 9.186e-14,
  9e-14, 9e-14, 9e-14, 9e-14, 9e-14, 9e-14, 9e-14, 9e-14, 9e-14, 9e-14,
  9e-14, 9e-14, 9e-14, 9e-14, 9e-14, 9e-14, 9e-14, 9e-14, 9e-14, 9e-14,
  9e-14, 9e-14, 9e-14, 9e-14, 9e-14, 9e-14, 9e-14, 9e-14, 9e-14, 9e-14,
  9e-14, 9e-14, 9e-14, 9e-14, 9e-14, 9e-14
};

static double o3[121] = {
  2.218e-08, 3.394e-08, 3.869e-08, 4.219e-08, 4.501e-08, 4.778e-08,
  5.067e-08, 5.402e-08, 5.872e-08, 6.521e-08, 7.709e-08, 9.461e-08,
  1.269e-07, 1.853e-07, 2.723e-07, 3.964e-07, 5.773e-07, 8.2e-07,
  1.155e-06, 1.59e-06, 2.076e-06, 2.706e-06, 3.249e-06, 3.848e-06,
  4.459e-06, 4.986e-06, 5.573e-06, 5.958e-06, 6.328e-06, 6.661e-06,
  6.9e-06, 7.146e-06, 7.276e-06, 7.374e-06, 7.447e-06, 7.383e-06,
  7.321e-06, 7.161e-06, 6.879e-06, 6.611e-06, 6.216e-06, 5.765e-06,
  5.355e-06, 4.905e-06, 4.471e-06, 4.075e-06, 3.728e-06, 3.413e-06,
  3.125e-06, 2.856e-06, 2.607e-06, 2.379e-06, 2.17e-06, 1.978e-06,
  1.8e-06, 1.646e-06, 1.506e-06, 1.376e-06, 1.233e-06, 1.102e-06,
  9.839e-07, 8.771e-07, 7.814e-07, 6.947e-07, 6.102e-07, 5.228e-07,
  4.509e-07, 3.922e-07, 3.501e-07, 3.183e-07, 2.909e-07, 2.686e-07,
  2.476e-07, 2.284e-07, 2.109e-07, 2.003e-07, 2.013e-07, 2.022e-07,
  2.032e-07, 2.042e-07, 2.097e-07, 2.361e-07, 2.656e-07, 2.989e-07,
  3.37e-07, 3.826e-07, 4.489e-07, 5.26e-07, 6.189e-07, 7.312e-07,
  8.496e-07, 8.444e-07, 8.392e-07, 8.339e-07, 8.286e-07, 8.234e-07,
  8.181e-07, 8.129e-07, 8.077e-07, 8.026e-07, 6.918e-07, 5.176e-07,
  3.865e-07, 2.885e-07, 2.156e-07, 1.619e-07, 1.219e-07, 9.161e-08,
  6.972e-08, 5.399e-08, 3.498e-08, 2.111e-08, 1.322e-08, 8.482e-09,
  5.527e-09, 3.423e-09, 2.071e-09, 1.314e-09, 8.529e-10, 5.503e-10,
  3.665e-10
};

static double ocs[121] = {
  6e-10, 6e-10, 6e-10, 6e-10, 6e-10, 6e-10, 6e-10, 6e-10, 6e-10, 5.997e-10,
  5.989e-10, 5.881e-10, 5.765e-10, 5.433e-10, 5.074e-10, 4.567e-10,
  4.067e-10, 3.601e-10, 3.093e-10, 2.619e-10, 2.232e-10, 1.805e-10,
  1.46e-10, 1.187e-10, 8.03e-11, 5.435e-11, 3.686e-11, 2.217e-11,
  1.341e-11, 8.756e-12, 4.511e-12, 2.37e-12, 1.264e-12, 8.28e-13,
  5.263e-13, 3.209e-13, 1.717e-13, 9.068e-14, 4.709e-14, 2.389e-14,
  1.236e-14, 1.127e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14,
  1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14,
  1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14,
  1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14,
  1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14,
  1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14,
  1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14,
  1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14,
  1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14,
  1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14,
  1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14,
  1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14,
  1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14,
  1.091e-14, 1.091e-14, 1.091e-14
};
```

```
  static double sf6[121] = {
    4.103e-12, 4.103e-12, 4.103e-12, 4.103e-12, 4.103e-12, 4.103e-12,
    4.103e-12, 4.103e-12, 4.103e-12, 4.087e-12, 4.064e-12, 4.023e-12,
    3.988e-12, 3.941e-12, 3.884e-12, 3.755e-12, 3.622e-12, 3.484e-12,
    3.32e-12, 3.144e-12, 2.978e-12, 2.811e-12, 2.653e-12, 2.489e-12,
    2.332e-12, 2.199e-12, 2.089e-12, 2.013e-12, 1.953e-12, 1.898e-12,
    1.859e-12, 1.826e-12, 1.798e-12, 1.776e-12, 1.757e-12, 1.742e-12,
    1.728e-12, 1.717e-12, 1.707e-12, 1.698e-12, 1.691e-12, 1.685e-12,
    1.679e-12, 1.675e-12, 1.671e-12, 1.668e-12, 1.665e-12, 1.663e-12,
    1.661e-12, 1.659e-12, 1.658e-12, 1.657e-12, 1.656e-12, 1.655e-12,
    1.654e-12, 1.653e-12, 1.653e-12, 1.652e-12, 1.652e-12, 1.652e-12,
    1.651e-12, 1.651e-12, 1.651e-12, 1.651e-12, 1.651e-12, 1.651e-12,
    1.651e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12,
    1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12,
    1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12,
    1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12,
    1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12,
    1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12,
    1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12,
    1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12
  };

  static double so2[121] = {
    1e-10, 1e-10, 1e-10, 1e-10, 1e-10, 1e-10, 1e-10, 1e-10, 1e-10,
    1e-10, 1e-10, 9.867e-11, 9.537e-11, 9e-11, 8.404e-11, 7.799e-11,
    7.205e-11, 6.616e-11, 6.036e-11, 5.475e-11, 5.007e-11, 4.638e-11,
    4.346e-11, 4.055e-11, 3.763e-11, 3.471e-11, 3.186e-11, 2.905e-11,
    2.631e-11, 2.358e-11, 2.415e-11, 2.949e-11, 3.952e-11, 5.155e-11,
    6.76e-11, 8.741e-11, 1.099e-10, 1.278e-10, 1.414e-10, 1.512e-10,
    1.607e-10, 1.699e-10, 1.774e-10, 1.832e-10, 1.871e-10, 1.907e-10,
    1.943e-10, 1.974e-10, 1.993e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10,
    2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10,
    2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10,
    2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10,
    2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10,
    2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10,
    2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10,
    2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10
  };

  static int ig_co2 = -999;

  double co2, *q[NG] = { NULL };

  int ig, ip, iw, iz;

  /* Find emitter index of CO2... */
  if (ig_co2 == -999)
    ig_co2 = find_emitter(ctl, "CO2");

  /* Identify variable... */
  for (ig = 0; ig < ctl->ng; ig++) {
    q[ig] = NULL;
    if (strcasecmp(ctl->emitter[ig], "C2H2") == 0)
      q[ig] = c2h2;
    if (strcasecmp(ctl->emitter[ig], "C2H6") == 0)
      q[ig] = c2h6;
    if (strcasecmp(ctl->emitter[ig], "CCl4") == 0)
      q[ig] = ccl4;
    if (strcasecmp(ctl->emitter[ig], "CH4") == 0)
      q[ig] = ch4;
    if (strcasecmp(ctl->emitter[ig], "ClO") == 0)
      q[ig] = clo;
    if (strcasecmp(ctl->emitter[ig], "ClONO2") == 0)
      q[ig] = clono2;
    if (strcasecmp(ctl->emitter[ig], "CO") == 0)
      q[ig] = co;
    if (strcasecmp(ctl->emitter[ig], "COF2") == 0)
      q[ig] = cof2;
    if (strcasecmp(ctl->emitter[ig], "F11") == 0)
      q[ig] = f11;
    if (strcasecmp(ctl->emitter[ig], "F12") == 0)
      q[ig] = f12;
    if (strcasecmp(ctl->emitter[ig], "F14") == 0)
      q[ig] = f14;
    if (strcasecmp(ctl->emitter[ig], "F22") == 0)
      q[ig] = f22;
    if (strcasecmp(ctl->emitter[ig], "H2O") == 0)
      q[ig] = h2o;
    if (strcasecmp(ctl->emitter[ig], "H2O2") == 0)
      q[ig] = h2o2;
    if (strcasecmp(ctl->emitter[ig], "HCN") == 0)
      q[ig] = hcn;
    if (strcasecmp(ctl->emitter[ig], "HNO3") == 0)
      q[ig] = hno3;
    if (strcasecmp(ctl->emitter[ig], "HNO4") == 0)
```

```
      q[ig] = hno4;
    if (strcasecmp(ctl->emitter[ig], "HOCl") == 0)
      q[ig] = hocl;
    if (strcasecmp(ctl->emitter[ig], "N2O") == 0)
      q[ig] = n2o;
    if (strcasecmp(ctl->emitter[ig], "N2O5") == 0)
      q[ig] = n2o5;
    if (strcasecmp(ctl->emitter[ig], "NH3") == 0)
      q[ig] = nh3;
    if (strcasecmp(ctl->emitter[ig], "NO") == 0)
      q[ig] = no;
    if (strcasecmp(ctl->emitter[ig], "NO2") == 0)
      q[ig] = no2;
    if (strcasecmp(ctl->emitter[ig], "O3") == 0)
      q[ig] = o3;
    if (strcasecmp(ctl->emitter[ig], "OCS") == 0)
      q[ig] = ocs;
    if (strcasecmp(ctl->emitter[ig], "SF6") == 0)
      q[ig] = sf6;
    if (strcasecmp(ctl->emitter[ig], "SO2") == 0)
      q[ig] = so2;
  }

  /* Loop over atmospheric data points... */
  for (ip = 0; ip < atm->np; ip++) {

    /* Get altitude index... */
    iz = locate(z, 121, atm->z[ip]);

    /* Interpolate pressure... */
    atm->p[ip] = EXP(z[iz], pre[iz], z[iz + 1], pre[iz + 1], atm->z[ip]);

    /* Interpolate temperature... */
    atm->t[ip] = LIN(z[iz], tem[iz], z[iz + 1], tem[iz + 1], atm->z[ip]);

    /* Interpolate trace gases... */
    for (ig = 0; ig < ctl->ng; ig++)
      if (q[ig] != NULL)
        atm->q[ig][ip] =
          LIN(z[iz], q[ig][iz], z[iz + 1], q[ig][iz + 1], atm->z[ip]);
      else
        atm->q[ig][ip] = 0;

    /* Set CO2... */
    if (ig_co2 >= 0) {
      co2 =
        371.789948e-6 + 2.026214e-6 * (atm->time[ip] - 63158400.) / 31557600.;
      atm->q[ig_co2][ip] = co2;
    }

    /* Set extinction to zero... */
    for (iw = 0; iw < ctl->nw; iw++)
      atm->k[iw][ip] = 0;
  }
}

/*****************************************************************************/

double ctmco2(
  double nu,
  double p,
  double t,
  double u) {

  static double co2296[2001] = { 9.3388e-5, 9.7711e-5, 1.0224e-4, 1.0697e-4,
    1.1193e-4, 1.1712e-4, 1.2255e-4, 1.2824e-4, 1.3419e-4, 1.4043e-4,
    1.4695e-4, 1.5378e-4, 1.6094e-4, 1.6842e-4, 1.7626e-4, 1.8447e-4,
    1.9307e-4, 2.0207e-4, 2.1149e-4, 2.2136e-4, 2.3169e-4, 2.4251e-4,
    2.5384e-4, 2.657e-4, 2.7813e-4, 2.9114e-4, 3.0477e-4, 3.1904e-4,
    3.3399e-4, 3.4965e-4, 3.6604e-4, 3.8322e-4, 4.0121e-4, 4.2006e-4,
    4.398e-4, 4.6047e-4, 4.8214e-4, 5.0483e-4, 5.286e-4, 5.535e-4,
    5.7959e-4, 6.0693e-4, 6.3557e-4, 6.6558e-4, 6.9702e-4, 7.2996e-4,
    7.6449e-4, 8.0066e-4, 8.3856e-4, 8.7829e-4, 9.1991e-4, 9.6354e-4,
    .0010093, .0010572, .0011074, .00116, .0012152, .001273,
    .0013336, .0013972, .0014638, .0015336, .0016068, .0016835,
    .001764, .0018483, .0019367, .0020295, .0021267, .0022286,
    .0023355, .0024476, .0025652, .0026885, .0028178, .0029534,
    .0030956, .0032448, .0034012, .0035654, .0037375, .0039181,
    .0041076, .0043063, .0045148, .0047336, .0049632, .005204,
    .0054567, .0057219, .0060002, .0062923, .0065988, .0069204,
    .007258, .0076123, .0079842, .0083746, .0087844, .0092146,
    .0096663, .01014, .010638, .011161, .01171, .012286, .012891,
    .013527, .014194, .014895, .015631, .016404, .017217, .01807,
    .018966, .019908, .020897, .021936, .023028, .024176, .025382,
    .026649, .027981, .02938, .030851, .032397, .034023, .035732,
    .037528, .039416, .041402, .04349, .045685, .047994, .050422,
    .052975, .055661, .058486, .061458, .064584, .067873, .071334,
```

```
.074975, .078807, .082839, .087082, .091549, .096249, .1012,
.10641, .11189, .11767, .12375, .13015, .13689, .14399, .15147,
.15935, .16765, .17639, .18561, .19531, .20554, .21632, .22769,
.23967, .25229, .2656, .27964, .29443, .31004, .3265, .34386,
.36218, .3815, .40188, .42339, .44609, .47004, .49533, .52202,
.5502, .57995, .61137, .64455, .6796, .71663, .75574, .79707,
.84075, .88691, .9357, .98728, 1.0418, 1.0995, 1.1605, 1.225,
1.2932, 1.3654, 1.4418, 1.5227, 1.6083, 1.6989, 1.7948, 1.8964,
2.004, 2.118, 2.2388, 2.3668, 2.5025, 2.6463, 2.7988, 2.9606,
3.1321, 3.314, 3.5071, 3.712, 3.9296, 4.1605, 4.4058, 4.6663,
4.9431, 5.2374, 5.5501, 5.8818, 6.2353, 6.6114, 7.0115, 7.4372,
7.8905, 8.3731, 8.8871, 9.4349, 10.019, 10.641, 11.305, 12.013,
12.769, 13.576, 14.437, 15.358, 16.342, 17.39, 18.513, 19.716,
21.003, 22.379, 23.854, 25.436, 27.126, 28.942, 30.89, 32.973,
35.219, 37.634, 40.224, 43.021, 46.037, 49.29, 52.803, 56.447,
60.418, 64.792, 69.526, 74.637, 80.182, 86.193, 92.713, 99.786,
107.47, 115.84, 124.94, 134.86, 145.69, 157.49, 170.3, 184.39,
199.83, 216.4, 234.55, 254.72, 276.82, 299.85, 326.16, 354.99,
386.51, 416.68, 449.89, 490.12, 534.35, 578.25, 632.26, 692.61,
756.43, 834.75, 924.11, 1016.9, 996.96, 1102.7, 1219.2, 1351.9,
1494.3, 1654.1, 1826.5, 2027.9, 2249., 2453.8, 2714.4, 2999.4,
3209.5, 3509., 3840.4, 3907.5, 4190.7, 4533.5, 4648.3, 5059.1,
5561.6, 6191.4, 6820.8, 7905.9, 9362.2, 2431.3, 2211.3, 2046.8,
2023.8, 1985.9, 1905.9, 1491.1, 1369.8, 1262.2, 1200.7, 887.74,
820.25, 885.23, 887.21, 816.73, 1126.9, 1216.2, 1272.4, 1579.5,
1634.2, 1656.3, 1657.9, 1789.5, 1670.8, 1509.5, 8474.6, 7489.2,
6793.6, 6117., 5574.1, 5141.2, 5084.6, 4745.1, 4413.2, 4102.8,
4024.7, 3715., 3398.6, 3100.8, 2900.4, 2629.2, 2374., 2144.7,
1955.8, 1760.8, 1591.2, 1435.2, 1296.2, 1174., 1065.1, 967.76,
999.48, 897.45, 809.23, 732.77, 670.26, 611.93, 560.11, 518.77,
476.84, 438.8, 408.48, 380.21, 349.24, 322.71, 296.65, 272.85,
251.96, 232.04, 213.88, 197.69, 182.41, 168.41, 155.79, 144.05,
133.31, 123.48, 114.5, 106.21, 98.591, 91.612, 85.156, 79.204,
73.719, 68.666, 63.975, 59.637, 56.35, 52.545, 49.042, 45.788,
42.78, 39.992, 37.441, 35.037, 32.8, 30.744, 28.801, 26.986,
25.297, 23.731, 22.258, 20.883, 19.603, 18.403, 17.295, 16.249,
15.271, 14.356, 13.501, 12.701, 11.954, 11.254, 10.6, 9.9864,
9.4118, 8.8745, 8.3714, 7.8997, 7.4578, 7.0446, 6.6573, 6.2949,
5.9577, 5.6395, 5.3419, 5.063, 4.8037, 4.5608, 4.3452, 4.1364,
3.9413, 3.7394, 3.562, 3.3932, 3.2325, 3.0789, 2.9318, 2.7898,
2.6537, 2.5225, 2.3958, 2.2305, 2.1215, 2.0245, 1.9427, 1.8795,
1.8336, 1.7604, 1.7016, 1.6419, 1.5282, 1.4611, 1.3443, 1.27,
1.1675, 1.0824, 1.0534, .99833, .95854, .92981, .90887, .89346,
.88113, .87068, .86102, .85096, .88262, .86151, .83565, .80518,
.77045, .73736, .74744, .74954, .75773, .82267, .83493, .89402,
.89725, .93426, .95564, .94045, .94174, .93404, .92035, .90456,
.88621, .86673, .78117, .7515, .72056, .68822, .65658, .62764,
.55984, .55598, .57407, .60963, .63763, .66198, .61132, .60972,
.52496, .50649, .41872, .3964, .32422, .27276, .24048, .23772,
.2286, .22711, .23999, .32038, .34371, .36621, .38561, .39953,
.40636, .44913, .42716, .3919, .35477, .33935, .3351, .39746,
.40993, .49398, .49956, .56157, .54742, .57295, .57386, .55417,
.50745, .471, .43446, .39102, .34993, .31269, .27888, .24912,
.22291, .19994, .17972, .16197, .14633, .13252, .12029, .10942,
.099745, .091118, .083404, .076494, .070292, .064716, .059697,
.055173, .051093, .047411, .044089, .041092, .038392, .035965,
.033789, .031846, .030122, .028607, .02729, .026169, .025209,
.024405, .023766, .023288, .022925, .022716, .022681, .022685,
.022768, .023133, .023325, .023486, .024004, .024126, .024083,
.023785, .024023, .023029, .021649, .021108, .019454, .017809,
.017292, .016635, .017037, .018068, .018977, .018756, .017847,
.016557, .016142, .014459, .012869, .012381, .010875, .0098701,
.009285, .0091698, .0091701, .0096145, .010553, .01106, .012613,
.014362, .015017, .016507, .017741, .01768, .017784, .0171,
.016357, .016172, .017257, .018978, .020935, .021741, .023567,
.025183, .025589, .026732, .027648, .028278, .028215, .02856,
.029015, .029062, .028851, .028497, .027825, .027801, .026523,
.02487, .022967, .022168, .020194, .018605, .017903, .018439,
.019697, .020311, .020855, .020057, .018608, .016738, .015963,
.013844, .011801, .011134, .0097573, .0086007, .0086226,
.0083721, .0090978, .0097616, .0098426, .011317, .012853, .01447,
.014657, .015771, .016351, .016079, .014829, .013431, .013185,
.013207, .01448, .016176, .017971, .018265, .019526, .020455,
.019797, .019802, .0194, .018176, .017505, .016197, .015339,
.014401, .013213, .012203, .011186, .010236, .0093288, .0084854,
.0076837, .0069375, .0062614, .0056628, .0051153, .0046015,
.0041501, .003752, .0033996, .0030865, .0028077, .0025586,
.0023355, .0021353, .0019553, .0017931, .0016466, .0015141,
.0013941, .0012852, .0011862, .0010962, .0010142, 9.3935e-4,
8.71e-4, 8.0851e-4, 7.5132e-4, 6.9894e-4, 6.5093e-4, 6.0689e-4,
5.6647e-4, 5.2935e-4, 4.9525e-4, 4.6391e-4, 4.3509e-4, 4.086e-4,
3.8424e-4, 3.6185e-4, 3.4126e-4, 3.2235e-4, 3.0498e-4, 2.8904e-4,
2.7444e-4, 2.6106e-4, 2.4883e-4, 2.3766e-4, 2.275e-4, 2.1827e-4,
2.0992e-4, 2.0239e-4, 1.9563e-4, 1.896e-4, 1.8427e-4, 1.796e-4,
1.7555e-4, 1.7209e-4, 1.692e-4, 1.6687e-4, 1.6505e-4, 1.6375e-4,
1.6294e-4, 1.6261e-4, 1.6274e-4, 1.6334e-4, 1.6438e-4, 1.6587e-4,
1.678e-4, 1.7017e-4, 1.7297e-4, 1.762e-4, 1.7988e-4, 1.8399e-4,
```

```
1.8855e-4, 1.9355e-4, 1.9902e-4, 2.0494e-4, 2.1134e-4, 2.1823e-4,
2.2561e-4, 2.335e-4, 2.4192e-4, 2.5088e-4, 2.604e-4, 2.705e-4,
2.8119e-4, 2.9251e-4, 3.0447e-4, 3.171e-4, 3.3042e-4, 3.4447e-4,
3.5927e-4, 3.7486e-4, 3.9127e-4, 4.0854e-4, 4.267e-4, 4.4579e-4,
4.6586e-4, 4.8696e-4, 5.0912e-4, 5.324e-4, 5.5685e-4, 5.8253e-4,
6.0949e-4, 6.378e-4, 6.6753e-4, 6.9873e-4, 7.3149e-4, 7.6588e-4,
8.0198e-4, 8.3987e-4, 8.7964e-4, 9.2139e-4, 9.6522e-4, .0010112,
.0010595, .0011102, .0011634, .0012193, .001278, .0013396,
.0014043, .0014722, .0015436, .0016185, .0016972, .0017799,
.0018668, .001958, .0020539, .0021547, .0022606, .0023719,
.002489, .002612, .0027414, .0028775, .0030206, .0031712,
.0033295, .0034962, .0036716, .0038563, .0040506, .0042553,
.0044709, .004698, .0049373, .0051894, .0054552, .0057354,
.006031, .0063427, .0066717, .0070188, .0073854, .0077726,
.0081816, .0086138, .0090709, .0095543, .010066, .010607,
.011181, .011789, .012433, .013116, .013842, .014613, .015432,
.016304, .017233, .018224, .019281, .020394, .021574, .022836,
.024181, .025594, .027088, .028707, .030401, .032245, .034219,
.036262, .038539, .040987, .043578, .04641, .04949, .052726,
.056326, .0602, .064093, .068521, .073278, .077734, .083064,
.088731, .093885, .1003, .1072, .11365, .12187, .13078, .13989,
.15095, .16299, .17634, .19116, .20628, .22419, .24386, .26587,
.28811, .31399, .34321, .36606, .39675, .42742, .44243, .47197,
.49993, .49027, .51147, .52803, .48931, .49729, .5026, .43854,
.441, .44766, .43414, .46151, .50029, .55247, .43855, .32115,
.32607, .3431, .36119, .38029, .41179, .43996, .47144, .51853,
.55362, .59122, .66338, .69877, .74001, .82923, .86907, .90361,
1.0025, 1.031, 1.0559, 1.104, 1.1178, 1.1341, 1.1547, 1.351,
1.4772, 1.4812, 1.4907, 1.512, 1.5442, 1.5853, 1.6358, 1.6963,
1.7674, 1.8474, 1.9353, 2.0335, 2.143, 2.2592, 2.3853, 2.5217,
2.6686, 2.8273, 2.9998, 3.183, 3.3868, 3.6109, 3.8564, 4.1159,
4.4079, 4.7278, 5.0497, 5.3695, 5.758, 6.0834, 6.4976, 6.9312,
7.38, 7.5746, 7.9833, 8.3791, 8.3956, 8.7501, 9.1067, 9.072,
9.4649, 9.9112, 10.402, 10.829, 11.605, 12.54, 12.713, 10.443,
10.825, 11.375, 11.955, 12.623, 13.326, 14.101, 15.041, 15.547,
16.461, 17.439, 18.716, 19.84, 21.036, 22.642, 23.901, 25.244,
27.03, 28.411, 29.871, 31.403, 33.147, 34.744, 36.456, 39.239,
43.605, 45.162, 47.004, 49.093, 51.391, 53.946, 56.673, 59.629,
63.167, 66.576, 70.254, 74.222, 78.477, 83.034, 87.914, 93.18,
98.77, 104.74, 111.15, 117.95, 125.23, 133.01, 141.33, 150.21,
159.71, 169.89, 180.93, 192.54, 204.99, 218.34, 232.65, 248.,
264.47, 282.14, 301.13, 321.53, 343.48, 367.08, 392.5, 419.88,
449.4, 481.26, 515.64, 552.79, 592.99, 636.48, 683.61, 734.65,
789.99, 850.02, 915.14, 985.81, 1062.5, 1147.1, 1237.8, 1336.4,
1443.2, 1558.9, 1684.2, 1819.2, 1965.2, 2122.6, 2291.7, 2470.8,
2665.7, 2874.9, 3099.4, 3337.9, 3541., 3813.3, 4111.9, 4439.3,
4798.9, 5196., 5639.2, 6087.5, 6657.7, 7306.7, 8040.7, 8845.5,
9702.2, 10670., 11739., 12842., 14141., 15498., 17068., 18729.,
20557., 22559., 25248., 27664., 30207., 32915., 35611., 38081.,
40715., 43191., 41651., 42750., 43785., 44353., 44366., 44189.,
43618., 42862., 41878., 35133., 35215., 36383., 39420., 44055.,
44155., 45850., 46853., 39197., 38274., 29942., 28553., 21792.,
21228., 17106., 14955., 18181., 19557., 21427., 23728., 26301.,
28584., 30775., 32536., 33867., 40089., 39204., 37329., 34452.,
31373., 33921., 34800., 36043., 44415., 45162., 52181., 50895.,
54140., 50840., 50468., 48302., 44915., 40910., 36754., 32755.,
29093., 25860., 22962., 20448., 18247., 16326., 14645., 13165.,
11861., 10708., 9686.9, 8779.7, 7971.9, 7250.8, 6605.7, 6027.2,
5507.3, 5039.1, 4616.6, 4234.8, 3889., 3575.4, 3290.5, 3031.3,
2795.2, 2579.9, 2383.1, 2203.3, 2038.6, 1887.6, 1749.1, 1621.9,
1505., 1397.4, 1298.3, 1207., 1122.8, 1045., 973.1, 906.64,
845.16, 788.22, 735.48, 686.57, 641.21, 599.1, 559.99, 523.64,
489.85, 458.42, 429.16, 401.92, 376.54, 352.88, 330.82, 310.24,
291.03, 273.09, 256.34, 240.69, 226.05, 212.37, 199.57, 187.59,
176.37, 165.87, 156.03, 146.82, 138.17, 130.07, 122.47, 115.34,
108.65, 102.37, 96.473, 90.934, 85.73, 80.84, 76.243, 71.922,
67.858, 64.034, 60.438, 57.052, 53.866, 50.866, 48.04, 45.379,
42.872, 40.51, 38.285, 36.188, 34.211, 32.347, 30.588, 28.929,
27.362, 25.884, 24.489, 23.171, 21.929, 20.755, 19.646, 18.599,
17.61, 16.677, 15.795, 14.961, 14.174, 13.43, 12.725, 12.06,
11.431, 10.834, 10.27, 9.7361, 9.2302, 8.7518, 8.2997, 7.8724,
7.4674, 7.0848, 6.7226, 6.3794, 6.054, 5.745, 5.4525, 5.1752,
4.9121, 4.6625, 4.4259, 4.2015, 3.9888, 3.7872, 3.5961, 3.4149,
3.2431, 3.0802, 2.9257, 2.7792, 2.6402, 2.5084, 2.3834, 2.2648,
2.1522, 2.0455, 1.9441, 1.848, 1.7567, 1.6701, 1.5878, 1.5097,
1.4356, 1.3651, 1.2981, 1.2345, 1.174, 1.1167, 1.062, 1.0101,
.96087, .91414, .86986, .82781, .78777, .74971, .71339, .67882,
.64604, .61473, .58507, .55676, .52987, .5044, .48014, .45715,
.43527, .41453, .3948, .37609, .35831, .34142, .32524, .30995,
.29536, .28142, .26807, .25527, .24311, .23166, .22077, .21053,
.20081, .19143, .18261, .17407, .16603, .15833, .15089, .14385,
.13707, .13065, .12449, .11865, .11306, .10774, .10266, .097818,
.093203, .088815, .084641, .080671, .076892, .073296, .069873,
.066613, .06351, .060555, .05774, .055058, .052504, .050071,
.047752, .045543, .043438, .041432, .039521, .037699, .035962,
.034307, .032729, .031225, .029791, .028423, .02712, .025877,
.024692, .023563, .022485, .021458, .020478, .019543, .018652,
```

```
    .017802, .016992, .016219, .015481, .014778, .014107, .013467,
    .012856, .012274, .011718, .011188, .010682, .0102, .0097393,
    .0093001, .008881, .0084812, .0080997, .0077358, .0073885,
    .0070571, .0067409, .0064393, .0061514, .0058768, .0056147,
    .0053647, .0051262, .0048987, .0046816, .0044745, .0042769,
    .0040884, .0039088, .0037373, .0035739, .003418, .0032693,
    .0031277, .0029926, .0028639, .0027413, .0026245, .0025133,
    .0024074, .0023066, .0022108, .0021196, .002033, .0019507,
    .0018726, .0017985, .0017282, .0016617, .0015988, .0015394,
    .0014834, .0014306, .0013811, .0013346, .0012911, .0012506,
    .0012131, .0011784, .0011465, .0011175, .0010912, .0010678,
    .0010472, .0010295, .0010147, .001003, 9.9428e-4, 9.8883e-4,
    9.8673e-4, 9.8821e-4, 9.9343e-4, .0010027, .0010164, .0010348,
    .0010586, .0010882, .0011245, .0011685, .0012145, .0012666,
    .0013095, .0013688, .0014048, .0014663, .0015309, .0015499,
    .0016144, .0016312, .001705, .0017892, .0018499, .0019715,
    .0021102, .0022442, .0024284, .0025893, .0027703, .0029445,
    .0031193, .003346, .0034552, .0036906, .0037584, .0040084,
    .0041934, .0044587, .0047093, .0049759, .0053421, .0055134,
    .0059048, .0058663, .0061036, .0063259, .0059657, .0060653,
    .0060972, .0055539, .0055653, .0055772, .005331, .0054953,
    .0055919, .0058684, .006183, .0066675, .0069808, .0075142,
    .0078536, .0084282, .0089454, .0094625, .0093703, .0095857,
    .0099283, .010063, .010521, .0097778, .0098175, .010379, .010447,
    .0105, .010617, .010706, .01078, .011177, .011212, .011304,
    .011446, .011603, .011816, .012165, .012545, .013069, .013539,
    .01411, .014776, .016103, .017016, .017994, .018978, .01998,
    .021799, .022745, .023681, .024627, .025562, .026992, .027958,
    .029013, .030154, .031402, .03228, .033651, .035272, .037088,
    .039021, .041213, .043597, .045977, .04877, .051809, .054943,
    .058064, .061528, .06537, .069309, .071928, .075752, .079589,
    .083352, .084096, .087497, .090817, .091198, .094966, .099045,
    .10429, .10867, .11518, .12269, .13126, .14087, .15161, .16388,
    .16423, .1759, .18721, .19994, .21275, .22513, .23041, .24231,
    .25299, .25396, .26396, .27696, .27929, .2908, .30595, .31433,
    .3282, .3429, .35944, .37467, .39277, .41245, .43326, .45649,
    .48152, .51897, .54686, .57877, .61263, .64962, .68983, .73945,
    .78619, .83537, .89622, .95002, 1.0067, 1.0742, 1.1355, 1.2007,
    1.2738, 1.347, 1.4254, 1.5094, 1.6009, 1.6976, 1.8019, 1.9148,
    2.0357, 2.166, 2.3066, 2.4579, 2.6208, 2.7966, 2.986, 3.188,
    3.4081, 3.6456, 3.9, 4.1747, 4.4712, 4.7931, 5.1359, 5.5097,
    5.9117, 6.3435, 6.8003, 7.3001, 7.8385, 8.3945, 9.011, 9.6869,
    10.392, 11.18, 12.036, 12.938, 13.944, 14.881, 16.029, 17.255,
    18.574, 19.945, 21.38, 22.9, 24.477, 26.128, 27.87, 29.037,
    30.988, 33.145, 35.506, 37.76, 40.885, 44.487, 48.505, 52.911,
    57.56, 61.964, 67.217, 72.26, 78.343, 85.08, 91.867, 99.435,
    107.68, 116.97, 127.12, 138.32, 150.26, 163.04, 174.81, 189.26,
    205.61, 224.68, 240.98, 261.88, 285.1, 307.58, 334.35, 363.53,
    394.68, 427.85, 458.85, 489.25, 472.87, 486.93, 496.27, 501.52,
    501.57, 497.14, 488.09, 476.32, 393.76, 388.51, 393.42, 414.45,
    455.12, 514.62, 520.38, 547.42, 562.6, 487.47, 480.83, 391.06,
    376.92, 303.7, 295.91, 256.03, 236.73, 280.38, 310.71, 335.53,
    367.88, 401.94, 435.52, 469.13, 497.94, 588.82, 597.94, 597.2,
    588.28, 571.2, 555.75, 603.56, 638.15, 680.75, 801.72, 848.01,
    962.15, 990.06, 1068.1, 1076.2, 1115.3, 1134.2, 1136.6, 1119.1,
    1108.9, 1090.6, 1068.7, 1041.9, 1005.4, 967.98, 927.08, 780.1,
    751.41, 733.12, 742.65, 785.56, 855.16, 852.45, 878.1, 784.59,
    777.81, 765.13, 622.93, 498.09, 474.89, 386.9, 378.48, 336.17,
    322.04, 329.57, 350.5, 383.38, 420.02, 462.39, 499.71, 531.98,
    654.99, 653.43, 639.99, 605.16, 554.16, 504.42, 540.64, 552.33,
    679.46, 699.51, 713.91, 832.17, 919.91, 884.96, 907.57, 846.56,
    818.56, 768.93, 706.71, 642.17, 575.95, 515.38, 459.07, 409.02,
    364.61, 325.46, 291.1, 260.89, 234.39, 211.01, 190.38, 172.11,
    155.91, 141.49, 128.63, 117.13, 106.84, 97.584, 89.262, 81.756,
    74.975, 68.842, 63.28, 58.232, 53.641, 49.46, 45.649, 42.168,
    38.991, 36.078, 33.409, 30.96, 28.71, 26.642, 24.737, 22.985,
    21.37, 19.882, 18.512, 17.242, 16.073, 14.987, 13.984, 13.05,
    12.186, 11.384, 10.637, 9.9436, 9.2988, 8.6991, 8.141, 7.6215,
    7.1378, 6.6872, 6.2671, 5.8754, 5.51, 5.1691, 4.851, 4.5539,
    4.2764, 4.0169, 3.7742, 3.5472, 3.3348, 3.1359, 2.9495, 2.7749,
    2.6113, 2.4578, 2.3139, 2.1789, 2.0523, 1.9334, 1.8219, 1.7171,
    1.6188, 1.5263, 1.4395, 1.3579, 1.2812, 1.209, 1.1411, 1.0773,
    1.0171, .96048, .90713, .85684, .80959, .76495, .72282, .68309,
    .64563, .61035, .57707, .54573, .51622, .48834, .46199, .43709,
    .41359, .39129, .37034, .35064, .33198, .31442, .29784, .28218,
    .26732, .25337, .24017, .22774, .21601, .20479, .19426
};

static double co2260[2001] = { 5.7971e-5, 6.0733e-5, 6.3628e-5, 6.6662e-5,
    6.9843e-5, 7.3176e-5, 7.6671e-5, 8.0334e-5, 8.4175e-5, 8.8201e-5,
    9.2421e-5, 9.6846e-5, 1.0149e-4, 1.0635e-4, 1.1145e-4, 1.1679e-4,
    1.224e-4, 1.2828e-4, 1.3444e-4, 1.409e-4, 1.4768e-4, 1.5479e-4,
    1.6224e-4, 1.7006e-4, 1.7826e-4, 1.8685e-4, 1.9587e-4, 2.0532e-4,
    2.1524e-4, 2.2565e-4, 2.3656e-4, 2.48e-4, 2.6001e-4, 2.7261e-4,
    2.8582e-4, 2.9968e-4, 3.1422e-4, 3.2948e-4, 3.4548e-4, 3.6228e-4,
    3.799e-4, 3.9838e-4, 4.1778e-4, 4.3814e-4, 4.595e-4, 4.8191e-4,
    5.0543e-4, 5.3012e-4, 5.5603e-4, 5.8321e-4, 6.1175e-4, 6.417e-4,
```

```
6.7314e-4, 7.0614e-4, 7.4078e-4, 7.7714e-4, 8.1531e-4, 8.5538e-4,
8.9745e-4, 9.4162e-4, 9.8798e-4, .0010367, .0010878, .0011415,
.0011978, .001257, .0013191, .0013844, .001453, .0015249,
.0016006, .00168, .0017634, .001851, .001943, .0020397, .0021412,
.0022479, .00236, .0024778, .0026015, .0027316, .0028682,
.0030117, .0031626, .0033211, .0034877, .0036628, .0038469,
.0040403, .0042436, .0044574, .004682, .0049182, .0051665,
.0054276, .0057021, .0059907, .0062942, .0066133, .0069489,
.0073018, .0076729, .0080632, .0084738, .0089056, .0093599,
.0098377, .01034, .010869, .011426, .012011, .012627, .013276,
.013958, .014676, .015431, .016226, .017063, .017944, .018872,
.019848, .020876, .021958, .023098, .024298, .025561, .026892,
.028293, .029769, .031323, .032961, .034686, .036503, .038418,
.040435, .042561, .044801, .047161, .049649, .052271, .055035,
.057948, .061019, .064256, .06767, .07127, .075066, .079069,
.083291, .087744, .092441, .097396, .10262, .10814, .11396,
.1201, .12658, .13342, .14064, .14826, .1563, .1648, .17376,
.18323, .19324, .2038, .21496, .22674, .23919, .25234, .26624,
.28093, .29646, .31287, .33021, .34855, .36794, .38844, .41012,
.43305, .45731, .48297, .51011, .53884, .56924, .60141, .63547,
.67152, .70969, .75012, .79292, .83826, .8863, .93718, .99111,
1.0482, 1.1088, 1.173, 1.2411, 1.3133, 1.3898, 1.471, 1.5571,
1.6485, 1.7455, 1.8485, 1.9577, 2.0737, 2.197, 2.3278, 2.4668,
2.6145, 2.7715, 2.9383, 3.1156, 3.3042, 3.5047, 3.7181, 3.9451,
4.1866, 4.4437, 4.7174, 5.0089, 5.3192, 5.65, 6.0025, 6.3782,
6.7787, 7.206, 7.6617, 8.1479, 8.6669, 9.221, 9.8128, 10.445,
11.12, 11.843, 12.615, 13.441, 14.325, 15.271, 16.283, 17.367,
18.529, 19.776, 21.111, 22.544, 24.082, 25.731, 27.504, 29.409,
31.452, 33.654, 36.024, 38.573, 41.323, 44.29, 47.492, 50.951,
54.608, 58.588, 62.929, 67.629, 72.712, 78.226, 84.207, 90.699,
97.749, 105.42, 113.77, 122.86, 132.78, 143.61, 155.44, 168.33,
182.48, 198.01, 214.87, 233.39, 253.86, 276.34, 300.3, 327.28,
356.89, 389.48, 422.29, 458.99, 501.39, 548.13, 595.62, 652.74,
716.54, 784.57, 866.78, 960.59, 1062.8, 1072.5, 1189.5, 1319.4,
1467.6, 1630.2, 1813.7, 2016.9, 2253., 2515.3, 2773.5, 3092.8,
3444.4, 3720.4, 4104.3, 4527.5, 4645.9, 5021.7, 5462.2, 5597.,
6110.6, 6732.5, 7513.8, 8270.6, 9640.6, 11487., 2796.1, 2680.1,
2441.6, 2404.2, 2334.8, 2215.2, 1642.5, 1477.9, 1328.1, 1223.5,
843.34, 766.96, 831.65, 834.84, 774.85, 1156.3, 1275.6, 1366.1,
1795.6, 1885., 1936.5, 1953.4, 2154.4, 2002.7, 1789.8, 10381.,
9040., 8216.5, 7384.7, 6721.9, 6187.7, 6143.8, 5703.9, 5276.6,
4873.1, 4736., 4325.3, 3927., 3554.1, 3286.1, 2950.1, 2642.4,
2368.7, 2138.9, 1914., 1719.6, 1543.9, 1388.6, 1252.1, 1132.2,
1024.1, 1025.4, 920.58, 829.59, 750.54, 685.01, 624.25, 570.14,
525.81, 481.85, 441.95, 408.71, 377.23, 345.86, 318.51, 292.26,
268.34, 247.04, 227.14, 209.02, 192.69, 177.59, 163.78, 151.26,
139.73, 129.19, 119.53, 110.7, 102.57, 95.109, 88.264, 81.948,
76.13, 70.768, 65.827, 61.251, 57.022, 53.495, 49.824, 46.443,
43.307, 40.405, 37.716, 35.241, 32.923, 30.77, 28.78, 26.915,
25.177, 23.56, 22.059, 20.654, 19.345, 18.126, 16.988, 15.93,
14.939, 14.014, 13.149, 12.343, 11.589, 10.884, 10.225, 9.6093,
9.0327, 8.4934, 7.9889, 7.5166, 7.0744, 6.6604, 6.2727, 5.9098,
5.5701, 5.2529, 4.955, 4.676, 4.4148, 4.171, 3.9426, 3.7332,
3.5347, 3.3493, 3.1677, 3.0025, 2.8466, 2.6994, 2.5601, 2.4277,
2.3016, 2.1814, 2.0664, 1.9564, 1.8279, 1.7311, 1.6427, 1.5645,
1.4982, 1.443, 1.374, 1.3146, 1.2562, 1.17, 1.1105, 1.0272,
.96863, .89718, .83654, .80226, .75908, .72431, .69573, .67174,
.65126, .63315, .61693, .60182, .58715, .59554, .57649, .55526,
.53177, .50622, .48176, .4813, .47642, .47492, .50273, .50293,
.52687, .52239, .53419, .53814, .52626, .52211, .51492, .50622,
.49746, .48841, .4792, .43534, .41999, .40349, .38586, .36799,
.35108, .31089, .30803, .3171, .33599, .35041, .36149, .32924,
.32462, .27309, .25961, .20922, .19504, .15683, .13098, .11588,
.11478, .11204, .11363, .12135, .16423, .17785, .19094, .20236,
.21084, .2154, .24108, .22848, .20871, .18797, .17963, .17834,
.21552, .22284, .26945, .27052, .30108, .28977, .29772, .29224,
.27658, .24956, .22777, .20654, .18392, .16338, .1452, .12916,
.1152, .10304, .092437, .083163, .075031, .067878, .061564,
.055976, .051018, .046609, .042679, .03917, .036032, .033223,
.030706, .02845, .026428, .024617, .022998, .021554, .02027,
.019136, .018141, .017278, .016541, .015926, .015432, .015058,
.014807, .014666, .014635, .014728, .014947, .01527, .015728,
.016345, .017026, .017798, .018839, .019752, .020636, .021886,
.022695, .02327, .023478, .024292, .023544, .022222, .021932,
.020052, .018143, .017722, .017031, .017782, .01938, .020734,
.020476, .019255, .017477, .016878, .014617, .012489, .011765,
.0099077, .0086446, .0079446, .0078644, .0079763, .008671,
.01001, .0108, .012933, .015349, .016341, .018484, .020254,
.020254, .020478, .019591, .018595, .018385, .019913, .022254,
.024847, .025809, .028053, .029924, .030212, .031367, .03222,
.032739, .032537, .03286, .033344, .033507, .033499, .033339,
.032809, .033041, .031723, .029837, .027511, .026603, .024032,
.021914, .020948, .021701, .023425, .024259, .024987, .023818,
.021768, .019223, .018144, .015282, .012604, .01163, .0097907,
.008336, .0082473, .0079582, .0088077, .009779, .010129, .012145,
.014378, .016761, .01726, .018997, .019998, .019809, .01819,
.016358, .016099, .01617, .017939, .020223, .022521, .02277,
```

```
    .024279, .025247, .024222, .023989, .023224, .021493, .020362,
    .018596, .017309, .015975, .014466, .013171, .011921, .01078,
    .0097229, .0087612, .0078729, .0070682, .0063494, .0057156,
    .0051459, .0046273, .0041712, .0037686, .0034119, .003095,
    .0028126, .0025603, .0023342, .0021314, .0019489, .0017845,
    .001636, .0015017, .00138, .0012697, .0011694, .0010782,
    9.9507e-4, 9.1931e-4, 8.5013e-4, 7.869e-4, 7.2907e-4, 6.7611e-4,
    6.2758e-4, 5.8308e-4, 5.4223e-4, 5.0473e-4, 4.7027e-4, 4.3859e-4,
    4.0946e-4, 3.8265e-4, 3.5798e-4, 3.3526e-4, 3.1436e-4, 2.9511e-4,
    2.7739e-4, 2.6109e-4, 2.4609e-4, 2.3229e-4, 2.1961e-4, 2.0797e-4,
    1.9729e-4, 1.875e-4, 1.7855e-4, 1.7038e-4, 1.6294e-4, 1.5619e-4,
    1.5007e-4, 1.4456e-4, 1.3961e-4, 1.3521e-4, 1.3131e-4, 1.2789e-4,
    1.2494e-4, 1.2242e-4, 1.2032e-4, 1.1863e-4, 1.1733e-4, 1.1641e-4,
    1.1585e-4, 1.1565e-4, 1.158e-4, 1.1629e-4, 1.1712e-4, 1.1827e-4,
    1.1976e-4, 1.2158e-4, 1.2373e-4, 1.262e-4, 1.2901e-4, 1.3214e-4,
    1.3562e-4, 1.3944e-4, 1.4361e-4, 1.4814e-4, 1.5303e-4, 1.5829e-4,
    1.6394e-4, 1.6999e-4, 1.7644e-4, 1.8332e-4, 1.9063e-4, 1.984e-4,
    2.0663e-4, 2.1536e-4, 2.246e-4, 2.3436e-4, 2.4468e-4, 2.5558e-4,
    2.6708e-4, 2.7921e-4, 2.92e-4, 3.0548e-4, 3.1968e-4, 3.3464e-4,
    3.5039e-4, 3.6698e-4, 3.8443e-4, 4.0281e-4, 4.2214e-4, 4.4248e-4,
    4.6389e-4, 4.864e-4, 5.1009e-4, 5.3501e-4, 5.6123e-4, 5.888e-4,
    6.1781e-4, 6.4833e-4, 6.8043e-4, 7.142e-4, 7.4973e-4, 7.8711e-4,
    8.2644e-4, 8.6783e-4, 9.1137e-4, 9.5721e-4, .0010054, .0010562,
    .0011096, .0011659, .0012251, .0012875, .0013532, .0014224,
    .0014953, .001572, .0016529, .0017381, .0018279, .0019226,
    .0020224, .0021277, .0022386, .0023557, .0024792, .0026095,
    .002747, .0028921, .0030453, .0032071, .003378, .0035586,
    .0037494, .003951, .0041642, .0043897, .0046282, .0048805,
    .0051476, .0054304, .00573, .0060473, .0063837, .0067404,
    .0071188, .0075203, .0079466, .0083994, .0088806, .0093922,
    .0099366, .010516, .011134, .011792, .012494, .013244, .014046,
    .014898, .015808, .016781, .017822, .018929, .020108, .02138,
    .022729, .02419, .02576, .027412, .029233, .031198, .033301,
    .035594, .038092, .040767, .04372, .046918, .050246, .053974,
    .058009, .061976, .066586, .071537, .076209, .081856, .087998,
    .093821, .10113, .10913, .11731, .12724, .13821, .15025, .1639,
    .17807, .19472, .21356, .23496, .25758, .28387, .31389, .34104,
    .37469, .40989, .43309, .46845, .5042, .5023, .52981, .55275,
    .51075, .51976, .52457, .44779, .44721, .4503, .4243, .45244,
    .49491, .55399, .39021, .24802, .2501, .2618, .27475, .28879,
    .31317, .33643, .36257, .4018, .43275, .46525, .53333, .56599,
    .60557, .70142, .74194, .77736, .88567, .91182, .93294, .98407,
    .98772, .99176, .9995, 1.2405, 1.3602, 1.338, 1.3255, 1.3267,
    1.3404, 1.3634, 1.3967, 1.4407, 1.4961, 1.5603, 1.6328, 1.7153,
    1.8094, 1.9091, 2.018, 2.1367, 2.264, 2.4035, 2.5562, 2.7179,
    2.9017, 3.1052, 3.3304, 3.5731, 3.8488, 4.1553, 4.4769, 4.7818,
    5.1711, 5.5204, 5.9516, 6.4097, 6.8899, 7.1118, 7.5469, 7.9735,
    7.9511, 8.3014, 8.6418, 8.4757, 8.8256, 9.2294, 9.6923, 10.033,
    10.842, 11.851, 11.78, 8.8435, 9.1381, 9.5956, 10.076, 10.629,
    11.22, 11.883, 12.69, 13.163, 13.974, 14.846, 16.027, 17.053,
    18.148, 19.715, 20.907, 22.163, 23.956, 25.235, 26.566, 27.94,
    29.576, 30.956, 32.432, 35.337, 39.911, 41.128, 42.625, 44.386,
    46.369, 48.619, 51.031, 53.674, 56.825, 59.921, 63.286, 66.929,
    70.859, 75.081, 79.618, 84.513, 89.739, 95.335, 101.35, 107.76,
    114.63, 121.98, 129.87, 138.3, 147.34, 157.04, 167.56, 178.67,
    190.61, 203.43, 217.19, 231.99, 247.88, 264.98, 283.37, 303.17,
    324.49, 347.47, 372.25, 398.98, 427.85, 459.06, 492.8, 529.31,
    568.89, 611.79, 658.35, 708.91, 763.87, 823.65, 888.72, 959.58,
    1036.8, 1121.8, 1213.9, 1314.3, 1423.8, 1543., 1672.8, 1813.4,
    1966.1, 2131.4, 2309.5, 2499.3, 2705., 2925.7, 3161.6, 3411.3,
    3611.5, 3889.2, 4191.1, 4519.3, 4877.9, 5272.9, 5712.9, 6142.7,
    6719.6, 7385., 8145., 8977.7, 9831.9, 10827., 11934., 13063.,
    14434., 15878., 17591., 19435., 21510., 23835., 26835., 29740.,
    32878., 36305., 39830., 43273., 46931., 50499., 49586., 51598.,
    53429., 54619., 55081., 55102., 54485., 53487., 52042., 42689.,
    42607., 44020., 47994., 54169., 53916., 55808., 56642., 46049.,
    44243., 32929., 30658., 21963., 20835., 15962., 13679., 17652.,
    19680., 22388., 25625., 29184., 32520., 35720., 38414., 40523.,
    49228., 48173., 45678., 41768., 37600., 41313., 42654., 44465.,
    55736., 56630., 65409., 63308., 66572., 61845., 60379., 56777.,
    51920., 46601., 41367., 36529., 32219., 28470., 25192., 22362.,
    19907., 17772., 15907., 14273., 12835., 11567., 10445., 9450.2,
    8565.1, 7776., 7070.8, 6439.2, 5872.3, 5362.4, 4903., 4488.3,
    4113.4, 3773.8, 3465.8, 3186.1, 2931.7, 2700.1, 2488.8, 2296.,
    2119.8, 1958.6, 1810.9, 1675.6, 1551.4, 1437.3, 1332.4, 1236.,
    1147.2, 1065.3, 989.86, 920.22, 855.91, 796.48, 741.53, 690.69,
    643.62, 600.02, 559.6, 522.13, 487.35, 455.06, 425.08, 397.21,
    371.3, 347.2, 324.78, 303.9, 284.46, 266.34, 249.45, 233.7,
    219.01, 205.3, 192.5, 180.55, 169.38, 158.95, 149.2, 140.07,
    131.54, 123.56, 116.09, 109.09, 102.54, 96.405, 90.655, 85.266,
    80.213, 75.475, 71.031, 66.861, 62.948, 59.275, 55.827, 52.587,
    49.544, 46.686, 43.998, 41.473, 39.099, 36.867, 34.768, 32.795,
    30.939, 29.192, 27.546, 25.998, 24.539, 23.164, 21.869, 20.65,
    19.501, 18.419, 17.399, 16.438, 15.532, 14.678, 13.874, 13.115,
    12.4, 11.726, 11.088, 10.488, 9.921, 9.3846, 8.8784, 8.3996,
    7.9469, 7.5197, 7.1174, 6.738, 6.379, 6.0409, 5.7213, 5.419,
    5.1327, 4.8611, 4.6046, 4.3617, 4.1316, 3.9138, 3.7077, 3.5125,
```

```
3.3281, 3.1536, 2.9885, 2.8323, 2.6846, 2.5447, 2.4124, 2.2871,
2.1686, 2.0564, 1.9501, 1.8495, 1.7543, 1.6641, 1.5787, 1.4978,
1.4212, 1.3486, 1.2799, 1.2147, 1.1529, 1.0943, 1.0388, .98602,
.93596, .8886, .84352, .80078, .76029, .722, .68585, .65161,
.61901, .58808, .55854, .53044, .5039, .47853, .45459, .43173,
.41008, .38965, .37021, .35186, .33444, .31797, .30234, .28758,
.2736, .26036, .24764, .2357, .22431, .21342, .20295, .19288,
.18334, .17444, .166, .15815, .15072, .14348, .13674, .13015,
.12399, .11807, .11231, .10689, .10164, .096696, .091955,
.087476, .083183, .079113, .075229, .071536, .068026, .064698,
.06154, .058544, .055699, .052997, .050431, .047993, .045676,
.043475, .041382, .039392, .037501, .035702, .033991, .032364,
.030817, .029345, .027945, .026613, .025345, .024139, .022991,
.021899, .02086, .019871, .018929, .018033, .01718, .016368,
.015595, .014859, .014158, .013491, .012856, .012251, .011675,
.011126, .010604, .010107, .0096331, .009182, .0087523, .0083431,
.0079533, .0075821, .0072284, .0068915, .0065706, .0062649,
.0059737, .0056963, .005432, .0051802, .0049404, .0047118,
.0044941, .0042867, .0040891, .0039009, .0037216, .0035507,
.003388, .0032329, .0030852, .0029445, .0028105, .0026829,
.0025613, .0024455, .0023353, .0022303, .0021304, .0020353,
.0019448, .0018587, .0017767, .0016988, .0016247, .0015543,
.0014874, .0014238, .0013635, .0013062, .0012519, .0012005,
.0011517, .0011057, .0010621, .001021, 9.8233e-4, 9.4589e-4,
9.1167e-4, 8.7961e-4, 8.4964e-4, 8.2173e-4, 7.9582e-4, 7.7189e-4,
7.499e-4, 7.2983e-4, 7.1167e-4, 6.9542e-4, 6.8108e-4, 6.6866e-4,
6.5819e-4, 6.4971e-4, 6.4328e-4, 6.3895e-4, 6.3681e-4, 6.3697e-4,
6.3956e-4, 6.4472e-4, 6.5266e-4, 6.6359e-4, 6.778e-4, 6.9563e-4,
7.1749e-4, 7.4392e-4, 7.7556e-4, 8.1028e-4, 8.4994e-4, 8.8709e-4,
9.3413e-4, 9.6953e-4, .0010202, .0010738, .0010976, .0011507,
.0011686, .0012264, .001291, .0013346, .0014246, .0015293,
.0016359, .0017824, .0019255, .0020854, .002247, .0024148,
.0026199, .0027523, .0029704, .0030702, .0033047, .0035013,
.0037576, .0040275, .0043089, .0046927, .0049307, .0053486,
.0053809, .0056699, .0059325, .0055488, .005634, .0056392,
.004946, .0048855, .0048208, .0044386, .0045498, .0046377,
.0048939, .0052396, .0057324, .0060859, .0066906, .0071148,
.0077224, .0082687, .008769, .0084471, .008572, .0087729,
.008775, .0090742, .0080704, .0080288, .0085747, .0086087,
.0086408, .0088752, .0089381, .0089757, .0093532, .0092824,
.0092566, .0092645, .0092735, .009342, .0095806, .0097991,
.010213, .010611, .011129, .011756, .013237, .01412, .015034,
.015936, .01682, .018597, .019315, .019995, .020658, .021289,
.022363, .022996, .023716, .024512, .025434, .026067, .027118,
.028396, .029865, .031442, .033253, .03525, .037296, .039701,
.042356, .045154, .048059, .051294, .054893, .058636, .061407,
.065172, .068974, .072676, .073379, .076547, .079556, .079134,
.082308, .085739, .090192, .09359, .099599, .10669, .11496,
.1244, .13512, .14752, .14494, .15647, .1668, .17863, .19029,
.20124, .20254, .21179, .21982, .21625, .22364, .23405, .23382,
.2434, .25708, .26406, .27621, .28909, .30395, .31717, .33271,
.3496, .36765, .38774, .40949, .446, .46985, .49846, .5287, .562,
.59841, .64598, .68834, .7327, .78978, .8373, .88708, .94744,
1.0006, 1.0574, 1.1215, 1.1856, 1.2546, 1.3292, 1.4107, 1.4974,
1.5913, 1.6931, 1.8028, 1.9212, 2.0492, 2.1874, 2.3365, 2.4978,
2.6718, 2.8588, 3.062, 3.2818, 3.5188, 3.7752, 4.0527, 4.3542,
4.6782, 5.0312, 5.4123, 5.8246, 6.2639, 6.7435, 7.2636, 7.8064,
8.4091, 9.0696, 9.7677, 10.548, 11.4, 12.309, 13.324, 14.284,
15.445, 16.687, 18.019, 19.403, 20.847, 22.366, 23.925, 25.537,
27.213, 28.069, 29.864, 31.829, 33.988, 35.856, 38.829, 42.321,
46.319, 50.606, 55.126, 59.126, 64.162, 68.708, 74.615, 81.176,
87.739, 95.494, 103.83, 113.38, 123.99, 135.8, 148.7, 162.58,
176.32, 192.6, 211.47, 232.7, 252.64, 277.41, 305.38, 333.44,
366.42, 402.66, 442.14, 484.53, 526.42, 568.15, 558.78, 582.6,
600.98, 613.94, 619.44, 618.24, 609.84, 595.96, 484.86, 475.59,
478.49, 501.56, 552.19, 628.44, 630.39, 658.92, 671.96, 562.7,
545.88, 423.43, 400.14, 306.59, 294.13, 246.8, 226.51, 278.21,
314.39, 347.22, 389.13, 433.16, 477.48, 521.67, 560.54, 683.6,
696.37, 695.91, 683.1, 658.24, 634.89, 698.85, 742.87, 796.66,
954.49, 1009.5, 1150.5, 1179.1, 1267.9, 1272.4, 1312.7, 1330.4,
1331.6, 1315.8, 1308.3, 1293.3, 1274.6, 1249.5, 1213.2, 1172.1,
1124.4, 930.33, 893.36, 871.27, 883.54, 940.76, 1036., 1025.6,
1053.1, 914.51, 894.15, 865.03, 670.63, 508.41, 475.15, 370.85,
361.06, 319.38, 312.75, 331.87, 367.13, 415., 467.94, 525.49,
578.41, 624.66, 794.82, 796.97, 780.29, 736.49, 670.18, 603.75,
659.67, 679.8, 857.12, 884.05, 900.65, 1046.1, 1141.9, 1083.,
1089.2, 1e3, 947.08, 872.31, 787.91, 704.75, 624.93, 553.68,
489.91, 434.21, 385.64, 343.3, 306.42, 274.18, 245.94, 221.11,
199.23, 179.88, 162.73, 147.48, 133.88, 121.73, 110.86, 101.1,
92.323, 84.417, 77.281, 70.831, 64.991, 59.694, 54.884, 50.509,
46.526, 42.893, 39.58, 36.549, 33.776, 31.236, 28.907, 26.77,
24.805, 23., 21.339, 19.81, 18.404, 17.105, 15.909, 14.801,
13.778, 12.83, 11.954, 11.142, 10.389, 9.691, 9.0434, 8.4423,
7.8842, 7.3657, 6.8838, 6.4357, 6.0189, 5.6308, 5.2696, 4.9332,
4.6198, 4.3277, 4.0553, 3.8012, 3.5639, 3.3424, 3.1355, 2.9422,
2.7614, 2.5924, 2.4343, 2.2864, 2.148, 2.0184, 1.8971, 1.7835,
1.677, 1.5773, 1.4838, 1.3961, 1.3139, 1.2369, 1.1645, 1.0966,
```

```
    1.0329, .97309, .91686, .86406, .81439, .76767, .72381, .68252,
    .64359, .60695, .57247, .54008, .50957, .48092, .45401, .42862,
    .40465, .38202, .36072, .34052, .3216, .30386, .28711, .27135,
    .25651, .24252, .2293, .21689, .20517, .19416, .18381, .17396,
    .16469
  };

  static double co2230[2001] = { 2.743e-5, 2.8815e-5, 3.027e-5, 3.1798e-5,
    3.3405e-5, 3.5094e-5, 3.6869e-5, 3.8734e-5, 4.0694e-5, 4.2754e-5,
    4.492e-5, 4.7196e-5, 4.9588e-5, 5.2103e-5, 5.4747e-5, 5.7525e-5,
    6.0446e-5, 6.3516e-5, 6.6744e-5, 7.0137e-5, 7.3704e-5, 7.7455e-5,
    8.1397e-5, 8.5543e-5, 8.9901e-5, 9.4484e-5, 9.9302e-5, 1.0437e-4,
    1.097e-4, 1.153e-4, 1.2119e-4, 1.2738e-4, 1.3389e-4, 1.4074e-4,
    1.4795e-4, 1.5552e-4, 1.6349e-4, 1.7187e-4, 1.8068e-4, 1.8995e-4,
    1.997e-4, 2.0996e-4, 2.2075e-4, 2.321e-4, 2.4403e-4, 2.5659e-4,
    2.698e-4, 2.837e-4, 2.9832e-4, 3.137e-4, 3.2988e-4, 3.4691e-4,
    3.6483e-4, 3.8368e-4, 4.0351e-4, 4.2439e-4, 4.4635e-4, 4.6947e-4,
    4.9379e-4, 5.1939e-4, 5.4633e-4, 5.7468e-4, 6.0452e-4, 6.3593e-4,
    6.69e-4, 7.038e-4, 7.4043e-4, 7.79e-4, 8.1959e-4, 8.6233e-4,
    9.0732e-4, 9.5469e-4, .0010046, .0010571, .0011124, .0011706,
    .0012319, .0012964, .0013644, .001436, .0015114, .0015908,
    .0016745, .0017625, .0018553, .0019531, .002056, .0021645,
    .0022788, .0023992, .002526, .0026596, .0028004, .0029488,
    .0031052, .0032699, .0034436, .0036265, .0038194, .0040227,
    .0042369, .0044628, .0047008, .0049518, .0052164, .0054953,
    .0057894, .0060995, .0064265, .0067713, .007135, .0075184,
    .0079228, .0083494, .0087993, .0092738, .0097745, .010303,
    .01086, .011448, .012068, .012722, .013413, .014142, .014911,
    .015723, .01658, .017484, .018439, .019447, .020511, .021635,
    .022821, .024074, .025397, .026794, .02827, .029829, .031475,
    .033215, .035052, .036994, .039045, .041213, .043504, .045926,
    .048485, .05119, .05405, .057074, .060271, .063651, .067225,
    .071006, .075004, .079233, .083708, .088441, .093449, .098749,
    .10436, .11029, .11657, .12322, .13026, .13772, .14561, .15397,
    .16282, .1722, .18214, .19266, .20381, .21563, .22816, .24143,
    .2555, .27043, .28625, .30303, .32082, .3397, .35972, .38097,
    .40352, .42746, .45286, .47983, .50847, .53888, .57119, .6055,
    .64196, .6807, .72187, .76564, .81217, .86165, .91427, .97025,
    1.0298, 1.0932, 1.1606, 1.2324, 1.3088, 1.3902, 1.477, 1.5693,
    1.6678, 1.7727, 1.8845, 2.0038, 2.131, 2.2666, 2.4114, 2.5659,
    2.7309, 2.907, 3.0951, 3.2961, 3.5109, 3.7405, 3.986, 4.2485,
    4.5293, 4.8299, 5.1516, 5.4961, 5.8651, 6.2605, 6.6842, 7.1385,
    7.6256, 8.1481, 8.7089, 9.3109, 9.9573, 10.652, 11.398, 12.2,
    13.063, 13.992, 14.99, 16.064, 17.222, 18.469, 19.813, 21.263,
    22.828, 24.516, 26.34, 28.31, 30.437, 32.738, 35.226, 37.914,
    40.824, 43.974, 47.377, 51.061, 55.011, 59.299, 63.961, 69.013,
    74.492, 80.444, 86.919, 93.836, 101.23, 109.25, 117.98, 127.47,
    137.81, 149.07, 161.35, 174.75, 189.42, 205.49, 223.02, 242.26,
    263.45, 286.75, 311.94, 340.01, 370.86, 404.92, 440.44, 480.27,
    525.17, 574.71, 626.22, 686.8, 754.38, 827.07, 913.38, 1011.7,
    1121.5, 1161.6, 1289.5, 1432.2, 1595.4, 1777., 1983.3, 2216.1,
    2485.7, 2788.3, 3101.5, 3481., 3902.1, 4257.1, 4740., 5272.8,
    5457.9, 5946.2, 6505.3, 6668.4, 7302.4, 8061.6, 9015.8, 9908.3,
    11613., 13956., 3249.6, 3243., 2901.5, 2841.3, 2729.6, 2558.2,
    1797.8, 1583.2, 1386., 1233.5, 787.74, 701.46, 761.66, 767.21,
    722.83, 1180.6, 1332.1, 1461.6, 2032.9, 2166., 2255.9, 2294.7,
    2587.2, 2396.5, 2122.4, 12553., 10784., 9832.5, 8827.3, 8029.1,
    7377.9, 7347.1, 6783.8, 6239.1, 5721.1, 5503., 4975.1, 4477.8,
    4021.3, 3676.8, 3275.3, 2914.9, 2597.4, 2328.2, 2075.4, 1857.6,
    1663.6, 1493.3, 1343.8, 1213.3, 1095.6, 1066.5, 958.91, 865.15,
    783.31, 714.35, 650.77, 593.98, 546.2, 499.9, 457.87, 421.75,
    387.61, 355.25, 326.62, 299.7, 275.21, 253.17, 232.83, 214.31,
    197.5, 182.08, 167.98, 155.12, 143.32, 132.5, 122.58, 113.48,
    105.11, 97.415, 90.182, 83.463, 77.281, 71.587, 66.341, 61.493,
    57.014, 53.062, 49.21, 45.663, 42.38, 39.348, 36.547, 33.967,
    31.573, 29.357, 27.314, 25.415, 23.658, 22.03, 20.524, 19.125,
    17.829, 16.627, 15.511, 14.476, 13.514, 12.618, 11.786, 11.013,
    10.294, 9.6246, 9.0018, 8.4218, 7.8816, 7.3783, 6.9092, 6.4719,
    6.0641, 5.6838, 5.3289, 4.998, 4.6893, 4.4014, 4.1325, 3.8813,
    3.6469, 3.4283, 3.2241, 3.035, 2.8576, 2.6922, 2.5348, 2.3896,
    2.2535, 2.1258, 2.0059, 1.8929, 1.7862, 1.6854, 1.5898, 1.4992,
    1.4017, 1.3218, 1.2479, 1.1809, 1.1215, 1.0693, 1.0116, .96016,
    .9105, .84859, .80105, .74381, .69982, .65127, .60899, .57843,
    .54592, .51792, .49336, .47155, .45201, .43426, .41807, .40303,
    .38876, .3863, .37098, .35492, .33801, .32032, .30341, .29874,
    .29193, .28689, .29584, .29155, .29826, .29195, .29287, .2904,
    .28199, .27709, .27162, .26622, .26133, .25676, .25235, .23137,
    .22365, .21519, .20597, .19636, .18699, .16485, .16262, .16643,
    .17542, .18198, .18631, .16759, .16338, .13505, .1267, .10053,
    .092554, .074093, .062159, .055523, .054849, .05401, .05528,
    .058982, .07952, .08647, .093244, .099285, .10393, .10661,
    .12072, .11417, .10396, .093265, .089137, .088909, .10902,
    .11277, .13625, .13565, .14907, .14167, .1428, .13744, .12768,
    .11382, .10244, .091686, .08109, .071739, .063616, .056579,
    .050504, .045251, .040689, .036715, .033237, .030181, .027488,
    .025107, .022998, .021125, .01946, .017979, .016661, .015489,
    .014448, .013526, .012712, .011998, .011375, .010839, .010384,
```

```
.010007, .0097053, .0094783, .0093257, .0092489, .0092504,
.0093346, .0095077, .0097676, .01012, .01058, .011157, .011844,
.012672, .013665, .014766, .015999, .017509, .018972, .020444,
.022311, .023742, .0249, .025599, .026981, .026462, .025143,
.025066, .022814, .020458, .020026, .019142, .020189, .022371,
.024163, .023728, .02199, .019506, .018591, .015576, .012784,
.011744, .0094777, .0079148, .0070652, .006986, .0071758,
.008086, .0098025, .01087, .013609, .016764, .018137, .021061,
.023498, .023576, .023965, .022828, .021519, .021283, .023364,
.026457, .029782, .030856, .033486, .035515, .035543, .036558,
.037198, .037472, .037045, .037284, .03777, .038085, .038366,
.038526, .038282, .038915, .037697, .035667, .032941, .031959,
.028692, .025918, .024596, .025592, .027873, .028935, .02984,
.028148, .025305, .021912, .020454, .016732, .013357, .01205,
.009731, .0079881, .0077704, .0074387, .0083895, .0096776,
.010326, .01293, .015955, .019247, .020145, .02267, .024231,
.024184, .022131, .019784, .01955, .01971, .022119, .025116,
.027978, .028107, .029808, .030701, .029164, .028551, .027286,
.024946, .023259, .020982, .019221, .017471, .015643, .014074,
.01261, .011301, .010116, .0090582, .0081036, .0072542, .0065034,
.0058436, .0052571, .0047321, .0042697, .0038607, .0034977,
.0031747, .0028864, .0026284, .002397, .002189, .0020017,
.0018326, .0016798, .0015414, .0014159, .0013019, .0011983,
.0011039, .0010177, 9.391e-4, 8.6717e-4, 8.0131e-4, 7.4093e-4,
6.8553e-4, 6.3464e-4, 5.8787e-4, 5.4487e-4, 5.0533e-4, 4.69e-4,
4.3556e-4, 4.0474e-4, 3.7629e-4, 3.5e-4, 3.2569e-4, 3.032e-4,
2.8239e-4, 2.6314e-4, 2.4535e-4, 2.2891e-4, 2.1374e-4, 1.9975e-4,
1.8685e-4, 1.7498e-4, 1.6406e-4, 1.5401e-4, 1.4479e-4, 1.3633e-4,
1.2858e-4, 1.2148e-4, 1.1499e-4, 1.0907e-4, 1.0369e-4, 9.8791e-5,
9.4359e-5, 9.0359e-5, 8.6766e-5, 8.3555e-5, 8.0703e-5, 7.8192e-5,
7.6003e-5, 7.4119e-5, 7.2528e-5, 7.1216e-5, 7.0171e-5, 6.9385e-5,
6.8848e-5, 6.8554e-5, 6.8496e-5, 6.8669e-5, 6.9069e-5, 6.9694e-5,
7.054e-5, 7.1608e-5, 7.2896e-5, 7.4406e-5, 7.6139e-5, 7.8097e-5,
8.0283e-5, 8.2702e-5, 8.5357e-5, 8.8255e-5, 9.1402e-5, 9.4806e-5,
9.8473e-5, 1.0241e-4, 1.0664e-4, 1.1115e-4, 1.1598e-4, 1.2112e-4,
1.2659e-4, 1.3241e-4, 1.3859e-4, 1.4515e-4, 1.521e-4, 1.5947e-4,
1.6728e-4, 1.7555e-4, 1.8429e-4, 1.9355e-4, 2.0334e-4, 2.1369e-4,
2.2463e-4, 2.3619e-4, 2.4841e-4, 2.6132e-4, 2.7497e-4, 2.8938e-4,
3.0462e-4, 3.2071e-4, 3.3771e-4, 3.5567e-4, 3.7465e-4, 3.947e-4,
4.1588e-4, 4.3828e-4, 4.6194e-4, 4.8695e-4, 5.1338e-4, 5.4133e-4,
5.7087e-4, 6.0211e-4, 6.3515e-4, 6.701e-4, 7.0706e-4, 7.4617e-4,
7.8756e-4, 8.3136e-4, 8.7772e-4, 9.2681e-4, 9.788e-4, .0010339,
.0010922, .001154, .0012195, .0012889, .0013626, .0014407,
.0015235, .0016114, .0017048, .0018038, .001909, .0020207,
.0021395, .0022657, .0023998, .0025426, .0026944, .002856,
.0030281, .0032114, .0034068, .003615, .0038371, .004074,
.004327, .0045971, .0048857, .0051942, .0055239, .0058766,
.0062538, .0066573, .0070891, .007551, .0080455, .0085747,
.0091412, .0097481, .010397, .011092, .011837, .012638, .013495,
.014415, .01541, .016475, .017621, .018857, .020175, .02162,
.023185, .024876, .02672, .028732, .030916, .033319, .035939,
.038736, .041847, .04524, .048715, .052678, .056977, .061203,
.066184, .07164, .076952, .083477, .090674, .098049, .10697,
.1169, .1277, .14011, .15323, .1684, .18601, .20626, .22831,
.25417, .28407, .31405, .34957, .38823, .41923, .46026, .50409,
.51227, .54805, .57976, .53818, .55056, .557, .46741, .46403,
.4636, .42265, .45166, .49852, .56663, .34306, .17779, .17697,
.18346, .19129, .20014, .21778, .23604, .25649, .28676, .31238,
.33856, .39998, .4288, .46568, .56654, .60786, .64473, .76466,
.7897, .80778, .86443, .85736, .84798, .84157, 1.1385, 1.2446,
1.1923, 1.1552, 1.1338, 1.1266, 1.1292, 1.1431, 1.1683, 1.2059,
1.2521, 1.3069, 1.3712, 1.4471, 1.5275, 1.6165, 1.7145, 1.8189,
1.9359, 2.065, 2.2007, 2.3591, 2.5362, 2.7346, 2.9515, 3.2021,
3.4851, 3.7935, 4.0694, 4.4463, 4.807, 5.2443, 5.7178, 6.2231,
6.4796, 6.9461, 7.4099, 7.3652, 7.7182, 8.048, 7.7373, 8.0363,
8.3855, 8.8044, 9.0257, 9.8574, 10.948, 10.563, 6.8979, 7.0744,
7.4121, 7.7663, 8.1768, 8.6243, 9.1437, 9.7847, 10.182, 10.849,
11.572, 12.602, 13.482, 14.431, 15.907, 16.983, 18.11, 19.884,
21.02, 22.18, 23.355, 24.848, 25.954, 27.13, 30.186, 34.893,
35.682, 36.755, 38.111, 39.703, 41.58, 43.606, 45.868, 48.573,
51.298, 54.291, 57.559, 61.116, 64.964, 69.124, 73.628, 78.471,
83.683, 89.307, 95.341, 101.84, 108.83, 116.36, 124.46, 133.18,
142.57, 152.79, 163.69, 175.43, 188.11, 201.79, 216.55, 232.51,
249.74, 268.38, 288.54, 310.35, 333.97, 359.55, 387.26, 417.3,
449.88, 485.2, 523.54, 565.14, 610.28, 659.31, 712.56, 770.43,
833.36, 901.82, 976.36, 1057.6, 1146.8, 1243.8, 1350., 1466.3,
1593.6, 1732.7, 1884.1, 2049.1, 2228.2, 2421.9, 2629.4, 2853.7,
3094.4, 3351.1, 3622.3, 3829.8, 4123.1, 4438.3, 4777.2, 5144.1,
5545.4, 5990.5, 6404.5, 6996.8, 7687.6, 8482.9, 9349.4, 10203.,
11223., 12358., 13493., 14916., 16416., 18236., 20222., 22501.,
25102., 28358., 31707., 35404., 39538., 43911., 48391., 53193.,
58028., 58082., 61276., 64193., 66294., 67480., 67921., 67423.,
66254., 64341., 51737., 51420., 53072., 58145., 66195., 65358.,
67377., 67869., 53509., 50553., 35737., 32425., 21704., 19974.,
14457., 12142., 16798., 19489., 23049., 27270., 31910., 36457.,
40877., 44748., 47876., 59793., 58626., 55454., 50337., 44893.,
50228., 52216., 54747., 69541., 70455., 81014., 77694., 80533.,
```

```
73953., 70927., 65539., 59002., 52281., 45953., 40292., 35360.,
31124., 27478., 24346., 21647., 19308., 17271., 15491., 13927.,
12550., 11331., 10250., 9288.8, 8431.4, 7664.9, 6978.3, 6361.8,
5807.4, 5307.7, 4856.8, 4449., 4079.8, 3744.9, 3440.8, 3164.2,
2912.3, 2682.7, 2473., 2281.4, 2106., 1945.3, 1797.9, 1662.5,
1538.1, 1423.6, 1318.1, 1221., 1131.5, 1049., 972.99, 902.87,
838.01, 777.95, 722.2, 670.44, 622.35, 577.68, 536.21, 497.76,
462.12, 429.13, 398.61, 370.39, 344.29, 320.16, 297.85, 277.2,
258.08, 240.38, 223.97, 208.77, 194.66, 181.58, 169.43, 158.15,
147.67, 137.92, 128.86, 120.44, 112.6, 105.3, 98.499, 92.166,
86.264, 80.763, 75.632, 70.846, 66.381, 62.213, 58.321, 54.685,
51.288, 48.114, 45.145, 42.368, 39.772, 37.341, 35.065, 32.937,
30.943, 29.077, 27.33, 25.693, 24.158, 22.717, 21.367, 20.099,
18.909, 17.792, 16.744, 15.761, 14.838, 13.971, 13.157, 12.393,
11.676, 11.003, 10.369, 9.775, 9.2165, 8.6902, 8.1963, 7.7314,
7.2923, 6.8794, 6.4898, 6.122, 5.7764, 5.4525, 5.1484, 4.8611,
4.5918, 4.3379, 4.0982, 3.8716, 3.6567, 3.4545, 3.2634, 3.0828,
2.9122, 2.7512, 2.5993, 2.4561, 2.3211, 2.1938, 2.0737, 1.9603,
1.8534, 1.7525, 1.6572, 1.5673, 1.4824, 1.4022, 1.3265, 1.2551,
1.1876, 1.1239, 1.0637, 1.0069, .9532, .90248, .85454, .80921,
.76631, .72569, .6872, .65072, .61635, .5836, .55261, .52336,
.49581, .46998, .44559, .42236, .40036, .37929, .35924, .34043,
.32238, .30547, .28931, .27405, .25975, .24616, .23341, .22133,
.20997, .19924, .18917, .17967, .17075, .16211, .15411, .14646,
.13912, .13201, .12509, .11857, .11261, .10698, .10186, .097039,
.092236, .087844, .083443, .07938, .075452, .071564, .067931,
.064389, .061078, .057901, .054921, .052061, .049364, .046789,
.04435, .042044, .039866, .037808, .035863, .034023, .032282,
.030634, .029073, .027595, .026194, .024866, .023608, .022415,
.021283, .02021, .019193, .018228, .017312, .016443, .015619,
.014837, .014094, .01339, .012721, .012086, .011483, .010911,
.010368, .009852, .0093623, .0088972, .0084556, .0080362,
.0076379, .0072596, .0069003, .006559, .0062349, .0059269,
.0056344, .0053565, .0050925, .0048417, .0046034, .004377,
.0041618, .0039575, .0037633, .0035788, .0034034, .0032368,
.0030785, .002928, .0027851, .0026492, .0025201, .0023975,
.0022809, .0021701, .0020649, .0019649, .0018699, .0017796,
.0016938, .0016122, .0015348, .0014612, .0013913, .001325,
.0012619, .0012021, .0011452, .0010913, .0010401, 9.9149e-4,
9.454e-4, 9.0169e-4, 8.6024e-4, 8.2097e-4, 7.8377e-4, 7.4854e-4,
7.1522e-4, 6.8371e-4, 6.5393e-4, 6.2582e-4, 5.9932e-4, 5.7435e-4,
5.5087e-4, 5.2882e-4, 5.0814e-4, 4.8881e-4, 4.7076e-4, 4.5398e-4,
4.3843e-4, 4.2407e-4, 4.109e-4, 3.9888e-4, 3.88e-4, 3.7826e-4,
3.6963e-4, 3.6213e-4, 3.5575e-4, 3.505e-4, 3.464e-4, 3.4346e-4,
3.4173e-4, 3.4125e-4, 3.4206e-4, 3.4424e-4, 3.4787e-4, 3.5303e-4,
3.5986e-4, 3.6847e-4, 3.7903e-4, 3.9174e-4, 4.0681e-4, 4.2455e-4,
4.4527e-4, 4.6942e-4, 4.9637e-4, 5.2698e-4, 5.5808e-4, 5.9514e-4,
6.2757e-4, 6.689e-4, 7.1298e-4, 7.3955e-4, 7.8403e-4, 8.0449e-4,
8.5131e-4, 9.0256e-4, 9.3692e-4, .0010051, .0010846, .0011678,
.001282, .0014016, .0015355, .0016764, .0018272, .0020055,
.0021455, .0023421, .0024615, .0026786, .0028787, .0031259,
.0034046, .0036985, .0040917, .0043902, .0048349, .0049531,
.0052989, .0056148, .0052452, .0053357, .005333, .0045069,
.0043851, .004253, .003738, .0038084, .0039013, .0041505,
.0045372, .0050569, .0054507, .0061267, .0066122, .0072449,
.0078012, .0082651, .0076538, .0076573, .0076806, .0075227,
.0076269, .0063758, .006254, .0067749, .0067909, .0068231,
.0072143, .0072762, .0072954, .007679, .0075107, .0073658,
.0072441, .0071074, .0070378, .007176, .0072472, .0075844,
.0079291, .008412, .0090165, .010688, .011535, .012375, .013166,
.013895, .015567, .016011, .016392, .016737, .017043, .017731,
.018031, .018419, .018877, .019474, .019868, .020604, .021538,
.022653, .023869, .025288, .026879, .028547, .030524, .03274,
.035132, .03769, .040567, .043793, .047188, .049962, .053542,
.057205, .060776, .061489, .064419, .067124, .065945, .068487,
.071209, .074783, .077039, .082444, .08902, .09692, .10617,
.11687, .12952, .12362, .13498, .14412, .15492, .16519, .1744,
.17096, .17714, .18208, .17363, .17813, .18564, .18295, .19045,
.20252, .20815, .21844, .22929, .24229, .25321, .26588, .2797,
.29465, .31136, .32961, .36529, .38486, .41027, .43694, .4667,
.49943, .54542, .58348, .62303, .67633, .71755, .76054, .81371,
.85934, .90841, .96438, 1.0207, 1.0821, 1.1491, 1.2226, 1.3018,
1.388, 1.4818, 1.5835, 1.6939, 1.8137, 1.9435, 2.0843, 2.237,
2.4026, 2.5818, 2.7767, 2.9885, 3.2182, 3.4679, 3.7391, 4.0349,
4.3554, 4.7053, 5.0849, 5.4986, 5.9436, 6.4294, 6.9598, 7.5203,
8.143, 8.8253, 9.5568, 10.371, 11.267, 12.233, 13.31, 14.357,
15.598, 16.93, 18.358, 19.849, 21.408, 23.04, 24.706, 26.409,
28.153, 28.795, 30.549, 32.43, 34.49, 36.027, 38.955, 42.465,
46.565, 50.875, 55.378, 59.002, 63.882, 67.949, 73.693, 80.095,
86.403, 94.264, 102.65, 112.37, 123.3, 135.54, 149.14, 163.83,
179.17, 196.89, 217.91, 240.94, 264.13, 292.39, 324.83, 358.21,
397.16, 440.5, 488.6, 541.04, 595.3, 650.43, 652.03, 688.74,
719.47, 743.54, 757.68, 762.35, 756.43, 741.42, 595.43, 580.97,
580.83, 605.68, 667.88, 764.49, 759.93, 789.12, 798.17, 645.66,
615.65, 455.05, 421.09, 306.45, 289.14, 235.7, 215.52, 274.57,
316.53, 357.73, 409.89, 465.06, 521.84, 579.02, 630.64, 794.46,
813., 813.56, 796.25, 761.57, 727.97, 812.14, 866.75, 932.5,
```

```
    1132.8, 1194.8, 1362.2, 1387.2, 1482.3, 1479.7, 1517.9, 1533.1,
    1534.2, 1523.3, 1522.5, 1515.5, 1505.2, 1486.5, 1454., 1412.,
    1358.8, 1107.8, 1060.9, 1033.5, 1048.2, 1122.4, 1248.9, 1227.1,
    1255.4, 1058.9, 1020.7, 970.59, 715.24, 512.56, 468.47, 349.3,
    338.26, 299.22, 301.26, 332.38, 382.08, 445.49, 515.87, 590.85,
    662.3, 726.05, 955.59, 964.11, 945.17, 891.48, 807.11, 720.9,
    803.36, 834.46, 1073.9, 1107.1, 1123.6, 1296., 1393.7, 1303.1,
    1284.3, 1161.8, 1078.8, 976.13, 868.72, 767.4, 674.72, 593.73,
    523.12, 462.24, 409.75, 364.34, 325., 290.73, 260.76, 234.46,
    211.28, 190.78, 172.61, 156.44, 142.01, 129.12, 117.57, 107.2,
    97.877, 89.47, 81.882, 75.021, 68.807, 63.171, 58.052, 53.396,
    49.155, 45.288, 41.759, 38.531, 35.576, 32.868, 30.384, 28.102,
    26.003, 24.071, 22.293, 20.655, 19.147, 17.756, 16.476, 15.292,
    14.198, 13.183, 12.241, 11.367, 10.554, 9.7989, 9.0978, 8.4475,
    7.845, 7.2868, 6.7704, 6.2927, 5.8508, 5.4421, 5.064, 4.714,
    4.3902, 4.0902, 3.8121, 3.5543, 3.315, 3.093, 2.8869, 2.6953,
    2.5172, 2.3517, 2.1977, 2.0544, 1.9211, 1.7969, 1.6812, 1.5735,
    1.4731, 1.3794, 1.2921, 1.2107, 1.1346, 1.0637, .99744, .93554,
    .87771, .82368, .77313, .72587, .6816, .64014, .60134, .565,
    .53086, .49883, .46881, .44074, .4144, .38979, .36679, .34513,
    .32474, .30552, .28751, .27045, .25458, .23976, .22584, .21278,
    .20051, .18899, .17815, .16801, .15846, .14954, .14117, .13328,
    .12584
  };

  double xw, dw, ew, cw296, cw260, cw230, dt230, dt260, dt296, ctw, ctmpth;

  int iw;

  /* Get CO2 continuum absorption... */
  xw = nu / 2 + 1;
  if (xw >= 1 && xw < 2001) {
    iw = (int) xw;
    dw = xw - iw;
    ew = 1 - dw;
    cw296 = ew * co2296[iw - 1] + dw * co2296[iw];
    cw260 = ew * co2260[iw - 1] + dw * co2260[iw];
    cw230 = ew * co2230[iw - 1] + dw * co2230[iw];
    dt230 = t - 230;
    dt260 = t - 260;
    dt296 = t - 296;
    ctw = dt260 * 5.050505e-4 * dt296 * cw230 - dt230 * 9.259259e-4
      * dt296 * cw260 + dt230 * 4.208754e-4 * dt260 * cw296;
    ctmpth = u / GSL_CONST_NUM_AVOGADRO / 1000 * p / P0 * ctw;
  } else
    ctmpth = 0;
  return ctmpth;
}

/*****************************************************************************/

double ctmh2o(
  double nu,
  double p,
  double t,
  double q,
  double u) {

  static double h2o296[2001] = { .17, .1695, .172, .168, .1687, .1624, .1606,
    .1508, .1447, .1344, .1214, .1133, .1009, .09217, .08297, .06989,
    .06513, .05469, .05056, .04417, .03779, .03484, .02994, .0272,
    .02325, .02063, .01818, .01592, .01405, .01251, .0108, .009647,
    .008424, .007519, .006555, .00588, .005136, .004511, .003989,
    .003509, .003114, .00274, .002446, .002144, .001895, .001676,
    .001486, .001312, .001164, .001031, 9.129e-4, 8.106e-4, 7.213e-4,
    6.4e-4, 5.687e-4, 5.063e-4, 4.511e-4, 4.029e-4, 3.596e-4,
    3.22e-4, 2.889e-4, 2.597e-4, 2.337e-4, 2.108e-4, 1.907e-4,
    1.728e-4, 1.57e-4, 1.43e-4, 1.305e-4, 1.195e-4, 1.097e-4,
    1.009e-4, 9.307e-5, 8.604e-5, 7.971e-5, 7.407e-5, 6.896e-5,
    6.433e-5, 6.013e-5, 5.631e-5, 5.283e-5, 4.963e-5, 4.669e-5,
    4.398e-5, 4.148e-5, 3.917e-5, 3.702e-5, 3.502e-5, 3.316e-5,
    3.142e-5, 2.978e-5, 2.825e-5, 2.681e-5, 2.546e-5, 2.419e-5,
    2.299e-5, 2.186e-5, 2.079e-5, 1.979e-5, 1.884e-5, 1.795e-5,
    1.711e-5, 1.633e-5, 1.559e-5, 1.49e-5, 1.426e-5, 1.367e-5,
    1.312e-5, 1.263e-5, 1.218e-5, 1.178e-5, 1.143e-5, 1.112e-5,
    1.088e-5, 1.07e-5, 1.057e-5, 1.05e-5, 1.051e-5, 1.059e-5,
    1.076e-5, 1.1e-5, 1.133e-5, 1.18e-5, 1.237e-5, 1.308e-5,
    1.393e-5, 1.483e-5, 1.614e-5, 1.758e-5, 1.93e-5, 2.123e-5,
    2.346e-5, 2.647e-5, 2.93e-5, 3.279e-5, 3.745e-5, 4.152e-5,
    4.813e-5, 5.477e-5, 6.203e-5, 7.331e-5, 8.056e-5, 9.882e-5,
    1.05e-4, 1.21e-4, 1.341e-4, 1.572e-4, 1.698e-4, 1.968e-4,
    2.175e-4, 2.431e-4, 2.735e-4, 2.867e-4, 3.19e-4, 3.371e-4,
    3.554e-4, 3.726e-4, 3.837e-4, 3.878e-4, 3.864e-4, 3.858e-4,
    3.841e-4, 3.852e-4, 3.815e-4, 3.762e-4, 3.618e-4, 3.579e-4,
    3.45e-4, 3.202e-4, 3.018e-4, 2.785e-4, 2.602e-4, 2.416e-4,
    2.097e-4, 1.939e-4, 1.689e-4, 1.498e-4, 1.308e-4, 1.17e-4,
    1.011e-4, 9.237e-5, 7.909e-5, 7.006e-5, 6.112e-5, 5.401e-5,
```

```
4.914e-5, 4.266e-5, 3.963e-5, 3.316e-5, 3.037e-5, 2.598e-5,
2.294e-5, 2.066e-5, 1.813e-5, 1.583e-5, 1.423e-5, 1.247e-5,
1.116e-5, 9.76e-6, 8.596e-6, 7.72e-6, 6.825e-6, 6.108e-6,
5.366e-6, 4.733e-6, 4.229e-6, 3.731e-6, 3.346e-6, 2.972e-6,
2.628e-6, 2.356e-6, 2.102e-6, 1.878e-6, 1.678e-6, 1.507e-6,
1.348e-6, 1.21e-6, 1.089e-6, 9.806e-7, 8.857e-7, 8.004e-7,
7.261e-7, 6.599e-7, 6.005e-7, 5.479e-7, 5.011e-7, 4.595e-7,
4.219e-7, 3.885e-7, 3.583e-7, 3.314e-7, 3.071e-7, 2.852e-7,
2.654e-7, 2.474e-7, 2.311e-7, 2.162e-7, 2.026e-7, 1.902e-7,
1.788e-7, 1.683e-7, 1.587e-7, 1.497e-7, 1.415e-7, 1.338e-7,
1.266e-7, 1.2e-7, 1.138e-7, 1.08e-7, 1.027e-7, 9.764e-8,
9.296e-8, 8.862e-8, 8.458e-8, 8.087e-8, 7.744e-8, 7.429e-8,
7.145e-8, 6.893e-8, 6.664e-8, 6.468e-8, 6.322e-8, 6.162e-8,
6.07e-8, 5.992e-8, 5.913e-8, 5.841e-8, 5.796e-8, 5.757e-8,
5.746e-8, 5.731e-8, 5.679e-8, 5.577e-8, 5.671e-8, 5.656e-8,
5.594e-8, 5.593e-8, 5.602e-8, 5.62e-8, 5.693e-8, 5.725e-8,
5.858e-8, 6.037e-8, 6.249e-8, 6.535e-8, 6.899e-8, 7.356e-8,
7.918e-8, 8.618e-8, 9.385e-8, 1.039e-7, 1.158e-7, 1.29e-7,
1.437e-7, 1.65e-7, 1.871e-7, 2.121e-7, 2.427e-7, 2.773e-7,
3.247e-7, 3.677e-7, 4.037e-7, 4.776e-7, 5.101e-7, 6.214e-7,
6.936e-7, 7.581e-7, 8.486e-7, 9.355e-7, 9.942e-7, 1.063e-6,
1.123e-6, 1.191e-6, 1.215e-6, 1.247e-6, 1.26e-6, 1.271e-6,
1.284e-6, 1.317e-6, 1.323e-6, 1.349e-6, 1.353e-6, 1.362e-6,
1.344e-6, 1.329e-6, 1.336e-6, 1.327e-6, 1.325e-6, 1.359e-6,
1.374e-6, 1.415e-6, 1.462e-6, 1.526e-6, 1.619e-6, 1.735e-6,
1.863e-6, 2.034e-6, 2.265e-6, 2.482e-6, 2.756e-6, 3.103e-6,
3.466e-6, 3.832e-6, 4.378e-6, 4.913e-6, 5.651e-6, 6.311e-6,
7.169e-6, 8.057e-6, 9.253e-6, 1.047e-5, 1.212e-5, 1.36e-5,
1.569e-5, 1.776e-5, 2.02e-5, 2.281e-5, 2.683e-5, 2.994e-5,
3.488e-5, 3.896e-5, 4.499e-5, 5.175e-5, 6.035e-5, 6.34e-5,
7.281e-5, 7.923e-5, 8.348e-5, 9.631e-5, 1.044e-4, 1.102e-4,
1.176e-4, 1.244e-4, 1.283e-4, 1.326e-4, 1.4e-4, 1.395e-4,
1.387e-4, 1.363e-4, 1.314e-4, 1.241e-4, 1.228e-4, 1.148e-4,
1.086e-4, 1.018e-4, 8.89e-5, 8.316e-5, 7.292e-5, 6.452e-5,
5.625e-5, 5.045e-5, 4.38e-5, 3.762e-5, 3.29e-5, 2.836e-5,
2.485e-5, 2.168e-5, 1.895e-5, 1.659e-5, 1.453e-5, 1.282e-5,
1.132e-5, 1.001e-5, 8.836e-6, 7.804e-6, 6.922e-6, 6.116e-6,
5.429e-6, 4.824e-6, 4.278e-6, 3.788e-6, 3.371e-6, 2.985e-6,
2.649e-6, 2.357e-6, 2.09e-6, 1.858e-6, 1.647e-6, 1.462e-6,
1.299e-6, 1.155e-6, 1.028e-6, 9.142e-7, 8.132e-7, 7.246e-7,
6.451e-7, 5.764e-7, 5.151e-7, 4.603e-7, 4.121e-7, 3.694e-7,
3.318e-7, 2.985e-7, 2.69e-7, 2.428e-7, 2.197e-7, 1.992e-7,
1.81e-7, 1.649e-7, 1.506e-7, 1.378e-7, 1.265e-7, 1.163e-7,
1.073e-7, 9.918e-8, 9.191e-8, 8.538e-8, 7.949e-8, 7.419e-8,
6.94e-8, 6.508e-8, 6.114e-8, 5.761e-8, 5.437e-8, 5.146e-8,
4.89e-8, 4.636e-8, 4.406e-8, 4.201e-8, 4.015e-8, 3.84e-8,
3.661e-8, 3.51e-8, 3.377e-8, 3.242e-8, 3.13e-8, 3.015e-8,
2.918e-8, 2.83e-8, 2.758e-8, 2.707e-8, 2.656e-8, 2.619e-8,
2.609e-8, 2.615e-8, 2.63e-8, 2.675e-8, 2.745e-8, 2.842e-8,
2.966e-8, 3.125e-8, 3.318e-8, 3.565e-8, 3.85e-8, 4.191e-8,
4.59e-8, 5.059e-8, 5.607e-8, 6.239e-8, 6.958e-8, 7.796e-8,
8.773e-8, 9.88e-8, 1.114e-7, 1.258e-7, 1.422e-7, 1.61e-7,
1.822e-7, 2.06e-7, 2.337e-7, 2.645e-7, 2.996e-7, 3.393e-7,
3.843e-7, 4.363e-7, 4.935e-7, 5.607e-7, 6.363e-7, 7.242e-7,
8.23e-7, 9.411e-7, 1.071e-6, 1.232e-6, 1.402e-6, 1.6e-6, 1.82e-6,
2.128e-6, 2.386e-6, 2.781e-6, 3.242e-6, 3.653e-6, 4.323e-6,
4.747e-6, 5.321e-6, 5.919e-6, 6.681e-6, 7.101e-6, 7.983e-6,
8.342e-6, 8.741e-6, 9.431e-6, 9.952e-6, 1.026e-5, 1.055e-5,
1.095e-5, 1.095e-5, 1.087e-5, 1.056e-5, 1.026e-5, 9.715e-6,
9.252e-6, 8.452e-6, 7.958e-6, 7.268e-6, 6.295e-6, 6.003e-6, 5e-6,
4.591e-6, 3.983e-6, 3.479e-6, 3.058e-6, 2.667e-6, 2.293e-6,
1.995e-6, 1.747e-6, 1.517e-6, 1.335e-6, 1.165e-6, 1.028e-6,
9.007e-7, 7.956e-7, 7.015e-7, 6.192e-7, 5.491e-7, 4.859e-7,
4.297e-7, 3.799e-7, 3.38e-7, 3.002e-7, 2.659e-7, 2.366e-7,
2.103e-7, 1.861e-7, 1.655e-7, 1.469e-7, 1.309e-7, 1.162e-7,
1.032e-7, 9.198e-8, 8.181e-8, 7.294e-8, 6.516e-8, 5.787e-8,
5.163e-8, 4.612e-8, 4.119e-8, 3.695e-8, 3.308e-8, 2.976e-8,
2.67e-8, 2.407e-8, 2.171e-8, 1.965e-8, 1.78e-8, 1.617e-8,
1.47e-8, 1.341e-8, 1.227e-8, 1.125e-8, 1.033e-8, 9.524e-9,
8.797e-9, 8.162e-9, 7.565e-9, 7.04e-9, 6.56e-9, 6.129e-9,
5.733e-9, 5.376e-9, 5.043e-9, 4.75e-9, 4.466e-9, 4.211e-9,
3.977e-9, 3.759e-9, 3.558e-9, 3.373e-9, 3.201e-9, 3.043e-9,
2.895e-9, 2.76e-9, 2.635e-9, 2.518e-9, 2.411e-9, 2.314e-9,
2.23e-9, 2.151e-9, 2.087e-9, 2.035e-9, 1.988e-9, 1.946e-9,
1.927e-9, 1.916e-9, 1.916e-9, 1.933e-9, 1.966e-9, 2.018e-9,
2.09e-9, 2.182e-9, 2.299e-9, 2.442e-9, 2.623e-9, 2.832e-9,
3.079e-9, 3.368e-9, 3.714e-9, 4.104e-9, 4.567e-9, 5.091e-9,
5.701e-9, 6.398e-9, 7.194e-9, 8.127e-9, 9.141e-9, 1.035e-8,
1.177e-8, 1.338e-8, 1.508e-8, 1.711e-8, 1.955e-8, 2.216e-8,
2.534e-8, 2.871e-8, 3.291e-8, 3.711e-8, 4.285e-8, 4.868e-8,
5.509e-8, 6.276e-8, 7.262e-8, 8.252e-8, 9.4e-8, 1.064e-7,
1.247e-7, 1.411e-7, 1.626e-7, 1.827e-7, 2.044e-7, 2.284e-7,
2.452e-7, 2.854e-7, 3.026e-7, 3.278e-7, 3.474e-7, 3.693e-7,
3.93e-7, 4.104e-7, 4.22e-7, 4.439e-7, 4.545e-7, 4.778e-7,
4.812e-7, 5.018e-7, 4.899e-7, 5.075e-7, 5.073e-7, 5.171e-7,
5.131e-7, 5.25e-7, 5.617e-7, 5.846e-7, 6.239e-7, 6.696e-7,
7.398e-7, 8.073e-7, 9.15e-7, 1.009e-6, 1.116e-6, 1.264e-6,
```

```
1.439e-6, 1.644e-6, 1.856e-6, 2.147e-6, 2.317e-6, 2.713e-6,
2.882e-6, 2.99e-6, 3.489e-6, 3.581e-6, 4.033e-6, 4.26e-6,
4.543e-6, 4.84e-6, 4.826e-6, 5.013e-6, 5.252e-6, 5.277e-6,
5.306e-6, 5.236e-6, 5.123e-6, 5.171e-6, 4.843e-6, 4.615e-6,
4.385e-6, 3.97e-6, 3.693e-6, 3.231e-6, 2.915e-6, 2.495e-6,
2.144e-6, 1.91e-6, 1.639e-6, 1.417e-6, 1.226e-6, 1.065e-6,
9.29e-7, 8.142e-7, 7.161e-7, 6.318e-7, 5.581e-7, 4.943e-7,
4.376e-7, 3.884e-7, 3.449e-7, 3.06e-7, 2.712e-7, 2.412e-7,
2.139e-7, 1.903e-7, 1.689e-7, 1.499e-7, 1.331e-7, 1.183e-7,
1.05e-7, 9.362e-8, 8.306e-8, 7.403e-8, 6.578e-8, 5.853e-8,
5.216e-8, 4.632e-8, 4.127e-8, 3.678e-8, 3.279e-8, 2.923e-8,
2.612e-8, 2.339e-8, 2.094e-8, 1.877e-8, 1.686e-8, 1.516e-8,
1.366e-8, 1.234e-8, 1.114e-8, 1.012e-8, 9.182e-9, 8.362e-9,
7.634e-9, 6.981e-9, 6.406e-9, 5.888e-9, 5.428e-9, 5.021e-9,
4.65e-9, 4.326e-9, 4.033e-9, 3.77e-9, 3.536e-9, 3.327e-9,
3.141e-9, 2.974e-9, 2.825e-9, 2.697e-9, 2.584e-9, 2.488e-9,
2.406e-9, 2.34e-9, 2.292e-9, 2.259e-9, 2.244e-9, 2.243e-9,
2.272e-9, 2.31e-9, 2.378e-9, 2.454e-9, 2.618e-9, 2.672e-9,
2.831e-9, 3.05e-9, 3.225e-9, 3.425e-9, 3.677e-9, 3.968e-9,
4.221e-9, 4.639e-9, 4.96e-9, 5.359e-9, 5.649e-9, 6.23e-9,
6.716e-9, 7.218e-9, 7.746e-9, 7.988e-9, 8.627e-9, 8.999e-9,
9.442e-9, 9.82e-9, 1.015e-8, 1.06e-8, 1.079e-8, 1.109e-8,
1.137e-8, 1.186e-8, 1.18e-8, 1.187e-8, 1.194e-8, 1.192e-8,
1.224e-8, 1.245e-8, 1.246e-8, 1.318e-8, 1.377e-8, 1.471e-8,
1.582e-8, 1.713e-8, 1.853e-8, 2.063e-8, 2.27e-8, 2.567e-8,
2.891e-8, 3.264e-8, 3.744e-8, 4.286e-8, 4.915e-8, 5.623e-8,
6.336e-8, 7.293e-8, 8.309e-8, 9.319e-8, 1.091e-7, 1.243e-7,
1.348e-7, 1.449e-7, 1.62e-7, 1.846e-7, 1.937e-7, 2.04e-7,
2.179e-7, 2.298e-7, 2.433e-7, 2.439e-7, 2.464e-7, 2.611e-7,
2.617e-7, 2.582e-7, 2.453e-7, 2.401e-7, 2.349e-7, 2.203e-7,
2.066e-7, 1.939e-7, 1.78e-7, 1.558e-7, 1.391e-7, 1.203e-7,
1.048e-7, 9.464e-8, 8.306e-8, 7.239e-8, 6.317e-8, 5.52e-8,
4.847e-8, 4.282e-8, 3.796e-8, 3.377e-8, 2.996e-8, 2.678e-8,
2.4e-8, 2.134e-8, 1.904e-8, 1.705e-8, 1.523e-8, 1.35e-8,
1.204e-8, 1.07e-8, 9.408e-9, 8.476e-9, 7.47e-9, 6.679e-9,
5.929e-9, 5.267e-9, 4.711e-9, 4.172e-9, 3.761e-9, 3.288e-9,
2.929e-9, 2.609e-9, 2.315e-9, 2.042e-9, 1.844e-9, 1.64e-9,
1.47e-9, 1.31e-9, 1.176e-9, 1.049e-9, 9.377e-10, 8.462e-10,
7.616e-10, 6.854e-10, 6.191e-10, 5.596e-10, 5.078e-10, 4.611e-10,
4.197e-10, 3.83e-10, 3.505e-10, 3.215e-10, 2.956e-10, 2.726e-10,
2.521e-10, 2.338e-10, 2.173e-10, 2.026e-10, 1.895e-10, 1.777e-10,
1.672e-10, 1.579e-10, 1.496e-10, 1.423e-10, 1.358e-10, 1.302e-10,
1.254e-10, 1.216e-10, 1.187e-10, 1.163e-10, 1.147e-10, 1.145e-10,
1.15e-10, 1.17e-10, 1.192e-10, 1.25e-10, 1.298e-10, 1.345e-10,
1.405e-10, 1.538e-10, 1.648e-10, 1.721e-10, 1.872e-10, 1.968e-10,
2.089e-10, 2.172e-10, 2.317e-10, 2.389e-10, 2.503e-10, 2.585e-10,
2.686e-10, 2.8e-10, 2.895e-10, 3.019e-10, 3.037e-10, 3.076e-10,
3.146e-10, 3.198e-10, 3.332e-10, 3.397e-10, 3.54e-10, 3.667e-10,
3.895e-10, 4.071e-10, 4.565e-10, 4.983e-10, 5.439e-10, 5.968e-10,
6.676e-10, 7.456e-10, 8.405e-10, 9.478e-10, 1.064e-9, 1.218e-9,
1.386e-9, 1.581e-9, 1.787e-9, 2.032e-9, 2.347e-9, 2.677e-9,
3.008e-9, 3.544e-9, 4.056e-9, 4.687e-9, 5.331e-9, 6.227e-9,
6.854e-9, 8.139e-9, 8.945e-9, 9.865e-9, 1.125e-8, 1.178e-8,
1.364e-8, 1.436e-8, 1.54e-8, 1.672e-8, 1.793e-8, 1.906e-8,
2.036e-8, 2.144e-8, 2.292e-8, 2.371e-8, 2.493e-8, 2.606e-8,
2.706e-8, 2.866e-8, 3.036e-8, 3.136e-8, 3.405e-8, 3.665e-8,
3.837e-8, 4.229e-8, 4.748e-8, 5.32e-8, 5.763e-8, 6.677e-8,
7.216e-8, 7.716e-8, 8.958e-8, 9.419e-8, 1.036e-7, 1.108e-7,
1.189e-7, 1.246e-7, 1.348e-7, 1.31e-7, 1.361e-7, 1.364e-7,
1.363e-7, 1.343e-7, 1.293e-7, 1.254e-7, 1.235e-7, 1.158e-7,
1.107e-7, 9.961e-8, 9.011e-8, 7.91e-8, 6.916e-8, 6.338e-8,
5.564e-8, 4.827e-8, 4.198e-8, 3.695e-8, 3.276e-8, 2.929e-8,
2.633e-8, 2.391e-8, 2.192e-8, 2.021e-8, 1.89e-8, 1.772e-8,
1.667e-8, 1.603e-8, 1.547e-8, 1.537e-8, 1.492e-8, 1.515e-8,
1.479e-8, 1.45e-8, 1.513e-8, 1.495e-8, 1.529e-8, 1.565e-8,
1.564e-8, 1.553e-8, 1.569e-8, 1.584e-8, 1.57e-8, 1.538e-8,
1.513e-8, 1.472e-8, 1.425e-8, 1.349e-8, 1.328e-8, 1.249e-8,
1.17e-8, 1.077e-8, 9.514e-9, 8.614e-9, 7.46e-9, 6.621e-9,
5.775e-9, 5.006e-9, 4.308e-9, 3.747e-9, 3.24e-9, 2.84e-9,
2.481e-9, 2.184e-9, 1.923e-9, 1.71e-9, 1.504e-9, 1.334e-9,
1.187e-9, 1.053e-9, 9.367e-10, 8.306e-10, 7.419e-10, 6.63e-10,
5.918e-10, 5.277e-10, 4.717e-10, 4.222e-10, 3.783e-10, 3.39e-10,
3.036e-10, 2.729e-10, 2.455e-10, 2.211e-10, 1.995e-10, 1.804e-10,
1.635e-10, 1.485e-10, 1.355e-10, 1.24e-10, 1.139e-10, 1.051e-10,
9.757e-11, 9.114e-11, 8.577e-11, 8.139e-11, 7.792e-11, 7.52e-11,
7.39e-11, 7.311e-11, 7.277e-11, 7.482e-11, 7.698e-11, 8.162e-11,
8.517e-11, 8.968e-11, 9.905e-11, 1.075e-10, 1.187e-10, 1.291e-10,
1.426e-10, 1.573e-10, 1.734e-10, 1.905e-10, 2.097e-10, 2.28e-10,
2.473e-10, 2.718e-10, 2.922e-10, 3.128e-10, 3.361e-10, 3.641e-10,
3.91e-10, 4.196e-10, 4.501e-10, 4.932e-10, 5.258e-10, 5.755e-10,
6.253e-10, 6.664e-10, 7.344e-10, 7.985e-10, 8.877e-10, 1.005e-9,
1.118e-9, 1.251e-9, 1.428e-9, 1.61e-9, 1.888e-9, 2.077e-9,
2.331e-9, 2.751e-9, 3.061e-9, 3.522e-9, 3.805e-9, 4.181e-9,
4.575e-9, 5.167e-9, 5.634e-9, 6.007e-9, 6.501e-9, 6.829e-9,
7.211e-9, 7.262e-9, 7.696e-9, 7.832e-9, 7.799e-9, 7.651e-9,
7.304e-9, 7.15e-9, 6.977e-9, 6.603e-9, 6.209e-9, 5.69e-9,
5.432e-9, 4.764e-9, 4.189e-9, 3.64e-9, 3.203e-9, 2.848e-9,
```

```
    2.51e-9, 2.194e-9, 1.946e-9, 1.75e-9, 1.567e-9, 1.426e-9,
    1.302e-9, 1.197e-9, 1.109e-9, 1.035e-9, 9.719e-10, 9.207e-10,
    8.957e-10, 8.578e-10, 8.262e-10, 8.117e-10, 7.987e-10, 7.875e-10,
    7.741e-10, 7.762e-10, 7.537e-10, 7.424e-10, 7.474e-10, 7.294e-10,
    7.216e-10, 7.233e-10, 7.075e-10, 6.892e-10, 6.618e-10, 6.314e-10,
    6.208e-10, 5.689e-10, 5.55e-10, 4.984e-10, 4.6e-10, 4.078e-10,
    3.879e-10, 3.459e-10, 2.982e-10, 2.626e-10, 2.329e-10, 1.988e-10,
    1.735e-10, 1.487e-10, 1.297e-10, 1.133e-10, 9.943e-11, 8.736e-11,
    7.726e-11, 6.836e-11, 6.053e-11, 5.384e-11, 4.789e-11, 4.267e-11,
    3.804e-11, 3.398e-11, 3.034e-11, 2.71e-11, 2.425e-11, 2.173e-11,
    1.95e-11, 1.752e-11, 1.574e-11, 1.418e-11, 1.278e-11, 1.154e-11,
    1.044e-11, 9.463e-12, 8.602e-12, 7.841e-12, 7.171e-12, 6.584e-12,
    6.073e-12, 5.631e-12, 5.254e-12, 4.937e-12, 4.679e-12, 4.476e-12,
    4.328e-12, 4.233e-12, 4.194e-12, 4.211e-12, 4.286e-12, 4.424e-12,
    4.628e-12, 4.906e-12, 5.262e-12, 5.708e-12, 6.254e-12, 6.914e-12,
    7.714e-12, 8.677e-12, 9.747e-12, 1.101e-11, 1.256e-11, 1.409e-11,
    1.597e-11, 1.807e-11, 2.034e-11, 2.316e-11, 2.622e-11, 2.962e-11,
    3.369e-11, 3.819e-11, 4.329e-11, 4.932e-11, 5.589e-11, 6.364e-11,
    7.284e-11, 8.236e-11, 9.447e-11, 1.078e-10, 1.229e-10, 1.417e-10,
    1.614e-10, 1.843e-10, 2.107e-10, 2.406e-10, 2.728e-10, 3.195e-10,
    3.595e-10, 4.153e-10, 4.736e-10, 5.41e-10, 6.088e-10, 6.769e-10,
    7.691e-10, 8.545e-10, 9.621e-10, 1.047e-9, 1.161e-9, 1.296e-9,
    1.424e-9, 1.576e-9, 1.739e-9, 1.893e-9, 2.08e-9, 2.336e-9,
    2.604e-9, 2.76e-9, 3.001e-9, 3.365e-9, 3.55e-9, 3.895e-9,
    4.183e-9, 4.614e-9, 4.846e-9, 5.068e-9, 5.427e-9, 5.541e-9,
    5.864e-9, 5.997e-9, 5.997e-9, 6.061e-9, 5.944e-9, 5.855e-9,
    5.661e-9, 5.523e-9, 5.374e-9, 4.94e-9, 4.688e-9, 4.17e-9,
    3.913e-9, 3.423e-9, 2.997e-9, 2.598e-9, 2.253e-9, 1.946e-9,
    1.71e-9, 1.507e-9, 1.336e-9, 1.19e-9, 1.068e-9, 9.623e-10,
    8.772e-10, 8.007e-10, 7.42e-10, 6.884e-10, 6.483e-10, 6.162e-10,
    5.922e-10, 5.688e-10, 5.654e-10, 5.637e-10, 5.701e-10, 5.781e-10,
    5.874e-10, 6.268e-10, 6.357e-10, 6.525e-10, 7.137e-10, 7.441e-10,
    8.024e-10, 8.485e-10, 9.143e-10, 9.536e-10, 9.717e-10, 1.018e-9,
    1.042e-9, 1.054e-9, 1.092e-9, 1.079e-9, 1.064e-9, 1.043e-9,
    1.02e-9, 9.687e-10, 9.273e-10, 9.208e-10, 9.068e-10, 7.687e-10,
    7.385e-10, 6.595e-10, 5.87e-10, 5.144e-10, 4.417e-10, 3.804e-10,
    3.301e-10, 2.866e-10, 2.509e-10, 2.202e-10, 1.947e-10, 1.719e-10,
    1.525e-10, 1.361e-10, 1.21e-10, 1.084e-10, 9.8e-11, 8.801e-11,
    7.954e-11, 7.124e-11, 6.335e-11, 5.76e-11, 5.132e-11, 4.601e-11,
    4.096e-11, 3.657e-11, 3.25e-11, 2.909e-11, 2.587e-11, 2.297e-11,
    2.05e-11, 1.828e-11, 1.632e-11, 1.462e-11, 1.314e-11, 1.185e-11,
    1.073e-11, 9.76e-12, 8.922e-12, 8.206e-12, 7.602e-12, 7.1e-12,
    6.694e-12, 6.378e-12, 6.149e-12, 6.004e-12, 5.941e-12, 5.962e-12,
    6.069e-12, 6.265e-12, 6.551e-12, 6.935e-12, 7.457e-12, 8.074e-12,
    8.811e-12, 9.852e-12, 1.086e-11, 1.207e-11, 1.361e-11, 1.553e-11,
    1.737e-11, 1.93e-11, 2.175e-11, 2.41e-11, 2.706e-11, 3.023e-11,
    3.313e-11, 3.657e-11, 4.118e-11, 4.569e-11, 5.025e-11, 5.66e-11,
    6.231e-11, 6.881e-11, 7.996e-11, 8.526e-11, 9.694e-11, 1.106e-10,
    1.222e-10, 1.355e-10, 1.525e-10, 1.775e-10, 1.924e-10, 2.181e-10,
    2.379e-10, 2.662e-10, 2.907e-10, 3.154e-10, 3.366e-10, 3.579e-10,
    3.858e-10, 4.046e-10, 4.196e-10, 4.166e-10, 4.457e-10, 4.466e-10,
    4.404e-10, 4.337e-10, 4.15e-10, 4.083e-10, 3.91e-10, 3.723e-10,
    3.514e-10, 3.303e-10, 2.847e-10, 2.546e-10, 2.23e-10, 1.994e-10,
    1.733e-10, 1.488e-10, 1.297e-10, 1.144e-10, 1.004e-10, 8.741e-11,
    7.928e-11, 7.034e-11, 6.323e-11, 5.754e-11, 5.25e-11, 4.85e-11,
    4.502e-11, 4.286e-11, 4.028e-11, 3.899e-11, 3.824e-11, 3.761e-11,
    3.804e-11, 3.839e-11, 3.845e-11, 4.244e-11, 4.382e-11, 4.582e-11,
    4.847e-11, 5.209e-11, 5.384e-11, 5.887e-11, 6.371e-11, 6.737e-11,
    7.168e-11, 7.415e-11, 7.827e-11, 8.037e-11, 8.12e-11, 8.071e-11,
    8.008e-11, 7.851e-11, 7.544e-11, 7.377e-11, 7.173e-11, 6.801e-11,
    6.267e-11, 5.727e-11, 5.288e-11, 4.853e-11, 4.082e-11, 3.645e-11,
    3.136e-11, 2.672e-11, 2.304e-11, 1.986e-11, 1.725e-11, 1.503e-11,
    1.315e-11, 1.153e-11, 1.014e-11, 8.942e-12, 7.901e-12, 6.993e-12,
    6.199e-12, 5.502e-12, 4.89e-12, 4.351e-12, 3.878e-12, 3.461e-12,
    3.094e-12, 2.771e-12, 2.488e-12, 2.241e-12, 2.025e-12, 1.838e-12,
    1.677e-12, 1.541e-12, 1.427e-12, 1.335e-12, 1.262e-12, 1.209e-12,
    1.176e-12, 1.161e-12, 1.165e-12, 1.189e-12, 1.234e-12, 1.3e-12,
    1.389e-12, 1.503e-12, 1.644e-12, 1.814e-12, 2.017e-12, 2.255e-12,
    2.534e-12, 2.858e-12, 3.231e-12, 3.661e-12, 4.153e-12, 4.717e-12,
    5.36e-12, 6.094e-12, 6.93e-12, 7.882e-12, 8.966e-12, 1.02e-11,
    1.162e-11, 1.324e-11, 1.51e-11, 1.72e-11, 1.965e-11, 2.237e-11,
    2.56e-11, 2.927e-11, 3.371e-11, 3.842e-11, 4.429e-11, 5.139e-11,
    5.798e-11, 6.697e-11, 7.626e-11, 8.647e-11, 1.022e-10, 1.136e-10,
    1.3e-10, 1.481e-10, 1.672e-10, 1.871e-10, 2.126e-10, 2.357e-10,
    2.583e-10, 2.997e-10, 3.289e-10, 3.702e-10, 4.012e-10, 4.319e-10,
    4.527e-10, 5.001e-10, 5.448e-10, 5.611e-10, 5.76e-10, 5.965e-10,
    6.079e-10, 6.207e-10, 6.276e-10, 6.222e-10, 6.137e-10, 6e-10,
    5.814e-10, 5.393e-10, 5.35e-10, 4.947e-10, 4.629e-10, 4.117e-10,
    3.712e-10, 3.372e-10, 2.923e-10, 2.55e-10, 2.232e-10, 1.929e-10,
    1.679e-10, 1.46e-10, 1.289e-10, 1.13e-10, 9.953e-11, 8.763e-11,
    7.76e-11, 6.9e-11, 6.16e-11, 5.525e-11, 4.958e-11, 4.489e-11,
    4.072e-11, 3.728e-11, 3.438e-11, 3.205e-11, 3.006e-11, 2.848e-11,
    2.766e-11, 2.688e-11, 2.664e-11, 2.67e-11, 2.696e-11, 2.786e-11,
    2.861e-11, 3.009e-11, 3.178e-11, 3.389e-11, 3.587e-11, 3.819e-11,
    4.054e-11, 4.417e-11, 4.703e-11, 5.137e-11, 5.46e-11, 6.055e-11,
    6.333e-11, 6.773e-11, 7.219e-11, 7.717e-11, 8.131e-11, 8.491e-11,
    8.574e-11, 9.01e-11, 9.017e-11, 8.999e-11, 8.959e-11, 8.838e-11,
```

```
    8.579e-11, 8.162e-11, 8.098e-11, 7.472e-11, 7.108e-11, 6.559e-11,
    5.994e-11, 5.172e-11, 4.424e-11, 3.951e-11, 3.34e-11, 2.902e-11,
    2.541e-11, 2.215e-11, 1.945e-11, 1.716e-11, 1.503e-11, 1.339e-11,
    1.185e-11, 1.05e-11, 9.336e-12, 8.307e-12, 7.312e-12, 6.55e-12,
    5.836e-12, 5.178e-12, 4.6e-12, 4.086e-12, 3.639e-12, 3.247e-12,
    2.904e-12, 2.604e-12, 2.341e-12, 2.112e-12, 1.914e-12, 1.744e-12,
    1.598e-12, 1.476e-12, 1.374e-12, 1.293e-12, 1.23e-12, 1.185e-12,
    1.158e-12, 1.147e-12, 1.154e-12, 1.177e-12, 1.219e-12, 1.28e-12,
    1.36e-12, 1.463e-12, 1.591e-12, 1.75e-12, 1.94e-12, 2.156e-12,
    2.43e-12, 2.748e-12, 3.052e-12, 3.533e-12, 3.967e-12, 4.471e-12,
    5.041e-12, 5.86e-12, 6.664e-12, 7.522e-12, 8.342e-12, 9.412e-12,
    1.072e-11, 1.213e-11, 1.343e-11, 1.496e-11, 1.664e-11, 1.822e-11,
    2.029e-11, 2.233e-11, 2.457e-11, 2.709e-11, 2.928e-11, 3.115e-11,
    3.356e-11, 3.592e-11, 3.818e-11, 3.936e-11, 4.061e-11, 4.149e-11,
    4.299e-11, 4.223e-11, 4.251e-11, 4.287e-11, 4.177e-11, 4.094e-11,
    3.942e-11, 3.772e-11, 3.614e-11, 3.394e-11, 3.222e-11, 2.791e-11,
    2.665e-11, 2.309e-11, 2.032e-11, 1.74e-11, 1.535e-11, 1.323e-11,
    1.151e-11, 9.803e-12, 8.65e-12, 7.54e-12, 6.619e-12, 5.832e-12,
    5.113e-12, 4.503e-12, 3.975e-12, 3.52e-12, 3.112e-12, 2.797e-12,
    2.5e-12, 2.24e-12, 2.013e-12, 1.819e-12, 1.653e-12, 1.513e-12,
    1.395e-12, 1.299e-12, 1.225e-12, 1.168e-12, 1.124e-12, 1.148e-12,
    1.107e-12, 1.128e-12, 1.169e-12, 1.233e-12, 1.307e-12, 1.359e-12,
    1.543e-12, 1.686e-12, 1.794e-12, 2.028e-12, 2.21e-12, 2.441e-12,
    2.653e-12, 2.828e-12, 3.093e-12, 3.28e-12, 3.551e-12, 3.677e-12,
    3.803e-12, 3.844e-12, 4.068e-12, 4.093e-12, 4.002e-12, 3.904e-12,
    3.624e-12, 3.633e-12, 3.622e-12, 3.443e-12, 3.184e-12, 2.934e-12,
    2.476e-12, 2.212e-12, 1.867e-12, 1.594e-12, 1.37e-12, 1.192e-12,
    1.045e-12, 9.211e-13, 8.17e-13, 7.29e-13, 6.55e-13, 5.929e-13,
    5.415e-13, 4.995e-13, 4.661e-13, 4.406e-13, 4.225e-13, 4.116e-13,
    4.075e-13, 4.102e-13, 4.198e-13, 4.365e-13, 4.606e-13, 4.925e-13,
    5.326e-13, 5.818e-13, 6.407e-13, 7.104e-13, 7.92e-13, 8.868e-13,
    9.964e-13, 1.123e-12, 1.268e-12, 1.434e-12, 1.626e-12, 1.848e-12,
    2.107e-12, 2.422e-12, 2.772e-12, 3.145e-12, 3.704e-12, 4.27e-12,
    4.721e-12, 5.361e-12, 6.083e-12, 7.095e-12, 7.968e-12, 9.228e-12,
    1.048e-11, 1.187e-11, 1.336e-11, 1.577e-11, 1.772e-11, 2.017e-11,
    2.25e-11, 2.63e-11, 2.911e-11, 3.356e-11, 3.82e-11, 4.173e-11,
    4.811e-11, 5.254e-11, 5.839e-11, 6.187e-11, 6.805e-11, 7.118e-11,
    7.369e-11, 7.664e-11, 7.794e-11, 7.947e-11, 8.036e-11, 7.954e-11,
    7.849e-11, 7.518e-11, 7.462e-11, 6.926e-11, 6.531e-11, 6.197e-11,
    5.421e-11, 4.777e-11, 4.111e-11, 3.679e-11, 3.166e-11, 2.786e-11,
    2.436e-11, 2.144e-11, 1.859e-11, 1.628e-11, 1.414e-11, 1.237e-11,
    1.093e-11, 9.558e-12
  };

  static double h2o260[2001] = { .2752, .2732, .2749, .2676, .2667, .2545,
    .2497, .2327, .2218, .2036, .1825, .1694, .1497, .1353, .121,
    .1014, .09405, .07848, .07195, .06246, .05306, .04853, .04138,
    .03735, .03171, .02785, .02431, .02111, .01845, .0164, .01405,
    .01255, .01098, .009797, .008646, .007779, .006898, .006099,
    .005453, .004909, .004413, .003959, .003581, .003199, .002871,
    .002583, .00233, .002086, .001874, .001684, .001512, .001361,
    .001225, .0011, 9.89e-4, 8.916e-4, 8.039e-4, 7.256e-4, 6.545e-4,
    5.918e-4, 5.359e-4, 4.867e-4, 4.426e-4, 4.033e-4, 3.682e-4,
    3.366e-4, 3.085e-4, 2.833e-4, 2.605e-4, 2.403e-4, 2.221e-4,
    2.055e-4, 1.908e-4, 1.774e-4, 1.653e-4, 1.544e-4, 1.443e-4,
    1.351e-4, 1.267e-4, 1.19e-4, 1.119e-4, 1.053e-4, 9.922e-5,
    9.355e-5, 8.831e-5, 8.339e-5, 7.878e-5, 7.449e-5, 7.043e-5,
    6.664e-5, 6.307e-5, 5.969e-5, 5.654e-5, 5.357e-5, 5.075e-5,
    4.81e-5, 4.56e-5, 4.322e-5, 4.102e-5, 3.892e-5, 3.696e-5,
    3.511e-5, 3.339e-5, 3.177e-5, 3.026e-5, 2.886e-5, 2.756e-5,
    2.636e-5, 2.527e-5, 2.427e-5, 2.337e-5, 2.257e-5, 2.185e-5,
    2.127e-5, 2.08e-5, 2.041e-5, 2.013e-5, 2e-5, 1.997e-5, 2.009e-5,
    2.031e-5, 2.068e-5, 2.124e-5, 2.189e-5, 2.267e-5, 2.364e-5,
    2.463e-5, 2.618e-5, 2.774e-5, 2.937e-5, 3.144e-5, 3.359e-5,
    3.695e-5, 4.002e-5, 4.374e-5, 4.947e-5, 5.431e-5, 6.281e-5,
    7.169e-5, 8.157e-5, 9.728e-5, 1.079e-4, 1.337e-4, 1.442e-4,
    1.683e-4, 1.879e-4, 2.223e-4, 2.425e-4, 2.838e-4, 3.143e-4,
    3.527e-4, 4.012e-4, 4.237e-4, 4.747e-4, 5.057e-4, 5.409e-4,
    5.734e-4, 5.944e-4, 6.077e-4, 6.175e-4, 6.238e-4, 6.226e-4,
    6.248e-4, 6.192e-4, 6.098e-4, 5.818e-4, 5.709e-4, 5.465e-4,
    5.043e-4, 4.699e-4, 4.294e-4, 3.984e-4, 3.672e-4, 3.152e-4,
    2.883e-4, 2.503e-4, 2.211e-4, 1.92e-4, 1.714e-4, 1.485e-4,
    1.358e-4, 1.156e-4, 1.021e-4, 8.887e-5, 7.842e-5, 7.12e-5,
    6.186e-5, 5.73e-5, 4.792e-5, 4.364e-5, 3.72e-5, 3.28e-5,
    2.946e-5, 2.591e-5, 2.261e-5, 2.048e-5, 1.813e-5, 1.63e-5,
    1.447e-5, 1.282e-5, 1.167e-5, 1.041e-5, 9.449e-6, 8.51e-6,
    7.596e-6, 6.961e-6, 6.272e-6, 5.728e-6, 5.198e-6, 4.667e-6,
    4.288e-6, 3.897e-6, 3.551e-6, 3.235e-6, 2.952e-6, 2.688e-6,
    2.449e-6, 2.241e-6, 2.05e-6, 1.879e-6, 1.722e-6, 1.582e-6,
    1.456e-6, 1.339e-6, 1.236e-6, 1.144e-6, 1.06e-6, 9.83e-7,
    9.149e-7, 8.535e-7, 7.973e-7, 7.466e-7, 6.999e-7, 6.574e-7,
    6.18e-7, 5.821e-7, 5.487e-7, 5.18e-7, 4.896e-7, 4.631e-7,
    4.386e-7, 4.16e-7, 3.945e-7, 3.748e-7, 3.562e-7, 3.385e-7,
    3.222e-7, 3.068e-7, 2.922e-7, 2.788e-7, 2.659e-7, 2.539e-7,
    2.425e-7, 2.318e-7, 2.219e-7, 2.127e-7, 2.039e-7, 1.958e-7,
    1.885e-7, 1.818e-7, 1.758e-7, 1.711e-7, 1.662e-7, 1.63e-7,
    1.605e-7, 1.58e-7, 1.559e-7, 1.545e-7, 1.532e-7, 1.522e-7,
```

```
1.51e-7, 1.495e-7, 1.465e-7, 1.483e-7, 1.469e-7, 1.448e-7,
1.444e-7, 1.436e-7, 1.426e-7, 1.431e-7, 1.425e-7, 1.445e-7,
1.477e-7, 1.515e-7, 1.567e-7, 1.634e-7, 1.712e-7, 1.802e-7,
1.914e-7, 2.024e-7, 2.159e-7, 2.295e-7, 2.461e-7, 2.621e-7,
2.868e-7, 3.102e-7, 3.394e-7, 3.784e-7, 4.223e-7, 4.864e-7,
5.501e-7, 6.039e-7, 7.193e-7, 7.728e-7, 9.514e-7, 1.073e-6,
1.18e-6, 1.333e-6, 1.472e-6, 1.566e-6, 1.677e-6, 1.784e-6,
1.904e-6, 1.953e-6, 2.02e-6, 2.074e-6, 2.128e-6, 2.162e-6,
2.219e-6, 2.221e-6, 2.249e-6, 2.239e-6, 2.235e-6, 2.185e-6,
2.141e-6, 2.124e-6, 2.09e-6, 2.068e-6, 2.1e-6, 2.104e-6,
2.142e-6, 2.181e-6, 2.257e-6, 2.362e-6, 2.5e-6, 2.664e-6,
2.884e-6, 3.189e-6, 3.48e-6, 3.847e-6, 4.313e-6, 4.79e-6,
5.25e-6, 5.989e-6, 6.692e-6, 7.668e-6, 8.52e-6, 9.606e-6,
1.073e-5, 1.225e-5, 1.377e-5, 1.582e-5, 1.761e-5, 2.029e-5,
2.284e-5, 2.602e-5, 2.94e-5, 3.483e-5, 3.928e-5, 4.618e-5,
5.24e-5, 6.132e-5, 7.183e-5, 8.521e-5, 9.111e-5, 1.07e-4,
1.184e-4, 1.264e-4, 1.475e-4, 1.612e-4, 1.704e-4, 1.818e-4,
1.924e-4, 1.994e-4, 2.061e-4, 2.18e-4, 2.187e-4, 2.2e-4,
2.196e-4, 2.131e-4, 2.015e-4, 1.988e-4, 1.847e-4, 1.729e-4,
1.597e-4, 1.373e-4, 1.262e-4, 1.087e-4, 9.439e-5, 8.061e-5,
7.093e-5, 6.049e-5, 5.12e-5, 4.435e-5, 3.817e-5, 3.34e-5,
2.927e-5, 2.573e-5, 2.291e-5, 2.04e-5, 1.827e-5, 1.636e-5,
1.463e-5, 1.309e-5, 1.17e-5, 1.047e-5, 9.315e-6, 8.328e-6,
7.458e-6, 6.665e-6, 5.94e-6, 5.316e-6, 4.752e-6, 4.252e-6,
3.825e-6, 3.421e-6, 3.064e-6, 2.746e-6, 2.465e-6, 2.216e-6,
1.99e-6, 1.79e-6, 1.609e-6, 1.449e-6, 1.306e-6, 1.177e-6,
1.063e-6, 9.607e-7, 8.672e-7, 7.855e-7, 7.118e-7, 6.46e-7,
5.871e-7, 5.34e-7, 4.868e-7, 4.447e-7, 4.068e-7, 3.729e-7,
3.423e-7, 3.151e-7, 2.905e-7, 2.686e-7, 2.484e-7, 2.306e-7,
2.142e-7, 1.995e-7, 1.86e-7, 1.738e-7, 1.626e-7, 1.522e-7,
1.427e-7, 1.338e-7, 1.258e-7, 1.183e-7, 1.116e-7, 1.056e-7,
9.972e-8, 9.46e-8, 9.007e-8, 8.592e-8, 8.195e-8, 7.816e-8,
7.483e-8, 7.193e-8, 6.892e-8, 6.642e-8, 6.386e-8, 6.154e-8,
5.949e-8, 5.764e-8, 5.622e-8, 5.479e-8, 5.364e-8, 5.301e-8,
5.267e-8, 5.263e-8, 5.313e-8, 5.41e-8, 5.55e-8, 5.745e-8,
6.003e-8, 6.311e-8, 6.713e-8, 7.173e-8, 7.724e-8, 8.368e-8,
9.121e-8, 9.986e-8, 1.097e-7, 1.209e-7, 1.338e-7, 1.486e-7,
1.651e-7, 1.837e-7, 2.048e-7, 2.289e-7, 2.557e-7, 2.857e-7,
3.195e-7, 3.587e-7, 4.015e-7, 4.497e-7, 5.049e-7, 5.665e-7,
6.366e-7, 7.121e-7, 7.996e-7, 8.946e-7, 1.002e-6, 1.117e-6,
1.262e-6, 1.416e-6, 1.611e-6, 1.807e-6, 2.056e-6, 2.351e-6,
2.769e-6, 3.138e-6, 3.699e-6, 4.386e-6, 5.041e-6, 6.074e-6,
6.812e-6, 7.79e-6, 8.855e-6, 1.014e-5, 1.095e-5, 1.245e-5,
1.316e-5, 1.39e-5, 1.504e-5, 1.583e-5, 1.617e-5, 1.652e-5,
1.713e-5, 1.724e-5, 1.715e-5, 1.668e-5, 1.629e-5, 1.552e-5,
1.478e-5, 1.34e-5, 1.245e-5, 1.121e-5, 9.575e-6, 8.956e-6,
7.345e-6, 6.597e-6, 5.612e-6, 4.818e-6, 4.165e-6, 3.579e-6,
3.041e-6, 2.623e-6, 2.29e-6, 1.984e-6, 1.748e-6, 1.534e-6,
1.369e-6, 1.219e-6, 1.092e-6, 9.8e-7, 8.762e-7, 7.896e-7,
7.104e-7, 6.364e-7, 5.691e-7, 5.107e-7, 4.575e-7, 4.09e-7,
3.667e-7, 3.287e-7, 2.931e-7, 2.633e-7, 2.356e-7, 2.111e-7,
1.895e-7, 1.697e-7, 1.525e-7, 1.369e-7, 1.233e-7, 1.114e-7,
9.988e-8, 9.004e-8, 8.149e-8, 7.352e-8, 6.662e-8, 6.03e-8,
5.479e-8, 4.974e-8, 4.532e-8, 4.129e-8, 3.781e-8, 3.462e-8,
3.176e-8, 2.919e-8, 2.687e-8, 2.481e-8, 2.292e-8, 2.119e-8,
1.967e-8, 1.828e-8, 1.706e-8, 1.589e-8, 1.487e-8, 1.393e-8,
1.307e-8, 1.228e-8, 1.156e-8, 1.089e-8, 1.028e-8, 9.696e-9,
9.159e-9, 8.658e-9, 8.187e-9, 7.746e-9, 7.34e-9, 6.953e-9,
6.594e-9, 6.259e-9, 5.948e-9, 5.66e-9, 5.386e-9, 5.135e-9,
4.903e-9, 4.703e-9, 4.515e-9, 4.362e-9, 4.233e-9, 4.117e-9,
4.017e-9, 3.962e-9, 3.924e-9, 3.905e-9, 3.922e-9, 3.967e-9,
4.046e-9, 4.165e-9, 4.32e-9, 4.522e-9, 4.769e-9, 5.083e-9,
5.443e-9, 5.872e-9, 6.366e-9, 6.949e-9, 7.601e-9, 8.371e-9,
9.22e-9, 1.02e-8, 1.129e-8, 1.251e-8, 1.393e-8, 1.542e-8,
1.72e-8, 1.926e-8, 2.152e-8, 2.392e-8, 2.678e-8, 3.028e-8,
3.39e-8, 3.836e-8, 4.309e-8, 4.9e-8, 5.481e-8, 6.252e-8,
7.039e-8, 7.883e-8, 8.849e-8, 1.012e-7, 1.142e-7, 1.3e-7,
1.475e-7, 1.732e-7, 1.978e-7, 2.304e-7, 2.631e-7, 2.988e-7,
3.392e-7, 3.69e-7, 4.355e-7, 4.672e-7, 5.11e-7, 5.461e-7,
5.828e-7, 6.233e-7, 6.509e-7, 6.672e-7, 6.969e-7, 7.104e-7,
7.439e-7, 7.463e-7, 7.708e-7, 7.466e-7, 7.668e-7, 7.549e-7,
7.586e-7, 7.384e-7, 7.439e-7, 7.785e-7, 7.915e-7, 8.31e-7,
8.745e-7, 9.558e-7, 1.038e-6, 1.173e-6, 1.304e-6, 1.452e-6,
1.671e-6, 1.931e-6, 2.239e-6, 2.578e-6, 3.032e-6, 3.334e-6,
3.98e-6, 4.3e-6, 4.518e-6, 5.321e-6, 5.508e-6, 6.211e-6, 6.59e-6,
7.046e-6, 7.555e-6, 7.558e-6, 7.875e-6, 8.319e-6, 8.433e-6,
8.59e-6, 8.503e-6, 8.304e-6, 8.336e-6, 7.739e-6, 7.301e-6,
6.827e-6, 6.078e-6, 5.551e-6, 4.762e-6, 4.224e-6, 3.538e-6,
2.984e-6, 2.619e-6, 2.227e-6, 1.923e-6, 1.669e-6, 1.462e-6,
1.294e-6, 1.155e-6, 1.033e-6, 9.231e-7, 8.238e-7, 7.36e-7,
6.564e-7, 5.869e-7, 5.236e-7, 4.673e-7, 4.174e-7, 3.736e-7,
3.33e-7, 2.976e-7, 2.657e-7, 2.367e-7, 2.106e-7, 1.877e-7,
1.671e-7, 1.494e-7, 1.332e-7, 1.192e-7, 1.065e-7, 9.558e-8,
8.586e-8, 7.717e-8, 6.958e-8, 6.278e-8, 5.666e-8, 5.121e-8,
4.647e-8, 4.213e-8, 3.815e-8, 3.459e-8, 3.146e-8, 2.862e-8,
2.604e-8, 2.375e-8, 2.162e-8, 1.981e-8, 1.817e-8, 1.67e-8,
1.537e-8, 1.417e-8, 1.31e-8, 1.215e-8, 1.128e-8, 1.05e-8,
```

```
9.793e-9, 9.158e-9, 8.586e-9, 8.068e-9, 7.595e-9, 7.166e-9,
6.778e-9, 6.427e-9, 6.108e-9, 5.826e-9, 5.571e-9, 5.347e-9,
5.144e-9, 4.968e-9, 4.822e-9, 4.692e-9, 4.589e-9, 4.506e-9,
4.467e-9, 4.44e-9, 4.466e-9, 4.515e-9, 4.718e-9, 4.729e-9,
4.937e-9, 5.249e-9, 5.466e-9, 5.713e-9, 6.03e-9, 6.436e-9,
6.741e-9, 7.33e-9, 7.787e-9, 8.414e-9, 8.908e-9, 9.868e-9,
1.069e-8, 1.158e-8, 1.253e-8, 1.3e-8, 1.409e-8, 1.47e-8,
1.548e-8, 1.612e-8, 1.666e-8, 1.736e-8, 1.763e-8, 1.812e-8,
1.852e-8, 1.923e-8, 1.897e-8, 1.893e-8, 1.888e-8, 1.868e-8,
1.895e-8, 1.899e-8, 1.876e-8, 1.96e-8, 2.02e-8, 2.121e-8,
2.239e-8, 2.379e-8, 2.526e-8, 2.766e-8, 2.994e-8, 3.332e-8,
3.703e-8, 4.158e-8, 4.774e-8, 5.499e-8, 6.355e-8, 7.349e-8,
8.414e-8, 9.846e-8, 1.143e-7, 1.307e-7, 1.562e-7, 1.817e-7,
2.011e-7, 2.192e-7, 2.485e-7, 2.867e-7, 3.035e-7, 3.223e-7,
3.443e-7, 3.617e-7, 3.793e-7, 3.793e-7, 3.839e-7, 4.081e-7,
4.117e-7, 4.085e-7, 3.92e-7, 3.851e-7, 3.754e-7, 3.49e-7,
3.229e-7, 2.978e-7, 2.691e-7, 2.312e-7, 2.029e-7, 1.721e-7,
1.472e-7, 1.308e-7, 1.132e-7, 9.736e-8, 8.458e-8, 7.402e-8,
6.534e-8, 5.811e-8, 5.235e-8, 4.762e-8, 4.293e-8, 3.896e-8,
3.526e-8, 3.165e-8, 2.833e-8, 2.551e-8, 2.288e-8, 2.036e-8,
1.82e-8, 1.626e-8, 1.438e-8, 1.299e-8, 1.149e-8, 1.03e-8,
9.148e-9, 8.122e-9, 7.264e-9, 6.425e-9, 5.777e-9, 5.06e-9,
4.502e-9, 4.013e-9, 3.567e-9, 3.145e-9, 2.864e-9, 2.553e-9,
2.311e-9, 2.087e-9, 1.886e-9, 1.716e-9, 1.556e-9, 1.432e-9,
1.311e-9, 1.202e-9, 1.104e-9, 1.013e-9, 9.293e-10, 8.493e-10,
7.79e-10, 7.185e-10, 6.642e-10, 6.141e-10, 5.684e-10, 5.346e-10,
5.032e-10, 4.725e-10, 4.439e-10, 4.176e-10, 3.93e-10, 3.714e-10,
3.515e-10, 3.332e-10, 3.167e-10, 3.02e-10, 2.887e-10, 2.769e-10,
2.665e-10, 2.578e-10, 2.503e-10, 2.436e-10, 2.377e-10, 2.342e-10,
2.305e-10, 2.296e-10, 2.278e-10, 2.321e-10, 2.355e-10, 2.402e-10,
2.478e-10, 2.67e-10, 2.848e-10, 2.982e-10, 3.263e-10, 3.438e-10,
3.649e-10, 3.829e-10, 4.115e-10, 4.264e-10, 4.473e-10, 4.63e-10,
4.808e-10, 4.995e-10, 5.142e-10, 5.313e-10, 5.318e-10, 5.358e-10,
5.452e-10, 5.507e-10, 5.698e-10, 5.782e-10, 5.983e-10, 6.164e-10,
6.532e-10, 6.811e-10, 7.624e-10, 8.302e-10, 9.067e-10, 9.937e-10,
1.104e-9, 1.221e-9, 1.361e-9, 1.516e-9, 1.675e-9, 1.883e-9,
2.101e-9, 2.349e-9, 2.614e-9, 2.92e-9, 3.305e-9, 3.724e-9,
4.142e-9, 4.887e-9, 5.614e-9, 6.506e-9, 7.463e-9, 8.817e-9,
9.849e-9, 1.187e-8, 1.321e-8, 1.474e-8, 1.698e-8, 1.794e-8,
2.09e-8, 2.211e-8, 2.362e-8, 2.556e-8, 2.729e-8, 2.88e-8,
3.046e-8, 3.167e-8, 3.367e-8, 3.457e-8, 3.59e-8, 3.711e-8,
3.826e-8, 4.001e-8, 4.211e-8, 4.315e-8, 4.661e-8, 5.01e-8,
5.249e-8, 5.84e-8, 6.628e-8, 7.512e-8, 8.253e-8, 9.722e-8,
1.067e-7, 1.153e-7, 1.347e-7, 1.428e-7, 1.577e-7, 1.694e-7,
1.833e-7, 1.938e-7, 2.108e-7, 2.059e-7, 2.157e-7, 2.185e-7,
2.208e-7, 2.182e-7, 2.093e-7, 2.014e-7, 1.962e-7, 1.819e-7,
1.713e-7, 1.51e-7, 1.34e-7, 1.154e-7, 9.89e-8, 8.88e-8, 7.673e-8,
6.599e-8, 5.73e-8, 5.081e-8, 4.567e-8, 4.147e-8, 3.773e-8,
3.46e-8, 3.194e-8, 2.953e-8, 2.759e-8, 2.594e-8, 2.442e-8,
2.355e-8, 2.283e-8, 2.279e-8, 2.231e-8, 2.279e-8, 2.239e-8,
2.21e-8, 2.309e-8, 2.293e-8, 2.352e-8, 2.415e-8, 2.43e-8,
2.426e-8, 2.465e-8, 2.5e-8, 2.496e-8, 2.465e-8, 2.445e-8,
2.383e-8, 2.299e-8, 2.165e-8, 2.113e-8, 1.968e-8, 1.819e-8,
1.644e-8, 1.427e-8, 1.27e-8, 1.082e-8, 9.428e-9, 8.091e-9,
6.958e-9, 5.988e-9, 5.246e-9, 4.601e-9, 4.098e-9, 3.664e-9,
3.287e-9, 2.942e-9, 2.656e-9, 2.364e-9, 2.118e-9, 1.903e-9,
1.703e-9, 1.525e-9, 1.365e-9, 1.229e-9, 1.107e-9, 9.96e-10,
8.945e-10, 8.08e-10, 7.308e-10, 6.616e-10, 5.994e-10, 5.422e-10,
4.929e-10, 4.478e-10, 4.07e-10, 3.707e-10, 3.379e-10, 3.087e-10,
2.823e-10, 2.592e-10, 2.385e-10, 2.201e-10, 2.038e-10, 1.897e-10,
1.774e-10, 1.667e-10, 1.577e-10, 1.502e-10, 1.437e-10, 1.394e-10,
1.358e-10, 1.324e-10, 1.329e-10, 1.324e-10, 1.36e-10, 1.39e-10,
1.424e-10, 1.544e-10, 1.651e-10, 1.817e-10, 1.984e-10, 2.195e-10,
2.438e-10, 2.7e-10, 2.991e-10, 3.322e-10, 3.632e-10, 3.957e-10,
4.36e-10, 4.701e-10, 5.03e-10, 5.381e-10, 5.793e-10, 6.19e-10,
6.596e-10, 7.004e-10, 7.561e-10, 7.934e-10, 8.552e-10, 9.142e-10,
9.57e-10, 1.027e-9, 1.097e-9, 1.193e-9, 1.334e-9, 1.47e-9,
1.636e-9, 1.871e-9, 2.122e-9, 2.519e-9, 2.806e-9, 3.203e-9,
3.846e-9, 4.362e-9, 5.114e-9, 5.643e-9, 6.305e-9, 6.981e-9,
7.983e-9, 8.783e-9, 9.419e-9, 1.017e-8, 1.063e-8, 1.121e-8,
1.13e-8, 1.201e-8, 1.225e-8, 1.232e-8, 1.223e-8, 1.177e-8,
1.151e-8, 1.116e-8, 1.047e-8, 9.698e-9, 8.734e-9, 8.202e-9,
7.041e-9, 6.074e-9, 5.172e-9, 4.468e-9, 3.913e-9, 3.414e-9,
2.975e-9, 2.65e-9, 2.406e-9, 2.173e-9, 2.009e-9, 1.861e-9,
1.727e-9, 1.612e-9, 1.514e-9, 1.43e-9, 1.362e-9, 1.333e-9,
1.288e-9, 1.249e-9, 1.238e-9, 1.228e-9, 1.217e-9, 1.202e-9,
1.209e-9, 1.177e-9, 1.157e-9, 1.165e-9, 1.142e-9, 1.131e-9,
1.138e-9, 1.117e-9, 1.1e-9, 1.069e-9, 1.023e-9, 1.005e-9,
9.159e-10, 8.863e-10, 7.865e-10, 7.153e-10, 6.247e-10, 5.846e-10,
5.133e-10, 4.36e-10, 3.789e-10, 3.335e-10, 2.833e-10, 2.483e-10,
2.155e-10, 1.918e-10, 1.709e-10, 1.529e-10, 1.374e-10, 1.235e-10,
1.108e-10, 9.933e-11, 8.932e-11, 8.022e-11, 7.224e-11, 6.52e-11,
5.896e-11, 5.328e-11, 4.813e-11, 4.365e-11, 3.961e-11, 3.594e-11,
3.266e-11, 2.967e-11, 2.701e-11, 2.464e-11, 2.248e-11, 2.054e-11,
1.878e-11, 1.721e-11, 1.579e-11, 1.453e-11, 1.341e-11, 1.241e-11,
1.154e-11, 1.078e-11, 1.014e-11, 9.601e-12, 9.167e-12, 8.838e-12,
8.614e-12, 8.493e-12, 8.481e-12, 8.581e-12, 8.795e-12, 9.131e-12,
```

```
9.601e-12, 1.021e-11, 1.097e-11, 1.191e-11, 1.303e-11, 1.439e-11,
1.601e-11, 1.778e-11, 1.984e-11, 2.234e-11, 2.474e-11, 2.766e-11,
3.085e-11, 3.415e-11, 3.821e-11, 4.261e-11, 4.748e-11, 5.323e-11,
5.935e-11, 6.619e-11, 7.418e-11, 8.294e-11, 9.26e-11, 1.039e-10,
1.156e-10, 1.297e-10, 1.46e-10, 1.641e-10, 1.858e-10, 2.1e-10,
2.383e-10, 2.724e-10, 3.116e-10, 3.538e-10, 4.173e-10, 4.727e-10,
5.503e-10, 6.337e-10, 7.32e-10, 8.298e-10, 9.328e-10, 1.059e-9,
1.176e-9, 1.328e-9, 1.445e-9, 1.593e-9, 1.77e-9, 1.954e-9,
2.175e-9, 2.405e-9, 2.622e-9, 2.906e-9, 3.294e-9, 3.713e-9,
3.98e-9, 4.384e-9, 4.987e-9, 5.311e-9, 5.874e-9, 6.337e-9,
7.027e-9, 7.39e-9, 7.769e-9, 8.374e-9, 8.605e-9, 9.165e-9,
9.415e-9, 9.511e-9, 9.704e-9, 9.588e-9, 9.45e-9, 9.086e-9,
8.798e-9, 8.469e-9, 7.697e-9, 7.168e-9, 6.255e-9, 5.772e-9,
4.97e-9, 4.271e-9, 3.653e-9, 3.154e-9, 2.742e-9, 2.435e-9,
2.166e-9, 1.936e-9, 1.731e-9, 1.556e-9, 1.399e-9, 1.272e-9,
1.157e-9, 1.066e-9, 9.844e-10, 9.258e-10, 8.787e-10, 8.421e-10,
8.083e-10, 8.046e-10, 8.067e-10, 8.181e-10, 8.325e-10, 8.517e-10,
9.151e-10, 9.351e-10, 9.677e-10, 1.071e-9, 1.126e-9, 1.219e-9,
1.297e-9, 1.408e-9, 1.476e-9, 1.517e-9, 1.6e-9, 1.649e-9,
1.678e-9, 1.746e-9, 1.742e-9, 1.728e-9, 1.699e-9, 1.655e-9,
1.561e-9, 1.48e-9, 1.451e-9, 1.411e-9, 1.171e-9, 1.106e-9,
9.714e-10, 8.523e-10, 7.346e-10, 6.241e-10, 5.371e-10, 4.704e-10,
4.144e-10, 3.683e-10, 3.292e-10, 2.942e-10, 2.62e-10, 2.341e-10,
2.104e-10, 1.884e-10, 1.7e-10, 1.546e-10, 1.394e-10, 1.265e-10,
1.14e-10, 1.019e-10, 9.279e-11, 8.283e-11, 7.458e-11, 6.668e-11,
5.976e-11, 5.33e-11, 4.794e-11, 4.289e-11, 3.841e-11, 3.467e-11,
3.13e-11, 2.832e-11, 2.582e-11, 2.356e-11, 2.152e-11, 1.97e-11,
1.808e-11, 1.664e-11, 1.539e-11, 1.434e-11, 1.344e-11, 1.269e-11,
1.209e-11, 1.162e-11, 1.129e-11, 1.108e-11, 1.099e-11, 1.103e-11,
1.119e-11, 1.148e-11, 1.193e-11, 1.252e-11, 1.329e-11, 1.421e-11,
1.555e-11, 1.685e-11, 1.839e-11, 2.054e-11, 2.317e-11, 2.571e-11,
2.839e-11, 3.171e-11, 3.49e-11, 3.886e-11, 4.287e-11, 4.645e-11,
5.047e-11, 5.592e-11, 6.109e-11, 6.628e-11, 7.381e-11, 8.088e-11,
8.966e-11, 1.045e-10, 1.12e-10, 1.287e-10, 1.486e-10, 1.662e-10,
1.866e-10, 2.133e-10, 2.524e-10, 2.776e-10, 3.204e-10, 3.559e-10,
4.028e-10, 4.448e-10, 4.882e-10, 5.244e-10, 5.605e-10, 6.018e-10,
6.328e-10, 6.579e-10, 6.541e-10, 7.024e-10, 7.074e-10, 7.068e-10,
7.009e-10, 6.698e-10, 6.545e-10, 6.209e-10, 5.834e-10, 5.412e-10,
5.001e-10, 4.231e-10, 3.727e-10, 3.211e-10, 2.833e-10, 2.447e-10,
2.097e-10, 1.843e-10, 1.639e-10, 1.449e-10, 1.27e-10, 1.161e-10,
1.033e-10, 9.282e-11, 8.407e-11, 7.639e-11, 7.023e-11, 6.474e-11,
6.142e-11, 5.76e-11, 5.568e-11, 5.472e-11, 5.39e-11, 5.455e-11,
5.54e-11, 5.587e-11, 6.23e-11, 6.49e-11, 6.868e-11, 7.382e-11,
8.022e-11, 8.372e-11, 9.243e-11, 1.004e-10, 1.062e-10, 1.13e-10,
1.176e-10, 1.244e-10, 1.279e-10, 1.298e-10, 1.302e-10, 1.312e-10,
1.295e-10, 1.244e-10, 1.211e-10, 1.167e-10, 1.098e-10, 9.927e-11,
8.854e-11, 8.011e-11, 7.182e-11, 5.923e-11, 5.212e-11, 4.453e-11,
3.832e-11, 3.371e-11, 2.987e-11, 2.651e-11, 2.354e-11, 2.093e-11,
1.863e-11, 1.662e-11, 1.486e-11, 1.331e-11, 1.193e-11, 1.071e-11,
9.628e-12, 8.66e-12, 7.801e-12, 7.031e-12, 6.347e-12, 5.733e-12,
5.182e-12, 4.695e-12, 4.26e-12, 3.874e-12, 3.533e-12, 3.235e-12,
2.979e-12, 2.76e-12, 2.579e-12, 2.432e-12, 2.321e-12, 2.246e-12,
2.205e-12, 2.196e-12, 2.223e-12, 2.288e-12, 2.387e-12, 2.525e-12,
2.704e-12, 2.925e-12, 3.191e-12, 3.508e-12, 3.876e-12, 4.303e-12,
4.793e-12, 5.347e-12, 5.978e-12, 6.682e-12, 7.467e-12, 8.34e-12,
9.293e-12, 1.035e-11, 1.152e-11, 1.285e-11, 1.428e-11, 1.586e-11,
1.764e-11, 1.972e-11, 2.214e-11, 2.478e-11, 2.776e-11, 3.151e-11,
3.591e-11, 4.103e-11, 4.66e-11, 5.395e-11, 6.306e-11, 7.172e-11,
8.358e-11, 9.67e-11, 1.11e-10, 1.325e-10, 1.494e-10, 1.736e-10,
2.007e-10, 2.296e-10, 2.608e-10, 3.004e-10, 3.361e-10, 3.727e-10,
4.373e-10, 4.838e-10, 5.483e-10, 6.006e-10, 6.535e-10, 6.899e-10,
7.687e-10, 8.444e-10, 8.798e-10, 9.135e-10, 9.532e-10, 9.757e-10,
9.968e-10, 1.006e-9, 9.949e-10, 9.789e-10, 9.564e-10, 9.215e-10,
8.51e-10, 8.394e-10, 7.707e-10, 7.152e-10, 6.274e-10, 5.598e-10,
5.028e-10, 4.3e-10, 3.71e-10, 3.245e-10, 2.809e-10, 2.461e-10,
2.154e-10, 1.91e-10, 1.685e-10, 1.487e-10, 1.313e-10, 1.163e-10,
1.031e-10, 9.172e-11, 8.221e-11, 7.382e-11, 6.693e-11, 6.079e-11,
5.581e-11, 5.167e-11, 4.811e-11, 4.506e-11, 4.255e-11, 4.083e-11,
3.949e-11, 3.881e-11, 3.861e-11, 3.858e-11, 3.951e-11, 4.045e-11,
4.24e-11, 4.487e-11, 4.806e-11, 5.133e-11, 5.518e-11, 5.919e-11,
6.533e-11, 7.031e-11, 7.762e-11, 8.305e-11, 9.252e-11, 9.727e-11,
1.045e-10, 1.117e-10, 1.2e-10, 1.275e-10, 1.341e-10, 1.362e-10,
1.438e-10, 1.45e-10, 1.455e-10, 1.455e-10, 1.434e-10, 1.381e-10,
1.301e-10, 1.276e-10, 1.163e-10, 1.089e-10, 9.911e-11, 8.943e-11,
7.618e-11, 6.424e-11, 5.717e-11, 4.866e-11, 4.257e-11, 3.773e-11,
3.331e-11, 2.958e-11, 2.629e-11, 2.316e-11, 2.073e-11, 1.841e-11,
1.635e-11, 1.464e-11, 1.31e-11, 1.16e-11, 1.047e-11, 9.408e-12,
8.414e-12, 7.521e-12, 6.705e-12, 5.993e-12, 5.371e-12, 4.815e-12,
4.338e-12, 3.921e-12, 3.567e-12, 3.265e-12, 3.01e-12, 2.795e-12,
2.613e-12, 2.464e-12, 2.346e-12, 2.256e-12, 2.195e-12, 2.165e-12,
2.166e-12, 2.198e-12, 2.262e-12, 2.364e-12, 2.502e-12, 2.682e-12,
2.908e-12, 3.187e-12, 3.533e-12, 3.946e-12, 4.418e-12, 5.013e-12,
5.708e-12, 6.379e-12, 7.43e-12, 8.39e-12, 9.51e-12, 1.078e-11,
1.259e-11, 1.438e-11, 1.63e-11, 1.814e-11, 2.055e-11, 2.348e-11,
2.664e-11, 2.956e-11, 3.3e-11, 3.677e-11, 4.032e-11, 4.494e-11,
4.951e-11, 5.452e-11, 6.014e-11, 6.5e-11, 6.915e-11, 7.45e-11,
7.971e-11, 8.468e-11, 8.726e-11, 8.995e-11, 9.182e-11, 9.509e-11,
```

```
  9.333e-11, 9.386e-11, 9.457e-11, 9.21e-11, 9.019e-11, 8.68e-11,
  8.298e-11, 7.947e-11, 7.46e-11, 7.082e-11, 6.132e-11, 5.855e-11,
  5.073e-11, 4.464e-11, 3.825e-11, 3.375e-11, 2.911e-11, 2.535e-11,
  2.16e-11, 1.907e-11, 1.665e-11, 1.463e-11, 1.291e-11, 1.133e-11,
  9.997e-12, 8.836e-12, 7.839e-12, 6.943e-12, 6.254e-12, 5.6e-12,
  5.029e-12, 4.529e-12, 4.102e-12, 3.737e-12, 3.428e-12, 3.169e-12,
  2.959e-12, 2.798e-12, 2.675e-12, 2.582e-12, 2.644e-12, 2.557e-12,
  2.614e-12, 2.717e-12, 2.874e-12, 3.056e-12, 3.187e-12, 3.631e-12,
  3.979e-12, 4.248e-12, 4.817e-12, 5.266e-12, 5.836e-12, 6.365e-12,
  6.807e-12, 7.47e-12, 7.951e-12, 8.636e-12, 8.972e-12, 9.314e-12,
  9.445e-12, 1.003e-11, 1.013e-11, 9.937e-12, 9.729e-12, 9.064e-12,
  9.119e-12, 9.124e-12, 8.704e-12, 8.078e-12, 7.47e-12, 6.329e-12,
  5.674e-12, 4.808e-12, 4.119e-12, 3.554e-12, 3.103e-12, 2.731e-12,
  2.415e-12, 2.15e-12, 1.926e-12, 1.737e-12, 1.578e-12, 1.447e-12,
  1.34e-12, 1.255e-12, 1.191e-12, 1.146e-12, 1.121e-12, 1.114e-12,
  1.126e-12, 1.156e-12, 1.207e-12, 1.278e-12, 1.372e-12, 1.49e-12,
  1.633e-12, 1.805e-12, 2.01e-12, 2.249e-12, 2.528e-12, 2.852e-12,
  3.228e-12, 3.658e-12, 4.153e-12, 4.728e-12, 5.394e-12, 6.176e-12,
  7.126e-12, 8.188e-12, 9.328e-12, 1.103e-11, 1.276e-11, 1.417e-11,
  1.615e-11, 1.84e-11, 2.155e-11, 2.429e-11, 2.826e-11, 3.222e-11,
  3.664e-11, 4.14e-11, 4.906e-11, 5.536e-11, 6.327e-11, 7.088e-11,
  8.316e-11, 9.242e-11, 1.07e-10, 1.223e-10, 1.341e-10, 1.553e-10,
  1.703e-10, 1.9e-10, 2.022e-10, 2.233e-10, 2.345e-10, 2.438e-10,
  2.546e-10, 2.599e-10, 2.661e-10, 2.703e-10, 2.686e-10, 2.662e-10,
  2.56e-10, 2.552e-10, 2.378e-10, 2.252e-10, 2.146e-10, 1.885e-10,
  1.668e-10, 1.441e-10, 1.295e-10, 1.119e-10, 9.893e-11, 8.687e-11,
  7.678e-11, 6.685e-11, 5.879e-11, 5.127e-11, 4.505e-11, 3.997e-11,
  3.511e-11
};

static double h2ofrn[2001] = { .01095, .01126, .01205, .01322, .0143,
  .01506, .01548, .01534, .01486, .01373, .01262, .01134, .01001,
  .008702, .007475, .006481, .00548, .0046, .003833, .00311,
  .002543, .002049, .00168, .001374, .001046, 8.193e-4, 6.267e-4,
  4.968e-4, 3.924e-4, 2.983e-4, 2.477e-4, 1.997e-4, 1.596e-4,
  1.331e-4, 1.061e-4, 8.942e-5, 7.168e-5, 5.887e-5, 4.848e-5,
  3.817e-5, 3.17e-5, 2.579e-5, 2.162e-5, 1.768e-5, 1.49e-5,
  1.231e-5, 1.013e-5, 8.555e-6, 7.328e-6, 6.148e-6, 5.207e-6,
  4.387e-6, 3.741e-6, 3.22e-6, 2.753e-6, 2.346e-6, 1.985e-6,
  1.716e-6, 1.475e-6, 1.286e-6, 1.122e-6, 9.661e-7, 8.284e-7,
  7.057e-7, 6.119e-7, 5.29e-7, 4.571e-7, 3.948e-7, 3.432e-7,
  2.983e-7, 2.589e-7, 2.265e-7, 1.976e-7, 1.704e-7, 1.456e-7,
  1.26e-7, 1.101e-7, 9.648e-8, 8.415e-8, 7.34e-8, 6.441e-8,
  5.643e-8, 4.94e-8, 4.276e-8, 3.703e-8, 3.227e-8, 2.825e-8,
  2.478e-8, 2.174e-8, 1.898e-8, 1.664e-8, 1.458e-8, 1.278e-8,
  1.126e-8, 9.891e-9, 8.709e-9, 7.652e-9, 6.759e-9, 5.975e-9,
  5.31e-9, 4.728e-9, 4.214e-9, 3.792e-9, 3.463e-9, 3.226e-9,
  2.992e-9, 2.813e-9, 2.749e-9, 2.809e-9, 2.913e-9, 3.037e-9,
  3.413e-9, 3.738e-9, 4.189e-9, 4.808e-9, 5.978e-9, 7.088e-9,
  8.071e-9, 9.61e-9, 1.21e-8, 1.5e-8, 1.764e-8, 2.221e-8, 2.898e-8,
  3.948e-8, 5.068e-8, 6.227e-8, 7.898e-8, 1.033e-7, 1.437e-7,
  1.889e-7, 2.589e-7, 3.59e-7, 4.971e-7, 7.156e-7, 9.983e-7,
  1.381e-6, 1.929e-6, 2.591e-6, 3.453e-6, 4.57e-6, 5.93e-6,
  7.552e-6, 9.556e-6, 1.183e-5, 1.425e-5, 1.681e-5, 1.978e-5,
  2.335e-5, 2.668e-5, 3.022e-5, 3.371e-5, 3.715e-5, 3.967e-5,
  4.06e-5, 4.01e-5, 3.809e-5, 3.491e-5, 3.155e-5, 2.848e-5,
  2.678e-5, 2.66e-5, 2.811e-5, 3.071e-5, 3.294e-5, 3.459e-5,
  3.569e-5, 3.56e-5, 3.434e-5, 3.186e-5, 2.916e-5, 2.622e-5,
  2.275e-5, 1.918e-5, 1.62e-5, 1.373e-5, 1.182e-5, 1.006e-5,
  8.556e-6, 7.26e-6, 6.107e-6, 5.034e-6, 4.211e-6, 3.426e-6,
  2.865e-6, 2.446e-6, 1.998e-6, 1.628e-6, 1.242e-6, 1.005e-6,
  7.853e-7, 6.21e-7, 5.071e-7, 4.156e-7, 3.548e-7, 2.825e-7,
  2.261e-7, 1.916e-7, 1.51e-7, 1.279e-7, 1.059e-7, 9.14e-8,
  7.707e-8, 6.17e-8, 5.311e-8, 4.263e-8, 3.518e-8, 2.961e-8,
  2.457e-8, 2.119e-8, 1.712e-8, 1.439e-8, 1.201e-8, 1.003e-8,
  8.564e-9, 7.199e-9, 6.184e-9, 5.206e-9, 4.376e-9, 3.708e-9,
  3.157e-9, 2.725e-9, 2.361e-9, 2.074e-9, 1.797e-9, 1.562e-9,
  1.364e-9, 1.196e-9, 1.042e-9, 8.862e-10, 7.648e-10, 6.544e-10,
  5.609e-10, 4.791e-10, 4.108e-10, 3.531e-10, 3.038e-10, 2.618e-10,
  2.268e-10, 1.969e-10, 1.715e-10, 1.496e-10, 1.308e-10, 1.147e-10,
  1.008e-10, 8.894e-11, 7.885e-11, 7.031e-11, 6.355e-11, 5.854e-11,
  5.534e-11, 5.466e-11, 5.725e-11, 6.447e-11, 7.943e-11, 1.038e-10,
  1.437e-10, 2.04e-10, 2.901e-10, 4.051e-10, 5.556e-10, 7.314e-10,
  9.291e-10, 1.134e-9, 1.321e-9, 1.482e-9, 1.596e-9, 1.669e-9,
  1.715e-9, 1.762e-9, 1.817e-9, 1.828e-9, 1.848e-9, 1.873e-9,
  1.902e-9, 1.894e-9, 1.864e-9, 1.841e-9, 1.797e-9, 1.704e-9,
  1.559e-9, 1.382e-9, 1.187e-9, 1.001e-9, 8.468e-10, 7.265e-10,
  6.521e-10, 6.381e-10, 6.66e-10, 7.637e-10, 9.705e-10, 1.368e-9,
  1.856e-9, 2.656e-9, 3.954e-9, 5.96e-9, 8.72e-9, 1.247e-8,
  1.781e-8, 2.491e-8, 3.311e-8, 4.272e-8, 5.205e-8, 6.268e-8,
  7.337e-8, 8.277e-8, 9.185e-8, 1.004e-7, 1.091e-7, 1.159e-7,
  1.188e-7, 1.175e-7, 1.124e-7, 1.033e-7, 9.381e-8, 8.501e-8,
  7.956e-8, 7.894e-8, 8.331e-8, 9.102e-8, 9.836e-8, 1.035e-7,
  1.064e-7, 1.06e-7, 1.032e-7, 9.808e-8, 9.139e-8, 8.442e-8,
  7.641e-8, 6.881e-8, 6.161e-8, 5.404e-8, 4.804e-8, 4.446e-8,
  4.328e-8, 4.259e-8, 4.421e-8, 4.673e-8, 4.985e-8, 5.335e-8,
  5.796e-8, 6.542e-8, 7.714e-8, 8.827e-8, 1.04e-7, 1.238e-7,
```

```
1.499e-7, 1.829e-7, 2.222e-7, 2.689e-7, 3.303e-7, 3.981e-7,
4.84e-7, 5.91e-7, 7.363e-7, 9.087e-7, 1.139e-6, 1.455e-6,
1.866e-6, 2.44e-6, 3.115e-6, 3.941e-6, 4.891e-6, 5.992e-6,
7.111e-6, 8.296e-6, 9.21e-6, 9.987e-6, 1.044e-5, 1.073e-5,
1.092e-5, 1.106e-5, 1.138e-5, 1.171e-5, 1.186e-5, 1.186e-5,
1.179e-5, 1.166e-5, 1.151e-5, 1.16e-5, 1.197e-5, 1.241e-5,
1.268e-5, 1.26e-5, 1.184e-5, 1.063e-5, 9.204e-6, 7.584e-6,
6.053e-6, 4.482e-6, 3.252e-6, 2.337e-6, 1.662e-6, 1.18e-6,
8.15e-7, 5.95e-7, 4.354e-7, 3.302e-7, 2.494e-7, 1.93e-7,
1.545e-7, 1.25e-7, 1.039e-7, 8.602e-8, 7.127e-8, 5.897e-8,
4.838e-8, 4.018e-8, 3.28e-8, 2.72e-8, 2.307e-8, 1.972e-8,
1.654e-8, 1.421e-8, 1.174e-8, 1.004e-8, 8.739e-9, 7.358e-9,
6.242e-9, 5.303e-9, 4.567e-9, 3.94e-9, 3.375e-9, 2.864e-9,
2.422e-9, 2.057e-9, 1.75e-9, 1.505e-9, 1.294e-9, 1.101e-9,
9.401e-10, 8.018e-10, 6.903e-10, 5.965e-10, 5.087e-10, 4.364e-10,
3.759e-10, 3.247e-10, 2.809e-10, 2.438e-10, 2.123e-10, 1.853e-10,
1.622e-10, 1.426e-10, 1.26e-10, 1.125e-10, 1.022e-10, 9.582e-11,
9.388e-11, 9.801e-11, 1.08e-10, 1.276e-10, 1.551e-10, 1.903e-10,
2.291e-10, 2.724e-10, 3.117e-10, 3.4e-10, 3.562e-10, 3.625e-10,
3.619e-10, 3.429e-10, 3.221e-10, 2.943e-10, 2.645e-10, 2.338e-10,
2.062e-10, 1.901e-10, 1.814e-10, 1.827e-10, 1.906e-10, 1.984e-10,
2.04e-10, 2.068e-10, 2.075e-10, 2.018e-10, 1.959e-10, 1.897e-10,
1.852e-10, 1.791e-10, 1.696e-10, 1.634e-10, 1.598e-10, 1.561e-10,
1.518e-10, 1.443e-10, 1.377e-10, 1.346e-10, 1.342e-10, 1.375e-10,
1.525e-10, 1.767e-10, 2.108e-10, 2.524e-10, 2.981e-10, 3.477e-10,
4.262e-10, 5.326e-10, 6.646e-10, 8.321e-10, 1.069e-9, 1.386e-9,
1.743e-9, 2.216e-9, 2.808e-9, 3.585e-9, 4.552e-9, 5.907e-9,
7.611e-9, 9.774e-9, 1.255e-8, 1.666e-8, 2.279e-8, 3.221e-8,
4.531e-8, 6.4e-8, 9.187e-8, 1.295e-7, 1.825e-7, 2.431e-7,
3.181e-7, 4.009e-7, 4.941e-7, 5.88e-7, 6.623e-7, 7.155e-7,
7.451e-7, 7.594e-7, 7.541e-7, 7.467e-7, 7.527e-7, 7.935e-7,
8.461e-7, 8.954e-7, 9.364e-7, 9.843e-7, 1.024e-6, 1.05e-6,
1.059e-6, 1.074e-6, 1.072e-6, 1.043e-6, 9.789e-7, 8.803e-7,
7.662e-7, 6.378e-7, 5.133e-7, 3.958e-7, 2.914e-7, 2.144e-7,
1.57e-7, 1.14e-7, 8.47e-8, 6.2e-8, 4.657e-8, 3.559e-8, 2.813e-8,
2.222e-8, 1.769e-8, 1.391e-8, 1.125e-8, 9.186e-9, 7.704e-9,
6.447e-9, 5.381e-9, 4.442e-9, 3.669e-9, 3.057e-9, 2.564e-9,
2.153e-9, 1.784e-9, 1.499e-9, 1.281e-9, 1.082e-9, 9.304e-10,
8.169e-10, 6.856e-10, 5.866e-10, 5.043e-10, 4.336e-10, 3.731e-10,
3.175e-10, 2.745e-10, 2.374e-10, 2.007e-10, 1.737e-10, 1.508e-10,
1.302e-10, 1.13e-10, 9.672e-11, 8.375e-11, 7.265e-11, 6.244e-11,
5.343e-11, 4.654e-11, 3.975e-11, 3.488e-11, 3.097e-11, 2.834e-11,
2.649e-11, 2.519e-11, 2.462e-11, 2.443e-11, 2.44e-11, 2.398e-11,
2.306e-11, 2.183e-11, 2.021e-11, 1.821e-11, 1.599e-11, 1.403e-11,
1.196e-11, 1.023e-11, 8.728e-12, 7.606e-12, 6.941e-12, 6.545e-12,
6.484e-12, 6.6e-12, 6.718e-12, 6.785e-12, 6.746e-12, 6.724e-12,
6.764e-12, 6.995e-12, 7.144e-12, 7.32e-12, 7.33e-12, 7.208e-12,
6.789e-12, 6.09e-12, 5.337e-12, 4.62e-12, 4.037e-12, 3.574e-12,
3.311e-12, 3.346e-12, 3.566e-12, 3.836e-12, 4.076e-12, 4.351e-12,
4.691e-12, 5.114e-12, 5.427e-12, 6.167e-12, 7.436e-12, 8.842e-12,
1.038e-11, 1.249e-11, 1.54e-11, 1.915e-11, 2.48e-11, 3.256e-11,
4.339e-11, 5.611e-11, 7.519e-11, 1.037e-10, 1.409e-10, 1.883e-10,
2.503e-10, 3.38e-10, 4.468e-10, 5.801e-10, 7.335e-10, 8.98e-10,
1.11e-9, 1.363e-9, 1.677e-9, 2.104e-9, 2.681e-9, 3.531e-9,
4.621e-9, 6.106e-9, 8.154e-9, 1.046e-8, 1.312e-8, 1.607e-8,
1.948e-8, 2.266e-8, 2.495e-8, 2.655e-8, 2.739e-8, 2.739e-8,
2.662e-8, 2.589e-8, 2.59e-8, 2.664e-8, 2.833e-8, 3.023e-8,
3.305e-8, 3.558e-8, 3.793e-8, 3.961e-8, 4.056e-8, 4.102e-8,
4.025e-8, 3.917e-8, 3.706e-8, 3.493e-8, 3.249e-8, 3.096e-8,
3.011e-8, 3.111e-8, 3.395e-8, 3.958e-8, 4.875e-8, 6.066e-8,
7.915e-8, 1.011e-7, 1.3e-7, 1.622e-7, 2.003e-7, 2.448e-7,
2.863e-7, 3.317e-7, 3.655e-7, 3.96e-7, 4.098e-7, 4.168e-7,
4.198e-7, 4.207e-7, 4.289e-7, 4.384e-7, 4.471e-7, 4.524e-7,
4.574e-7, 4.633e-7, 4.785e-7, 5.028e-7, 5.371e-7, 5.727e-7,
5.955e-7, 5.998e-7, 5.669e-7, 5.082e-7, 4.397e-7, 3.596e-7,
2.814e-7, 2.074e-7, 1.486e-7, 1.057e-7, 7.25e-8, 4.946e-8,
3.43e-8, 2.447e-8, 1.793e-8, 1.375e-8, 1.096e-8, 9.091e-9,
7.709e-9, 6.631e-9, 5.714e-9, 4.886e-9, 4.205e-9, 3.575e-9,
3.07e-9, 2.631e-9, 2.284e-9, 2.002e-9, 1.745e-9, 1.509e-9,
1.284e-9, 1.084e-9, 9.163e-10, 7.663e-10, 6.346e-10, 5.283e-10,
4.354e-10, 3.59e-10, 2.982e-10, 2.455e-10, 2.033e-10, 1.696e-10,
1.432e-10, 1.211e-10, 1.02e-10, 8.702e-11, 7.38e-11, 6.293e-11,
5.343e-11, 4.532e-11, 3.907e-11, 3.365e-11, 2.945e-11, 2.558e-11,
2.192e-11, 1.895e-11, 1.636e-11, 1.42e-11, 1.228e-11, 1.063e-11,
9.348e-12, 8.2e-12, 7.231e-12, 6.43e-12, 5.702e-12, 5.052e-12,
4.469e-12, 4e-12, 3.679e-12, 3.387e-12, 3.197e-12, 3.158e-12,
3.327e-12, 3.675e-12, 4.292e-12, 5.437e-12, 7.197e-12, 1.008e-11,
1.437e-11, 2.035e-11, 2.905e-11, 4.062e-11, 5.528e-11, 7.177e-11,
9.064e-11, 1.109e-10, 1.297e-10, 1.473e-10, 1.652e-10, 1.851e-10,
2.079e-10, 2.313e-10, 2.619e-10, 2.958e-10, 3.352e-10, 3.796e-10,
4.295e-10, 4.923e-10, 5.49e-10, 5.998e-10, 6.388e-10, 6.645e-10,
6.712e-10, 6.549e-10, 6.38e-10, 6.255e-10, 6.253e-10, 6.459e-10,
6.977e-10, 7.59e-10, 8.242e-10, 8.92e-10, 9.403e-10, 9.701e-10,
9.483e-10, 9.135e-10, 8.617e-10, 7.921e-10, 7.168e-10, 6.382e-10,
5.677e-10, 5.045e-10, 4.572e-10, 4.312e-10, 4.145e-10, 4.192e-10,
4.541e-10, 5.368e-10, 6.771e-10, 8.962e-10, 1.21e-9, 1.659e-9,
2.33e-9, 3.249e-9, 4.495e-9, 5.923e-9, 7.642e-9, 9.607e-9,
```

```
1.178e-8, 1.399e-8, 1.584e-8, 1.73e-8, 1.816e-8, 1.87e-8,
1.868e-8, 1.87e-8, 1.884e-8, 1.99e-8, 2.15e-8, 2.258e-8,
2.364e-8, 2.473e-8, 2.602e-8, 2.689e-8, 2.731e-8, 2.816e-8,
2.859e-8, 2.839e-8, 2.703e-8, 2.451e-8, 2.149e-8, 1.787e-8,
1.449e-8, 1.111e-8, 8.282e-9, 6.121e-9, 4.494e-9, 3.367e-9,
2.487e-9, 1.885e-9, 1.503e-9, 1.249e-9, 1.074e-9, 9.427e-10,
8.439e-10, 7.563e-10, 6.772e-10, 6.002e-10, 5.254e-10, 4.588e-10,
3.977e-10, 3.449e-10, 3.003e-10, 2.624e-10, 2.335e-10, 2.04e-10,
1.771e-10, 1.534e-10, 1.296e-10, 1.097e-10, 9.173e-11, 7.73e-11,
6.547e-11, 5.191e-11, 4.198e-11, 3.361e-11, 2.732e-11, 2.244e-11,
1.791e-11, 1.509e-11, 1.243e-11, 1.035e-11, 8.969e-12, 7.394e-12,
6.323e-12, 5.282e-12, 4.543e-12, 3.752e-12, 3.14e-12, 2.6e-12,
2.194e-12, 1.825e-12, 1.511e-12, 1.245e-12, 1.024e-12, 8.539e-13,
7.227e-13, 6.102e-13, 5.189e-13, 4.43e-13, 3.774e-13, 3.236e-13,
2.8e-13, 2.444e-13, 2.156e-13, 1.932e-13, 1.775e-13, 1.695e-13,
1.672e-13, 1.704e-13, 1.825e-13, 2.087e-13, 2.614e-13, 3.377e-13,
4.817e-13, 6.989e-13, 1.062e-12, 1.562e-12, 2.288e-12, 3.295e-12,
4.55e-12, 5.965e-12, 7.546e-12, 9.395e-12, 1.103e-11, 1.228e-11,
1.318e-11, 1.38e-11, 1.421e-11, 1.39e-11, 1.358e-11, 1.336e-11,
1.342e-11, 1.356e-11, 1.424e-11, 1.552e-11, 1.73e-11, 1.951e-11,
2.128e-11, 2.249e-11, 2.277e-11, 2.226e-11, 2.111e-11, 1.922e-11,
1.775e-11, 1.661e-11, 1.547e-11, 1.446e-11, 1.323e-11, 1.21e-11,
1.054e-11, 9.283e-12, 8.671e-12, 8.67e-12, 9.429e-12, 1.062e-11,
1.255e-11, 1.506e-11, 1.818e-11, 2.26e-11, 2.831e-11, 3.723e-11,
5.092e-11, 6.968e-11, 9.826e-11, 1.349e-10, 1.87e-10, 2.58e-10,
3.43e-10, 4.424e-10, 5.521e-10, 6.812e-10, 8.064e-10, 9.109e-10,
9.839e-10, 1.028e-9, 1.044e-9, 1.029e-9, 1.005e-9, 1.002e-9,
1.038e-9, 1.122e-9, 1.233e-9, 1.372e-9, 1.524e-9, 1.665e-9,
1.804e-9, 1.908e-9, 2.015e-9, 2.117e-9, 2.219e-9, 2.336e-9,
2.531e-9, 2.805e-9, 3.189e-9, 3.617e-9, 4.208e-9, 4.911e-9,
5.619e-9, 6.469e-9, 7.188e-9, 7.957e-9, 8.503e-9, 9.028e-9,
9.571e-9, 9.99e-9, 1.055e-8, 1.102e-8, 1.132e-8, 1.141e-8,
1.145e-8, 1.145e-8, 1.176e-8, 1.224e-8, 1.304e-8, 1.388e-8,
1.445e-8, 1.453e-8, 1.368e-8, 1.22e-8, 1.042e-8, 8.404e-9,
6.403e-9, 4.643e-9, 3.325e-9, 2.335e-9, 1.638e-9, 1.19e-9,
9.161e-10, 7.412e-10, 6.226e-10, 5.516e-10, 5.068e-10, 4.831e-10,
4.856e-10, 5.162e-10, 5.785e-10, 6.539e-10, 7.485e-10, 8.565e-10,
9.534e-10, 1.052e-9, 1.115e-9, 1.173e-9, 1.203e-9, 1.224e-9,
1.243e-9, 1.248e-9, 1.261e-9, 1.265e-9, 1.25e-9, 1.217e-9,
1.176e-9, 1.145e-9, 1.153e-9, 1.199e-9, 1.278e-9, 1.366e-9,
1.426e-9, 1.444e-9, 1.365e-9, 1.224e-9, 1.051e-9, 8.539e-10,
6.564e-10, 4.751e-10, 3.404e-10, 2.377e-10, 1.631e-10, 1.114e-10,
7.87e-11, 5.793e-11, 4.284e-11, 3.3e-11, 2.62e-11, 2.152e-11,
1.777e-11, 1.496e-11, 1.242e-11, 1.037e-11, 8.725e-12, 7.004e-12,
5.718e-12, 4.769e-12, 3.952e-12, 3.336e-12, 2.712e-12, 2.213e-12,
1.803e-12, 1.492e-12, 1.236e-12, 1.006e-12, 8.384e-13, 7.063e-13,
5.879e-13, 4.93e-13, 4.171e-13, 3.569e-13, 3.083e-13, 2.688e-13,
2.333e-13, 2.035e-13, 1.82e-13, 1.682e-13, 1.635e-13, 1.674e-13,
1.769e-13, 2.022e-13, 2.485e-13, 3.127e-13, 4.25e-13, 5.928e-13,
8.514e-13, 1.236e-12, 1.701e-12, 2.392e-12, 3.231e-12, 4.35e-12,
5.559e-12, 6.915e-12, 8.519e-12, 1.013e-11, 1.146e-11, 1.24e-11,
1.305e-11, 1.333e-11, 1.318e-11, 1.263e-11, 1.238e-11, 1.244e-11,
1.305e-11, 1.432e-11, 1.623e-11, 1.846e-11, 2.09e-11, 2.328e-11,
2.526e-11, 2.637e-11, 2.702e-11, 2.794e-11, 2.889e-11, 2.989e-11,
3.231e-11, 3.68e-11, 4.375e-11, 5.504e-11, 7.159e-11, 9.502e-11,
1.279e-10, 1.645e-10, 2.098e-10, 2.618e-10, 3.189e-10, 3.79e-10,
4.303e-10, 4.753e-10, 5.027e-10, 5.221e-10, 5.293e-10, 5.346e-10,
5.467e-10, 5.796e-10, 6.2e-10, 6.454e-10, 6.705e-10, 6.925e-10,
7.233e-10, 7.35e-10, 7.538e-10, 7.861e-10, 8.077e-10, 8.132e-10,
7.749e-10, 7.036e-10, 6.143e-10, 5.093e-10, 4.089e-10, 3.092e-10,
2.299e-10, 1.705e-10, 1.277e-10, 9.723e-11, 7.533e-11, 6.126e-11,
5.154e-11, 4.428e-11, 3.913e-11, 3.521e-11, 3.297e-11, 3.275e-11,
3.46e-11, 3.798e-11, 4.251e-11, 4.745e-11, 5.232e-11, 5.606e-11,
5.82e-11, 5.88e-11, 5.79e-11, 5.661e-11, 5.491e-11, 5.366e-11,
5.341e-11, 5.353e-11, 5.336e-11, 5.293e-11, 5.248e-11, 5.235e-11,
5.208e-11, 5.322e-11, 5.521e-11, 5.725e-11, 5.827e-11, 5.685e-11,
5.245e-11, 4.612e-11, 3.884e-11, 3.129e-11, 2.404e-11, 1.732e-11,
1.223e-11, 8.574e-12, 5.888e-12, 3.986e-12, 2.732e-12, 1.948e-12,
1.414e-12, 1.061e-12, 8.298e-13, 6.612e-13, 5.413e-13, 4.472e-13,
3.772e-13, 3.181e-13, 2.645e-13, 2.171e-13, 1.778e-13, 1.464e-13,
1.183e-13, 9.637e-14, 7.991e-14, 6.668e-14, 5.57e-14, 4.663e-14,
3.848e-14, 3.233e-14, 2.706e-14, 2.284e-14, 1.944e-14, 1.664e-14,
1.43e-14, 1.233e-14, 1.066e-14, 9.234e-15, 8.023e-15, 6.993e-15,
6.119e-15, 5.384e-15, 4.774e-15, 4.283e-15, 3.916e-15, 3.695e-15,
3.682e-15, 4.004e-15, 4.912e-15, 6.853e-15, 1.056e-14, 1.712e-14,
2.804e-14, 4.516e-14, 7.113e-14, 1.084e-13, 1.426e-13, 1.734e-13,
1.978e-13, 2.194e-13, 2.388e-13, 2.489e-13, 2.626e-13, 2.865e-13,
3.105e-13, 3.387e-13, 3.652e-13, 3.984e-13, 4.398e-13, 4.906e-13,
5.55e-13, 6.517e-13, 7.813e-13, 9.272e-13, 1.164e-12, 1.434e-12,
1.849e-12, 2.524e-12, 3.328e-12, 4.523e-12, 6.108e-12, 8.207e-12,
1.122e-11, 1.477e-11, 1.9e-11, 2.412e-11, 2.984e-11, 3.68e-11,
4.353e-11, 4.963e-11, 5.478e-11, 5.903e-11, 6.233e-11, 6.483e-11,
6.904e-11, 7.569e-11, 8.719e-11, 1.048e-10, 1.278e-10, 1.557e-10,
1.869e-10, 2.218e-10, 2.61e-10, 2.975e-10, 3.371e-10, 3.746e-10,
4.065e-10, 4.336e-10, 4.503e-10, 4.701e-10, 4.8e-10, 4.917e-10,
5.038e-10, 5.128e-10, 5.143e-10, 5.071e-10, 5.019e-10, 5.025e-10,
5.183e-10, 5.496e-10, 5.877e-10, 6.235e-10, 6.42e-10, 6.234e-10,
```

```
5.698e-10, 4.916e-10, 4.022e-10, 3.126e-10, 2.282e-10, 1.639e-10,
1.142e-10, 7.919e-11, 5.69e-11, 4.313e-11, 3.413e-11, 2.807e-11,
2.41e-11, 2.166e-11, 2.024e-11, 1.946e-11, 1.929e-11, 1.963e-11,
2.035e-11, 2.162e-11, 2.305e-11, 2.493e-11, 2.748e-11, 3.048e-11,
3.413e-11, 3.754e-11, 4.155e-11, 4.635e-11, 5.11e-11, 5.734e-11,
6.338e-11, 6.99e-11, 7.611e-11, 8.125e-11, 8.654e-11, 8.951e-11,
9.182e-11, 9.31e-11, 9.273e-11, 9.094e-11, 8.849e-11, 8.662e-11,
8.67e-11, 8.972e-11, 9.566e-11, 1.025e-10, 1.083e-10, 1.111e-10,
1.074e-10, 9.771e-11, 8.468e-11, 6.958e-11, 5.47e-11, 4.04e-11,
2.94e-11, 2.075e-11, 1.442e-11, 1.01e-11, 7.281e-12, 5.409e-12,
4.138e-12, 3.304e-12, 2.784e-12, 2.473e-12, 2.273e-12, 2.186e-12,
2.118e-12, 2.066e-12, 1.958e-12, 1.818e-12, 1.675e-12, 1.509e-12,
1.349e-12, 1.171e-12, 9.838e-13, 8.213e-13, 6.765e-13, 5.378e-13,
4.161e-13, 3.119e-13, 2.279e-13, 1.637e-13, 1.152e-13, 8.112e-14,
5.919e-14, 4.47e-14, 3.492e-14, 2.811e-14, 2.319e-14, 1.948e-14,
1.66e-14, 1.432e-14, 1.251e-14, 1.109e-14, 1.006e-14, 9.45e-15,
9.384e-15, 1.012e-14, 1.216e-14, 1.636e-14, 2.305e-14, 3.488e-14,
5.572e-14, 8.479e-14, 1.265e-13, 1.905e-13, 2.73e-13, 3.809e-13,
4.955e-13, 6.303e-13, 7.861e-13, 9.427e-13, 1.097e-12, 1.212e-12,
1.328e-12, 1.415e-12, 1.463e-12, 1.495e-12, 1.571e-12, 1.731e-12,
1.981e-12, 2.387e-12, 2.93e-12, 3.642e-12, 4.584e-12, 5.822e-12,
7.278e-12, 9.193e-12, 1.135e-11, 1.382e-11, 1.662e-11, 1.958e-11,
2.286e-11, 2.559e-11, 2.805e-11, 2.988e-11, 3.106e-11, 3.182e-11,
3.2e-11, 3.258e-11, 3.362e-11, 3.558e-11, 3.688e-11, 3.8e-11,
3.929e-11, 4.062e-11, 4.186e-11, 4.293e-11, 4.48e-11, 4.643e-11,
4.704e-11, 4.571e-11, 4.206e-11, 3.715e-11, 3.131e-11, 2.541e-11,
1.978e-11, 1.508e-11, 1.146e-11, 8.7e-12, 6.603e-12, 5.162e-12,
4.157e-12, 3.408e-12, 2.829e-12, 2.405e-12, 2.071e-12, 1.826e-12,
1.648e-12, 1.542e-12, 1.489e-12, 1.485e-12, 1.493e-12, 1.545e-12,
1.637e-12, 1.814e-12, 2.061e-12, 2.312e-12, 2.651e-12, 3.03e-12,
3.46e-12, 3.901e-12, 4.306e-12, 4.721e-12, 5.008e-12, 5.281e-12,
5.541e-12, 5.791e-12, 6.115e-12, 6.442e-12, 6.68e-12, 6.791e-12,
6.831e-12, 6.839e-12, 6.946e-12, 7.128e-12, 7.537e-12, 8.036e-12,
8.392e-12, 8.526e-12, 8.11e-12, 7.325e-12, 6.329e-12, 5.183e-12,
4.081e-12, 2.985e-12, 2.141e-12, 1.492e-12, 1.015e-12, 6.684e-13,
4.414e-13, 2.987e-13, 2.038e-13, 1.391e-13, 9.86e-14, 7.24e-14,
5.493e-14, 4.288e-14, 3.427e-14, 2.787e-14, 2.296e-14, 1.909e-14,
1.598e-14, 1.344e-14, 1.135e-14, 9.616e-15, 8.169e-15, 6.957e-15,
5.938e-15, 5.08e-15, 4.353e-15, 3.738e-15, 3.217e-15, 2.773e-15,
2.397e-15, 2.077e-15, 1.805e-15, 1.575e-15, 1.382e-15, 1.221e-15,
1.09e-15, 9.855e-16, 9.068e-16, 8.537e-16, 8.27e-16, 8.29e-16,
8.634e-16, 9.359e-16, 1.055e-15, 1.233e-15, 1.486e-15, 1.839e-15,
2.326e-15, 2.998e-15, 3.934e-15, 5.256e-15, 7.164e-15, 9.984e-15,
1.427e-14, 2.099e-14, 3.196e-14, 5.121e-14, 7.908e-14, 1.131e-13,
1.602e-13, 2.239e-13, 3.075e-13, 4.134e-13, 5.749e-13, 7.886e-13,
1.071e-12, 1.464e-12, 2.032e-12, 2.8e-12, 3.732e-12, 4.996e-12,
6.483e-12, 8.143e-12, 1.006e-11, 1.238e-11, 1.484e-11, 1.744e-11,
2.02e-11, 2.274e-11, 2.562e-11, 2.848e-11, 3.191e-11, 3.617e-11,
4.081e-11, 4.577e-11, 4.937e-11, 5.204e-11, 5.401e-11, 5.462e-11,
5.507e-11, 5.51e-11, 5.605e-11, 5.686e-11, 5.739e-11, 5.766e-11,
5.74e-11, 5.754e-11, 5.761e-11, 5.777e-11, 5.712e-11, 5.51e-11,
5.088e-11, 4.438e-11, 3.728e-11, 2.994e-11, 2.305e-11, 1.715e-11,
1.256e-11, 9.208e-12, 6.745e-12, 5.014e-12, 3.785e-12, 2.9e-12,
2.239e-12, 1.757e-12, 1.414e-12, 1.142e-12, 9.482e-13, 8.01e-13,
6.961e-13, 6.253e-13, 5.735e-13, 5.433e-13, 5.352e-13, 5.493e-13,
5.706e-13, 6.068e-13, 6.531e-13, 7.109e-13, 7.767e-13, 8.59e-13,
9.792e-13, 1.142e-12, 1.371e-12, 1.65e-12, 1.957e-12, 2.302e-12,
2.705e-12, 3.145e-12, 3.608e-12, 4.071e-12, 4.602e-12, 5.133e-12,
5.572e-12, 5.987e-12, 6.248e-12, 6.533e-12, 6.757e-12, 6.935e-12,
7.224e-12, 7.422e-12, 7.538e-12, 7.547e-12, 7.495e-12, 7.543e-12,
7.725e-12, 8.139e-12, 8.627e-12, 9.146e-12, 9.443e-12, 9.318e-12,
8.649e-12, 7.512e-12, 6.261e-12, 4.915e-12, 3.647e-12, 2.597e-12,
1.785e-12, 1.242e-12, 8.66e-13, 6.207e-13, 4.61e-13, 3.444e-13,
2.634e-13, 2.1e-13, 1.725e-13, 1.455e-13, 1.237e-13, 1.085e-13,
9.513e-14, 7.978e-14, 6.603e-14, 5.288e-14, 4.084e-14, 2.952e-14,
2.157e-14, 1.593e-14, 1.199e-14, 9.267e-15, 7.365e-15, 6.004e-15,
4.995e-15, 4.218e-15, 3.601e-15, 3.101e-15, 2.692e-15, 2.36e-15,
2.094e-15, 1.891e-15, 1.755e-15, 1.699e-15, 1.755e-15, 1.987e-15,
2.506e-15, 3.506e-15, 5.289e-15, 8.311e-15, 1.325e-14, 2.129e-14,
3.237e-14, 4.595e-14, 6.441e-14, 8.433e-14, 1.074e-13, 1.383e-13,
1.762e-13, 2.281e-13, 2.831e-13, 3.523e-13, 4.38e-13, 5.304e-13,
6.29e-13, 7.142e-13, 8.032e-13, 8.934e-13, 9.888e-13, 1.109e-12,
1.261e-12, 1.462e-12, 1.74e-12, 2.099e-12, 2.535e-12, 3.008e-12,
3.462e-12, 3.856e-12, 4.098e-12, 4.239e-12, 4.234e-12, 4.132e-12,
3.986e-12, 3.866e-12, 3.829e-12, 3.742e-12, 3.705e-12, 3.694e-12,
3.765e-12, 3.849e-12, 3.929e-12, 4.056e-12, 4.092e-12, 4.047e-12,
3.792e-12, 3.407e-12, 2.953e-12, 2.429e-12, 1.931e-12, 1.46e-12,
1.099e-12, 8.199e-13, 6.077e-13, 4.449e-13, 3.359e-13, 2.524e-13,
1.881e-13, 1.391e-13, 1.02e-13, 7.544e-14, 5.555e-14, 4.22e-14,
3.321e-14, 2.686e-14, 2.212e-14, 1.78e-14, 1.369e-14, 1.094e-14,
9.13e-15, 8.101e-15, 7.828e-15, 8.393e-15, 1.012e-14, 1.259e-14,
1.538e-14, 1.961e-14, 2.619e-14, 3.679e-14, 5.049e-14, 6.917e-14,
8.88e-14, 1.115e-13, 1.373e-13, 1.619e-13, 1.878e-13, 2.111e-13,
2.33e-13, 2.503e-13, 2.613e-13, 2.743e-13, 2.826e-13, 2.976e-13,
3.162e-13, 3.36e-13, 3.491e-13, 3.541e-13, 3.595e-13, 3.608e-13,
3.709e-13, 3.869e-13, 4.12e-13, 4.366e-13, 4.504e-13, 4.379e-13,
3.955e-13, 3.385e-13, 2.741e-13, 2.089e-13, 1.427e-13, 9.294e-14,
```

```
    5.775e-14, 3.565e-14, 2.21e-14, 1.398e-14, 9.194e-15, 6.363e-15,
    4.644e-15, 3.55e-15, 2.808e-15, 2.274e-15, 1.871e-15, 1.557e-15,
    1.308e-15, 1.108e-15, 9.488e-16, 8.222e-16, 7.238e-16, 6.506e-16,
    6.008e-16, 5.742e-16, 5.724e-16, 5.991e-16, 6.625e-16, 7.775e-16,
    9.734e-16, 1.306e-15, 1.88e-15, 2.879e-15, 4.616e-15, 7.579e-15,
    1.248e-14, 2.03e-14, 3.244e-14, 5.171e-14, 7.394e-14, 9.676e-14,
    1.199e-13, 1.467e-13, 1.737e-13, 2.02e-13, 2.425e-13, 3.016e-13,
    3.7e-13, 4.617e-13, 5.949e-13, 7.473e-13, 9.378e-13, 1.191e-12,
    1.481e-12, 1.813e-12, 2.232e-12, 2.722e-12, 3.254e-12, 3.845e-12,
    4.458e-12, 5.048e-12, 5.511e-12, 5.898e-12, 6.204e-12, 6.293e-12,
    6.386e-12, 6.467e-12, 6.507e-12, 6.466e-12, 6.443e-12, 6.598e-12,
    6.873e-12, 7.3e-12, 7.816e-12, 8.368e-12, 8.643e-12, 8.466e-12,
    7.871e-12, 6.853e-12, 5.714e-12, 4.482e-12, 3.392e-12, 2.613e-12,
    2.008e-12, 1.562e-12, 1.228e-12, 9.888e-13, 7.646e-13, 5.769e-13,
    4.368e-13, 3.324e-13, 2.508e-13, 1.916e-13
  };

  static double xfcrev[15] =
    { 1.003, 1.009, 1.015, 1.023, 1.029, 1.033, 1.037,
    1.039, 1.04, 1.046, 1.036, 1.027, 1.01, 1.002, 1.
  };

  double a1, a2, a3, dw, ew, dx, xw, xx, vf2, vf6, cw260, cw296,
    sfac, fscal, cwfrn, ctmpth, ctwfrn, ctwslf;

  int iw, ix;

  /* Get H2O continuum absorption... */
  xw = nu / 10 + 1;
  if (xw >= 1 && xw < 2001) {
    iw = (int) xw;
    dw = xw - iw;
    ew = 1 - dw;
    cw296 = ew * h2o296[iw - 1] + dw * h2o296[iw];
    cw260 = ew * h2o260[iw - 1] + dw * h2o260[iw];
    cwfrn = ew * h2ofrn[iw - 1] + dw * h2ofrn[iw];
    if (nu <= 820 || nu >= 960) {
      sfac = 1;
    } else {
      xx = (nu - 820) / 10;
      ix = (int) xx;
      dx = xx - ix;
      sfac = (1 - dx) * xfcrev[ix] + dx * xfcrev[ix + 1];
    }
    ctwslf = sfac * cw296 * pow(cw260 / cw296, (296 - t) / (296 - 260));
    vf2 = gsl_pow_2(nu - 370);
    vf6 = gsl_pow_3(vf2);
    fscal = 36100 / (vf2 + vf6 * 1e-8 + 36100) * -.25 + 1;
    ctwfrn = cwfrn * fscal;
    a1 = nu * u * tanh(.7193876 / t * nu);
    a2 = 296 / t;
    a3 = p / P0 * (q * ctwslf + (1 - q) * ctwfrn) * 1e-20;
    ctmpth = a1 * a2 * a3;
  } else
    ctmpth = 0;
  return ctmpth;
}

/*****************************************************************************/

double ctmn2(
  double nu,
  double p,
  double t) {

  static double ba[98] = { 0., 4.45e-8, 5.22e-8, 6.46e-8, 7.75e-8, 9.03e-8,
    1.06e-7, 1.21e-7, 1.37e-7, 1.57e-7, 1.75e-7, 2.01e-7, 2.3e-7,
    2.59e-7, 2.95e-7, 3.26e-7, 3.66e-7, 4.05e-7, 4.47e-7, 4.92e-7,
    5.34e-7, 5.84e-7, 6.24e-7, 6.67e-7, 7.14e-7, 7.26e-7, 7.54e-7,
    7.84e-7, 8.09e-7, 8.42e-7, 8.62e-7, 8.87e-7, 9.11e-7, 9.36e-7,
    9.76e-7, 1.03e-6, 1.11e-6, 1.23e-6, 1.39e-6, 1.61e-6, 1.76e-6,
    1.94e-6, 1.97e-6, 1.87e-6, 1.75e-6, 1.56e-6, 1.42e-6, 1.35e-6,
    1.32e-6, 1.29e-6, 1.29e-6, 1.29e-6, 1.3e-6, 1.32e-6, 1.33e-6,
    1.34e-6, 1.35e-6, 1.33e-6, 1.31e-6, 1.29e-6, 1.24e-6, 1.2e-6,
    1.16e-6, 1.1e-6, 1.04e-6, 9.96e-7, 9.38e-7, 8.63e-7, 7.98e-7,
    7.26e-7, 6.55e-7, 5.94e-7, 5.35e-7, 4.74e-7, 4.24e-7, 3.77e-7,
    3.33e-7, 2.96e-7, 2.63e-7, 2.34e-7, 2.08e-7, 1.85e-7, 1.67e-7,
    1.47e-7, 1.32e-7, 1.2e-7, 1.09e-7, 9.85e-8, 9.08e-8, 8.18e-8,
    7.56e-8, 6.85e-8, 6.14e-8, 5.83e-8, 5.77e-8, 5e-8, 4.32e-8, 0.
  };

  static double betaa[98] = { 802., 802., 761., 722., 679., 646., 609., 562.,
    511., 472., 436., 406., 377., 355., 338., 319., 299., 278., 255.,
    233., 208., 184., 149., 107., 66., 25., -13., -49., -82., -104.,
    -119., -130., -139., -144., -146., -146., -147., -148., -150.,
    -153., -160., -169., -181., -189., -195., -200., -205., -209.,
    -211., -210., -210., -209., -205., -199., -190., -180., -168.,
```

```
    -157., -143., -126., -108., -89., -63., -32., 1., 35., 65., 95.,
    121., 141., 152., 161., 164., 164., 161., 155., 148., 143., 137.,
    133., 131., 133., 139., 150., 165., 187., 213., 248., 284., 321.,
    372., 449., 514., 569., 609., 642., 673., 673.
  };

  static double nua[98] = { 2120., 2125., 2130., 2135., 2140., 2145., 2150.,
    2155., 2160., 2165., 2170., 2175., 2180., 2185., 2190., 2195.,
    2200., 2205., 2210., 2215., 2220., 2225., 2230., 2235., 2240.,
    2245., 2250., 2255., 2260., 2265., 2270., 2275., 2280., 2285.,
    2290., 2295., 2300., 2305., 2310., 2315., 2320., 2325., 2330.,
    2335., 2340., 2345., 2350., 2355., 2360., 2365., 2370., 2375.,
    2380., 2385., 2390., 2395., 2400., 2405., 2410., 2415., 2420.,
    2425., 2430., 2435., 2440., 2445., 2450., 2455., 2460., 2465.,
    2470., 2475., 2480., 2485., 2490., 2495., 2500., 2505., 2510.,
    2515., 2520., 2525., 2530., 2535., 2540., 2545., 2550., 2555.,
    2560., 2565., 2570., 2575., 2580., 2585., 2590., 2595., 2600., 2605.
  };

  double b, beta, q_n2 = 0.79, t0 = 273, tr = 296;

  int idx;

  /* Check wavenumber range... */
  if (nu < nua[0] || nu > nua[97])
    return 0;

  /* Interpolate B and beta... */
  idx = locate(nua, 98, nu);
  b = LIN(nua[idx], ba[idx], nua[idx + 1], ba[idx + 1], nu);
  beta = LIN(nua[idx], betaa[idx], nua[idx + 1], betaa[idx + 1], nu);

  /* Compute absorption coefficient... */
  return 0.1 * gsl_pow_2(p / P0) * gsl_pow_2(t0 / t)
    * exp(beta * (1 / tr - 1 / t))
    * q_n2 * b * (q_n2 + (1 - q_n2) * (1.294 - 0.4545 * t / tr));
}

/*****************************************************************************/

double ctmo2(
  double nu,
  double p,
  double t) {

  static double ba[90] = { 0., .061, .074, .084, .096, .12, .162, .208, .246,
    .285, .314, .38, .444, .5, .571, .673, .768, .853, .966, 1.097,
    1.214, 1.333, 1.466, 1.591, 1.693, 1.796, 1.922, 2.037, 2.154,
    2.264, 2.375, 2.508, 2.671, 2.847, 3.066, 3.417, 3.828, 4.204,
    4.453, 4.599, 4.528, 4.284, 3.955, 3.678, 3.477, 3.346, 3.29,
    3.251, 3.231, 3.226, 3.212, 3.192, 3.108, 3.033, 2.911, 2.798,
    2.646, 2.508, 2.322, 2.13, 1.928, 1.757, 1.588, 1.417, 1.253,
    1.109, .99, .888, .791, .678, .587, .524, .464, .403, .357, .32,
    .29, .267, .242, .215, .182, .16, .146, .128, .103, .087, .081,
    .071, .064, 0.
  };

  static double betaa[90] = { 467., 467., 400., 315., 379., 368., 475., 521.,
    531., 512., 442., 444., 430., 381., 335., 324., 296., 248., 215.,
    193., 158., 127., 101., 71., 31., -6., -26., -47., -63., -79.,
    -88., -88., -87., -90., -98., -99., -109., -134., -160., -167.,
    -164., -158., -153., -151., -156., -166., -168., -173., -170.,
    -161., -145., -126., -108., -84., -59., -29., 4., 41., 73., 97.,
    123., 159., 198., 220., 242., 256., 281., 311., 334., 319., 313.,
    321., 323., 310., 315., 320., 335., 361., 378., 373., 338., 319.,
    346., 322., 291., 290., 350., 371., 504., 504.
  };

  static double nua[90] = { 1360., 1365., 1370., 1375., 1380., 1385., 1390.,
    1395., 1400., 1405., 1410., 1415., 1420., 1425., 1430., 1435.,
    1440., 1445., 1450., 1455., 1460., 1465., 1470., 1475., 1480.,
    1485., 1490., 1495., 1500., 1505., 1510., 1515., 1520., 1525.,
    1530., 1535., 1540., 1545., 1550., 1555., 1560., 1565., 1570.,
    1575., 1580., 1585., 1590., 1595., 1600., 1605., 1610., 1615.,
    1620., 1625., 1630., 1635., 1640., 1645., 1650., 1655., 1660.,
    1665., 1670., 1675., 1680., 1685., 1690., 1695., 1700., 1705.,
    1710., 1715., 1720., 1725., 1730., 1735., 1740., 1745., 1750.,
    1755., 1760., 1765., 1770., 1775., 1780., 1785., 1790., 1795.,
    1800., 1805.
  };

  double b, beta, q_o2 = 0.21, t0 = 273, tr = 296;

  int idx;

  /* Check wavenumber range... */
  if (nu < nua[0] || nu > nua[89])
```

```
    return 0;

  /* Interpolate B and beta... */
  idx = locate(nua, 90, nu);
  b = LIN(nua[idx], ba[idx], nua[idx + 1], ba[idx + 1], nu);
  beta = LIN(nua[idx], betaa[idx], nua[idx + 1], betaa[idx + 1], nu);

  /* Compute absorption coefficient... */
  return 0.1 * gsl_pow_2(p / P0) * gsl_pow_2(t0 / t)
    * exp(beta * (1 / tr - 1 / t)) * q_o2 * b;
}

/*****************************************************************************/

void copy_atm(
  ctl_t * ctl,
  atm_t * atm_dest,
  atm_t * atm_src,
  int init) {

  int ig, ip, iw;

  size_t s;

  /* Data size... */
  s = (size_t) atm_src->np * sizeof(double);

  /* Copy data... */
  atm_dest->np = atm_src->np;
  memcpy(atm_dest->time, atm_src->time, s);
  memcpy(atm_dest->z, atm_src->z, s);
  memcpy(atm_dest->lon, atm_src->lon, s);
  memcpy(atm_dest->lat, atm_src->lat, s);
  memcpy(atm_dest->p, atm_src->p, s);
  memcpy(atm_dest->t, atm_src->t, s);
  for (ig = 0; ig < ctl->ng; ig++)
    memcpy(atm_dest->q[ig], atm_src->q[ig], s);
  for (iw = 0; iw < ctl->nw; iw++)
    memcpy(atm_dest->k[iw], atm_src->k[iw], s);

  /* Initialize... */
  if (init)
    for (ip = 0; ip < atm_dest->np; ip++) {
      atm_dest->p[ip] = 0;
      atm_dest->t[ip] = 0;
      for (ig = 0; ig < ctl->ng; ig++)
        atm_dest->q[ig][ip] = 0;
      for (iw = 0; iw < ctl->nw; iw++)
        atm_dest->k[iw][ip] = 0;
    }
}

/*****************************************************************************/

void copy_obs(
  ctl_t * ctl,
  obs_t * obs_dest,
  obs_t * obs_src,
  int init) {

  int id, ir;

  size_t s;

  /* Data size... */
  s = (size_t) obs_src->nr * sizeof(double);

  /* Copy data... */
  obs_dest->nr = obs_src->nr;
  memcpy(obs_dest->time, obs_src->time, s);
  memcpy(obs_dest->obsz, obs_src->obsz, s);
  memcpy(obs_dest->obslon, obs_src->obslon, s);
  memcpy(obs_dest->obslat, obs_src->obslat, s);
  memcpy(obs_dest->vpz, obs_src->vpz, s);
  memcpy(obs_dest->vplon, obs_src->vplon, s);
  memcpy(obs_dest->vplat, obs_src->vplat, s);
  memcpy(obs_dest->tpz, obs_src->tpz, s);
  memcpy(obs_dest->tplon, obs_src->tplon, s);
  memcpy(obs_dest->tplat, obs_src->tplat, s);
  for (id = 0; id < ctl->nd; id++)
    memcpy(obs_dest->rad[id], obs_src->rad[id], s);
  for (id = 0; id < ctl->nd; id++)
    memcpy(obs_dest->tau[id], obs_src->tau[id], s);

  /* Initialize... */
  if (init)
    for (id = 0; id < ctl->nd; id++)
```

```
      for (ir = 0; ir < obs_dest->nr; ir++)
        if (gsl_finite(obs_dest->rad[id][ir])) {
          obs_dest->rad[id][ir] = 0;
          obs_dest->tau[id][ir] = 0;
        }
}

/*****************************************************************************/

int find_emitter(
  ctl_t * ctl,
  const char *emitter) {

  int ig;

  for (ig = 0; ig < ctl->ng; ig++)
    if (strcasecmp(ctl->emitter[ig], emitter) == 0)
      return ig;

  return -1;
}

/*****************************************************************************/

void formod(
  ctl_t * ctl,
  atm_t * atm,
  obs_t * obs) {

  int id, ir, *mask;

  /* Allocate... */
  ALLOC(mask, int,
        ND * NR);

  /* Save observation mask... */
  for (id = 0; id < ctl->nd; id++)
    for (ir = 0; ir < obs->nr; ir++)
      mask[id * NR + ir] = !gsl_finite(obs->rad[id][ir]);

  /* Hydrostatic equilibrium... */
  hydrostatic(ctl, atm);

  /* Claculate pencil beams... */
  for (ir = 0; ir < obs->nr; ir++)
    formod_pencil(ctl, atm, obs, ir);

  /* Apply field-of-view convolution... */
  formod_fov(ctl, obs);

  /* Convert radiance to brightness temperature... */
  if (ctl->write_bbt)
    for (id = 0; id < ctl->nd; id++)
      for (ir = 0; ir < obs->nr; ir++)
        obs->rad[id][ir] = brightness(obs->rad[id][ir], ctl->nu[id]);

  /* Apply observation mask... */
  for (id = 0; id < ctl->nd; id++)
    for (ir = 0; ir < obs->nr; ir++)
      if (mask[id * NR + ir])
        obs->rad[id][ir] = GSL_NAN;

  /* Free... */
  free(mask);
}

/*****************************************************************************/

void formod_continua(
  ctl_t * ctl,
  los_t * los,
  int ip,
  double *beta) {

  static int ig_co2 = -999, ig_h2o = -999;

  int id;

  /* Extinction... */
  for (id = 0; id < ctl->nd; id++)
    beta[id] = los->k[ctl->window[id]][ip];

  /* CO2 continuum... */
  if (ctl->ctm_co2) {
    if (ig_co2 == -999)
      ig_co2 = find_emitter(ctl, "CO2");
    if (ig_co2 >= 0)
```

```
      for (id = 0; id < ctl->nd; id++)
        beta[id] += ctmco2(ctl->nu[id], los->p[ip], los->t[ip],
                           los->u[ig_co2][ip]) / los->ds[ip];
  }

  /* H2O continuum... */
  if (ctl->ctm_h2o) {
    if (ig_h2o == -999)
      ig_h2o = find_emitter(ctl, "H2O");
    if (ig_h2o >= 0)
      for (id = 0; id < ctl->nd; id++)
        beta[id] += ctmh2o(ctl->nu[id], los->p[ip], los->t[ip],
                           los->q[ig_h2o][ip],
                           los->u[ig_h2o][ip]) / los->ds[ip];
  }

  /* N2 continuum... */
  if (ctl->ctm_n2)
    for (id = 0; id < ctl->nd; id++)
      beta[id] += ctmn2(ctl->nu[id], los->p[ip], los->t[ip]);

  /* O2 continuum... */
  if (ctl->ctm_o2)
    for (id = 0; id < ctl->nd; id++)
      beta[id] += ctmo2(ctl->nu[id], los->p[ip], los->t[ip]);
}

/*****************************************************************************/

void formod_fov(
  ctl_t * ctl,
  obs_t * obs) {

  static double dz[NSHAPE], w[NSHAPE];

  static int init = 0, n;

  obs_t *obs2;

  double rad[ND][NR], tau[ND][NR], wsum, z[NR], zfov;

  int i, id, idx, ir, ir2, nz;

  /* Do not take into account FOV... */
  if (ctl->fov[0] == '-')
    return;

  /* Initialize FOV data... */
  if (!init) {
    init = 1;
    read_shape(ctl->fov, dz, w, &n);
  }

  /* Allocate... */
  ALLOC(obs2, obs_t, 1);

  /* Copy observation data... */
  copy_obs(ctl, obs2, obs, 0);

  /* Loop over ray paths... */
  for (ir = 0; ir < obs->nr; ir++) {

    /* Get radiance and transmittance profiles... */
    nz = 0;
    for (ir2 = GSL_MAX(ir - NFOV, 0); ir2 < GSL_MIN(ir + 1 + NFOV, obs->nr);
         ir2++)
      if (obs->time[ir2] == obs->time[ir]) {
        z[nz] = obs2->vpz[ir2];
        for (id = 0; id < ctl->nd; id++) {
          rad[id][nz] = obs2->rad[id][ir2];
          tau[id][nz] = obs2->tau[id][ir2];
        }
        nz++;
      }
    if (nz < 2)
      ERRMSG("Cannot apply FOV convolution!");

    /* Convolute profiles with FOV... */
    wsum = 0;
    for (id = 0; id < ctl->nd; id++) {
      obs->rad[id][ir] = 0;
      obs->tau[id][ir] = 0;
    }
    for (i = 0; i < n; i++) {
      zfov = obs->vpz[ir] + dz[i];
      idx = locate(z, nz, zfov);
      for (id = 0; id < ctl->nd; id++) {
```

```
        obs->rad[id][ir] += w[i]
          * LIN(z[idx], rad[id][idx], z[idx + 1], rad[id][idx + 1], zfov);
        obs->tau[id][ir] += w[i]
          * LIN(z[idx], tau[id][idx], z[idx + 1], tau[id][idx + 1], zfov);
      }
      wsum += w[i];
    }
    for (id = 0; id < ctl->nd; id++) {
      obs->rad[id][ir] /= wsum;
      obs->tau[id][ir] /= wsum;
    }
  }

  /* Free... */
  free(obs2);
}

/*****************************************************************************/

void formod_pencil(
  ctl_t * ctl,
  atm_t * atm,
  obs_t * obs,
  int ir) {

  static tbl_t *tbl;

  static int init = 0;

  los_t *los;

  double beta_ctm[ND], eps, src_planck[ND], tau_path[NG][ND], tau_gas[ND];

  int id, ip;

  /* Initialize look-up tables... */
  if (!init) {
    init = 1;
    ALLOC(tbl, tbl_t, 1);
    init_tbl(ctl, tbl);
  }

  /* Allocate... */
  ALLOC(los, los_t, 1);

  /* Initialize... */
  for (id = 0; id < ctl->nd; id++) {
    obs->rad[id][ir] = 0;
    obs->tau[id][ir] = 1;
  }

  /* Raytracing... */
  raytrace(ctl, atm, obs, los, ir);

  /* Loop over LOS points... */
  for (ip = 0; ip < los->np; ip++) {

    /* Get trace gas transmittance... */
    intpol_tbl(ctl, tbl, los, ip, tau_path, tau_gas);

    /* Get continuum absorption... */
    formod_continua(ctl, los, ip, beta_ctm);

    /* Compute Planck function... */
    formod_srcfunc(ctl, tbl, los->t[ip], src_planck);

    /* Loop over channels... */
    for (id = 0; id < ctl->nd; id++)
      if (tau_gas[id] > 0) {

        /* Get segment emissivity... */
        eps = 1 - tau_gas[id] * exp(-beta_ctm[id] * los->ds[ip]);

        /* Compute radiance... */
        obs->rad[id][ir] += src_planck[id] * eps * obs->tau[id][ir];

        /* Compute path transmittance... */
        obs->tau[id][ir] *= (1 - eps);
      }
  }

  /* Add surface... */
  if (los->tsurf > 0) {
    formod_srcfunc(ctl, tbl, los->tsurf, src_planck);
    for (id = 0; id < ctl->nd; id++)
      obs->rad[id][ir] += src_planck[id] * obs->tau[id][ir];
  }
```

```
  /* Free... */
  free(los);
}

/*****************************************************************************/

void formod_srcfunc(
  ctl_t * ctl,
  tbl_t * tbl,
  double t,
  double *src) {

  int id, it;

  /* Determine index in temperature array... */
  it = locate(tbl->st, TBLNS, t);

  /* Interpolate Planck function value... */
  for (id = 0; id < ctl->nd; id++)
    src[id] = LIN(tbl->st[it], tbl->sr[id][it],
                  tbl->st[it + 1], tbl->sr[id][it + 1], t);
}

/*****************************************************************************/

void geo2cart(
  double z,
  double lon,
  double lat,
  double *x) {

  double radius;

  radius = z + RE;
  x[0] = radius * cos(lat / 180 * M_PI) * cos(lon / 180 * M_PI);
  x[1] = radius * cos(lat / 180 * M_PI) * sin(lon / 180 * M_PI);
  x[2] = radius * sin(lat / 180 * M_PI);
}

/*****************************************************************************/

double gravity(
  double z,
  double lat) {

  /* Compute gravity according to 1967 Geodetic Reference System... */
  return 9.780318 * (1 + 0.0053024 * gsl_pow_2(sin(lat / 180 * M_PI))
                     - 0.0000058 * gsl_pow_2(sin(2 * lat / 180 * M_PI))) -
    3.086e-3 * z;
}

/*****************************************************************************/

void hydrostatic(
  ctl_t * ctl,
  atm_t * atm) {

  static int ig_h2o = -999;

  double dzmin = 1e99, e = 0, mean, mmair = 28.96456e-3, mmh2o =
    18.0153e-3, z;

  int i, ip, ipref = 0, ipts = 20;

  /* Check reference height... */
  if (ctl->hydz < 0)
    return;

  /* Determine emitter index of H2O... */
  if (ig_h2o == -999)
    ig_h2o = find_emitter(ctl, "H2O");

  /* Find air parcel next to reference height... */
  for (ip = 0; ip < atm->np; ip++)
    if (fabs(atm->z[ip] - ctl->hydz) < dzmin) {
      dzmin = fabs(atm->z[ip] - ctl->hydz);
      ipref = ip;
    }

  /* Upper part of profile... */
  for (ip = ipref + 1; ip < atm->np; ip++) {
    mean = 0;
    for (i = 0; i < ipts; i++) {
      z = LIN(0.0, atm->z[ip - 1], ipts - 1.0, atm->z[ip], (double) i);
      if (ig_h2o >= 0)
        e = LIN(0.0, atm->q[ig_h2o][ip - 1],
```

```
                ipts - 1.0, atm->q[ig_h2o][ip], (double) i);
      mean += (e * mmh2o + (1 - e) * mmair)
        * gravity(z, atm->lat[ipref]) / GSL_CONST_MKSA_MOLAR_GAS
        / LIN(0.0, atm->t[ip - 1], ipts - 1.0, atm->t[ip], (double) i) / ipts;
    }

    /* Compute p(z,T)... */
    atm->p[ip] =
      exp(log(atm->p[ip - 1]) - mean * 1000 * (atm->z[ip] - atm->z[ip - 1]));
  }

  /* Lower part of profile... */
  for (ip = ipref - 1; ip >= 0; ip--) {
    mean = 0;
    for (i = 0; i < ipts; i++) {
      z = LIN(0.0, atm->z[ip + 1], ipts - 1.0, atm->z[ip], (double) i);
      if (ig_h2o >= 0)
        e = LIN(0.0, atm->q[ig_h2o][ip + 1],
                ipts - 1.0, atm->q[ig_h2o][ip], (double) i);
      mean += (e * mmh2o + (1 - e) * mmair)
        * gravity(z, atm->lat[ipref]) / GSL_CONST_MKSA_MOLAR_GAS
        / LIN(0.0, atm->t[ip + 1], ipts - 1.0, atm->t[ip], (double) i) / ipts;
    }

    /* Compute p(z,T)... */
    atm->p[ip] =
      exp(log(atm->p[ip + 1]) - mean * 1000 * (atm->z[ip] - atm->z[ip + 1]));
  }
}

/*****************************************************************************/

void idx2name(
  ctl_t * ctl,
  int idx,
  char *quantity) {

  int ig, iw;

  if (idx == IDXP)
    sprintf(quantity, "PRESSURE");

  if (idx == IDXT)
    sprintf(quantity, "TEMPERATURE");

  for (ig = 0; ig < ctl->ng; ig++)
    if (idx == IDXQ(ig))
      sprintf(quantity, "%s", ctl->emitter[ig]);

  for (iw = 0; iw < ctl->nw; iw++)
    if (idx == IDXK(iw))
      sprintf(quantity, "EXTINCT_WINDOW%d", iw);
}

/*****************************************************************************/

void init_tbl(
  ctl_t * ctl,
  tbl_t * tbl) {

  FILE *in;

  char filename[LEN], line[LEN];

  double eps, eps_old, press, press_old, temp, temp_old, u, u_old,
    f[NSHAPE], fsum, nu[NSHAPE];

  int i, id, ig, ip, it, n;

  /* Loop over trace gases and channels... */
  for (ig = 0; ig < ctl->ng; ig++)
#pragma omp parallel for default(none) shared(ctl,tbl,ig) private(in,filename,line,eps,eps_old,press,press_old,temp,temp_old,u,u_old,id,ip,it)
    for (id = 0; id < ctl->nd; id++) {

      /* Initialize... */
      tbl->np[ig][id] = -1;
      eps_old = -999;
      press_old = -999;
      temp_old = -999;
      u_old = -999;

      /* Try to open file... */
      sprintf(filename, "%s_%.4f_%s.tab",
              ctl->tblbase, ctl->nu[id], ctl->emitter[ig]);
      if (!(in = fopen(filename, "r"))) {
        printf("Missing emissivity table: %s\n", filename);
```

```
        continue;
      }
      printf("Read emissivity table: %s\n", filename);

      /* Read data... */
      while (fgets(line, LEN, in)) {

        /* Parse line... */
        if (sscanf(line, "%lg %lg %lg %lg", &press, &temp, &u, &eps) != 4)
          continue;

        /* Determine pressure index... */
        if (press != press_old) {
          press_old = press;
          if ((++tbl->np[ig][id]) >= TBLNP)
            ERRMSG("Too many pressure levels!");
          tbl->nt[ig][id][tbl->np[ig][id]] = -1;
        }

        /* Determine temperature index... */
        if (temp != temp_old) {
          temp_old = temp;
          if ((++tbl->nt[ig][id][tbl->np[ig][id]]) >= TBLNT)
            ERRMSG("Too many temperatures!");
          tbl->nu[ig][id][tbl->np[ig][id]]
            [tbl->nt[ig][id][tbl->np[ig][id]]] = -1;
        }

        /* Determine column density index... */
        if ((eps > eps_old && u > u_old) || tbl->nu[ig][id][tbl->np[ig][id]]
            [tbl->nt[ig][id][tbl->np[ig][id]]] < 0) {
          eps_old = eps;
          u_old = u;
          if ((++tbl->nu[ig][id][tbl->np[ig][id]]
               [tbl->nt[ig][id][tbl->np[ig][id]]]) >= TBLNU) {
            tbl->nu[ig][id][tbl->np[ig][id]]
              [tbl->nt[ig][id][tbl->np[ig][id]]]--;
            continue;
          }
        }

        /* Store data... */
        tbl->p[ig][id][tbl->np[ig][id]] = press;
        tbl->t[ig][id][tbl->np[ig][id]][tbl->nt[ig][id][tbl->np[ig][id]]]
          = temp;
        tbl->u[ig][id][tbl->np[ig][id]][tbl->nt[ig][id][tbl->np[ig][id]]]
          [tbl->nu[ig][id][tbl->np[ig][id]]
           [tbl->nt[ig][id][tbl->np[ig][id]]]] = (float) u;
        tbl->eps[ig][id][tbl->np[ig][id]][tbl->nt[ig][id][tbl->np[ig][id]]]
          [tbl->nu[ig][id][tbl->np[ig][id]]
           [tbl->nt[ig][id][tbl->np[ig][id]]]] = (float) eps;
      }

      /* Increment counters... */
      tbl->np[ig][id]++;
      for (ip = 0; ip < tbl->np[ig][id]; ip++) {
        tbl->nt[ig][id][ip]++;
        for (it = 0; it < tbl->nt[ig][id][ip]; it++)
          tbl->nu[ig][id][ip][it]++;
      }

      /* Close file... */
      fclose(in);
    }

  /* Write info... */
  printf("Initialize source function table...\n");

  /* Loop over channels... */
#pragma omp parallel for default(none) shared(ctl,tbl,ig) private(filename,it,i,n,f,fsum,nu)
  for (id = 0; id < ctl->nd; id++) {

    /* Read filter function... */
    sprintf(filename, "%s_%.4f.filt", ctl->tblbase, ctl->nu[id]);
    read_shape(filename, nu, f, &n);

    /* Compute source function table... */
    for (it = 0; it < TBLNS; it++) {

      /* Set temperature... */
      tbl->st[it] = LIN(0.0, TMIN, TBLNS - 1.0, TMAX, (double) it);

      /* Integrate Planck function... */
      fsum = 0;
      tbl->sr[id][it] = 0;
      for (i = 0; i < n; i++) {
        fsum += f[i];
```

```
        tbl->sr[id][it] += f[i] * planck(tbl->st[it], nu[i]);
      }
      tbl->sr[id][it] /= fsum;
    }
  }
}

/*****************************************************************************/

void intpol_atm(
  ctl_t * ctl,
  atm_t * atm,
  double z,
  double *p,
  double *t,
  double *q,
  double *k) {

  int ig, ip, iw;

  /* Get array index... */
  ip = locate(atm->z, atm->np, z);

  /* Interpolate... */
  *p = EXP(atm->z[ip], atm->p[ip], atm->z[ip + 1], atm->p[ip + 1], z);
  *t = LIN(atm->z[ip], atm->t[ip], atm->z[ip + 1], atm->t[ip + 1], z);
  for (ig = 0; ig < ctl->ng; ig++)
    q[ig] =
      LIN(atm->z[ip], atm->q[ig][ip], atm->z[ip + 1], atm->q[ig][ip + 1], z);
  for (iw = 0; iw < ctl->nw; iw++)
    k[iw] =
      LIN(atm->z[ip], atm->k[iw][ip], atm->z[ip + 1], atm->k[iw][ip + 1], z);
}

/*****************************************************************************/

void intpol_tbl(
  ctl_t * ctl,
  tbl_t * tbl,
  los_t * los,
  int ip,
  double tau_path[NG][ND],
  double tau_seg[ND]) {

  double eps, eps00, eps01, eps10, eps11, u;

  int id, ig, ipr, it0, it1;

  /* Initialize... */
  if (ip <= 0)
    for (ig = 0; ig < ctl->ng; ig++)
      for (id = 0; id < ctl->nd; id++)
        tau_path[ig][id] = 1;

  /* Loop over channels... */
  for (id = 0; id < ctl->nd; id++) {

    /* Initialize... */
    tau_seg[id] = 1;

    /* Loop over emitters.... */
    for (ig = 0; ig < ctl->ng; ig++) {

      /* Check size of table (pressure)... */
      if (tbl->np[ig][id] < 2)
        eps = 0;

      /* Check transmittance... */
      else if (tau_path[ig][id] < 1e-9)
        eps = 1;

      /* Interpolate... */
      else {

        /* Determine pressure and temperature indices... */
        ipr = locate(tbl->p[ig][id], tbl->np[ig][id], los->p[ip]);
        it0 = locate(tbl->t[ig][id][ipr], tbl->nt[ig][id][ipr], los->t[ip]);
        it1 =
          locate(tbl->t[ig][id][ipr + 1], tbl->nt[ig][id][ipr + 1],
                 los->t[ip]);

        /* Check size of table (temperature and column density)... */
        if (tbl->nt[ig][id][ipr] < 2 || tbl->nt[ig][id][ipr + 1] < 2
            || tbl->nu[ig][id][ipr][it0] < 2
            || tbl->nu[ig][id][ipr][it0 + 1] < 2
            || tbl->nu[ig][id][ipr + 1][it1] < 2
            || tbl->nu[ig][id][ipr + 1][it1 + 1] < 2)
```

```
          eps = 0;

        else {

          /* Get emissivities of extended path... */
          u = intpol_tbl_u(tbl, ig, id, ipr, it0, 1 - tau_path[ig][id]);
          eps00 = intpol_tbl_eps(tbl, ig, id, ipr, it0, u + los->u[ig][ip]);

          u = intpol_tbl_u(tbl, ig, id, ipr, it0 + 1, 1 - tau_path[ig][id]);
          eps01 =
            intpol_tbl_eps(tbl, ig, id, ipr, it0 + 1, u + los->u[ig][ip]);

          u = intpol_tbl_u(tbl, ig, id, ipr + 1, it1, 1 - tau_path[ig][id]);
          eps10 =
            intpol_tbl_eps(tbl, ig, id, ipr + 1, it1, u + los->u[ig][ip]);

          u =
            intpol_tbl_u(tbl, ig, id, ipr + 1, it1 + 1, 1 - tau_path[ig][id]);
          eps11 =
            intpol_tbl_eps(tbl, ig, id, ipr + 1, it1 + 1, u + los->
u[ig][ip]);

          /* Interpolate with respect to temperature... */
          eps00 = LIN(tbl->t[ig][id][ipr][it0], eps00,
                      tbl->t[ig][id][ipr][it0 + 1], eps01, los->t[ip]);
          eps11 = LIN(tbl->t[ig][id][ipr + 1][it1], eps10,
                      tbl->t[ig][id][ipr + 1][it1 + 1], eps11, los->t[ip]);

          /* Interpolate with respect to pressure... */
          eps00 = LIN(tbl->p[ig][id][ipr], eps00,
                      tbl->p[ig][id][ipr + 1], eps11, los->p[ip]);

          /* Check emssivity range... */
          eps00 = GSL_MAX(GSL_MIN(eps00, 1), 0);

          /* Determine segment emissivity... */
          eps = 1 - (1 - eps00) / tau_path[ig][id];
        }
      }

      /* Get transmittance of extended path... */
      tau_path[ig][id] *= (1 - eps);

      /* Get segment transmittance... */
      tau_seg[id] *= (1 - eps);
    }
  }
}

/*****************************************************************************/

double intpol_tbl_eps(
  tbl_t * tbl,
  int ig,
  int id,
  int ip,
  int it,
  double u) {

  int idx;

  /* Lower boundary... */
  if (u < tbl->u[ig][id][ip][it][0])
    return LIN(0, 0, tbl->u[ig][id][ip][it][0], tbl->eps[ig][id][ip][it][0],
               u);

  /* Upper boundary... */
  else if (u > tbl->u[ig][id][ip][it][tbl->nu[ig][id][ip][it] - 1])
    return LIN(tbl->u[ig][id][ip][it][tbl->nu[ig][id][ip][it] - 1],
               tbl->eps[ig][id][ip][it][tbl->nu[ig][id][ip][it] - 1],
               1e30, 1, u);

  /* Interpolation... */
  else {

    /* Get index... */
    idx = locate_tbl(tbl->u[ig][id][ip][it], tbl->nu[ig][id][ip][it], u);

    /* Interpolate... */
    return
      LIN(tbl->u[ig][id][ip][it][idx], tbl->eps[ig][id][ip][it][idx],
          tbl->u[ig][id][ip][it][idx + 1], tbl->eps[ig][id][ip][it][idx + 1],
          u);
  }
}

/*****************************************************************************/
```

```
double intpol_tbl_u(
  tbl_t * tbl,
  int ig,
  int id,
  int ip,
  int it,
  double eps) {

  int idx;

  /* Lower boundary... */
  if (eps < tbl->eps[ig][id][ip][it][0])
    return LIN(0, 0, tbl->eps[ig][id][ip][it][0], tbl->u[ig][id][ip][it][0],
               eps);

  /* Upper boundary... */
  else if (eps > tbl->eps[ig][id][ip][it][tbl->nu[ig][id][ip][it] - 1])
    return LIN(tbl->eps[ig][id][ip][it][tbl->nu[ig][id][ip][it] - 1],
               tbl->u[ig][id][ip][it][tbl->nu[ig][id][ip][it] - 1],
               1, 1e30, eps);

  /* Interpolation... */
  else {

    /* Get index... */
    idx = locate_tbl(tbl->eps[ig][id][ip][it], tbl->nu[ig][id][ip][it], eps);

    /* Interpolate... */
    return
      LIN(tbl->eps[ig][id][ip][it][idx], tbl->u[ig][id][ip][it][idx],
          tbl->eps[ig][id][ip][it][idx + 1], tbl->u[ig][id][ip][it][idx + 1],
          eps);
  }
}

/*****************************************************************************/

void jsec2time(
  double jsec,
  int *year,
  int *mon,
  int *day,
  int *hour,
  int *min,
  int *sec,
  double *remain) {

  struct tm t0, *t1;

  time_t jsec0;

  t0.tm_year = 100;
  t0.tm_mon = 0;
  t0.tm_mday = 1;
  t0.tm_hour = 0;
  t0.tm_min = 0;
  t0.tm_sec = 0;

  jsec0 = (time_t) jsec + timegm(&t0);
  t1 = gmtime(&jsec0);

  *year = t1->tm_year + 1900;
  *mon = t1->tm_mon + 1;
  *day = t1->tm_mday;
  *hour = t1->tm_hour;
  *min = t1->tm_min;
  *sec = t1->tm_sec;
  *remain = jsec - floor(jsec);
}

/*****************************************************************************/

void kernel(
  ctl_t * ctl,
  atm_t * atm,
  obs_t * obs,
  gsl_matrix * k) {

  atm_t *atm1;
  obs_t *obs1;

  gsl_vector *x0, *x1, *yy0, *yy1;

  int *iqa, j;

  double h;
```

```

  size_t i, n, m;

  /* Get sizes... */
  m = k->size1;
  n = k->size2;

  /* Allocate... */
  x0 = gsl_vector_alloc(n);
  yy0 = gsl_vector_alloc(m);
  ALLOC(iqa, int,
        N);

  /* Compute radiance for undisturbed atmospheric data... */
  formod(ctl, atm, obs);

  /* Compose vectors... */
  atm2x(ctl, atm, x0, iqa, NULL);
  obs2y(ctl, obs, yy0, NULL, NULL);

  /* Initialize kernel matrix... */
  gsl_matrix_set_zero(k);

  /* Loop over state vector elements... */
#pragma omp parallel for default(none) shared(ctl,atm,obs,k,x0,yy0,n,m,iqa) private(i, j, h, x1, yy1, atm1,
 obs1)
  for (j = 0; j < (int) n; j++) {

    /* Allocate... */
    x1 = gsl_vector_alloc(n);
    yy1 = gsl_vector_alloc(m);
    ALLOC(atm1, atm_t, 1);
    ALLOC(obs1, obs_t, 1);

    /* Set perturbation size... */
    if (iqa[j] == IDXP)
      h = GSL_MAX(fabs(0.01 * gsl_vector_get(x0, (size_t) j)), 1e-7);
    else if (iqa[j] == IDXT)
      h = 1;
    else if (iqa[j] >= IDXQ(0) && iqa[j] < IDXQ(ctl->ng))
      h = GSL_MAX(fabs(0.01 * gsl_vector_get(x0, (size_t) j)), 1e-15);
    else if (iqa[j] >= IDXK(0) && iqa[j] < IDXK(ctl->nw))
      h = 1e-4;
    else
      ERRMSG("Cannot set perturbation size!");

    /* Disturb state vector element... */
    gsl_vector_memcpy(x1, x0);
    gsl_vector_set(x1, (size_t) j, gsl_vector_get(x1, (size_t) j) + h);
    copy_atm(ctl, atm1, atm, 0);
    copy_obs(ctl, obs1, obs, 0);
    x2atm(ctl, x1, atm1);

    /* Compute radiance for disturbed atmospheric data... */
    formod(ctl, atm1, obs1);

    /* Compose measurement vector for disturbed radiance data... */
    obs2y(ctl, obs1, yy1, NULL, NULL);

    /* Compute derivatives... */
    for (i = 0; i < m; i++)
      gsl_matrix_set(k, i, (size_t) j,
                     (gsl_vector_get(yy1, i) - gsl_vector_get(yy0, i)) / h);

    /* Free... */
    gsl_vector_free(x1);
    gsl_vector_free(yy1);
    free(atm1);
    free(obs1);
  }

  /* Free... */
  gsl_vector_free(x0);
  gsl_vector_free(yy0);
  free(iqa);
}

/*****************************************************************************/

int locate(
  double *xx,
  int n,
  double x) {

  int i, ilo, ihi;

  ilo = 0;
```

```
  ihi = n - 1;
  i = (ihi + ilo) >> 1;

  if (xx[i] < xx[i + 1])
    while (ihi > ilo + 1) {
      i = (ihi + ilo) >> 1;
      if (xx[i] > x)
        ihi = i;
      else
        ilo = i;
  } else
    while (ihi > ilo + 1) {
      i = (ihi + ilo) >> 1;
      if (xx[i] <= x)
        ihi = i;
      else
        ilo = i;
    }

  return ilo;
}

/*****************************************************************************/

int locate_tbl(
  float *xx,
  int n,
  double x) {

  int i, ilo, ihi;

  ilo = 0;
  ihi = n - 1;
  i = (ihi + ilo) >> 1;

  while (ihi > ilo + 1) {
    i = (ihi + ilo) >> 1;
    if (xx[i] > x)
      ihi = i;
    else
      ilo = i;
  }

  return ilo;
}

/*****************************************************************************/

size_t obs2y(
  ctl_t * ctl,
  obs_t * obs,
  gsl_vector * y,
  int *ida,
  int *ira) {

  int id, ir;

  size_t m = 0;

  /* Determine measurement vector... */
  for (ir = 0; ir < obs->nr; ir++)
    for (id = 0; id < ctl->nd; id++)
      if (gsl_finite(obs->rad[id][ir])) {
        if (y != NULL)
          gsl_vector_set(y, m, obs->rad[id][ir]);
        if (ida != NULL)
          ida[m] = id;
        if (ira != NULL)
          ira[m] = ir;
        m++;
      }

  return m;
}

/*****************************************************************************/

double planck(
  double t,
  double nu) {

  return C1 * gsl_pow_3(nu) / gsl_expm1(C2 * nu / t);
}

/*****************************************************************************/

void raytrace(
```

```
ctl_t * ctl,
atm_t * atm,
obs_t * obs,
los_t * los,
int ir) {

double cosa, d, dmax, dmin = 0, ds, ex0[3], ex1[3], frac, h = 0.02, k[NW],
  lat, lon, n, naux, ng[3], norm, p, q[NG], t, x[3], xh[3],
  xobs[3], xvp[3], z = 1e99, zmax, zmin, zrefrac = 60;

int i, ig, ip, iw, stop = 0;

/* Initialize... */
los->np = 0;
los->tsurf = -999;
obs->tpz[ir] = obs->vpz[ir];
obs->tplon[ir] = obs->vplon[ir];
obs->tplat[ir] = obs->vplat[ir];

/* Get altitude range of atmospheric data... */
gsl_stats_minmax(&zmin, &zmax, atm->z, 1, (size_t) atm->np);

/* Check observer altitude... */
if (obs->obsz[ir] < zmin)
  ERRMSG("Observer below surface!");

/* Check view point altitude... */
if (obs->vpz[ir] > zmax)
  return;

/* Determine Cartesian coordinates for observer and view point... */
geo2cart(obs->obsz[ir], obs->obslon[ir], obs->obslat[ir], xobs);
geo2cart(obs->vpz[ir], obs->vplon[ir], obs->vplat[ir], xvp);

/* Determine initial tangent vector... */
for (i = 0; i < 3; i++)
  ex0[i] = xvp[i] - xobs[i];
norm = NORM(ex0);
for (i = 0; i < 3; i++)
  ex0[i] /= norm;

/* Observer within atmosphere... */
for (i = 0; i < 3; i++)
  x[i] = xobs[i];

/* Observer above atmosphere (search entry point)... */
if (obs->obsz[ir] > zmax) {
  dmax = norm;
  while (fabs(dmin - dmax) > 0.001) {
    d = (dmax + dmin) / 2;
    for (i = 0; i < 3; i++)
      x[i] = xobs[i] + d * ex0[i];
    cart2geo(x, &z, &lon, &lat);
    if (z <= zmax && z > zmax - 0.001)
      break;
    if (z < zmax - 0.0005)
      dmax = d;
    else
      dmin = d;
  }
}

/* Ray-tracing... */
while (1) {

  /* Set step length... */
  ds = ctl->rayds;
  if (ctl->raydz > 0) {
    norm = NORM(x);
    for (i = 0; i < 3; i++)
      xh[i] = x[i] / norm;
    cosa = fabs(DOTP(ex0, xh));
    if (cosa != 0)
      ds = GSL_MIN(ctl->rayds, ctl->raydz / cosa);
  }

  /* Determine geolocation... */
  cart2geo(x, &z, &lon, &lat);

  /* Check if LOS hits the ground or has left atmosphere... */
  if (z < zmin || z > zmax) {
    stop = (z < zmin ? 2 : 1);
    frac =
      ((z <
        zmin ? zmin : zmax) - los->z[los->np - 1]) / (z - los->z[los->np -
                                                                   1]);
    geo2cart(los->z[los->np - 1], los->lon[los->np - 1],
```

```
              los->lat[los->np - 1], xh);
      for (i = 0; i < 3; i++)
        x[i] = xh[i] + frac * (x[i] - xh[i]);
      cart2geo(x, &z, &lon, &lat);
      los->ds[los->np - 1] = ds * frac;
      ds = 0;
    }

    /* Interpolate atmospheric data... */
    intpol_atm(ctl, atm, z, &p, &t, q, k);

    /* Save data... */
    los->lon[los->np] = lon;
    los->lat[los->np] = lat;
    los->z[los->np] = z;
    los->p[los->np] = p;
    los->t[los->np] = t;
    for (ig = 0; ig < ctl->ng; ig++)
      los->q[ig][los->np] = q[ig];
    for (iw = 0; iw < ctl->nw; iw++)
      los->k[iw][los->np] = k[iw];
    los->ds[los->np] = ds;

    /* Increment and check number of LOS points... */
    if ((++los->np) > NLOS)
      ERRMSG("Too many LOS points!");

    /* Check stop flag... */
    if (stop) {
      los->tsurf = (stop == 2 ? t : -999);
      break;
    }

    /* Determine refractivity... */
    if (ctl->refrac && z <= zrefrac)
      n = 1 + refractivity(p, t);
    else
      n = 1;

    /* Construct new tangent vector (first term)... */
    for (i = 0; i < 3; i++)
      ex1[i] = ex0[i] * n;

    /* Compute gradient of refractivity... */
    if (ctl->refrac && z <= zrefrac) {
      for (i = 0; i < 3; i++)
        xh[i] = x[i] + 0.5 * ds * ex0[i];
      cart2geo(xh, &z, &lon, &lat);
      intpol_atm(ctl, atm, z, &p, &t, q, k);
      n = refractivity(p, t);
      for (i = 0; i < 3; i++) {
        xh[i] += h;
        cart2geo(xh, &z, &lon, &lat);
        intpol_atm(ctl, atm, z, &p, &t, q, k);
        naux = refractivity(p, t);
        ng[i] = (naux - n) / h;
        xh[i] -= h;
      }
    } else
      for (i = 0; i < 3; i++)
        ng[i] = 0;

    /* Construct new tangent vector (second term)... */
    for (i = 0; i < 3; i++)
      ex1[i] += ds * ng[i];

    /* Normalize new tangent vector... */
    norm = NORM(ex1);
    for (i = 0; i < 3; i++)
      ex1[i] /= norm;

    /* Determine next point of LOS... */
    for (i = 0; i < 3; i++)
      x[i] += 0.5 * ds * (ex0[i] + ex1[i]);

    /* Copy tangent vector... */
    for (i = 0; i < 3; i++)
      ex0[i] = ex1[i];
  }

  /* Get tangent point (to be done before changing segment lengths!)... */
  tangent_point(los, &obs->tpz[ir], &obs->tplon[ir], &obs->
tplat[ir]);

  /* Change segment lengths according to trapezoid rule... */
  for (ip = los->np - 1; ip >= 1; ip--)
    los->ds[ip] = 0.5 * (los->ds[ip - 1] + los->ds[ip]);
```

```
  los->ds[0] *= 0.5;

  /* Compute column density... */
  for (ip = 0; ip < los->np; ip++)
    for (ig = 0; ig < ctl->ng; ig++)
      los->u[ig][ip] = 10 * los->q[ig][ip] * los->p[ip]
        / (GSL_CONST_MKSA_BOLTZMANN * los->t[ip]) * los->ds[ip];
}

/*****************************************************************************/

void read_atm(
  const char *dirname,
  const char *filename,
  ctl_t * ctl,
  atm_t * atm) {

  FILE *in;

  char file[LEN], line[LEN], *tok;

  int ig, iw;

  /* Init... */
  atm->np = 0;

  /* Set filename... */
  if (dirname != NULL)
    sprintf(file, "%s/%s", dirname, filename);
  else
    sprintf(file, "%s", filename);

  /* Write info... */
  printf("Read atmospheric data: %s\n", file);

  /* Open file... */
  if (!(in = fopen(file, "r")))
    ERRMSG("Cannot open file!");

  /* Read line... */
  while (fgets(line, LEN, in)) {

    /* Read data... */
    TOK(line, tok, "%lg", atm->time[atm->np]);
    TOK(NULL, tok, "%lg", atm->z[atm->np]);
    TOK(NULL, tok, "%lg", atm->lon[atm->np]);
    TOK(NULL, tok, "%lg", atm->lat[atm->np]);
    TOK(NULL, tok, "%lg", atm->p[atm->np]);
    TOK(NULL, tok, "%lg", atm->t[atm->np]);
    for (ig = 0; ig < ctl->ng; ig++)
      TOK(NULL, tok, "%lg", atm->q[ig][atm->np]);
    for (iw = 0; iw < ctl->nw; iw++)
      TOK(NULL, tok, "%lg", atm->k[iw][atm->np]);

    /* Increment data point counter... */
    if ((++atm->np) > NP)
      ERRMSG("Too many data points!");
  }

  /* Close file... */
  fclose(in);

  /* Check number of points... */
  if (atm->np < 1)
    ERRMSG("Could not read any data!");
}

/*****************************************************************************/

void read_ctl(
  int argc,
  char *argv[],
  ctl_t * ctl) {

  int id, ig, iw;

  /* Write info... */
  printf("\nJuelich Rapid Spectral Simulation Code (JURASSIC)\n"
         "(executable: %s | compiled: %s, %s)\n\n",
         argv[0], __DATE__, __TIME__);

  /* Emitters... */
  ctl->ng = (int) scan_ctl(argc, argv, "NG", -1, "0", NULL);
  if (ctl->ng < 0 || ctl->ng > NG)
    ERRMSG("Set 0 <= NG <= MAX!");
  for (ig = 0; ig < ctl->ng; ig++)
    scan_ctl(argc, argv, "EMITTER", ig, "", ctl->emitter[ig]);
```

```c

  /* Radiance channels... */
  ctl->nd = (int) scan_ctl(argc, argv, "ND", -1, "0", NULL);
  if (ctl->nd < 0 || ctl->nd > ND)
    ERRMSG("Set 0 <= ND <= MAX!");
  for (id = 0; id < ctl->nd; id++)
    ctl->nu[id] = scan_ctl(argc, argv, "NU", id, "", NULL);

  /* Spectral windows... */
  ctl->nw = (int) scan_ctl(argc, argv, "NW", -1, "1", NULL);
  if (ctl->nw < 0 || ctl->nw > NW)
    ERRMSG("Set 0 <= NW <= MAX!");
  for (id = 0; id < ctl->nd; id++)
    ctl->window[id] = (int) scan_ctl(argc, argv, "WINDOW", id, "0", NULL);

  /* Emissivity look-up tables... */
  scan_ctl(argc, argv, "TBLBASE", -1, "-", ctl->tblbase);

  /* Hydrostatic equilibrium... */
  ctl->hydz = scan_ctl(argc, argv, "HYDZ", -1, "-999", NULL);

  /* Continua... */
  ctl->ctm_co2 = (int) scan_ctl(argc, argv, "CTM_CO2", -1, "1", NULL);
  ctl->ctm_h2o = (int) scan_ctl(argc, argv, "CTM_H2O", -1, "1", NULL);
  ctl->ctm_n2 = (int) scan_ctl(argc, argv, "CTM_N2", -1, "1", NULL);
  ctl->ctm_o2 = (int) scan_ctl(argc, argv, "CTM_O2", -1, "1", NULL);

  /* Ray-tracing... */
  ctl->refrac = (int) scan_ctl(argc, argv, "REFRAC", -1, "1", NULL);
  ctl->rayds = scan_ctl(argc, argv, "RAYDS", -1, "10", NULL);
  ctl->raydz = scan_ctl(argc, argv, "RAYDZ", -1, "0.5", NULL);

  /* Field of view... */
  scan_ctl(argc, argv, "FOV", -1, "-", ctl->fov);

  /* Retrieval interface... */
  ctl->retp_zmin = scan_ctl(argc, argv, "RETP_ZMIN", -1, "-999", NULL);
  ctl->retp_zmax = scan_ctl(argc, argv, "RETP_ZMAX", -1, "-999", NULL);
  ctl->rett_zmin = scan_ctl(argc, argv, "RETT_ZMIN", -1, "-999", NULL);
  ctl->rett_zmax = scan_ctl(argc, argv, "RETT_ZMAX", -1, "-999", NULL);
  for (ig = 0; ig < ctl->ng; ig++) {
    ctl->retq_zmin[ig] = scan_ctl(argc, argv, "RETQ_ZMIN", ig, "-999", NULL);
    ctl->retq_zmax[ig] = scan_ctl(argc, argv, "RETQ_ZMAX", ig, "-999", NULL);
  }
  for (iw = 0; iw < ctl->nw; iw++) {
    ctl->retk_zmin[iw] = scan_ctl(argc, argv, "RETK_ZMIN", iw, "-999", NULL);
    ctl->retk_zmax[iw] = scan_ctl(argc, argv, "RETK_ZMAX", iw, "-999", NULL);
  }

  /* Output flags... */
  ctl->write_bbt = (int) scan_ctl(argc, argv, "WRITE_BBT", -1, "0", NULL);
  ctl->write_matrix =
    (int) scan_ctl(argc, argv, "WRITE_MATRIX", -1, "0", NULL);
}

/*****************************************************************************/

void read_matrix(
  const char *dirname,
  const char *filename,
  gsl_matrix * matrix) {

  FILE *in;

  char dum[LEN], file[LEN], line[LEN];

  double value;

  int i, j;

  /* Set filename... */
  if (dirname != NULL)
    sprintf(file, "%s/%s", dirname, filename);
  else
    sprintf(file, "%s", filename);

  /* Write info... */
  printf("Read matrix: %s\n", file);

  /* Open file... */
  if (!(in = fopen(file, "r")))
    ERRMSG("Cannot open file!");

  /* Read data... */
  gsl_matrix_set_zero(matrix);
  while (fgets(line, LEN, in))
    if (sscanf(line, "%d %s %s %s %s %s %d %s %s %s %s %s %lg",
```

```
                &i, dum, dum, dum, dum, dum,
                &j, dum, dum, dum, dum, dum, &value) == 13)
      gsl_matrix_set(matrix, (size_t) i, (size_t) j, value);

  /* Close file... */
  fclose(in);
}

/*****************************************************************************/

void read_obs(
  const char *dirname,
  const char *filename,
  ctl_t * ctl,
  obs_t * obs) {

  FILE *in;

  char file[LEN], line[LEN], *tok;

  int id;

  /* Init... */
  obs->nr = 0;

  /* Set filename... */
  if (dirname != NULL)
    sprintf(file, "%s/%s", dirname, filename);
  else
    sprintf(file, "%s", filename);

  /* Write info... */
  printf("Read observation data: %s\n", file);

  /* Open file... */
  if (!(in = fopen(file, "r")))
    ERRMSG("Cannot open file!");

  /* Read line... */
  while (fgets(line, LEN, in)) {

    /* Read data... */
    TOK(line, tok, "%lg", obs->time[obs->nr]);
    TOK(NULL, tok, "%lg", obs->obsz[obs->nr]);
    TOK(NULL, tok, "%lg", obs->obslon[obs->nr]);
    TOK(NULL, tok, "%lg", obs->obslat[obs->nr]);
    TOK(NULL, tok, "%lg", obs->vpz[obs->nr]);
    TOK(NULL, tok, "%lg", obs->vplon[obs->nr]);
    TOK(NULL, tok, "%lg", obs->vplat[obs->nr]);
    TOK(NULL, tok, "%lg", obs->tpz[obs->nr]);
    TOK(NULL, tok, "%lg", obs->tplon[obs->nr]);
    TOK(NULL, tok, "%lg", obs->tplat[obs->nr]);
    for (id = 0; id < ctl->nd; id++)
      TOK(NULL, tok, "%lg", obs->rad[id][obs->nr]);
    for (id = 0; id < ctl->nd; id++)
      TOK(NULL, tok, "%lg", obs->tau[id][obs->nr]);

    /* Increment counter... */
    if ((++obs->nr) > NR)
      ERRMSG("Too many rays!");
  }

  /* Close file... */
  fclose(in);

  /* Check number of points... */
  if (obs->nr < 1)
    ERRMSG("Could not read any data!");
}

/*****************************************************************************/

void read_shape(
  const char *filename,
  double *x,
  double *y,
  int *n) {

  FILE *in;

  char line[LEN];

  /* Write info... */
  printf("Read shape function: %s\n", filename);

  /* Open file... */
  if (!(in = fopen(filename, "r")))
```

```
    ERRMSG("Cannot open file!");

  /* Read data... */
  *n = 0;
  while (fgets(line, LEN, in))
    if (sscanf(line, "%lg %lg", &x[*n], &y[*n]) == 2)
      if ((++(*n)) > NSHAPE)
        ERRMSG("Too many data points!");

  /* Check number of points... */
  if (*n < 1)
    ERRMSG("Could not read any data!");

  /* Close file... */
  fclose(in);
}

/*****************************************************************************/

double refractivity(
  double p,
  double t) {

  /* Refractivity of air at 4 to 15 micron... */
  return 7.753e-05 * p / t;
}

/*****************************************************************************/

double scan_ctl(
  int argc,
  char *argv[],
  const char *varname,
  int arridx,
  const char *defvalue,
  char *value) {

  FILE *in = NULL;

  char dummy[LEN], fullname1[LEN], fullname2[LEN], line[LEN],
    msg[LEN], rvarname[LEN], rval[LEN];

  int contain = 0, i;

  /* Open file... */
  if (argv[1][0] != '-')
    if (!(in = fopen(argv[1], "r")))
      ERRMSG("Cannot open file!");

  /* Set full variable name... */
  if (arridx >= 0) {
    sprintf(fullname1, "%s[%d]", varname, arridx);
    sprintf(fullname2, "%s[*]", varname);
  } else {
    sprintf(fullname1, "%s", varname);
    sprintf(fullname2, "%s", varname);
  }

  /* Read data... */
  if (in != NULL)
    while (fgets(line, LEN, in))
      if (sscanf(line, "%s %s %s", rvarname, dummy, rval) == 3)
        if (strcasecmp(rvarname, fullname1) == 0 ||
            strcasecmp(rvarname, fullname2) == 0) {
          contain = 1;
          break;
        }
  for (i = 1; i < argc - 1; i++)
    if (strcasecmp(argv[i], fullname1) == 0 ||
        strcasecmp(argv[i], fullname2) == 0) {
      sprintf(rval, "%s", argv[i + 1]);
      contain = 1;
      break;
    }

  /* Close file... */
  if (in != NULL)
    fclose(in);

  /* Check for missing variables... */
  if (!contain) {
    if (strlen(defvalue) > 0)
      sprintf(rval, "%s", defvalue);
    else {
      sprintf(msg, "Missing variable %s!\n", fullname1);
      ERRMSG(msg);
    }
```

```
  }

  /* Write info... */
  printf("%s = %s\n", fullname1, rval);

  /* Return values... */
  if (value != NULL)
    sprintf(value, "%s", rval);
  return atof(rval);
}

/*****************************************************************************/

void tangent_point(
  los_t * los,
  double *tpz,
  double *tplon,
  double *tplat) {

  double a, b, c, dummy, v[3], v0[3], v2[3], x, x1, x2, yy0, yy1, yy2;

  size_t i, ip;

  /* Find minimum altitude... */
  ip = gsl_stats_min_index(los->z, 1, (size_t) los->np);

  /* Nadir or zenith... */
  if (ip <= 0 || ip >= (size_t) los->np - 1) {
    *tpz = los->z[los->np - 1];
    *tplon = los->lon[los->np - 1];
    *tplat = los->lat[los->np - 1];
  }

  /* Limb... */
  else {

    /* Determine interpolating polynomial y=a*x^2+b*x+c... */
    yy0 = los->z[ip - 1];
    yy1 = los->z[ip];
    yy2 = los->z[ip + 1];
    x1 = sqrt(gsl_pow_2(los->ds[ip]) - gsl_pow_2(yy1 - yy0));
    x2 = x1 + sqrt(gsl_pow_2(los->ds[ip + 1]) - gsl_pow_2(yy2 - yy1));
    a = 1 / (x1 - x2) * (-(yy0 - yy1) / x1 + (yy0 - yy2) / x2);
    b = -(yy0 - yy1) / x1 - a * x1;
    c = yy0;

    /* Get tangent point location... */
    x = -b / (2 * a);
    *tpz = a * x * x + b * x + c;
    geo2cart(los->z[ip - 1], los->lon[ip - 1], los->lat[ip - 1], v0);
    geo2cart(los->z[ip + 1], los->lon[ip + 1], los->lat[ip + 1], v2);
    for (i = 0; i < 3; i++)
      v[i] = LIN(0.0, v0[i], x2, v2[i], x);
    cart2geo(v, &dummy, tplon, tplat);
  }
}

/*****************************************************************************/

void time2jsec(
  int year,
  int mon,
  int day,
  int hour,
  int min,
  int sec,
  double remain,
  double *jsec) {

  struct tm t0, t1;

  t0.tm_year = 100;
  t0.tm_mon = 0;
  t0.tm_mday = 1;
  t0.tm_hour = 0;
  t0.tm_min = 0;
  t0.tm_sec = 0;

  t1.tm_year = year - 1900;
  t1.tm_mon = mon - 1;
  t1.tm_mday = day;
  t1.tm_hour = hour;
  t1.tm_min = min;
  t1.tm_sec = sec;

  *jsec = (double) timegm(&t1) - (double) timegm(&t0) + remain;
}
```

```c

/*****************************************************************************/

void timer(
  const char *name,
  const char *file,
  const char *func,
  int line,
  int mode) {

  static double dt_w, w0[10];

  static int l0[10], nt;

  struct timeval tim;

  /* Start new timer... */
  if (mode == 1) {
    gettimeofday(&tim, NULL);
    w0[nt] = (double) tim.tv_sec + (double) tim.tv_usec / 1e6;
    l0[nt] = line;
    if ((++nt) >= 10)
      ERRMSG("Too many timers!");
  }

  /* Write elapsed time... */
  else {

    /* Check timer index... */
    if (nt - 1 < 0)
      ERRMSG("Coding error!");

    /* Get time differences... */
    gettimeofday(&tim, NULL);
    dt_w = (double) tim.tv_sec + (double) tim.tv_usec / 1e6 - w0[nt - 1];

    /* Write elapsed time... */
    printf("Timer '%s' (%s, %s, l%d-%d): %.3f sec\n",
           name, file, func, l0[nt - 1], line, dt_w);
  }

  /* Stop timer... */
  if (mode == 3)
    nt--;
}

/*****************************************************************************/

void write_atm(
  const char *dirname,
  const char *filename,
  ctl_t * ctl,
  atm_t * atm) {

  FILE *out;

  char file[LEN];

  int ig, ip, iw, n = 6;

  /* Set filename... */
  if (dirname != NULL)
    sprintf(file, "%s/%s", dirname, filename);
  else
    sprintf(file, "%s", filename);

  /* Write info... */
  printf("Write atmospheric data: %s\n", file);

  /* Create file... */
  if (!(out = fopen(file, "w")))
    ERRMSG("Cannot create file!");

  /* Write header... */
  fprintf(out,
          "# $1 = time (seconds since 2000-01-01T00:00Z)\n"
          "# $2 = altitude [km]\n"
          "# $3 = longitude [deg]\n"
          "# $4 = latitude [deg]\n"
          "# $5 = pressure [hPa]\n" "# $6 = temperature [K]\n");
  for (ig = 0; ig < ctl->ng; ig++)
    fprintf(out, "# $%d = %s volume mixing ratio\n", ++n, ctl->emitter[ig]);
  for (iw = 0; iw < ctl->nw; iw++)
    fprintf(out, "# $%d = window %d: extinction [1/km]\n", ++n, iw);

  /* Write data... */
  for (ip = 0; ip < atm->np; ip++) {
```

```
    if (ip == 0 || atm->lat[ip] != atm->lat[ip - 1]
        || atm->lon[ip] != atm->lon[ip - 1])
      fprintf(out, "\n");
    fprintf(out, "%.2f %g %g %g %g %g", atm->time[ip], atm->z[ip],
            atm->lon[ip], atm->lat[ip], atm->p[ip], atm->t[ip]);
    for (ig = 0; ig < ctl->ng; ig++)
      fprintf(out, " %g", atm->q[ig][ip]);
    for (iw = 0; iw < ctl->nw; iw++)
      fprintf(out, " %g", atm->k[iw][ip]);
    fprintf(out, "\n");
  }

  /* Close file... */
  fclose(out);
}

/*****************************************************************************/

void write_matrix(
  const char *dirname,
  const char *filename,
  ctl_t * ctl,
  gsl_matrix * matrix,
  atm_t * atm,
  obs_t * obs,
  const char *rowspace,
  const char *colspace,
  const char *sort) {

  FILE *out;

  char file[LEN], quantity[LEN];

  int *cida, *ciqa, *cipa, *cira, *rida, *riqa, *ripa, *rira;

  size_t i, j, nc, nr;

  /* Check output flag... */
  if (!ctl->write_matrix)
    return;

  /* Allocate... */
  ALLOC(cida, int, M);
  ALLOC(ciqa, int,
        N);
  ALLOC(cipa, int,
        N);
  ALLOC(cira, int,
        M);
  ALLOC(rida, int,
        M);
  ALLOC(riqa, int,
        N);
  ALLOC(ripa, int,
        N);
  ALLOC(rira, int,
        M);

  /* Set filename... */
  if (dirname != NULL)
    sprintf(file, "%s/%s", dirname, filename);
  else
    sprintf(file, "%s", filename);

  /* Write info... */
  printf("Write matrix: %s\n", file);

  /* Create file... */
  if (!(out = fopen(file, "w")))
    ERRMSG("Cannot create file!");

  /* Write header (row space)... */
  if (rowspace[0] == 'y') {

    fprintf(out,
            "# $1 = Row: index (measurement space)\n"
            "# $2 = Row: channel wavenumber [cm^-1]\n"
            "# $3 = Row: time (seconds since 2000-01-01T00:00Z)\n"
            "# $4 = Row: view point altitude [km]\n"
            "# $5 = Row: view point longitude [deg]\n"
            "# $6 = Row: view point latitude [deg]\n");

    /* Get number of rows... */
    nr = obs2y(ctl, obs, NULL, rida, rira);

  } else {

```

```
    fprintf(out,
            "# $1 = Row: index (state space)\n"
            "# $2 = Row: name of quantity\n"
            "# $3 = Row: time (seconds since 2000-01-01T00:00Z)\n"
            "# $4 = Row: altitude [km]\n"
            "# $5 = Row: longitude [deg]\n" "# $6 = Row: latitude [deg]\n");

    /* Get number of rows... */
    nr = atm2x(ctl, atm, NULL, riqa, ripa);
  }

  /* Write header (column space)... */
  if (colspace[0] == 'y') {

    fprintf(out,
            "# $7 = Col: index (measurement space)\n"
            "# $8 = Col: channel wavenumber [cm^-1]\n"
            "# $9 = Col: time (seconds since 2000-01-01T00:00Z)\n"
            "# $10 = Col: view point altitude [km]\n"
            "# $11 = Col: view point longitude [deg]\n"
            "# $12 = Col: view point latitude [deg]\n");

    /* Get number of columns... */
    nc = obs2y(ctl, obs, NULL, cida, cira);

  } else {

    fprintf(out,
            "# $7 = Col: index (state space)\n"
            "# $8 = Col: name of quantity\n"
            "# $9 = Col: time (seconds since 2000-01-01T00:00Z)\n"
            "# $10 = Col: altitude [km]\n"
            "# $11 = Col: longitude [deg]\n" "# $12 = Col: latitude [deg]\n");

    /* Get number of columns... */
    nc = atm2x(ctl, atm, NULL, ciqa, cipa);
  }

  /* Write header entry... */
  fprintf(out, "# $13 = Matrix element\n\n");

  /* Write matrix data... */
  i = j = 0;
  while (i < nr && j < nc) {

    /* Write info about the row... */
    if (rowspace[0] == 'y')
      fprintf(out, "%d %g %.2f %g %g %g",
              (int) i, ctl->nu[rida[i]],
              obs->time[rira[i]], obs->vpz[rira[i]],
              obs->vplon[rira[i]], obs->vplat[rira[i]]);
    else {
      idx2name(ctl, riqa[i], quantity);
      fprintf(out, "%d %s %.2f %g %g %g", (int) i, quantity,
              atm->time[ripa[i]], atm->z[ripa[i]],
              atm->lon[ripa[i]], atm->lat[ripa[i]]);
    }

    /* Write info about the column... */
    if (colspace[0] == 'y')
      fprintf(out, " %d %g %.2f %g %g %g",
              (int) j, ctl->nu[cida[j]],
              obs->time[cira[j]], obs->vpz[cira[j]],
              obs->vplon[cira[j]], obs->vplat[cira[j]]);
    else {
      idx2name(ctl, ciqa[j], quantity);
      fprintf(out, " %d %s %.2f %g %g %g", (int) j, quantity,
              atm->time[cipa[j]], atm->z[cipa[j]],
              atm->lon[cipa[j]], atm->lat[cipa[j]]);
    }

    /* Write matrix entry... */
    fprintf(out, " %g\n", gsl_matrix_get(matrix, i, j));

    /* Set matrix indices... */
    if (sort[0] == 'r') {
      j++;
      if (j >= nc) {
        j = 0;
        i++;
        fprintf(out, "\n");
      }
    } else {
      i++;
      if (i >= nr) {
        i = 0;
        j++;
```

```
      fprintf(out, "\n");
    }
  }
}

/* Close file... */
fclose(out);

/* Free... */
free(cida);
free(ciqa);
free(cipa);
free(cira);
free(rida);
free(riqa);
free(ripa);
free(rira);
}

/*****************************************************************************/

void write_obs(
  const char *dirname,
  const char *filename,
  ctl_t * ctl,
  obs_t * obs) {

  FILE *out;

  char file[LEN];

  int id, ir, n = 10;

  /* Set filename... */
  if (dirname != NULL)
    sprintf(file, "%s/%s", dirname, filename);
  else
    sprintf(file, "%s", filename);

  /* Write info... */
  printf("Write observation data: %s\n", file);

  /* Create file... */
  if (!(out = fopen(file, "w")))
    ERRMSG("Cannot create file!");

  /* Write header... */
  fprintf(out,
          "# $1 = time (seconds since 2000-01-01T00:00Z)\n"
          "# $2 = observer altitude [km]\n"
          "# $3 = observer longitude [deg]\n"
          "# $4 = observer latitude [deg]\n"
          "# $5 = view point altitude [km]\n"
          "# $6 = view point longitude [deg]\n"
          "# $7 = view point latitude [deg]\n"
          "# $8 = tangent point altitude [km]\n"
          "# $9 = tangent point longitude [deg]\n"
          "# $10 = tangent point latitude [deg]\n");
  for (id = 0; id < ctl->nd; id++)
    fprintf(out, "# $%d = channel %g: radiance [W/(m^2 sr cm^-1)]\n",
            ++n, ctl->nu[id]);
  for (id = 0; id < ctl->nd; id++)
    fprintf(out, "# $%d = channel %g: transmittance\n", ++n, ctl->nu[id]);

  /* Write data... */
  for (ir = 0; ir < obs->nr; ir++) {
    if (ir == 0 || obs->time[ir] != obs->time[ir - 1])
      fprintf(out, "\n");
    fprintf(out, "%.2f %g %g %g %g %g %g %g %g %g", obs->time[ir],
            obs->obsz[ir], obs->obslon[ir], obs->obslat[ir],
            obs->vpz[ir], obs->vplon[ir], obs->vplat[ir],
            obs->tpz[ir], obs->tplon[ir], obs->tplat[ir]);
    for (id = 0; id < ctl->nd; id++)
      fprintf(out, " %g", obs->rad[id][ir]);
    for (id = 0; id < ctl->nd; id++)
      fprintf(out, " %g", obs->tau[id][ir]);
    fprintf(out, "\n");
  }

  /* Close file... */
  fclose(out);
}

/*****************************************************************************/

void x2atm(
  ctl_t * ctl,
```

```
  gsl_vector * x,
  atm_t * atm) {

  int ig, iw;

  size_t n = 0;

  /* Set pressure... */
  x2atm_help(atm, ctl->retp_zmin, ctl->retp_zmax, atm->p, x, &n);

  /* Set temperature... */
  x2atm_help(atm, ctl->rett_zmin, ctl->rett_zmax, atm->t, x, &n);

  /* Set volume mixing ratio... */
  for (ig = 0; ig < ctl->ng; ig++)
    x2atm_help(atm, ctl->retq_zmin[ig], ctl->retq_zmax[ig],
               atm->q[ig], x, &n);

  /* Set extinction... */
  for (iw = 0; iw < ctl->nw; iw++)
    x2atm_help(atm, ctl->retk_zmin[iw], ctl->retk_zmax[iw],
               atm->k[iw], x, &n);
}

/*****************************************************************************/

void x2atm_help(
  atm_t * atm,
  double zmin,
  double zmax,
  double *value,
  gsl_vector * x,
  size_t * n) {

  int ip;

  /* Extract state vector elements... */
  for (ip = 0; ip < atm->np; ip++)
    if (atm->z[ip] >= zmin && atm->z[ip] <= zmax) {
      value[ip] = gsl_vector_get(x, *n);
      (*n)++;
    }
}

/*****************************************************************************/

void y2obs(
  ctl_t * ctl,
  gsl_vector * y,
  obs_t * obs) {

  int id, ir;

  size_t m = 0;

  /* Decompose measurement vector... */
  for (ir = 0; ir < obs->nr; ir++)
    for (id = 0; id < ctl->nd; id++)
      if (gsl_finite(obs->rad[id][ir])) {
        obs->rad[id][ir] = gsl_vector_get(y, m);
        m++;
      }
}
```

5.33 jurassic.h File Reference

JURASSIC library declarations.

Data Structures

- struct atm_t

  Atmospheric data.
- struct ctl_t

  Forward model control parameters.

- struct los_t

    Line-of-sight data.
- struct obs_t

    Observation geometry and radiance data.
- struct tbl_t

    Emissivity look-up tables.

Functions

- size_t atm2x (ctl_t ∗ctl, atm_t ∗atm, gsl_vector ∗x, int ∗iqa, int ∗ipa)

    Compose state vector or parameter vector.
- void atm2x_help (atm_t ∗atm, double zmin, double zmax, double ∗value, int val_iqa, gsl_vector ∗x, int ∗iqa, int ∗ipa, size_t ∗n)

    Add elements to state vector.
- double brightness (double rad, double nu)

    Compute brightness temperature.
- void cart2geo (double ∗x, double ∗z, double ∗lon, double ∗lat)

    Convert Cartesian coordinates to geolocation.
- void climatology (ctl_t ∗ctl, atm_t ∗atm_mean)

    Interpolate climatological data.
- double ctmco2 (double nu, double p, double t, double u)

    Compute carbon dioxide continuum (optical depth).
- double ctmh2o (double nu, double p, double t, double q, double u)

    Compute water vapor continuum (optical depth).
- double ctmn2 (double nu, double p, double t)

    Compute nitrogen continuum (absorption coefficient).
- double ctmo2 (double nu, double p, double t)

    Compute oxygen continuum (absorption coefficient).
- void copy_atm (ctl_t ∗ctl, atm_t ∗atm_dest, atm_t ∗atm_src, int init)

    Copy and initialize atmospheric data.
- void copy_obs (ctl_t ∗ctl, obs_t ∗obs_dest, obs_t ∗obs_src, int init)

    Copy and initialize observation data.
- int find_emitter (ctl_t ∗ctl, const char ∗emitter)

    Find index of an emitter.
- void formod (ctl_t ∗ctl, atm_t ∗atm, obs_t ∗obs)

    Determine ray paths and compute radiative transfer.
- void formod_continua (ctl_t ∗ctl, los_t ∗los, int ip, double ∗beta)

    Compute absorption coefficient of continua.
- void formod_fov (ctl_t ∗ctl, obs_t ∗obs)

    Apply field of view convolution.
- void formod_pencil (ctl_t ∗ctl, atm_t ∗atm, obs_t ∗obs, int ir)

    Compute radiative transfer for a pencil beam.
- void formod_srcfunc (ctl_t ∗ctl, tbl_t ∗tbl, double t, double ∗src)

    Compute Planck source function.
- void geo2cart (double z, double lon, double lat, double ∗x)

    Convert geolocation to Cartesian coordinates.
- double gravity (double z, double lat)

    Determine gravity of Earth.
- void hydrostatic (ctl_t ∗ctl, atm_t ∗atm)

    Set hydrostatic equilibrium.

- void idx2name (ctl_t *ctl, int idx, char *quantity)

  Determine name of state vector quantity for given index.
- void init_tbl (ctl_t *ctl, tbl_t *tbl)

  Initialize look-up tables.
- void intpol_atm (ctl_t *ctl, atm_t *atm, double z, double *p, double *t, double *q, double *k)

  Interpolate atmospheric data.
- void intpol_tbl (ctl_t *ctl, tbl_t *tbl, los_t *los, int ip, double tau_path[NG][ND], double tau_seg[ND])

  Get transmittance from look-up tables.
- double intpol_tbl_eps (tbl_t *tbl, int ig, int id, int ip, int it, double u)

  Interpolate emissivity from look-up tables.
- double intpol_tbl_u (tbl_t *tbl, int ig, int id, int ip, int it, double eps)

  Interpolate column density from look-up tables.
- void jsec2time (double jsec, int *year, int *mon, int *day, int *hour, int *min, int *sec, double *remain)

  Convert seconds to date.
- void kernel (ctl_t *ctl, atm_t *atm, obs_t *obs, gsl_matrix *k)

  Compute Jacobians.
- int locate (double *xx, int n, double x)

  Find array index.
- int locate_tbl (float *xx, int n, double x)

  Find array index in float array.
- size_t obs2y (ctl_t *ctl, obs_t *obs, gsl_vector *y, int *ida, int *ira)

  Compose measurement vector.
- double planck (double t, double nu)

  Compute Planck function.
- void raytrace (ctl_t *ctl, atm_t *atm, obs_t *obs, los_t *los, int ir)

  Do ray-tracing to determine LOS.
- void read_atm (const char *dirname, const char *filename, ctl_t *ctl, atm_t *atm)

  Read atmospheric data.
- void read_ctl (int argc, char *argv[ ], ctl_t *ctl)

  Read forward model control parameters.
- void read_matrix (const char *dirname, const char *filename, gsl_matrix *matrix)

  Read matrix.
- void read_obs (const char *dirname, const char *filename, ctl_t *ctl, obs_t *obs)

  Read observation data.
- void read_shape (const char *filename, double *x, double *y, int *n)

  Read shape function.
- double refractivity (double p, double t)

  Compute refractivity (return value is n - 1).
- double scan_ctl (int argc, char *argv[ ], const char *varname, int arridx, const char *defvalue, char *value)

  Search control parameter file for variable entry.
- void tangent_point (los_t *los, double *tpz, double *tplon, double *tplat)

  Find tangent point of a given LOS.
- void time2jsec (int year, int mon, int day, int hour, int min, int sec, double remain, double *jsec)

  Convert date to seconds.
- void timer (const char *name, const char *file, const char *func, int line, int mode)

  Measure wall-clock time.
- void write_atm (const char *dirname, const char *filename, ctl_t *ctl, atm_t *atm)

  Write atmospheric data.
- void write_matrix (const char *dirname, const char *filename, ctl_t *ctl, gsl_matrix *matrix, atm_t *atm, obs_t *obs, const char *rowspace, const char *colspace, const char *sort)

  Write matrix.

- void write_obs (const char ∗dirname, const char ∗filename, ctl_t ∗ctl, obs_t ∗obs)

    Write observation data.
- void x2atm (ctl_t ∗ctl, gsl_vector ∗x, atm_t ∗atm)

    Decompose parameter vector or state vector.
- void x2atm_help (atm_t ∗atm, double zmin, double zmax, double ∗value, gsl_vector ∗x, size_t ∗n)

    Extract elements from state vector.
- void y2obs (ctl_t ∗ctl, gsl_vector ∗y, obs_t ∗obs)

    Decompose measurement vector.

5.33.1 Detailed Description

JURASSIC library declarations.

Definition in file jurassic.h.

5.33.2 Function Documentation

5.33.2.1 size_t atm2x (ctl_t ∗ *ctl,* atm_t ∗ *atm,* gsl_vector ∗ *x,* int ∗ *iqa,* int ∗ *ipa*)

Compose state vector or parameter vector.

Definition at line 29 of file jurassic.c.

```
               {

  int ig, iw;

  size_t n = 0;

  /* Add pressure... */
  atm2x_help(atm, ctl->retp_zmin, ctl->retp_zmax,
             atm->p, IDXP, x, iqa, ipa, &n);

  /* Add temperature... */
  atm2x_help(atm, ctl->rett_zmin, ctl->rett_zmax,
             atm->t, IDXT, x, iqa, ipa, &n);

  /* Add volume mixing ratios... */
  for (ig = 0; ig < ctl->ng; ig++)
    atm2x_help(atm, ctl->retq_zmin[ig], ctl->retq_zmax[ig],
               atm->q[ig], IDXQ(ig), x, iqa, ipa, &n);

  /* Add extinction... */
  for (iw = 0; iw < ctl->nw; iw++)
    atm2x_help(atm, ctl->retk_zmin[iw], ctl->retk_zmax[iw],
               atm->k[iw], IDXK(iw), x, iqa, ipa, &n);

  return n;
}
```

Here is the call graph for this function:

5.33.2.2 void atm2x_help (atm_t * *atm,* double *zmin,* double *zmax,* double * *value,* int *val_iqa,* gsl_vector * *x,* int * *iqa,* int * *ipa,* size_t * *n*)

Add elements to state vector.

Definition at line 63 of file jurassic.c.

```
                {

  int ip;

  /* Add elements to state vector... */
  for (ip = 0; ip < atm->np; ip++)
    if (atm->z[ip] >= zmin && atm->z[ip] <= zmax) {
      if (x != NULL)
        gsl_vector_set(x, *n, value[ip]);
      if (iqa != NULL)
        iqa[*n] = val_iqa;
      if (ipa != NULL)
        ipa[*n] = ip;
      (*n)++;
    }
}
```

5.33.2.3 double brightness (double *rad,* double *nu*)

Compute brightness temperature.

Definition at line 91 of file jurassic.c.

```
               {

  return C2 * nu / gsl_log1p(C1 * gsl_pow_3(nu) / rad);
}
```

5.33.2.4 void cart2geo (double * *x,* double * *z,* double * *lon,* double * *lat*)

Convert Cartesian coordinates to geolocation.

Definition at line 101 of file jurassic.c.

```
                 {

  double radius;

  radius = NORM(x);
  *lat = asin(x[2] / radius) * 180 / M_PI;
  *lon = atan2(x[1], x[0]) * 180 / M_PI;
  *z = radius - RE;
}
```

5.33.2.5 void climatology (ctl_t * *ctl*, atm_t * *atm_mean*)

Interpolate climatological data.

Definition at line 117 of file jurassic.c.

```
              {

  static double z[121] = {
    0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19,
    20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37,
    38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55,
    56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73,
    74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91,
    92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107,
    108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 120
  };

  static double pre[121] = {
    1017, 901.083, 796.45, 702.227, 617.614, 541.644, 473.437, 412.288,
    357.603, 308.96, 265.994, 228.348, 195.619, 167.351, 143.039, 122.198,
    104.369, 89.141, 76.1528, 65.0804, 55.641, 47.591, 40.7233, 34.8637,
    29.8633, 25.5956, 21.9534, 18.8445, 16.1909, 13.9258, 11.9913,
    10.34, 8.92988, 7.72454, 6.6924, 5.80701, 5.04654, 4.39238, 3.82902,
    3.34337, 2.92413, 2.56128, 2.2464, 1.97258, 1.73384, 1.52519, 1.34242,
    1.18197, 1.04086, 0.916546, 0.806832, 0.709875, 0.624101, 0.548176,
    0.480974, 0.421507, 0.368904, 0.322408, 0.281386, 0.245249, 0.213465,
    0.185549, 0.161072, 0.139644, 0.120913, 0.104568, 0.0903249, 0.0779269,
    0.0671493, 0.0577962, 0.0496902, 0.0426736, 0.0366093, 0.0313743,
    0.0268598, 0.0229699, 0.0196206, 0.0167399, 0.0142646, 0.0121397,
    0.0103181, 0.00875775, 0.00742226, 0.00628076, 0.00530519, 0.00447183,
    0.00376124, 0.00315632, 0.00264248, 0.00220738, 0.00184003, 0.00153095,
    0.00127204, 0.00105608, 0.000876652, 0.00072798, 0.00060492,
    0.000503201, 0.000419226, 0.000349896, 0.000292659, 0.000245421,
    0.000206394, 0.000174125, 0.000147441, 0.000125333, 0.000106985,
    9.173e-05, 7.90172e-05, 6.84172e-05, 5.95574e-05, 5.21183e-05,
    4.58348e-05, 4.05127e-05, 3.59987e-05, 3.21583e-05, 2.88718e-05,
    2.60322e-05, 2.35687e-05, 2.14263e-05, 1.95489e-05
  };

  static double tem[121] = {
    285.14, 279.34, 273.91, 268.3, 263.24, 256.55, 250.2, 242.82, 236.17,
    229.87, 225.04, 221.19, 218.85, 217.19, 216.2, 215.68, 215.42, 215.55,
    215.92, 216.4, 216.93, 217.45, 218, 218.68, 219.39, 220.25, 221.3,
    222.41, 223.88, 225.42, 227.2, 229.52, 231.89, 234.51, 236.85, 239.42,
    241.94, 244.57, 247.36, 250.32, 253.34, 255.82, 258.27, 260.39,
    262.03, 263.45, 264.2, 264.78, 264.67, 264.38, 263.24, 262.03, 260.02,
    258.09, 255.63, 253.28, 250.43, 247.81, 245.26, 242.77, 240.38,
    237.94, 235.79, 233.53, 231.5, 229.53, 227.6, 225.62, 223.77, 222.06,
    220.33, 218.69, 217.18, 215.64, 214.13, 212.52, 210.86, 209.25,
    207.49, 205.81, 204.11, 202.22, 200.32, 198.39, 195.92, 193.46,
    190.94, 188.31, 185.82, 183.57, 181.43, 179.74, 178.64, 178.1, 178.25,
    178.7, 179.41, 180.67, 182.31, 184.18, 186.6, 189.53, 192.66, 196.54,
    201.13, 205.93, 211.73, 217.86, 225, 233.53, 242.57, 252.14, 261.48,
    272.97, 285.26, 299.12, 312.2, 324.17, 338.34, 352.56, 365.28
  };

  static double c2h2[121] = {
    1.352e-09, 2.83e-10, 1.269e-10, 6.926e-11, 4.346e-11, 2.909e-11,
    2.014e-11, 1.363e-11, 8.71e-12, 5.237e-12, 2.718e-12, 1.375e-12,
    5.786e-13, 2.16e-13, 7.317e-14, 2.551e-14, 1.055e-14, 4.758e-15,
    2.056e-15, 7.703e-16, 2.82e-16, 1.035e-16, 4.382e-17, 1.946e-17,
    9.638e-18, 5.2e-18, 2.811e-18, 1.494e-18, 7.925e-19, 4.213e-19,
    1.998e-19, 8.78e-20, 3.877e-20, 1.728e-20, 7.743e-21, 3.536e-21,
    1.623e-21, 7.508e-22, 3.508e-22, 1.65e-22, 7.837e-23, 3.733e-23,
    1.808e-23, 8.77e-24, 4.285e-24, 2.095e-24, 1.032e-24, 5.082e-25,
    2.506e-25, 1.236e-25, 6.088e-26, 2.996e-26, 1.465e-26, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
  };

  static double c2h6[121] = {
    2.667e-09, 2.02e-09, 1.658e-09, 1.404e-09, 1.234e-09, 1.109e-09,
    1.012e-09, 9.262e-10, 8.472e-10, 7.71e-10, 6.932e-10, 6.216e-10,
    5.503e-10, 4.87e-10, 4.342e-10, 3.861e-10, 3.347e-10, 2.772e-10,
    2.209e-10, 1.672e-10, 1.197e-10, 8.536e-11, 5.783e-11, 3.846e-11,
    2.495e-11, 1.592e-11, 1.017e-11, 6.327e-12, 3.895e-12, 2.403e-12,
    1.416e-12, 8.101e-13, 4.649e-13, 2.686e-13, 1.557e-13, 9.14e-14,
    5.386e-14, 3.19e-14, 1.903e-14, 1.14e-14, 6.875e-15, 4.154e-15,
    2.538e-15, 1.553e-15, 9.548e-16, 5.872e-16, 3.63e-16, 2.244e-16,
    1.388e-16, 8.587e-17, 5.308e-17, 3.279e-17, 2.017e-17, 1.238e-17,
    7.542e-18, 4.585e-18, 2.776e-18, 1.671e-18, 9.985e-19, 5.937e-19,
```

```
  3.518e-19, 2.07e-19, 1.215e-19, 7.06e-20, 4.097e-20, 2.37e-20,
  1.363e-20, 7.802e-21, 4.441e-21, 2.523e-21, 1.424e-21, 8.015e-22,
  4.497e-22, 2.505e-22, 1.391e-22, 7.691e-23, 4.238e-23, 2.331e-23,
  1.274e-23, 6.929e-24, 3.752e-24, 2.02e-24, 1.083e-24, 5.774e-25,
  3.041e-25, 1.593e-25, 8.308e-26, 4.299e-26, 2.195e-26, 1.112e-26,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0
};

static double ccl4[121] = {
  1.075e-10, 1.075e-10, 1.075e-10, 1.075e-10, 1.075e-10, 1.075e-10,
  1.075e-10, 1.075e-10, 1.075e-10, 1.06e-10, 1.024e-10, 9.69e-11,
  8.93e-11, 8.078e-11, 7.213e-11, 6.307e-11, 5.383e-11, 4.49e-11,
  3.609e-11, 2.705e-11, 1.935e-11, 1.385e-11, 8.35e-12, 5.485e-12,
  3.853e-12, 2.22e-12, 5.875e-13, 3.445e-13, 1.015e-13, 6.075e-14,
  4.383e-14, 2.692e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14,
  1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14,
  1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14,
  1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14,
  1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14,
  1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14,
  1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14,
  1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14,
  1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14,
  1e-14, 1e-14, 1e-14
};

static double ch4[121] = {
  1.864e-06, 1.835e-06, 1.819e-06, 1.805e-06, 1.796e-06, 1.788e-06,
  1.782e-06, 1.776e-06, 1.769e-06, 1.761e-06, 1.749e-06, 1.734e-06,
  1.716e-06, 1.692e-06, 1.654e-06, 1.61e-06, 1.567e-06, 1.502e-06,
  1.433e-06, 1.371e-06, 1.323e-06, 1.277e-06, 1.232e-06, 1.188e-06,
  1.147e-06, 1.108e-06, 1.07e-06, 1.027e-06, 9.854e-07, 9.416e-07,
  8.933e-07, 8.478e-07, 7.988e-07, 7.515e-07, 7.07e-07, 6.64e-07,
  6.239e-07, 5.864e-07, 5.512e-07, 5.184e-07, 4.87e-07, 4.571e-07,
  4.296e-07, 4.04e-07, 3.802e-07, 3.578e-07, 3.383e-07, 3.203e-07,
  3.032e-07, 2.889e-07, 2.76e-07, 2.635e-07, 2.519e-07, 2.409e-07,
  2.302e-07, 2.219e-07, 2.144e-07, 2.071e-07, 1.999e-07, 1.93e-07,
  1.862e-07, 1.795e-07, 1.731e-07, 1.668e-07, 1.607e-07, 1.548e-07,
  1.49e-07, 1.434e-07, 1.38e-07, 1.328e-07, 1.277e-07, 1.227e-07,
  1.18e-07, 1.134e-07, 1.089e-07, 1.046e-07, 1.004e-07, 9.635e-08,
  9.245e-08, 8.867e-08, 8.502e-08, 8.15e-08, 7.809e-08, 7.48e-08,
  7.159e-08, 6.849e-08, 6.55e-08, 6.262e-08, 5.98e-08, 5.708e-08,
  5.448e-08, 5.194e-08, 4.951e-08, 4.72e-08, 4.5e-08, 4.291e-08,
  4.093e-08, 3.905e-08, 3.729e-08, 3.563e-08, 3.408e-08, 3.265e-08,
  3.128e-08, 2.996e-08, 2.87e-08, 2.76e-08, 2.657e-08, 2.558e-08,
  2.467e-08, 2.385e-08, 2.307e-08, 2.234e-08, 2.168e-08, 2.108e-08,
  2.05e-08, 1.998e-08, 1.947e-08, 1.902e-08, 1.86e-08, 1.819e-08,
  1.782e-08
};

static double clo[121] = {
  7.419e-15, 1.061e-14, 1.518e-14, 2.195e-14, 3.175e-14, 4.666e-14,
  6.872e-14, 1.03e-13, 1.553e-13, 2.375e-13, 3.664e-13, 5.684e-13,
  8.915e-13, 1.402e-12, 2.269e-12, 4.125e-12, 7.501e-12, 1.257e-11,
  2.048e-11, 3.338e-11, 5.44e-11, 8.846e-11, 1.008e-10, 1.082e-10,
  1.157e-10, 1.232e-10, 1.312e-10, 1.539e-10, 1.822e-10, 2.118e-10,
  2.387e-10, 2.687e-10, 2.875e-10, 3.031e-10, 3.23e-10, 3.648e-10,
  4.117e-10, 4.477e-10, 4.633e-10, 4.794e-10, 4.95e-10, 5.104e-10,
  5.259e-10, 5.062e-10, 4.742e-10, 4.443e-10, 4.051e-10, 3.659e-10,
  3.305e-10, 2.911e-10, 2.54e-10, 2.215e-10, 1.927e-10, 1.675e-10,
  1.452e-10, 1.259e-10, 1.09e-10, 9.416e-11, 8.119e-11, 6.991e-11,
  6.015e-11, 5.163e-11, 4.43e-11, 3.789e-11, 3.24e-11, 2.769e-11,
  2.361e-11, 2.011e-11, 1.71e-11, 1.453e-11, 1.233e-11, 1.045e-11,
  8.851e-12, 7.48e-12, 6.316e-12, 5.326e-12, 4.487e-12, 3.778e-12,
  3.176e-12, 2.665e-12, 2.234e-12, 1.87e-12, 1.563e-12, 1.304e-12,
  1.085e-12, 9.007e-13, 7.468e-13, 6.179e-13, 5.092e-13, 4.188e-13,
  3.442e-13, 2.816e-13, 2.304e-13, 1.885e-13, 1.542e-13, 1.263e-13,
  1.035e-13, 8.5e-14, 7.004e-14, 5.783e-14, 4.795e-14, 4.007e-14,
  3.345e-14, 2.792e-14, 2.33e-14, 1.978e-14, 1.686e-14, 1.438e-14,
  1.234e-14, 1.07e-14, 9.312e-15, 8.131e-15, 7.164e-15, 6.367e-15,
  5.67e-15, 5.088e-15, 4.565e-15, 4.138e-15, 3.769e-15, 3.432e-15,
  3.148e-15
};

static double clono2[121] = {
  1.011e-13, 1.515e-13, 2.272e-13, 3.446e-13, 5.231e-13, 8.085e-13,
  1.253e-12, 1.979e-12, 3.149e-12, 5.092e-12, 8.312e-12, 1.366e-11,
  2.272e-11, 3.791e-11, 6.209e-11, 9.101e-11, 1.334e-10, 1.951e-10,
  2.853e-10, 3.94e-10, 4.771e-10, 5.771e-10, 6.675e-10, 7.665e-10,
  8.504e-10, 8.924e-10, 9.363e-10, 8.923e-10, 8.411e-10, 7.646e-10,
  6.525e-10, 5.576e-10, 4.398e-10, 3.403e-10, 2.612e-10, 1.915e-10,
  1.407e-10, 1.028e-10, 7.455e-11, 5.42e-11, 3.708e-11, 2.438e-11,
  1.618e-11, 1.075e-11, 7.17e-12, 4.784e-12, 3.205e-12, 2.147e-12,
  1.44e-12, 9.654e-13, 6.469e-13, 4.332e-13, 2.891e-13, 1.926e-13,
  1.274e-13, 8.422e-14, 5.547e-14, 3.636e-14, 2.368e-14, 1.536e-14,
  9.937e-15, 6.39e-15, 4.101e-15, 2.61e-15, 1.659e-15, 1.052e-15,
```

```
  6.638e-16, 4.172e-16, 2.61e-16, 1.63e-16, 1.013e-16, 6.275e-17,
  3.879e-17, 2.383e-17, 1.461e-17, 8.918e-18, 5.43e-18, 3.301e-18,
  1.997e-18, 1.203e-18, 7.216e-19, 4.311e-19, 2.564e-19, 1.519e-19,
  8.911e-20, 5.203e-20, 3.026e-20, 1.748e-20, 9.99e-21, 5.673e-21,
  3.215e-21, 1.799e-21, 1.006e-21, 5.628e-22, 3.146e-22, 1.766e-22,
  9.94e-23, 5.614e-23, 3.206e-23, 1.841e-23, 1.071e-23, 6.366e-24,
  3.776e-24, 2.238e-24, 1.326e-24, 8.253e-25, 5.201e-25, 3.279e-25,
  2.108e-25, 1.395e-25, 9.326e-26, 6.299e-26, 4.365e-26, 3.104e-26,
  2.219e-26, 1.621e-26, 1.185e-26, 8.92e-27, 6.804e-27, 5.191e-27,
  4.041e-27
};

static double co[121] = {
  1.907e-07, 1.553e-07, 1.362e-07, 1.216e-07, 1.114e-07, 1.036e-07,
  9.737e-08, 9.152e-08, 8.559e-08, 7.966e-08, 7.277e-08, 6.615e-08,
  5.884e-08, 5.22e-08, 4.699e-08, 4.284e-08, 3.776e-08, 3.274e-08,
  2.845e-08, 2.479e-08, 2.246e-08, 2.054e-08, 1.991e-08, 1.951e-08,
  1.94e-08, 2.009e-08, 2.1e-08, 2.201e-08, 2.322e-08, 2.45e-08,
  2.602e-08, 2.73e-08, 2.867e-08, 2.998e-08, 3.135e-08, 3.255e-08,
  3.352e-08, 3.426e-08, 3.484e-08, 3.53e-08, 3.593e-08, 3.671e-08,
  3.759e-08, 3.945e-08, 4.192e-08, 4.49e-08, 5.03e-08, 5.703e-08,
  6.538e-08, 7.878e-08, 9.644e-08, 1.196e-07, 1.498e-07, 1.904e-07,
  2.422e-07, 3.055e-07, 3.804e-07, 4.747e-07, 5.899e-07, 7.272e-07,
  8.91e-07, 1.071e-06, 1.296e-06, 1.546e-06, 1.823e-06, 2.135e-06,
  2.44e-06, 2.714e-06, 2.967e-06, 3.189e-06, 3.391e-06, 3.58e-06,
  3.773e-06, 4.022e-06, 4.346e-06, 4.749e-06, 5.199e-06, 5.668e-06,
  6.157e-06, 6.688e-06, 7.254e-06, 7.867e-06, 8.539e-06, 9.26e-06,
  1.009e-05, 1.119e-05, 1.228e-05, 1.365e-05, 1.506e-05, 1.641e-05,
  1.784e-05, 1.952e-05, 2.132e-05, 2.323e-05, 2.531e-05, 2.754e-05,
  3.047e-05, 3.459e-05, 3.922e-05, 4.439e-05, 4.825e-05, 5.077e-05,
  5.34e-05, 5.618e-05, 5.909e-05, 6.207e-05, 6.519e-05, 6.845e-05,
  6.819e-05, 6.726e-05, 6.622e-05, 6.512e-05, 6.671e-05, 6.862e-05,
  7.048e-05, 7.264e-05, 7.3e-05, 7.3e-05, 7.3e-05, 7.3e-05, 7.3e-05
};

static double cof2[121] = {
  7.5e-14, 1.055e-13, 1.485e-13, 2.111e-13, 3.001e-13, 4.333e-13,
  6.269e-13, 9.221e-13, 1.364e-12, 2.046e-12, 3.093e-12, 4.703e-12,
  7.225e-12, 1.113e-11, 1.66e-11, 2.088e-11, 2.626e-11, 3.433e-11,
  4.549e-11, 5.886e-11, 7.21e-11, 8.824e-11, 1.015e-10, 1.155e-10,
  1.288e-10, 1.388e-10, 1.497e-10, 1.554e-10, 1.606e-10, 1.639e-10,
  1.64e-10, 1.64e-10, 1.596e-10, 1.542e-10, 1.482e-10, 1.382e-10,
  1.289e-10, 1.198e-10, 1.109e-10, 1.026e-10, 9.484e-11, 8.75e-11,
  8.086e-11, 7.49e-11, 6.948e-11, 6.446e-11, 5.961e-11, 5.505e-11,
  5.085e-11, 4.586e-11, 4.1e-11, 3.665e-11, 3.235e-11, 2.842e-11,
  2.491e-11, 2.11e-11, 1.769e-11, 1.479e-11, 1.197e-11, 9.631e-12,
  7.74e-12, 6.201e-12, 4.963e-12, 3.956e-12, 3.151e-12, 2.507e-12,
  1.99e-12, 1.576e-12, 1.245e-12, 9.83e-13, 7.742e-13, 6.088e-13,
  4.782e-13, 3.745e-13, 2.929e-13, 2.286e-13, 1.782e-13, 1.388e-13,
  1.079e-13, 8.362e-14, 6.471e-14, 4.996e-14, 3.85e-14, 2.96e-14,
  2.265e-14, 1.729e-14, 1.317e-14, 9.998e-15, 7.549e-15, 5.683e-15,
  4.273e-15, 3.193e-15, 2.385e-15, 1.782e-15, 1.331e-15, 9.957e-16,
  7.461e-16, 5.601e-16, 4.228e-16, 3.201e-16, 2.438e-16, 1.878e-16,
  1.445e-16, 1.111e-16, 8.544e-17, 6.734e-17, 5.341e-17, 4.237e-17,
  3.394e-17, 2.759e-17, 2.254e-17, 1.851e-17, 1.54e-17, 1.297e-17,
  1.096e-17, 9.365e-18, 8e-18, 6.938e-18, 6.056e-18, 5.287e-18,
  4.662e-18
};

static double f11[121] = {
  2.65e-10, 2.65e-10, 2.65e-10, 2.65e-10, 2.65e-10, 2.65e-10, 2.65e-10,
  2.65e-10, 2.65e-10, 2.65e-10, 2.65e-10, 2.65e-10, 2.635e-10, 2.536e-10,
  2.44e-10, 2.348e-10, 2.258e-10, 2.153e-10, 2.046e-10, 1.929e-10,
  1.782e-10, 1.648e-10, 1.463e-10, 1.291e-10, 1.1e-10, 8.874e-11,
  7.165e-11, 5.201e-11, 3.744e-11, 2.577e-11, 1.64e-11, 1.048e-11,
  5.993e-12, 3.345e-12, 1.839e-12, 9.264e-13, 4.688e-13, 2.329e-13,
  1.129e-13, 5.505e-14, 2.825e-14, 1.492e-14, 7.997e-15, 5.384e-15,
  3.988e-15, 2.955e-15, 2.196e-15, 1.632e-15, 1.214e-15, 9.025e-16,
  6.708e-16, 4.984e-16, 3.693e-16, 2.733e-16, 2.013e-16, 1.481e-16,
  1.087e-16, 7.945e-17, 5.782e-17, 4.195e-17, 3.038e-17, 2.19e-17,
  1.577e-17, 1.128e-17, 8.063e-18, 5.753e-18, 4.09e-18, 2.899e-18,
  2.048e-18, 1.444e-18, 1.015e-18, 7.12e-19, 4.985e-19, 3.474e-19,
  2.417e-19, 1.677e-19, 1.161e-19, 8.029e-20, 5.533e-20, 3.799e-20,
  2.602e-20, 1.776e-20, 1.209e-20, 8.202e-21, 5.522e-21, 3.707e-21,
  2.48e-21, 1.652e-21, 1.091e-21, 7.174e-22, 4.709e-22, 3.063e-22,
  1.991e-22, 1.294e-22, 8.412e-23, 5.483e-23, 3.581e-23, 2.345e-23,
  1.548e-23, 1.027e-23, 6.869e-24, 4.673e-24, 3.173e-24, 2.153e-24,
  1.461e-24, 1.028e-24, 7.302e-25, 5.188e-25, 3.739e-25, 2.753e-25,
  2.043e-25, 1.528e-25, 1.164e-25, 9.041e-26, 7.051e-26, 5.587e-26,
  4.428e-26, 3.588e-26, 2.936e-26, 2.402e-26, 1.995e-26
};

static double f12[121] = {
  5.45e-10, 5.45e-10, 5.45e-10, 5.45e-10, 5.45e-10, 5.45e-10, 5.45e-10,
  5.45e-10, 5.45e-10, 5.45e-10, 5.45e-10, 5.45e-10, 5.429e-10, 5.291e-10,
  5.155e-10, 5.022e-10, 4.893e-10, 4.772e-10, 4.655e-10, 4.497e-10,
  4.249e-10, 4.015e-10, 3.632e-10, 3.261e-10, 2.858e-10, 2.408e-10,
```

```
    2.03e-10, 1.685e-10, 1.4e-10, 1.163e-10, 9.65e-11, 8.02e-11, 6.705e-11,
    5.624e-11, 4.764e-11, 4.249e-11, 3.792e-11, 3.315e-11, 2.819e-11,
    2.4e-11, 1.999e-11, 1.64e-11, 1.352e-11, 1.14e-11, 9.714e-12,
    8.28e-12, 7.176e-12, 6.251e-12, 5.446e-12, 4.72e-12, 4.081e-12,
    3.528e-12, 3.08e-12, 2.699e-12, 2.359e-12, 2.111e-12, 1.901e-12,
    1.709e-12, 1.534e-12, 1.376e-12, 1.233e-12, 1.103e-12, 9.869e-13,
    8.808e-13, 7.859e-13, 7.008e-13, 6.241e-13, 5.553e-13, 4.935e-13,
    4.383e-13, 3.889e-13, 3.447e-13, 3.054e-13, 2.702e-13, 2.389e-13,
    2.11e-13, 1.862e-13, 1.643e-13, 1.448e-13, 1.274e-13, 1.121e-13,
    9.844e-14, 8.638e-14, 7.572e-14, 6.62e-14, 5.782e-14, 5.045e-14,
    4.394e-14, 3.817e-14, 3.311e-14, 2.87e-14, 2.48e-14, 2.142e-14,
    1.851e-14, 1.599e-14, 1.383e-14, 1.196e-14, 1.036e-14, 9e-15,
    7.828e-15, 6.829e-15, 5.992e-15, 5.254e-15, 4.606e-15, 4.037e-15,
    3.583e-15, 3.19e-15, 2.841e-15, 2.542e-15, 2.291e-15, 2.07e-15,
    1.875e-15, 1.71e-15, 1.57e-15, 1.442e-15, 1.333e-15, 1.232e-15,
    1.147e-15, 1.071e-15, 1.001e-15, 9.396e-16
  };

  static double f14[121] = {
    9e-11, 9e-11, 9e-11, 9e-11, 9e-11, 9e-11, 9e-11, 9e-11, 9e-11, 9e-11,
    9e-11, 9e-11, 9e-11, 9e-11, 9e-11, 8.91e-11, 8.73e-11, 8.46e-11,
    8.19e-11, 7.92e-11, 7.74e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11,
    7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11,
    7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11,
    7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11,
    7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11,
    7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11,
    7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11,
    7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11,
    7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11,
    7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11,
    7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11,
    7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11,
    7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11,
    7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11,
    7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11
  };

  static double f22[121] = {
    1.4e-10, 1.4e-10, 1.4e-10, 1.4e-10, 1.4e-10, 1.4e-10, 1.4e-10,
    1.4e-10, 1.4e-10, 1.4e-10, 1.372e-10, 1.317e-10, 1.235e-10, 1.153e-10,
    1.075e-10, 1.002e-10, 9.332e-11, 8.738e-11, 8.194e-11, 7.7e-11,
    7.165e-11, 6.753e-11, 6.341e-11, 5.971e-11, 5.6e-11, 5.229e-11,
    4.859e-11, 4.488e-11, 4.118e-11, 3.83e-11, 3.568e-11, 3.308e-11,
    3.047e-11, 2.82e-11, 2.594e-11, 2.409e-11, 2.237e-11, 2.065e-11,
    1.894e-11, 1.771e-11, 1.647e-11, 1.532e-11, 1.416e-11, 1.332e-11,
    1.246e-11, 1.161e-11, 1.087e-11, 1.017e-11, 9.471e-12, 8.853e-12,
    8.235e-12, 7.741e-12, 7.247e-12, 6.836e-12, 6.506e-12, 6.176e-12,
    5.913e-12, 5.65e-12, 5.419e-12, 5.221e-12, 5.024e-12, 4.859e-12,
    4.694e-12, 4.546e-12, 4.414e-12, 4.282e-12, 4.15e-12, 4.019e-12,
    3.903e-12, 3.805e-12, 3.706e-12, 3.607e-12, 3.508e-12, 3.41e-12,
    3.31e-12, 3.212e-12, 3.129e-12, 3.047e-12, 2.964e-12, 2.882e-12,
    2.8e-12, 2.734e-12, 2.668e-12, 2.602e-12, 2.537e-12, 2.471e-12,
    2.421e-12, 2.372e-12, 2.322e-12, 2.273e-12, 2.224e-12, 2.182e-12,
    2.141e-12, 2.1e-12, 2.059e-12, 2.018e-12, 1.977e-12, 1.935e-12,
    1.894e-12, 1.853e-12, 1.812e-12, 1.77e-12, 1.73e-12, 1.688e-12,
    1.647e-12, 1.606e-12, 1.565e-12, 1.524e-12, 1.483e-12, 1.441e-12,
    1.4e-12, 1.359e-12, 1.317e-12, 1.276e-12, 1.235e-12, 1.194e-12,
    1.153e-12, 1.112e-12, 1.071e-12, 1.029e-12, 9.883e-13
  };

  static double h2o[121] = {
    0.01166, 0.008269, 0.005742, 0.003845, 0.00277, 0.001897, 0.001272,
    0.000827, 0.000539, 0.0003469, 0.0001579, 3.134e-05, 1.341e-05,
    6.764e-06, 4.498e-06, 3.703e-06, 3.724e-06, 3.899e-06, 4.002e-06,
    4.122e-06, 4.277e-06, 4.438e-06, 4.558e-06, 4.673e-06, 4.763e-06,
    4.809e-06, 4.856e-06, 4.936e-06, 5.021e-06, 5.114e-06, 5.222e-06,
    5.331e-06, 5.414e-06, 5.488e-06, 5.563e-06, 5.633e-06, 5.704e-06,
    5.767e-06, 5.819e-06, 5.872e-06, 5.914e-06, 5.949e-06, 5.984e-06,
    6.015e-06, 6.044e-06, 6.073e-06, 6.104e-06, 6.136e-06, 6.167e-06,
    6.189e-06, 6.208e-06, 6.226e-06, 6.212e-06, 6.185e-06, 6.158e-06,
    6.114e-06, 6.066e-06, 6.018e-06, 5.877e-06, 5.728e-06, 5.582e-06,
    5.437e-06, 5.296e-06, 5.156e-06, 5.02e-06, 4.886e-06, 4.754e-06,
    4.625e-06, 4.498e-06, 4.374e-06, 4.242e-06, 4.096e-06, 3.955e-06,
    3.817e-06, 3.683e-06, 3.491e-06, 3.204e-06, 2.94e-06, 2.696e-06,
    2.47e-06, 2.252e-06, 2.019e-06, 1.808e-06, 1.618e-06, 1.445e-06,
    1.285e-06, 1.105e-06, 9.489e-07, 8.121e-07, 6.938e-07, 5.924e-07,
    5.04e-07, 4.288e-07, 3.648e-07, 3.103e-07, 2.642e-07, 2.252e-07,
    1.921e-07, 1.643e-07, 1.408e-07, 1.211e-07, 1.048e-07, 9.063e-08,
    7.835e-08, 6.774e-08, 5.936e-08, 5.221e-08, 4.592e-08, 4.061e-08,
    3.62e-08, 3.236e-08, 2.902e-08, 2.62e-08, 2.383e-08, 2.171e-08,
    1.989e-08, 1.823e-08, 1.684e-08, 1.562e-08, 1.449e-08, 1.351e-08
  };

  static double h2o2[121] = {
    1.779e-10, 7.938e-10, 8.953e-10, 8.032e-10, 6.564e-10, 5.159e-10,
    4.003e-10, 3.026e-10, 2.222e-10, 1.58e-10, 1.044e-10, 6.605e-11,
```

```
  3.413e-11, 1.453e-11, 1.062e-11, 1.009e-11, 9.597e-12, 1.175e-11,
  1.572e-11, 2.091e-11, 2.746e-11, 3.603e-11, 4.791e-11, 6.387e-11,
  8.239e-11, 1.007e-10, 1.23e-10, 1.363e-10, 1.489e-10, 1.585e-10,
  1.608e-10, 1.632e-10, 1.576e-10, 1.502e-10, 1.423e-10, 1.302e-10,
  1.192e-10, 1.085e-10, 9.795e-11, 8.854e-11, 8.057e-11, 7.36e-11,
  6.736e-11, 6.362e-11, 6.087e-11, 5.825e-11, 5.623e-11, 5.443e-11,
  5.27e-11, 5.098e-11, 4.931e-11, 4.769e-11, 4.611e-11, 4.458e-11,
  4.308e-11, 4.102e-11, 3.887e-11, 3.682e-11, 3.521e-11, 3.369e-11,
  3.224e-11, 3.082e-11, 2.946e-11, 2.814e-11, 2.687e-11, 2.566e-11,
  2.449e-11, 2.336e-11, 2.227e-11, 2.123e-11, 2.023e-11, 1.927e-11,
  1.835e-11, 1.746e-11, 1.661e-11, 1.58e-11, 1.502e-11, 1.428e-11,
  1.357e-11, 1.289e-11, 1.224e-11, 1.161e-11, 1.102e-11, 1.045e-11,
  9.895e-12, 9.369e-12, 8.866e-12, 8.386e-12, 7.922e-12, 7.479e-12,
  7.06e-12, 6.656e-12, 6.274e-12, 5.914e-12, 5.575e-12, 5.257e-12,
  4.959e-12, 4.679e-12, 4.42e-12, 4.178e-12, 3.954e-12, 3.75e-12,
  3.557e-12, 3.372e-12, 3.198e-12, 3.047e-12, 2.908e-12, 2.775e-12,
  2.653e-12, 2.544e-12, 2.442e-12, 2.346e-12, 2.26e-12, 2.183e-12,
  2.11e-12, 2.044e-12, 1.98e-12, 1.924e-12, 1.871e-12, 1.821e-12,
  1.775e-12
};

static double hcn[121] = {
  5.5e-10, 5.5e-10, 5.5e-10, 5.5e-10, 5.5e-10, 5.5e-10, 5.5e-10,
  5.5e-10, 5.5e-10, 5.5e-10, 5.5e-10, 5.498e-10, 5.495e-10, 5.493e-10,
  5.49e-10, 5.488e-10, 4.717e-10, 3.946e-10, 3.174e-10, 2.4e-10,
  1.626e-10, 1.619e-10, 1.612e-10, 1.602e-10, 1.593e-10, 1.582e-10,
  1.572e-10, 1.56e-10, 1.549e-10, 1.539e-10, 1.53e-10, 1.519e-10,
  1.506e-10, 1.487e-10, 1.467e-10, 1.449e-10, 1.43e-10, 1.413e-10,
  1.397e-10, 1.382e-10, 1.368e-10, 1.354e-10, 1.337e-10, 1.315e-10,
  1.292e-10, 1.267e-10, 1.241e-10, 1.215e-10, 1.19e-10, 1.165e-10,
  1.141e-10, 1.118e-10, 1.096e-10, 1.072e-10, 1.047e-10, 1.021e-10,
  9.968e-11, 9.739e-11, 9.539e-11, 9.339e-11, 9.135e-11, 8.898e-11,
  8.664e-11, 8.439e-11, 8.249e-11, 8.075e-11, 7.904e-11, 7.735e-11,
  7.565e-11, 7.399e-11, 7.245e-11, 7.109e-11, 6.982e-11, 6.863e-11,
  6.755e-11, 6.657e-11, 6.587e-11, 6.527e-11, 6.476e-11, 6.428e-11,
  6.382e-11, 6.343e-11, 6.307e-11, 6.272e-11, 6.238e-11, 6.205e-11,
  6.17e-11, 6.137e-11, 6.102e-11, 6.072e-11, 6.046e-11, 6.03e-11,
  6.018e-11, 6.01e-11, 6.001e-11, 5.992e-11, 5.984e-11, 5.975e-11,
  5.967e-11, 5.958e-11, 5.95e-11, 5.941e-11, 5.933e-11, 5.925e-11,
  5.916e-11, 5.908e-11, 5.899e-11, 5.891e-11, 5.883e-11, 5.874e-11,
  5.866e-11, 5.858e-11, 5.85e-11, 5.841e-11, 5.833e-11, 5.825e-11,
  5.817e-11, 5.808e-11, 5.8e-11, 5.792e-11, 5.784e-11
};

static double hno3[121] = {
  1.809e-10, 7.234e-10, 5.899e-10, 4.342e-10, 3.277e-10, 2.661e-10,
  2.35e-10, 2.267e-10, 2.389e-10, 2.651e-10, 3.255e-10, 4.099e-10,
  5.42e-10, 6.978e-10, 8.807e-10, 1.112e-09, 1.405e-09, 2.04e-09,
  3.111e-09, 4.5e-09, 5.762e-09, 7.37e-09, 7.852e-09, 8.109e-09,
  8.067e-09, 7.554e-09, 7.076e-09, 6.268e-09, 5.524e-09, 4.749e-09,
  3.909e-09, 3.223e-09, 2.517e-09, 1.942e-09, 1.493e-09, 1.122e-09,
  8.449e-10, 6.361e-10, 4.787e-10, 3.611e-10, 2.804e-10, 2.215e-10,
  1.758e-10, 1.441e-10, 1.197e-10, 9.953e-11, 8.505e-11, 7.334e-11,
  6.325e-11, 5.625e-11, 5.058e-11, 4.548e-11, 4.122e-11, 3.748e-11,
  3.402e-11, 3.088e-11, 2.8e-11, 2.536e-11, 2.293e-11, 2.072e-11,
  1.871e-11, 1.687e-11, 1.52e-11, 1.368e-11, 1.23e-11, 1.105e-11,
  9.922e-12, 8.898e-12, 7.972e-12, 7.139e-12, 6.385e-12, 5.708e-12,
  5.099e-12, 4.549e-12, 4.056e-12, 3.613e-12, 3.216e-12, 2.862e-12,
  2.544e-12, 2.259e-12, 2.004e-12, 1.776e-12, 1.572e-12, 1.391e-12,
  1.227e-12, 1.082e-12, 9.528e-13, 8.379e-13, 7.349e-13, 6.436e-13,
  5.634e-13, 4.917e-13, 4.291e-13, 3.745e-13, 3.267e-13, 2.854e-13,
  2.494e-13, 2.181e-13, 1.913e-13, 1.68e-13, 1.479e-13, 1.31e-13,
  1.159e-13, 1.025e-13, 9.067e-14, 8.113e-14, 7.281e-14, 6.535e-14,
  5.892e-14, 5.348e-14, 4.867e-14, 4.439e-14, 4.073e-14, 3.76e-14,
  3.476e-14, 3.229e-14, 3e-14, 2.807e-14, 2.635e-14, 2.473e-14,
  2.332e-14
};

static double hno4[121] = {
  6.118e-12, 3.594e-12, 2.807e-12, 3.04e-12, 4.458e-12, 7.986e-12,
  1.509e-11, 2.661e-11, 3.738e-11, 4.652e-11, 4.429e-11, 3.992e-11,
  3.347e-11, 3.005e-11, 3.173e-11, 4.055e-11, 5.812e-11, 8.489e-11,
  1.19e-10, 1.482e-10, 1.766e-10, 2.103e-10, 2.35e-10, 2.598e-10,
  2.801e-10, 2.899e-10, 3e-10, 2.817e-10, 2.617e-10, 2.332e-10,
  1.933e-10, 1.605e-10, 1.232e-10, 9.285e-11, 6.941e-11, 4.951e-11,
  3.539e-11, 2.402e-11, 1.522e-11, 9.676e-12, 6.056e-12, 3.745e-12,
  2.34e-12, 1.463e-12, 9.186e-13, 5.769e-13, 3.322e-13, 1.853e-13,
  1.035e-13, 7.173e-14, 5.382e-14, 4.036e-14, 3.401e-14, 2.997e-14,
  2.635e-14, 2.316e-14, 2.034e-14, 1.783e-14, 1.56e-14, 1.363e-14,
  1.19e-14, 1.037e-14, 9.032e-15, 7.846e-15, 6.813e-15, 5.912e-15,
  5.121e-15, 4.431e-15, 3.829e-15, 3.306e-15, 2.851e-15, 2.456e-15,
  2.114e-15, 1.816e-15, 1.559e-15, 1.337e-15, 1.146e-15, 9.811e-16,
  8.389e-16, 7.162e-16, 6.109e-16, 5.203e-16, 4.425e-16, 3.76e-16,
  3.184e-16, 2.692e-16, 2.274e-16, 1.917e-16, 1.61e-16, 1.35e-16,
  1.131e-16, 9.437e-17, 7.874e-17, 6.57e-17, 5.481e-17, 4.579e-17,
  3.828e-17, 3.204e-17, 2.691e-17, 2.264e-17, 1.912e-17, 1.626e-17,
  1.382e-17, 1.174e-17, 9.972e-18, 8.603e-18, 7.45e-18, 6.453e-18,
```

```
  5.623e-18, 4.944e-18, 4.361e-18, 3.859e-18, 3.443e-18, 3.096e-18,
  2.788e-18, 2.528e-18, 2.293e-18, 2.099e-18, 1.929e-18, 1.773e-18,
  1.64e-18
};

static double hocl[121] = {
  1.056e-12, 1.194e-12, 1.35e-12, 1.531e-12, 1.737e-12, 1.982e-12,
  2.263e-12, 2.599e-12, 2.991e-12, 3.459e-12, 4.012e-12, 4.662e-12,
  5.438e-12, 6.35e-12, 7.425e-12, 8.686e-12, 1.016e-11, 1.188e-11,
  1.389e-11, 1.659e-11, 2.087e-11, 2.621e-11, 3.265e-11, 4.064e-11,
  4.859e-11, 5.441e-11, 6.09e-11, 6.373e-11, 6.611e-11, 6.94e-11,
  7.44e-11, 7.97e-11, 8.775e-11, 9.722e-11, 1.064e-10, 1.089e-10,
  1.114e-10, 1.106e-10, 1.053e-10, 1.004e-10, 9.006e-11, 7.778e-11,
  6.739e-11, 5.636e-11, 4.655e-11, 3.845e-11, 3.042e-11, 2.368e-11,
  1.845e-11, 1.442e-11, 1.127e-11, 8.814e-12, 6.544e-12, 4.763e-12,
  3.449e-12, 2.612e-12, 1.999e-12, 1.526e-12, 1.16e-12, 8.793e-13,
  6.655e-13, 5.017e-13, 3.778e-13, 2.829e-13, 2.117e-13, 1.582e-13,
  1.178e-13, 8.755e-14, 6.486e-14, 4.799e-14, 3.54e-14, 2.606e-14,
  1.916e-14, 1.403e-14, 1.026e-14, 7.48e-15, 5.446e-15, 3.961e-15,
  2.872e-15, 2.076e-15, 1.498e-15, 1.077e-15, 7.726e-16, 5.528e-16,
  3.929e-16, 2.785e-16, 1.969e-16, 1.386e-16, 9.69e-17, 6.747e-17,
  4.692e-17, 3.236e-17, 2.232e-17, 1.539e-17, 1.061e-17, 7.332e-18,
  5.076e-18, 3.522e-18, 2.461e-18, 1.726e-18, 1.22e-18, 8.75e-19,
  6.264e-19, 4.482e-19, 3.207e-19, 2.368e-19, 1.762e-19, 1.312e-19,
  9.891e-20, 7.595e-20, 5.87e-20, 4.567e-20, 3.612e-20, 2.904e-20,
  2.343e-20, 1.917e-20, 1.568e-20, 1.308e-20, 1.1e-20, 9.25e-21,
  7.881e-21
};

static double n2o[121] = {
  3.17e-07, 3.17e-07, 3.17e-07, 3.17e-07, 3.17e-07, 3.17e-07, 3.17e-07,
  3.17e-07, 3.17e-07, 3.17e-07, 3.124e-07, 3.077e-07, 3.03e-07,
  2.984e-07, 2.938e-07, 2.892e-07, 2.847e-07, 2.779e-07, 2.705e-07,
  2.631e-07, 2.557e-07, 2.484e-07, 2.345e-07, 2.201e-07, 2.01e-07,
  1.754e-07, 1.532e-07, 1.329e-07, 1.154e-07, 1.003e-07, 8.735e-08,
  7.617e-08, 6.512e-08, 5.547e-08, 4.709e-08, 3.915e-08, 3.259e-08,
  2.738e-08, 2.327e-08, 1.98e-08, 1.711e-08, 1.493e-08, 1.306e-08,
  1.165e-08, 1.049e-08, 9.439e-09, 8.375e-09, 7.391e-09, 6.525e-09,
  5.759e-09, 5.083e-09, 4.485e-09, 3.953e-09, 3.601e-09, 3.27e-09,
  2.975e-09, 2.757e-09, 2.556e-09, 2.37e-09, 2.195e-09, 2.032e-09,
  1.912e-09, 1.79e-09, 1.679e-09, 1.572e-09, 1.482e-09, 1.402e-09,
  1.326e-09, 1.254e-09, 1.187e-09, 1.127e-09, 1.071e-09, 1.02e-09,
  9.673e-10, 9.193e-10, 8.752e-10, 8.379e-10, 8.017e-10, 7.66e-10,
  7.319e-10, 7.004e-10, 6.721e-10, 6.459e-10, 6.199e-10, 5.942e-10,
  5.703e-10, 5.488e-10, 5.283e-10, 5.082e-10, 4.877e-10, 4.696e-10,
  4.52e-10, 4.355e-10, 4.198e-10, 4.039e-10, 3.888e-10, 3.754e-10,
  3.624e-10, 3.499e-10, 3.381e-10, 3.267e-10, 3.163e-10, 3.058e-10,
  2.959e-10, 2.864e-10, 2.77e-10, 2.686e-10, 2.604e-10, 2.534e-10,
  2.462e-10, 2.386e-10, 2.318e-10, 2.247e-10, 2.189e-10, 2.133e-10,
  2.071e-10, 2.014e-10, 1.955e-10, 1.908e-10, 1.86e-10, 1.817e-10
};

static double n2o5[121] = {
  1.231e-11, 3.035e-12, 1.702e-12, 9.877e-13, 8.081e-13, 9.039e-13,
  1.169e-12, 1.474e-12, 1.651e-12, 1.795e-12, 1.998e-12, 2.543e-12,
  4.398e-12, 7.698e-12, 1.28e-11, 2.131e-11, 3.548e-11, 5.894e-11,
  7.645e-11, 1.089e-10, 1.391e-10, 1.886e-10, 2.386e-10, 2.986e-10,
  3.487e-10, 3.994e-10, 4.5e-10, 4.6e-10, 4.591e-10, 4.1e-10, 3.488e-10,
  2.846e-10, 2.287e-10, 1.696e-10, 1.011e-10, 6.428e-11, 4.324e-11,
  2.225e-11, 6.214e-12, 3.608e-12, 8.793e-13, 4.491e-13, 1.04e-13,
  6.1e-14, 3.436e-14, 6.671e-15, 1.171e-15, 5.848e-16, 1.212e-16,
  1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16,
  1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16,
  1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16,
  1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16,
  1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16,
  1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16,
  1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16,
  1e-16, 1e-16
};

static double nh3[121] = {
  1e-10, 1e-10, 1e-10, 1e-10, 1e-10, 1e-10, 1e-10, 1e-10, 1e-10, 1e-10,
  1e-10, 1e-10, 1e-10, 1e-10, 9.444e-11, 8.488e-11, 7.241e-11, 5.785e-11,
  4.178e-11, 3.018e-11, 2.18e-11, 1.574e-11, 1.137e-11, 8.211e-12,
  5.973e-12, 4.327e-12, 3.118e-12, 2.234e-12, 1.573e-12, 1.04e-12,
  6.762e-13, 4.202e-13, 2.406e-13, 1.335e-13, 6.938e-14, 3.105e-14,
  1.609e-14, 1.033e-14, 6.432e-15, 4.031e-15, 2.555e-15, 1.656e-15,
  1.115e-15, 7.904e-16, 5.63e-16, 4.048e-16, 2.876e-16, 2.004e-16,
  1.356e-16, 9.237e-17, 6.235e-17, 4.223e-17, 3.009e-17, 2.328e-17,
  2.002e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17,
  1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17,
  1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17,
  1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17,
  1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17,
  1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17,
  1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17,
```

```
    1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17,
    1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17,
    1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17,
    1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17,
    1.914e-17
  };

  static double no[121] = {
    2.586e-10, 4.143e-11, 1.566e-11, 9.591e-12, 8.088e-12, 8.462e-12,
    1.013e-11, 1.328e-11, 1.855e-11, 2.678e-11, 3.926e-11, 5.464e-11,
    7.012e-11, 8.912e-11, 1.127e-10, 1.347e-10, 1.498e-10, 1.544e-10,
    1.602e-10, 1.824e-10, 2.078e-10, 2.366e-10, 2.691e-10, 5.141e-10,
    8.259e-10, 1.254e-09, 1.849e-09, 2.473e-09, 3.294e-09, 4.16e-09,
    5.095e-09, 6.11e-09, 6.93e-09, 7.888e-09, 8.903e-09, 9.713e-09,
    1.052e-08, 1.115e-08, 1.173e-08, 1.21e-08, 1.228e-08, 1.239e-08,
    1.231e-08, 1.213e-08, 1.192e-08, 1.138e-08, 1.085e-08, 1.008e-08,
    9.224e-09, 8.389e-09, 7.262e-09, 6.278e-09, 5.335e-09, 4.388e-09,
    3.589e-09, 2.761e-09, 2.129e-09, 1.633e-09, 1.243e-09, 9.681e-10,
    8.355e-10, 7.665e-10, 7.442e-10, 8.584e-10, 9.732e-10, 1.063e-09,
    1.163e-09, 1.286e-09, 1.472e-09, 1.707e-09, 2.032e-09, 2.474e-09,
    2.977e-09, 3.506e-09, 4.102e-09, 5.013e-09, 6.493e-09, 8.414e-09,
    1.077e-08, 1.367e-08, 1.777e-08, 2.625e-08, 3.926e-08, 5.545e-08,
    7.195e-08, 9.464e-08, 1.404e-07, 2.183e-07, 3.329e-07, 4.535e-07,
    6.158e-07, 8.187e-07, 1.075e-06, 1.422e-06, 1.979e-06, 2.71e-06,
    3.58e-06, 4.573e-06, 5.951e-06, 7.999e-06, 1.072e-05, 1.372e-05,
    1.697e-05, 2.112e-05, 2.643e-05, 3.288e-05, 3.994e-05, 4.794e-05,
    5.606e-05, 6.383e-05, 7.286e-05, 8.156e-05, 8.883e-05, 9.469e-05,
    9.848e-05, 0.0001023, 0.0001066, 0.0001115, 0.0001145, 0.0001142,
    0.0001133
  };

  static double no2[121] = {
    3.036e-09, 2.945e-10, 9.982e-11, 5.069e-11, 3.485e-11, 2.982e-11,
    2.947e-11, 3.164e-11, 3.714e-11, 4.586e-11, 6.164e-11, 8.041e-11,
    9.982e-11, 1.283e-10, 1.73e-10, 2.56e-10, 3.909e-10, 5.959e-10,
    9.081e-10, 1.384e-09, 1.788e-09, 2.189e-09, 2.686e-09, 3.091e-09,
    3.49e-09, 3.796e-09, 4.2e-09, 5.103e-09, 6.005e-09, 6.3e-09, 6.706e-09,
    7.07e-09, 7.434e-09, 7.663e-09, 7.788e-09, 7.8e-09, 7.597e-09,
    7.482e-09, 7.227e-09, 6.403e-09, 5.585e-09, 4.606e-09, 3.703e-09,
    2.984e-09, 2.183e-09, 1.48e-09, 8.441e-10, 5.994e-10, 3.799e-10,
    2.751e-10, 1.927e-10, 1.507e-10, 1.102e-10, 6.971e-11, 5.839e-11,
    3.904e-11, 3.087e-11, 2.176e-11, 1.464e-11, 1.209e-11, 8.497e-12,
    6.477e-12, 4.371e-12, 2.914e-12, 2.424e-12, 1.753e-12, 1.35e-12,
    9.417e-13, 6.622e-13, 5.148e-13, 3.841e-13, 3.446e-13, 3.01e-13,
    2.551e-13, 2.151e-13, 1.829e-13, 1.64e-13, 1.475e-13, 1.352e-13,
    1.155e-13, 9.963e-14, 9.771e-14, 9.577e-14, 9.384e-14, 9.186e-14,
    9e-14, 9e-14, 9e-14, 9e-14, 9e-14, 9e-14, 9e-14, 9e-14, 9e-14, 9e-14,
    9e-14, 9e-14, 9e-14, 9e-14, 9e-14, 9e-14, 9e-14, 9e-14, 9e-14, 9e-14,
    9e-14, 9e-14, 9e-14, 9e-14, 9e-14, 9e-14, 9e-14, 9e-14, 9e-14, 9e-14,
    9e-14, 9e-14, 9e-14, 9e-14, 9e-14, 9e-14
  };

  static double o3[121] = {
    2.218e-08, 3.394e-08, 3.869e-08, 4.219e-08, 4.501e-08, 4.778e-08,
    5.067e-08, 5.402e-08, 5.872e-08, 6.521e-08, 7.709e-08, 9.461e-08,
    1.269e-07, 1.853e-07, 2.723e-07, 3.964e-07, 5.773e-07, 8.2e-07,
    1.155e-06, 1.59e-06, 2.076e-06, 2.706e-06, 3.249e-06, 3.848e-06,
    4.459e-06, 4.986e-06, 5.573e-06, 5.958e-06, 6.328e-06, 6.661e-06,
    6.9e-06, 7.146e-06, 7.276e-06, 7.374e-06, 7.447e-06, 7.383e-06,
    7.321e-06, 7.161e-06, 6.879e-06, 6.611e-06, 6.216e-06, 5.765e-06,
    5.355e-06, 4.905e-06, 4.471e-06, 4.075e-06, 3.728e-06, 3.413e-06,
    3.125e-06, 2.856e-06, 2.607e-06, 2.379e-06, 2.17e-06, 1.978e-06,
    1.8e-06, 1.646e-06, 1.506e-06, 1.376e-06, 1.233e-06, 1.102e-06,
    9.839e-07, 8.771e-07, 7.814e-07, 6.947e-07, 6.102e-07, 5.228e-07,
    4.509e-07, 3.922e-07, 3.501e-07, 3.183e-07, 2.909e-07, 2.686e-07,
    2.476e-07, 2.284e-07, 2.109e-07, 2.003e-07, 2.013e-07, 2.022e-07,
    2.032e-07, 2.042e-07, 2.097e-07, 2.361e-07, 2.656e-07, 2.989e-07,
    3.37e-07, 3.826e-07, 4.489e-07, 5.26e-07, 6.189e-07, 7.312e-07,
    8.496e-07, 8.444e-07, 8.392e-07, 8.339e-07, 8.286e-07, 8.234e-07,
    8.181e-07, 8.129e-07, 8.077e-07, 8.026e-07, 6.918e-07, 5.176e-07,
    3.865e-07, 2.885e-07, 2.156e-07, 1.619e-07, 1.219e-07, 9.161e-08,
    6.972e-08, 5.399e-08, 3.498e-08, 2.111e-08, 1.322e-08, 8.482e-09,
    5.527e-09, 3.423e-09, 2.071e-09, 1.314e-09, 8.529e-10, 5.503e-10,
    3.665e-10
  };

  static double ocs[121] = {
    6e-10, 6e-10, 6e-10, 6e-10, 6e-10, 6e-10, 6e-10, 6e-10, 6e-10, 5.997e-10,
    5.989e-10, 5.881e-10, 5.765e-10, 5.433e-10, 5.074e-10, 4.567e-10,
    4.067e-10, 3.601e-10, 3.093e-10, 2.619e-10, 2.232e-10, 1.805e-10,
    1.46e-10, 1.187e-10, 8.03e-11, 5.435e-11, 3.686e-11, 2.217e-11,
    1.341e-11, 8.756e-12, 4.511e-12, 2.37e-12, 1.264e-12, 8.28e-13,
    5.263e-13, 3.209e-13, 1.717e-13, 9.068e-14, 4.709e-14, 2.389e-14,
    1.236e-14, 1.127e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14,
    1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14,
    1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14,
    1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14,
```

```
    1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14,
    1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14,
    1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14,
    1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14,
    1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14,
    1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14,
    1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14,
    1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14,
    1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14,
    1.091e-14, 1.091e-14, 1.091e-14
  };

  static double sf6[121] = {
    4.103e-12, 4.103e-12, 4.103e-12, 4.103e-12, 4.103e-12, 4.103e-12,
    4.103e-12, 4.103e-12, 4.103e-12, 4.087e-12, 4.064e-12, 4.023e-12,
    3.988e-12, 3.941e-12, 3.884e-12, 3.755e-12, 3.622e-12, 3.484e-12,
    3.32e-12, 3.144e-12, 2.978e-12, 2.811e-12, 2.653e-12, 2.489e-12,
    2.332e-12, 2.199e-12, 2.089e-12, 2.013e-12, 1.953e-12, 1.898e-12,
    1.859e-12, 1.826e-12, 1.798e-12, 1.776e-12, 1.757e-12, 1.742e-12,
    1.728e-12, 1.717e-12, 1.707e-12, 1.698e-12, 1.691e-12, 1.685e-12,
    1.679e-12, 1.675e-12, 1.671e-12, 1.668e-12, 1.665e-12, 1.663e-12,
    1.661e-12, 1.659e-12, 1.658e-12, 1.657e-12, 1.656e-12, 1.655e-12,
    1.654e-12, 1.653e-12, 1.653e-12, 1.652e-12, 1.652e-12, 1.652e-12,
    1.651e-12, 1.651e-12, 1.651e-12, 1.651e-12, 1.651e-12, 1.651e-12,
    1.651e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12,
    1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12,
    1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12,
    1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12,
    1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12,
    1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12,
    1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12,
    1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12
  };

  static double so2[121] = {
    1e-10, 1e-10, 1e-10, 1e-10, 1e-10, 1e-10, 1e-10, 1e-10, 1e-10, 1e-10,
    1e-10, 1e-10, 9.867e-11, 9.537e-11, 9e-11, 8.404e-11, 7.799e-11,
    7.205e-11, 6.616e-11, 6.036e-11, 5.475e-11, 5.007e-11, 4.638e-11,
    4.346e-11, 4.055e-11, 3.763e-11, 3.471e-11, 3.186e-11, 2.905e-11,
    2.631e-11, 2.358e-11, 2.415e-11, 2.949e-11, 3.952e-11, 5.155e-11,
    6.76e-11, 8.741e-11, 1.099e-10, 1.278e-10, 1.414e-10, 1.512e-10,
    1.607e-10, 1.699e-10, 1.774e-10, 1.832e-10, 1.871e-10, 1.907e-10,
    1.943e-10, 1.974e-10, 1.993e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10,
    2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10,
    2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10,
    2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10,
    2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10,
    2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10,
    2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10,
    2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10
  };

  static int ig_co2 = -999;

  double co2, *q[NG] = { NULL };

  int ig, ip, iw, iz;

  /* Find emitter index of CO2... */
  if (ig_co2 == -999)
    ig_co2 = find_emitter(ctl, "CO2");

  /* Identify variable... */
  for (ig = 0; ig < ctl->ng; ig++) {
    q[ig] = NULL;
    if (strcasecmp(ctl->emitter[ig], "C2H2") == 0)
      q[ig] = c2h2;
    if (strcasecmp(ctl->emitter[ig], "C2H6") == 0)
      q[ig] = c2h6;
    if (strcasecmp(ctl->emitter[ig], "CCl4") == 0)
      q[ig] = ccl4;
    if (strcasecmp(ctl->emitter[ig], "CH4") == 0)
      q[ig] = ch4;
    if (strcasecmp(ctl->emitter[ig], "ClO") == 0)
      q[ig] = clo;
    if (strcasecmp(ctl->emitter[ig], "ClONO2") == 0)
      q[ig] = clono2;
    if (strcasecmp(ctl->emitter[ig], "CO") == 0)
      q[ig] = co;
    if (strcasecmp(ctl->emitter[ig], "COF2") == 0)
      q[ig] = cof2;
    if (strcasecmp(ctl->emitter[ig], "F11") == 0)
      q[ig] = f11;
    if (strcasecmp(ctl->emitter[ig], "F12") == 0)
      q[ig] = f12;
    if (strcasecmp(ctl->emitter[ig], "F14") == 0)
      q[ig] = f14;
```

```
    if (strcasecmp(ctl->emitter[ig], "F22") == 0)
      q[ig] = f22;
    if (strcasecmp(ctl->emitter[ig], "H2O") == 0)
      q[ig] = h2o;
    if (strcasecmp(ctl->emitter[ig], "H2O2") == 0)
      q[ig] = h2o2;
    if (strcasecmp(ctl->emitter[ig], "HCN") == 0)
      q[ig] = hcn;
    if (strcasecmp(ctl->emitter[ig], "HNO3") == 0)
      q[ig] = hno3;
    if (strcasecmp(ctl->emitter[ig], "HNO4") == 0)
      q[ig] = hno4;
    if (strcasecmp(ctl->emitter[ig], "HOCl") == 0)
      q[ig] = hocl;
    if (strcasecmp(ctl->emitter[ig], "N2O") == 0)
      q[ig] = n2o;
    if (strcasecmp(ctl->emitter[ig], "N2O5") == 0)
      q[ig] = n2o5;
    if (strcasecmp(ctl->emitter[ig], "NH3") == 0)
      q[ig] = nh3;
    if (strcasecmp(ctl->emitter[ig], "NO") == 0)
      q[ig] = no;
    if (strcasecmp(ctl->emitter[ig], "NO2") == 0)
      q[ig] = no2;
    if (strcasecmp(ctl->emitter[ig], "O3") == 0)
      q[ig] = o3;
    if (strcasecmp(ctl->emitter[ig], "OCS") == 0)
      q[ig] = ocs;
    if (strcasecmp(ctl->emitter[ig], "SF6") == 0)
      q[ig] = sf6;
    if (strcasecmp(ctl->emitter[ig], "SO2") == 0)
      q[ig] = so2;
  }

  /* Loop over atmospheric data points... */
  for (ip = 0; ip < atm->np; ip++) {

    /* Get altitude index... */
    iz = locate(z, 121, atm->z[ip]);

    /* Interpolate pressure... */
    atm->p[ip] = EXP(z[iz], pre[iz], z[iz + 1], pre[iz + 1], atm->z[ip]);

    /* Interpolate temperature... */
    atm->t[ip] = LIN(z[iz], tem[iz], z[iz + 1], tem[iz + 1], atm->z[ip]);

    /* Interpolate trace gases... */
    for (ig = 0; ig < ctl->ng; ig++)
      if (q[ig] != NULL)
        atm->q[ig][ip] =
          LIN(z[iz], q[ig][iz], z[iz + 1], q[ig][iz + 1], atm->z[ip]);
      else
        atm->q[ig][ip] = 0;

    /* Set CO2... */
    if (ig_co2 >= 0) {
      co2 =
        371.789948e-6 + 2.026214e-6 * (atm->time[ip] - 63158400.) / 31557600.;
      atm->q[ig_co2][ip] = co2;
    }

    /* Set extinction to zero... */
    for (iw = 0; iw < ctl->nw; iw++)
      atm->k[iw][ip] = 0;
  }
}
```

Here is the call graph for this function:

5.33.2.6 double ctmco2 (double *nu*, double *p*, double *t*, double *u*)

Compute carbon dioxide continuum (optical depth).

Definition at line 874 of file jurassic.c.

```
            {

  static double co2296[2001] = { 9.3388e-5, 9.7711e-5, 1.0224e-4, 1.0697e-4,
    1.1193e-4, 1.1712e-4, 1.2255e-4, 1.2824e-4, 1.3419e-4, 1.4043e-4,
```

```
1.4695e-4, 1.5378e-4, 1.6094e-4, 1.6842e-4, 1.7626e-4, 1.8447e-4,
1.9307e-4, 2.0207e-4, 2.1149e-4, 2.2136e-4, 2.3169e-4, 2.4251e-4,
2.5384e-4, 2.657e-4, 2.7813e-4, 2.9114e-4, 3.0477e-4, 3.1904e-4,
3.3399e-4, 3.4965e-4, 3.6604e-4, 3.8322e-4, 4.0121e-4, 4.2006e-4,
4.398e-4, 4.6047e-4, 4.8214e-4, 5.0483e-4, 5.286e-4, 5.535e-4,
5.7959e-4, 6.0693e-4, 6.3557e-4, 6.6558e-4, 6.9702e-4, 7.2996e-4,
7.6449e-4, 8.0066e-4, 8.3856e-4, 8.7829e-4, 9.1991e-4, 9.6354e-4,
.0010093, .0010572, .0011074, .00116, .0012152, .001273,
.0013336, .0013972, .0014638, .0015336, .0016068, .0016835,
.001764, .0018483, .0019367, .0020295, .0021267, .0022286,
.0023355, .0024476, .0025652, .0026885, .0028178, .0029534,
.0030956, .0032448, .0034012, .0035654, .0037375, .0039181,
.0041076, .0043063, .0045148, .0047336, .0049632, .005204,
.0054567, .0057219, .0060002, .0062923, .0065988, .0069204,
.007258, .0076123, .0079842, .0083746, .0087844, .0092146,
.0096663, .01014, .010638, .011161, .01171, .012286, .012891,
.013527, .014194, .014895, .015631, .016404, .017217, .01807,
.018966, .019908, .020897, .021936, .023028, .024176, .025382,
.026649, .027981, .02938, .030851, .032397, .034023, .035732,
.037528, .039416, .041402, .04349, .045685, .047994, .050422,
.052975, .055661, .058486, .061458, .064584, .067873, .071334,
.074975, .078807, .082839, .087082, .091549, .096249, .1012,
.10641, .11189, .11767, .12375, .13015, .13689, .14399, .15147,
.15935, .16765, .17639, .18561, .19531, .20554, .21632, .22769,
.23967, .25229, .2656, .27964, .29443, .31004, .3265, .34386,
.36218, .3815, .40188, .42339, .44609, .47004, .49533, .52202,
.5502, .57995, .61137, .64455, .6796, .71663, .75574, .79707,
.84075, .88691, .9357, .98728, 1.0418, 1.0995, 1.1605, 1.225,
1.2932, 1.3654, 1.4418, 1.5227, 1.6083, 1.6989, 1.7948, 1.8964,
2.004, 2.118, 2.2388, 2.3668, 2.5025, 2.6463, 2.7988, 2.9606,
3.1321, 3.314, 3.5071, 3.712, 3.9296, 4.1605, 4.4058, 4.6663,
4.9431, 5.2374, 5.5501, 5.8818, 6.2353, 6.6114, 7.0115, 7.4372,
7.8905, 8.3731, 8.8871, 9.4349, 10.019, 10.641, 11.305, 12.013,
12.769, 13.576, 14.437, 15.358, 16.342, 17.39, 18.513, 19.716,
21.003, 22.379, 23.854, 25.436, 27.126, 28.942, 30.89, 32.973,
35.219, 37.634, 40.224, 43.021, 46.037, 49.29, 52.803, 56.447,
60.418, 64.792, 69.526, 74.637, 80.182, 86.193, 92.713, 99.786,
107.47, 115.84, 124.94, 134.86, 145.69, 157.49, 170.3, 184.39,
199.83, 216.4, 234.55, 254.72, 276.82, 299.85, 326.16, 354.99,
386.51, 416.68, 449.89, 490.12, 534.35, 578.25, 632.26, 692.61,
756.43, 834.75, 924.11, 1016.9, 996.96, 1102.7, 1219.2, 1351.9,
1494.3, 1654.1, 1826.5, 2027.9, 2249., 2453.8, 2714.4, 2999.4,
3209.5, 3509., 3840.4, 3907.5, 4190.7, 4533.5, 4648.3, 5059.1,
5561.6, 6191.4, 6820.8, 7905.9, 9362.2, 2431.3, 2211.3, 2046.8,
2023.8, 1985.9, 1905.9, 1491.1, 1369.8, 1262.2, 1200.7, 887.74,
820.25, 885.23, 887.21, 816.73, 1126.9, 1216.2, 1272.4, 1579.5,
1634.2, 1656.3, 1657.9, 1789.5, 1670.8, 1509.5, 8474.6, 7489.2,
6793.6, 6117., 5574.1, 5141.2, 5084.6, 4745.1, 4413.2, 4102.8,
4024.7, 3715., 3398.6, 3100.8, 2900.4, 2629.2, 2374., 2144.7,
1955.8, 1760.8, 1591.2, 1435.2, 1296.2, 1174., 1065.1, 967.76,
999.48, 897.45, 809.23, 732.77, 670.26, 611.93, 560.11, 518.77,
476.84, 438.8, 408.48, 380.21, 349.24, 322.71, 296.65, 272.85,
251.96, 232.04, 213.88, 197.69, 182.41, 168.41, 155.79, 144.05,
133.31, 123.48, 114.5, 106.21, 98.591, 91.612, 85.156, 79.204,
73.719, 68.666, 63.975, 59.637, 56.35, 52.545, 49.042, 45.788,
42.78, 39.992, 37.441, 35.037, 32.8, 30.744, 28.801, 26.986,
25.297, 23.731, 22.258, 20.883, 19.603, 18.403, 17.295, 16.249,
15.271, 14.356, 13.501, 12.701, 11.954, 11.254, 10.6, 9.9864,
9.4118, 8.8745, 8.3714, 7.8997, 7.4578, 7.0446, 6.6573, 6.2949,
5.9577, 5.6395, 5.3419, 5.063, 4.8037, 4.5608, 4.3452, 4.1364,
3.9413, 3.7394, 3.562, 3.3932, 3.2325, 3.0789, 2.9318, 2.7898,
2.6537, 2.5225, 2.3958, 2.2305, 2.1215, 2.0245, 1.9427, 1.8795,
1.8336, 1.7604, 1.7016, 1.6419, 1.5282, 1.4611, 1.3443, 1.27,
1.1675, 1.0824, 1.0534, .99833, .95854, .92981, .90887, .89346,
.88113, .87068, .86102, .85096, .88262, .86151, .83565, .80518,
.77045, .73736, .74744, .74954, .75773, .82267, .83493, .89402,
.89725, .93426, .95564, .94045, .94174, .93404, .92035, .90456,
.88621, .86673, .78117, .7515, .72056, .68822, .65658, .62764,
.55984, .55598, .57407, .60963, .63763, .66198, .61132, .60972,
.52496, .50649, .41872, .3964, .32422, .27276, .24048, .23772,
.2286, .22711, .23999, .32038, .34371, .36621, .38561, .39953,
.40636, .44913, .42716, .3919, .35477, .33935, .3351, .39746,
.40993, .49398, .49956, .56157, .54742, .57295, .57386, .55417,
.50745, .471, .43446, .39102, .34993, .31269, .27888, .24912,
.22291, .19994, .17972, .16197, .14633, .13252, .12029, .10942,
.099745, .091118, .083404, .076494, .070292, .064716, .059697,
.055173, .051093, .047411, .044089, .041092, .038392, .035965,
.033789, .031846, .030122, .028607, .02729, .026169, .025209,
.024405, .023766, .023288, .022925, .022716, .022681, .022685,
.022768, .023133, .023325, .023486, .024004, .024126, .024083,
.023785, .024023, .023029, .021649, .021108, .019454, .017809,
.017292, .016635, .017037, .018068, .018977, .018756, .017847,
.016557, .016142, .014459, .012869, .012381, .010875, .0098701,
.009285, .0091698, .0091701, .0096145, .010553, .01106, .012613,
.014362, .015017, .016507, .017741, .01768, .017784, .0171,
.016357, .016172, .017257, .018978, .020935, .021741, .023567,
.025183, .025589, .026732, .027648, .028278, .028215, .02856,
```

```
.029015, .029062, .028851, .028497, .027825, .027801, .026523,
.02487, .022967, .022168, .020194, .018605, .017903, .018439,
.019697, .020311, .020855, .020057, .018608, .016738, .015963,
.013844, .011801, .011134, .0097573, .0086007, .0086226,
.0083721, .0090978, .0097616, .0098426, .011317, .012853, .01447,
.014657, .015771, .016351, .016079, .014829, .013431, .013185,
.013207, .01448, .016176, .017971, .018265, .019526, .020455,
.019797, .019802, .0194, .018176, .017505, .016197, .015339,
.014401, .013213, .012203, .011186, .010236, .0093288, .0084854,
.0076837, .0069375, .0062614, .0056628, .0051153, .0046015,
.0041501, .003752, .0033996, .0030865, .0028077, .0025586,
.0023355, .0021353, .0019553, .0017931, .0016466, .0015141,
.0013941, .0012852, .0011862, .0010962, .0010142, 9.3935e-4,
8.71e-4, 8.0851e-4, 7.5132e-4, 6.9894e-4, 6.5093e-4, 6.0689e-4,
5.6647e-4, 5.2935e-4, 4.9525e-4, 4.6391e-4, 4.3509e-4, 4.086e-4,
3.8424e-4, 3.6185e-4, 3.4126e-4, 3.2235e-4, 3.0498e-4, 2.8904e-4,
2.7444e-4, 2.6106e-4, 2.4883e-4, 2.3766e-4, 2.275e-4, 2.1827e-4,
2.0992e-4, 2.0239e-4, 1.9563e-4, 1.896e-4, 1.8427e-4, 1.796e-4,
1.7555e-4, 1.7209e-4, 1.692e-4, 1.6687e-4, 1.6505e-4, 1.6375e-4,
1.6294e-4, 1.6261e-4, 1.6274e-4, 1.6334e-4, 1.6438e-4, 1.6587e-4,
1.678e-4, 1.7017e-4, 1.7297e-4, 1.762e-4, 1.7988e-4, 1.8399e-4,
1.8855e-4, 1.9355e-4, 1.9902e-4, 2.0494e-4, 2.1134e-4, 2.1823e-4,
2.2561e-4, 2.335e-4, 2.4192e-4, 2.5088e-4, 2.604e-4, 2.705e-4,
2.8119e-4, 2.9251e-4, 3.0447e-4, 3.171e-4, 3.3042e-4, 3.4447e-4,
3.5927e-4, 3.7486e-4, 3.9127e-4, 4.0854e-4, 4.267e-4, 4.4579e-4,
4.6586e-4, 4.8696e-4, 5.0912e-4, 5.324e-4, 5.5685e-4, 5.8253e-4,
6.0949e-4, 6.378e-4, 6.6753e-4, 6.9873e-4, 7.3149e-4, 7.6588e-4,
8.0198e-4, 8.3987e-4, 8.7964e-4, 9.2139e-4, 9.6522e-4, .0010112,
.0010595, .0011102, .0011634, .0012193, .001278, .0013396,
.0014043, .0014722, .0015436, .0016185, .0016972, .0017799,
.0018668, .001958, .0020539, .0021547, .0022606, .0023719,
.002489, .002612, .0027414, .0028775, .0030206, .0031712,
.0033295, .0034962, .0036716, .0038563, .0040506, .0042553,
.0044709, .004698, .0049373, .0051894, .0054552, .0057354,
.006031, .0063427, .0066717, .0070188, .0073854, .0077726,
.0081816, .0086138, .0090709, .0095543, .010066, .010607,
.011181, .011789, .012433, .013116, .013842, .014613, .015432,
.016304, .017233, .018224, .019281, .020394, .021574, .022836,
.024181, .025594, .027088, .028707, .030401, .032245, .034219,
.036262, .038539, .040987, .043578, .04641, .04949, .052726,
.056326, .0602, .064093, .068521, .073278, .077734, .083064,
.088731, .093885, .1003, .1072, .11365, .12187, .13078, .13989,
.15095, .16299, .17634, .19116, .20628, .22419, .24386, .26587,
.28811, .31399, .34321, .36606, .39675, .42742, .44243, .47197,
.49993, .49027, .51147, .52803, .48931, .49729, .5026, .43854,
.441, .44766, .43414, .46151, .50029, .55247, .43855, .32115,
.32607, .3431, .36119, .38029, .41179, .43996, .47144, .51853,
.55362, .59122, .66338, .69877, .74001, .82923, .86907, .90361,
1.0025, 1.031, 1.0559, 1.104, 1.1178, 1.1341, 1.1547, 1.351,
1.4772, 1.4812, 1.4907, 1.512, 1.5442, 1.5853, 1.6358, 1.6963,
1.7674, 1.8474, 1.9353, 2.0335, 2.143, 2.2592, 2.3853, 2.5217,
2.6686, 2.8273, 2.9998, 3.183, 3.3868, 3.6109, 3.8564, 4.1159,
4.4079, 4.7278, 5.0497, 5.3695, 5.758, 6.0834, 6.4976, 6.9312,
7.38, 7.5746, 7.9833, 8.3791, 8.3956, 8.7501, 9.1067, 9.072,
9.4649, 9.9112, 10.402, 10.829, 11.605, 12.54, 12.713, 10.443,
10.825, 11.375, 11.955, 12.623, 13.326, 14.101, 15.041, 15.547,
16.461, 17.439, 18.716, 19.84, 21.036, 22.642, 23.901, 25.244,
27.03, 28.411, 29.871, 31.403, 33.147, 34.744, 36.456, 39.239,
43.605, 45.162, 47.004, 49.093, 51.391, 53.946, 56.673, 59.629,
63.167, 66.576, 70.254, 74.222, 78.477, 83.034, 87.914, 93.18,
98.77, 104.74, 111.15, 117.95, 125.23, 133.01, 141.33, 150.21,
159.71, 169.89, 180.93, 192.54, 204.99, 218.34, 232.65, 248.,
264.47, 282.14, 301.13, 321.53, 343.48, 367.08, 392.5, 419.88,
449.4, 481.26, 515.64, 552.79, 592.99, 636.48, 683.61, 734.65,
789.99, 850.02, 915.14, 985.81, 1062.5, 1147.1, 1237.8, 1336.4,
1443.2, 1558.9, 1684.2, 1819.2, 1965.2, 2122.6, 2291.7, 2470.8,
2665.7, 2874.9, 3099.4, 3337.9, 3541., 3813.3, 4111.9, 4439.3,
4798.9, 5196., 5639.2, 6087.5, 6657.7, 7306.7, 8040.7, 8845.5,
9702.2, 10670., 11739., 12842., 14141., 15498., 17068., 18729.,
20557., 22559., 25248., 27664., 30207., 32915., 35611., 38081.,
40715., 43191., 41651., 42750., 43785., 44353., 44366., 44189.,
43618., 42862., 41878., 35133., 35215., 36383., 39420., 44055.,
44155., 45850., 46853., 39197., 38274., 29942., 28553., 21792.,
21228., 17106., 14955., 18181., 19557., 21427., 23728., 26301.,
28584., 30775., 32536., 33867., 40089., 39204., 37329., 34452.,
31373., 33921., 34800., 36043., 44415., 45162., 52181., 50895.,
54140., 50840., 50468., 48302., 44915., 40910., 36754., 32755.,
29093., 25860., 22962., 20448., 18247., 16326., 14645., 13165.,
11861., 10708., 9686.9, 8779.7, 7971.9, 7250.8, 6605.7, 6027.2,
5507.3, 5039.1, 4616.6, 4234.8, 3889., 3575.4, 3290.5, 3031.3,
2795.2, 2579.9, 2383.1, 2203.3, 2038.6, 1887.6, 1749.1, 1621.9,
1505., 1397.4, 1298.3, 1207., 1122.8, 1045., 973.1, 906.64,
845.16, 788.22, 735.48, 686.57, 641.21, 599.1, 559.99, 523.64,
489.85, 458.42, 429.16, 401.92, 376.54, 352.88, 330.82, 310.24,
291.03, 273.09, 256.34, 240.69, 226.05, 212.37, 199.57, 187.59,
176.37, 165.87, 156.03, 146.82, 138.17, 130.07, 122.47, 115.34,
108.65, 102.37, 96.473, 90.934, 85.73, 80.84, 76.243, 71.922,
```

```
67.858, 64.034, 60.438, 57.052, 53.866, 50.866, 48.04, 45.379,
42.872, 40.51, 38.285, 36.188, 34.211, 32.347, 30.588, 28.929,
27.362, 25.884, 24.489, 23.171, 21.929, 20.755, 19.646, 18.599,
17.61, 16.677, 15.795, 14.961, 14.174, 13.43, 12.725, 12.06,
11.431, 10.834, 10.27, 9.7361, 9.2302, 8.7518, 8.2997, 7.8724,
7.4674, 7.0848, 6.7226, 6.3794, 6.054, 5.745, 5.4525, 5.1752,
4.9121, 4.6625, 4.4259, 4.2015, 3.9888, 3.7872, 3.5961, 3.4149,
3.2431, 3.0802, 2.9257, 2.7792, 2.6402, 2.5084, 2.3834, 2.2648,
2.1522, 2.0455, 1.9441, 1.848, 1.7567, 1.6701, 1.5878, 1.5097,
1.4356, 1.3651, 1.2981, 1.2345, 1.174, 1.1167, 1.062, 1.0101,
.96087, .91414, .86986, .82781, .78777, .74971, .71339, .67882,
.64604, .61473, .58507, .55676, .52987, .5044, .48014, .45715,
.43527, .41453, .3948, .37609, .35831, .34142, .32524, .30995,
.29536, .28142, .26807, .25527, .24311, .23166, .22077, .21053,
.20081, .19143, .18261, .17407, .16603, .15833, .15089, .14385,
.13707, .13065, .12449, .11865, .11306, .10774, .10266, .097818,
.093203, .088815, .084641, .080671, .076892, .073296, .069873,
.066613, .06351, .060555, .05774, .055058, .052504, .050071,
.047752, .045543, .043438, .041432, .039521, .037699, .035962,
.034307, .032729, .031225, .029791, .028423, .02712, .025877,
.024692, .023563, .022485, .021458, .020478, .019543, .018652,
.017802, .016992, .016219, .015481, .014778, .014107, .013467,
.012856, .012274, .011718, .011188, .010682, .0102, .0097393,
.0093001, .008881, .0084812, .0080997, .0077358, .0073885,
.0070571, .0067409, .0064393, .0061514, .0058768, .0056147,
.0053647, .0051262, .0048987, .0046816, .0044745, .0042769,
.0040884, .0039088, .0037373, .0035739, .003418, .0032693,
.0031277, .0029926, .0028639, .0027413, .0026245, .0025133,
.0024074, .0023066, .0022108, .0021196, .002033, .0019507,
.0018726, .0017985, .0017282, .0016617, .0015988, .0015394,
.0014834, .0014306, .0013811, .0013346, .0012911, .0012506,
.0012131, .0011784, .0011465, .0011175, .0010912, .0010678,
.0010472, .0010295, .0010147, .001003, 9.9428e-4, 9.8883e-4,
9.8673e-4, 9.8821e-4, 9.9343e-4, .0010027, .0010164, .0010348,
.0010586, .0010882, .0011245, .0011685, .0012145, .0012666,
.0013095, .0013688, .0014048, .0014663, .0015309, .0015499,
.0016144, .0016312, .001705, .0017892, .0018499, .0019715,
.0021102, .0022442, .0024284, .0025893, .0027703, .0029445,
.0031193, .003346, .0034552, .0036906, .0037584, .0040084,
.0041934, .0044587, .0047093, .0049759, .0053421, .0055134,
.0059048, .0058663, .0061036, .0063259, .0059657, .0060653,
.0060972, .0055539, .0055653, .0055772, .005331, .0054953,
.0055919, .0058684, .006183, .0066675, .0069808, .0075142,
.0078536, .0084282, .0089454, .0094625, .0093703, .0095857,
.0099283, .010063, .010521, .0097778, .0098175, .010379, .010447,
.0105, .010617, .010706, .01078, .011177, .011212, .011304,
.011446, .011603, .011816, .012165, .012545, .013069, .013539,
.01411, .014776, .016103, .017016, .017994, .018978, .01998,
.021799, .022745, .023681, .024627, .025562, .026992, .027958,
.029013, .030154, .031402, .03228, .033651, .035272, .037088,
.039021, .041213, .043597, .045977, .04877, .051809, .054943,
.058064, .061528, .06537, .069309, .071928, .075752, .079589,
.083352, .084096, .087497, .090817, .091198, .094966, .099045,
.10429, .10867, .11518, .12269, .13126, .14087, .15161, .16388,
.16423, .1759, .18721, .19994, .21275, .22513, .23041, .24231,
.25299, .25396, .26396, .27696, .27929, .2908, .30595, .31433,
.3282, .3429, .35944, .37467, .39277, .41245, .43326, .45649,
.48152, .51897, .54686, .57877, .61263, .64962, .68983, .73945,
.78619, .83537, .89622, .95002, 1.0067, 1.0742, 1.1355, 1.2007,
1.2738, 1.347, 1.4254, 1.5094, 1.6009, 1.6976, 1.8019, 1.9148,
2.0357, 2.166, 2.3066, 2.4579, 2.6208, 2.7966, 2.986, 3.188,
3.4081, 3.6456, 3.9, 4.1747, 4.4712, 4.7931, 5.1359, 5.5097,
5.9117, 6.3435, 6.8003, 7.3001, 7.8385, 8.3945, 9.011, 9.6869,
10.392, 11.18, 12.036, 12.938, 13.944, 14.881, 16.029, 17.255,
18.574, 19.945, 21.38, 22.9, 24.477, 26.128, 27.87, 29.037,
30.988, 33.145, 35.506, 37.76, 40.885, 44.487, 48.505, 52.911,
57.56, 61.964, 67.217, 72.26, 78.343, 85.08, 91.867, 99.435,
107.68, 116.97, 127.12, 138.32, 150.26, 163.04, 174.81, 189.26,
205.61, 224.68, 240.98, 261.88, 285.1, 307.58, 334.35, 363.53,
394.68, 427.85, 458.85, 489.25, 472.87, 486.93, 496.27, 501.52,
501.57, 497.14, 488.09, 476.32, 393.76, 388.51, 393.42, 414.45,
455.12, 514.62, 520.38, 547.42, 562.6, 487.47, 480.83, 391.06,
376.92, 303.7, 295.91, 256.03, 236.73, 280.38, 310.71, 335.53,
367.88, 401.94, 435.52, 469.13, 497.94, 588.82, 597.94, 597.2,
588.28, 571.2, 555.75, 603.56, 638.15, 680.75, 801.72, 848.01,
962.15, 990.06, 1068.1, 1076.2, 1115.3, 1134.2, 1136.6, 1119.1,
1108.9, 1090.6, 1068.7, 1041.9, 1005.4, 967.98, 927.08, 780.1,
751.41, 733.12, 742.65, 785.56, 855.16, 852.45, 878.1, 784.59,
777.81, 765.13, 622.93, 498.09, 474.89, 386.9, 378.48, 336.17,
322.04, 329.57, 350.5, 383.38, 420.02, 462.39, 499.71, 531.98,
654.99, 653.43, 639.99, 605.16, 554.16, 504.42, 540.64, 552.33,
679.46, 699.51, 713.91, 832.17, 919.91, 884.96, 907.57, 846.56,
818.56, 768.93, 706.71, 642.17, 575.95, 515.38, 459.07, 409.02,
364.61, 325.46, 291.1, 260.89, 234.39, 211.01, 190.38, 172.11,
155.91, 141.49, 128.63, 117.13, 106.84, 97.584, 89.262, 81.756,
74.975, 68.842, 63.28, 58.232, 53.641, 49.46, 45.649, 42.168,
38.991, 36.078, 33.409, 30.96, 28.71, 26.642, 24.737, 22.985,
```

```
  21.37, 19.882, 18.512, 17.242, 16.073, 14.987, 13.984, 13.05,
  12.186, 11.384, 10.637, 9.9436, 9.2988, 8.6991, 8.141, 7.6215,
  7.1378, 6.6872, 6.2671, 5.8754, 5.51, 5.1691, 4.851, 4.5539,
  4.2764, 4.0169, 3.7742, 3.5472, 3.3348, 3.1359, 2.9495, 2.7749,
  2.6113, 2.4578, 2.3139, 2.1789, 2.0523, 1.9334, 1.8219, 1.7171,
  1.6188, 1.5263, 1.4395, 1.3579, 1.2812, 1.209, 1.1411, 1.0773,
  1.0171, .96048, .90713, .85684, .80959, .76495, .72282, .68309,
  .64563, .61035, .57707, .54573, .51622, .48834, .46199, .43709,
  .41359, .39129, .37034, .35064, .33198, .31442, .29784, .28218,
  .26732, .25337, .24017, .22774, .21601, .20479, .19426
};

static double co2260[2001] = { 5.7971e-5, 6.0733e-5, 6.3628e-5, 6.6662e-5,
  6.9843e-5, 7.3176e-5, 7.6671e-5, 8.0334e-5, 8.4175e-5, 8.8201e-5,
  9.2421e-5, 9.6846e-5, 1.0149e-4, 1.0635e-4, 1.1145e-4, 1.1679e-4,
  1.224e-4, 1.2828e-4, 1.3444e-4, 1.409e-4, 1.4768e-4, 1.5479e-4,
  1.6224e-4, 1.7006e-4, 1.7826e-4, 1.8685e-4, 1.9587e-4, 2.0532e-4,
  2.1524e-4, 2.2565e-4, 2.3656e-4, 2.48e-4, 2.6001e-4, 2.7261e-4,
  2.8582e-4, 2.9968e-4, 3.1422e-4, 3.2948e-4, 3.4548e-4, 3.6228e-4,
  3.799e-4, 3.9838e-4, 4.1778e-4, 4.3814e-4, 4.595e-4, 4.8191e-4,
  5.0543e-4, 5.3012e-4, 5.5603e-4, 5.8321e-4, 6.1175e-4, 6.417e-4,
  6.7314e-4, 7.0614e-4, 7.4078e-4, 7.7714e-4, 8.1531e-4, 8.5538e-4,
  8.9745e-4, 9.4162e-4, 9.8798e-4, .0010367, .0010878, .0011415,
  .0011978, .001257, .0013191, .0013844, .001453, .0015249,
  .0016006, .00168, .0017634, .001851, .001943, .0020397, .0021412,
  .0022479, .00236, .0024778, .0026015, .0027316, .0028682,
  .0030117, .0031626, .0033211, .0034877, .0036628, .0038469,
  .0040403, .0042436, .0044574, .004682, .0049182, .0051665,
  .0054276, .0057021, .0059907, .0062942, .0066133, .0069489,
  .0073018, .0076729, .0080632, .0084738, .0089056, .0093599,
  .0098377, .01034, .010869, .011426, .012011, .012627, .013276,
  .013958, .014676, .015431, .016226, .017063, .017944, .018872,
  .019848, .020876, .021958, .023098, .024298, .025561, .026892,
  .028293, .029769, .031323, .032961, .034686, .036503, .038418,
  .040435, .042561, .044801, .047161, .049649, .052271, .055035,
  .057948, .061019, .064256, .06767, .07127, .075066, .079069,
  .083291, .087744, .092441, .097396, .10262, .10814, .11396,
  .1201, .12658, .13342, .14064, .14826, .1563, .1648, .17376,
  .18323, .19324, .2038, .21496, .22674, .23919, .25234, .26624,
  .28093, .29646, .31287, .33021, .34855, .36794, .38844, .41012,
  .43305, .45731, .48297, .51011, .53884, .56924, .60141, .63547,
  .67152, .70969, .75012, .79292, .83826, .8863, .93718, .99111,
  1.0482, 1.1088, 1.173, 1.2411, 1.3133, 1.3898, 1.471, 1.5571,
  1.6485, 1.7455, 1.8485, 1.9577, 2.0737, 2.197, 2.3278, 2.4668,
  2.6145, 2.7715, 2.9383, 3.1156, 3.3042, 3.5047, 3.7181, 3.9451,
  4.1866, 4.4437, 4.7174, 5.0089, 5.3192, 5.65, 6.0025, 6.3782,
  6.7787, 7.206, 7.6617, 8.1479, 8.6669, 9.221, 9.8128, 10.445,
  11.12, 11.843, 12.615, 13.441, 14.325, 15.271, 16.283, 17.367,
  18.529, 19.776, 21.111, 22.544, 24.082, 25.731, 27.504, 29.409,
  31.452, 33.654, 36.024, 38.573, 41.323, 44.29, 47.492, 50.951,
  54.608, 58.588, 62.929, 67.629, 72.712, 78.226, 84.207, 90.699,
  97.749, 105.42, 113.77, 122.86, 132.78, 143.61, 155.44, 168.33,
  182.48, 198.01, 214.87, 233.39, 253.86, 276.34, 300.3, 327.28,
  356.89, 389.48, 422.29, 458.99, 501.39, 548.13, 595.62, 652.74,
  716.54, 784.57, 866.78, 960.59, 1062.8, 1072.5, 1189.5, 1319.4,
  1467.6, 1630.2, 1813.7, 2016.9, 2253., 2515.3, 2773.5, 3092.8,
  3444.4, 3720.4, 4104.3, 4527.5, 4645.9, 5021.7, 5462.2, 5597.,
  6110.6, 6732.5, 7513.8, 8270.6, 9640.6, 11487., 2796.1, 2680.1,
  2441.6, 2404.2, 2334.8, 2215.2, 1642.5, 1477.9, 1328.1, 1223.5,
  843.34, 766.96, 831.65, 834.84, 774.85, 1156.3, 1275.6, 1366.1,
  1795.6, 1885., 1936.5, 1953.4, 2154.4, 2002.7, 1789.8, 10381.,
  9040., 8216.5, 7384.7, 6721.9, 6187.7, 6143.8, 5703.9, 5276.6,
  4873.1, 4736., 4325.3, 3927., 3554.1, 3286.1, 2950.1, 2642.4,
  2368.7, 2138.9, 1914., 1719.6, 1543.9, 1388.6, 1252.1, 1132.2,
  1024.1, 1025.4, 920.58, 829.59, 750.54, 685.01, 624.25, 570.14,
  525.81, 481.85, 441.95, 408.71, 377.23, 345.86, 318.51, 292.26,
  268.34, 247.04, 227.14, 209.02, 192.69, 177.59, 163.78, 151.26,
  139.73, 129.19, 119.53, 110.7, 102.57, 95.109, 88.264, 81.948,
  76.13, 70.768, 65.827, 61.251, 57.022, 53.495, 49.824, 46.443,
  43.307, 40.405, 37.716, 35.241, 32.923, 30.77, 28.78, 26.915,
  25.177, 23.56, 22.059, 20.654, 19.345, 18.126, 16.988, 15.93,
  14.939, 14.014, 13.149, 12.343, 11.589, 10.884, 10.225, 9.6093,
  9.0327, 8.4934, 7.9889, 7.5166, 7.0744, 6.6604, 6.2727, 5.9098,
  5.5701, 5.2529, 4.955, 4.676, 4.4148, 4.171, 3.9426, 3.7332,
  3.5347, 3.3493, 3.1677, 3.0025, 2.8466, 2.6994, 2.5601, 2.4277,
  2.3016, 2.1814, 2.0664, 1.9564, 1.8279, 1.7311, 1.6427, 1.5645,
  1.4982, 1.443, 1.374, 1.3146, 1.2562, 1.17, 1.1105, 1.0272,
  .96863, .89718, .83654, .80226, .75908, .72431, .69573, .67174,
  .65126, .63315, .61693, .60182, .58715, .59554, .57649, .55526,
  .53177, .50622, .48176, .4813, .47642, .47492, .50273, .50293,
  .52687, .52239, .53419, .53814, .52626, .52211, .51492, .50622,
  .49746, .48841, .4792, .43534, .41999, .40349, .38586, .36799,
  .35108, .31089, .30803, .3171, .33599, .35041, .36149, .32924,
  .32462, .27309, .25961, .20922, .19504, .15683, .13098, .11588,
  .11478, .11204, .11363, .12135, .16423, .17785, .19094, .20236,
  .21084, .2154, .24108, .22848, .20871, .18797, .17963, .17834,
  .21552, .22284, .26945, .27052, .30108, .28977, .29772, .29224,
```

```
.27658, .24956, .22777, .20654, .18392, .16338, .1452, .12916,
.1152, .10304, .092437, .083163, .075031, .067878, .061564,
.055976, .051018, .046609, .042679, .03917, .036032, .033223,
.030706, .02845, .026428, .024617, .022998, .021554, .02027,
.019136, .018141, .017278, .016541, .015926, .015432, .015058,
.014807, .014666, .014635, .014728, .014947, .01527, .015728,
.016345, .017026, .017798, .018839, .019752, .020636, .021886,
.022695, .02327, .023478, .024292, .023544, .022222, .021932,
.020052, .018143, .017722, .017031, .017782, .01938, .020734,
.020476, .019255, .017477, .016878, .014617, .012489, .011765,
.0099077, .0086446, .0079446, .0078644, .0079763, .008671,
.01001, .0108, .012933, .015349, .016341, .018484, .020254,
.020254, .020478, .019591, .018595, .018385, .019913, .022254,
.024847, .025809, .028053, .029924, .030212, .031367, .03222,
.032739, .032537, .03286, .033344, .033507, .033499, .033339,
.032809, .033041, .031723, .029837, .027511, .026603, .024032,
.021914, .020948, .021701, .023425, .024259, .024987, .023818,
.021768, .019223, .018144, .015282, .012604, .01163, .0097907,
.008336, .0082473, .0079582, .0088077, .009779, .010129, .012145,
.014378, .016761, .01726, .018997, .019998, .019809, .01819,
.016358, .016099, .01617, .017939, .020223, .022521, .02277,
.024279, .025247, .024222, .023989, .023224, .021493, .020362,
.018596, .017309, .015975, .014466, .013171, .011921, .01078,
.0097229, .0087612, .0078729, .0070682, .0063494, .0057156,
.0051459, .0046273, .0041712, .0037686, .0034119, .003095,
.0028126, .0025603, .0023342, .0021314, .0019489, .0017845,
.001636, .0015017, .00138, .0012697, .0011694, .0010782,
9.9507e-4, 9.1931e-4, 8.5013e-4, 7.869e-4, 7.2907e-4, 6.7611e-4,
6.2758e-4, 5.8308e-4, 5.4223e-4, 5.0473e-4, 4.7027e-4, 4.3859e-4,
4.0946e-4, 3.8265e-4, 3.5798e-4, 3.3526e-4, 3.1436e-4, 2.9511e-4,
2.7739e-4, 2.6109e-4, 2.4609e-4, 2.3229e-4, 2.1961e-4, 2.0797e-4,
1.9729e-4, 1.875e-4, 1.7855e-4, 1.7038e-4, 1.6294e-4, 1.5619e-4,
1.5007e-4, 1.4456e-4, 1.3961e-4, 1.3521e-4, 1.3131e-4, 1.2789e-4,
1.2494e-4, 1.2242e-4, 1.2032e-4, 1.1863e-4, 1.1733e-4, 1.1641e-4,
1.1585e-4, 1.1565e-4, 1.158e-4, 1.1629e-4, 1.1712e-4, 1.1827e-4,
1.1976e-4, 1.2158e-4, 1.2373e-4, 1.262e-4, 1.2901e-4, 1.3214e-4,
1.3562e-4, 1.3944e-4, 1.4361e-4, 1.4814e-4, 1.5303e-4, 1.5829e-4,
1.6394e-4, 1.6999e-4, 1.7644e-4, 1.8332e-4, 1.9063e-4, 1.984e-4,
2.0663e-4, 2.1536e-4, 2.246e-4, 2.3436e-4, 2.4468e-4, 2.5558e-4,
2.6708e-4, 2.7921e-4, 2.92e-4, 3.0548e-4, 3.1968e-4, 3.3464e-4,
3.5039e-4, 3.6698e-4, 3.8443e-4, 4.0281e-4, 4.2214e-4, 4.4248e-4,
4.6389e-4, 4.864e-4, 5.1009e-4, 5.3501e-4, 5.6123e-4, 5.888e-4,
6.1781e-4, 6.4833e-4, 6.8043e-4, 7.142e-4, 7.4973e-4, 7.8711e-4,
8.2644e-4, 8.6783e-4, 9.1137e-4, 9.5721e-4, .0010054, .0010562,
.0011096, .0011659, .0012251, .0012875, .0013532, .0014224,
.0014953, .001572, .0016529, .0017381, .0018279, .0019226,
.0020224, .0021277, .0022386, .0023557, .0024792, .0026095,
.002747, .0028921, .0030453, .0032071, .003378, .0035586,
.0037494, .003951, .0041642, .0043897, .0046282, .0048805,
.0051476, .0054304, .00573, .0060473, .0063837, .0067404,
.0071188, .0075203, .0079466, .0083994, .0088806, .0093922,
.0099366, .010516, .011134, .011792, .012494, .013244, .014046,
.014898, .015808, .016781, .017822, .018929, .020108, .02138,
.022729, .02419, .02576, .027412, .029233, .031198, .033301,
.035594, .038092, .040767, .04372, .046918, .050246, .053974,
.058009, .061976, .066586, .071537, .076209, .081856, .087998,
.093821, .10113, .10913, .11731, .12724, .13821, .15025, .1639,
.17807, .19472, .21356, .23496, .25758, .28387, .31389, .34104,
.37469, .40989, .43309, .46845, .5042, .5023, .52981, .55275,
.51075, .51976, .52457, .44779, .44721, .4503, .4243, .45244,
.49491, .55399, .39021, .24802, .2501, .2618, .27475, .28879,
.31317, .33643, .36257, .4018, .43275, .46525, .53333, .56599,
.60557, .70142, .74194, .77736, .88567, .91182, .93294, .98407,
.98772, .99176, .9995, 1.2405, 1.3602, 1.338, 1.3255, 1.3267,
1.3404, 1.3634, 1.3967, 1.4407, 1.4961, 1.5603, 1.6328, 1.7153,
1.8094, 1.9091, 2.018, 2.1367, 2.264, 2.4035, 2.5562, 2.7179,
2.9017, 3.1052, 3.3304, 3.5731, 3.8488, 4.1553, 4.4769, 4.7818,
5.1711, 5.5204, 5.9516, 6.4097, 6.8899, 7.1118, 7.5469, 7.9735,
7.9511, 8.3014, 8.6418, 8.4757, 8.8256, 9.2294, 9.6923, 10.033,
10.842, 11.851, 11.78, 8.8435, 9.1381, 9.5956, 10.076, 10.629,
11.22, 11.883, 12.69, 13.163, 13.974, 14.846, 16.027, 17.053,
18.148, 19.715, 20.907, 22.163, 23.956, 25.235, 26.566, 27.94,
29.576, 30.956, 32.432, 35.337, 39.911, 41.128, 42.625, 44.386,
46.369, 48.619, 51.031, 53.674, 56.825, 59.921, 63.286, 66.929,
70.859, 75.081, 79.618, 84.513, 89.739, 95.335, 101.35, 107.76,
114.63, 121.98, 129.87, 138.3, 147.34, 157.04, 167.56, 178.67,
190.61, 203.43, 217.19, 231.99, 247.88, 264.98, 283.37, 303.17,
324.49, 347.47, 372.25, 398.98, 427.85, 459.06, 492.8, 529.31,
568.89, 611.79, 658.35, 708.91, 763.87, 823.65, 888.72, 959.58,
1036.8, 1121.8, 1213.9, 1314.3, 1423.8, 1543., 1672.8, 1813.4,
1966.1, 2131.4, 2309.5, 2499.3, 2705., 2925.7, 3161.6, 3411.3,
3611.5, 3889.2, 4191.1, 4519.3, 4877.9, 5272.9, 5712.9, 6142.7,
6719.6, 7385., 8145., 8977.7, 9831.9, 10827., 11934., 13063.,
14434., 15878., 17591., 19435., 21510., 23835., 26835., 29740.,
32878., 36305., 39830., 43273., 46931., 50499., 49586., 51598.,
53429., 54619., 55081., 55102., 54485., 53487., 52042., 42689.,
42607., 44020., 47994., 54169., 53916., 55808., 56642., 46049.,
```

```
44243., 32929., 30658., 21963., 20835., 15962., 13679., 17652.,
19680., 22388., 25625., 29184., 32520., 35720., 38414., 40523.,
49228., 48173., 45678., 41768., 37600., 41313., 42654., 44465.,
55736., 56630., 65409., 63308., 66572., 61845., 60379., 56777.,
51920., 46601., 41367., 36529., 32219., 28470., 25192., 22362.,
19907., 17772., 15907., 14273., 12835., 11567., 10445., 9450.2,
8565.1, 7776., 7070.8, 6439.2, 5872.3, 5362.4, 4903., 4488.3,
4113.4, 3773.8, 3465.8, 3186.1, 2931.7, 2700.1, 2488.8, 2296.,
2119.8, 1958.6, 1810.9, 1675.6, 1551.4, 1437.3, 1332.4, 1236.,
1147.2, 1065.3, 989.86, 920.22, 855.91, 796.48, 741.53, 690.69,
643.62, 600.02, 559.6, 522.13, 487.35, 455.06, 425.08, 397.21,
371.3, 347.2, 324.78, 303.9, 284.46, 266.34, 249.45, 233.7,
219.01, 205.3, 192.5, 180.55, 169.38, 158.95, 149.2, 140.07,
131.54, 123.56, 116.09, 109.09, 102.54, 96.405, 90.655, 85.266,
80.213, 75.475, 71.031, 66.861, 62.948, 59.275, 55.827, 52.587,
49.544, 46.686, 43.998, 41.473, 39.099, 36.867, 34.768, 32.795,
30.939, 29.192, 27.546, 25.998, 24.539, 23.164, 21.869, 20.65,
19.501, 18.419, 17.399, 16.438, 15.532, 14.678, 13.874, 13.115,
12.4, 11.726, 11.088, 10.488, 9.921, 9.3846, 8.8784, 8.3996,
7.9469, 7.5197, 7.1174, 6.738, 6.379, 6.0409, 5.7213, 5.419,
5.1327, 4.8611, 4.6046, 4.3617, 4.1316, 3.9138, 3.7077, 3.5125,
3.3281, 3.1536, 2.9885, 2.8323, 2.6846, 2.5447, 2.4124, 2.2871,
2.1686, 2.0564, 1.9501, 1.8495, 1.7543, 1.6641, 1.5787, 1.4978,
1.4212, 1.3486, 1.2799, 1.2147, 1.1529, 1.0943, 1.0388, .98602,
.93596, .8886, .84352, .80078, .76029, .722, .68585, .65161,
.61901, .58808, .55854, .53044, .5039, .47853, .45459, .43173,
.41008, .38965, .37021, .35186, .33444, .31797, .30234, .28758,
.2736, .26036, .24764, .2357, .22431, .21342, .20295, .19288,
.18334, .17444, .166, .15815, .15072, .14348, .13674, .13015,
.12399, .11807, .11231, .10689, .10164, .096696, .091955,
.087476, .083183, .079113, .075229, .071536, .068026, .064698,
.06154, .058544, .055699, .052997, .050431, .047993, .045676,
.043475, .041382, .039392, .037501, .035702, .033991, .032364,
.030817, .029345, .027945, .026613, .025345, .024139, .022991,
.021899, .02086, .019871, .018929, .018033, .01718, .016368,
.015595, .014859, .014158, .013491, .012856, .012251, .011675,
.011126, .010604, .010107, .0096331, .009182, .0087523, .0083431,
.0079533, .0075821, .0072284, .0068915, .0065706, .0062649,
.0059737, .0056963, .005432, .0051802, .0049404, .0047118,
.0044941, .0042867, .0040891, .0039009, .0037216, .0035507,
.003388, .0032329, .0030852, .0029445, .0028105, .0026829,
.0025613, .0024455, .0023353, .0022303, .0021304, .0020353,
.0019448, .0018587, .0017767, .0016988, .0016247, .0015543,
.0014874, .0014238, .0013635, .0013062, .0012519, .0012005,
.0011517, .0011057, .0010621, .001021, 9.8233e-4, 9.4589e-4,
9.1167e-4, 8.7961e-4, 8.4964e-4, 8.2173e-4, 7.9582e-4, 7.7189e-4,
7.499e-4, 7.2983e-4, 7.1167e-4, 6.9542e-4, 6.8108e-4, 6.6866e-4,
6.5819e-4, 6.4971e-4, 6.4328e-4, 6.3895e-4, 6.3681e-4, 6.3697e-4,
6.3956e-4, 6.4472e-4, 6.5266e-4, 6.6359e-4, 6.778e-4, 6.9563e-4,
7.1749e-4, 7.4392e-4, 7.7556e-4, 8.1028e-4, 8.4994e-4, 8.8709e-4,
9.3413e-4, 9.6953e-4, .0010202, .0010738, .0010976, .0011507,
.0011686, .0012264, .001291, .0013346, .0014246, .0015293,
.0016359, .0017824, .0019255, .0020854, .002247, .0024148,
.0026199, .0027523, .0029704, .0030702, .0033047, .0035013,
.0037576, .0040275, .0043089, .0046927, .0049307, .0053486,
.0053809, .0056699, .0059325, .0055488, .005634, .0056392,
.004946, .0048855, .0048208, .0044386, .0045498, .0046377,
.0048939, .0052396, .0057324, .0060859, .0066906, .0071148,
.0077224, .0082687, .008769, .0084471, .008572, .0087729,
.008775, .0090742, .0080704, .0080288, .0085747, .0086087,
.0086408, .0088752, .0089381, .0089757, .0093532, .0092824,
.0092566, .0092645, .0092735, .009342, .0095806, .0097991,
.010213, .010611, .011129, .011756, .013237, .01412, .015034,
.015936, .01682, .018597, .019315, .019995, .020658, .021289,
.022363, .022996, .023716, .024512, .025434, .026067, .027118,
.028396, .029865, .031442, .033253, .03525, .037296, .039701,
.042356, .045154, .048059, .051294, .054893, .058636, .061407,
.065172, .068974, .072676, .073379, .076547, .079556, .079134,
.082308, .085739, .090192, .09359, .099599, .10669, .11496,
.1244, .13512, .14752, .14494, .15647, .1668, .17863, .19029,
.20124, .20254, .21179, .21982, .21625, .22364, .23405, .23382,
.2434, .25708, .26406, .27621, .28909, .30395, .31717, .33271,
.3496, .36765, .38774, .40949, .446, .46985, .49846, .5287, .562,
.59841, .64598, .68834, .7327, .78978, .8373, .88708, .94744,
1.0006, 1.0574, 1.1215, 1.1856, 1.2546, 1.3292, 1.4107, 1.4974,
1.5913, 1.6931, 1.8028, 1.9212, 2.0492, 2.1874, 2.3365, 2.4978,
2.6718, 2.8588, 3.062, 3.2818, 3.5188, 3.7752, 4.0527, 4.3542,
4.6782, 5.0312, 5.4123, 5.8246, 6.2639, 6.7435, 7.2636, 7.8064,
8.4091, 9.0696, 9.7677, 10.548, 11.4, 12.309, 13.324, 14.284,
15.445, 16.687, 18.019, 19.403, 20.847, 22.366, 23.925, 25.537,
27.213, 28.069, 29.864, 31.829, 33.988, 35.856, 38.829, 42.321,
46.319, 50.606, 55.126, 59.126, 64.162, 68.708, 74.615, 81.176,
87.739, 95.494, 103.83, 113.38, 123.99, 135.8, 148.7, 162.58,
176.32, 192.6, 211.47, 232.7, 252.64, 277.41, 305.38, 333.44,
366.42, 402.66, 442.14, 484.53, 526.42, 568.15, 558.78, 582.6,
600.98, 613.94, 619.44, 618.24, 609.84, 595.96, 484.86, 475.59,
478.49, 501.56, 552.19, 628.44, 630.39, 658.92, 671.96, 562.7,
```

```
    545.88, 423.43, 400.14, 306.59, 294.13, 246.8, 226.51, 278.21,
    314.39, 347.22, 389.13, 433.16, 477.48, 521.67, 560.54, 683.6,
    696.37, 695.91, 683.1, 658.24, 634.89, 698.85, 742.87, 796.66,
    954.49, 1009.5, 1150.5, 1179.1, 1267.9, 1272.4, 1312.7, 1330.4,
    1331.6, 1315.8, 1308.3, 1293.3, 1274.6, 1249.5, 1213.2, 1172.1,
    1124.4, 930.33, 893.36, 871.27, 883.54, 940.76, 1036., 1025.6,
    1053.1, 914.51, 894.15, 865.03, 670.63, 508.41, 475.15, 370.85,
    361.06, 319.38, 312.75, 331.87, 367.13, 415., 467.94, 525.49,
    578.41, 624.66, 794.82, 796.97, 780.29, 736.49, 670.18, 603.75,
    659.67, 679.8, 857.12, 884.05, 900.65, 1046.1, 1141.9, 1083.,
    1089.2, 1e3, 947.08, 872.31, 787.91, 704.75, 624.93, 553.68,
    489.91, 434.21, 385.64, 343.3, 306.42, 274.18, 245.94, 221.11,
    199.23, 179.88, 162.73, 147.48, 133.88, 121.73, 110.86, 101.1,
    92.323, 84.417, 77.281, 70.831, 64.991, 59.694, 54.884, 50.509,
    46.526, 42.893, 39.58, 36.549, 33.776, 31.236, 28.907, 26.77,
    24.805, 23., 21.339, 19.81, 18.404, 17.105, 15.909, 14.801,
    13.778, 12.83, 11.954, 11.142, 10.389, 9.691, 9.0434, 8.4423,
    7.8842, 7.3657, 6.8838, 6.4357, 6.0189, 5.6308, 5.2696, 4.9332,
    4.6198, 4.3277, 4.0553, 3.8012, 3.5639, 3.3424, 3.1355, 2.9422,
    2.7614, 2.5924, 2.4343, 2.2864, 2.148, 2.0184, 1.8971, 1.7835,
    1.677, 1.5773, 1.4838, 1.3961, 1.3139, 1.2369, 1.1645, 1.0966,
    1.0329, .97309, .91686, .86406, .81439, .76767, .72381, .68252,
    .64359, .60695, .57247, .54008, .50957, .48092, .45401, .42862,
    .40465, .38202, .36072, .34052, .3216, .30386, .28711, .27135,
    .25651, .24252, .2293, .21689, .20517, .19416, .18381, .17396,
    .16469
  };

  static double co2230[2001] = { 2.743e-5, 2.8815e-5, 3.027e-5, 3.1798e-5,
    3.3405e-5, 3.5094e-5, 3.6869e-5, 3.8734e-5, 4.0694e-5, 4.2754e-5,
    4.492e-5, 4.7196e-5, 4.9588e-5, 5.2103e-5, 5.4747e-5, 5.7525e-5,
    6.0446e-5, 6.3516e-5, 6.6744e-5, 7.0137e-5, 7.3704e-5, 7.7455e-5,
    8.1397e-5, 8.5543e-5, 8.9901e-5, 9.4484e-5, 9.9302e-5, 1.0437e-4,
    1.097e-4, 1.153e-4, 1.2119e-4, 1.2738e-4, 1.3389e-4, 1.4074e-4,
    1.4795e-4, 1.5552e-4, 1.6349e-4, 1.7187e-4, 1.8068e-4, 1.8995e-4,
    1.997e-4, 2.0996e-4, 2.2075e-4, 2.321e-4, 2.4403e-4, 2.5659e-4,
    2.698e-4, 2.837e-4, 2.9832e-4, 3.137e-4, 3.2988e-4, 3.4691e-4,
    3.6483e-4, 3.8368e-4, 4.0351e-4, 4.2439e-4, 4.4635e-4, 4.6947e-4,
    4.9379e-4, 5.1939e-4, 5.4633e-4, 5.7468e-4, 6.0452e-4, 6.3593e-4,
    6.69e-4, 7.038e-4, 7.4043e-4, 7.79e-4, 8.1959e-4, 8.6233e-4,
    9.0732e-4, 9.5469e-4, .0010046, .0010571, .0011124, .0011706,
    .0012319, .0012964, .0013644, .001436, .0015114, .0015908,
    .0016745, .0017625, .0018553, .0019531, .002056, .0021645,
    .0022788, .0023992, .002526, .0026596, .0028004, .0029488,
    .0031052, .0032699, .0034436, .0036265, .0038194, .0040227,
    .0042369, .0044628, .0047008, .0049518, .0052164, .0054953,
    .0057894, .0060995, .0064265, .0067713, .007135, .0075184,
    .0079228, .0083494, .0087993, .0092738, .0097745, .010303,
    .01086, .011448, .012068, .012722, .013413, .014142, .014911,
    .015723, .01658, .017484, .018439, .019447, .020511, .021635,
    .022821, .024074, .025397, .026794, .02827, .029829, .031475,
    .033215, .035052, .036994, .039045, .041213, .043504, .045926,
    .048485, .05119, .05405, .057074, .060271, .063651, .067225,
    .071006, .075004, .079233, .083708, .088441, .093449, .098749,
    .10436, .11029, .11657, .12322, .13026, .13772, .14561, .15397,
    .16282, .1722, .18214, .19266, .20381, .21563, .22816, .24143,
    .2555, .27043, .28625, .30303, .32082, .3397, .35972, .38097,
    .40352, .42746, .45286, .47983, .50847, .53888, .57119, .6055,
    .64196, .6807, .72187, .76564, .81217, .86165, .91427, .97025,
    1.0298, 1.0932, 1.1606, 1.2324, 1.3088, 1.3902, 1.477, 1.5693,
    1.6678, 1.7727, 1.8845, 2.0038, 2.131, 2.2666, 2.4114, 2.5659,
    2.7309, 2.907, 3.0951, 3.2961, 3.5109, 3.7405, 3.986, 4.2485,
    4.5293, 4.8299, 5.1516, 5.4961, 5.8651, 6.2605, 6.6842, 7.1385,
    7.6256, 8.1481, 8.7089, 9.3109, 9.9573, 10.652, 11.398, 12.2,
    13.063, 13.992, 14.99, 16.064, 17.222, 18.469, 19.813, 21.263,
    22.828, 24.516, 26.34, 28.31, 30.437, 32.738, 35.226, 37.914,
    40.824, 43.974, 47.377, 51.061, 55.011, 59.299, 63.961, 69.013,
    74.492, 80.444, 86.919, 93.836, 101.23, 109.25, 117.98, 127.47,
    137.81, 149.07, 161.35, 174.75, 189.42, 205.49, 223.02, 242.26,
    263.45, 286.75, 311.94, 340.01, 370.86, 404.92, 440.44, 480.27,
    525.17, 574.71, 626.22, 686.8, 754.38, 827.07, 913.38, 1011.7,
    1121.5, 1161.6, 1289.5, 1432.2, 1595.4, 1777., 1983.3, 2216.1,
    2485.7, 2788.3, 3101.5, 3481., 3902.1, 4257.1, 4740., 5272.8,
    5457.9, 5946.2, 6505.3, 6668.4, 7302.4, 8061.6, 9015.8, 9908.3,
    11613., 13956., 3249.6, 3243., 2901.5, 2841.3, 2729.6, 2558.2,
    1797.8, 1583.2, 1386., 1233.5, 787.74, 701.46, 761.66, 767.21,
    722.83, 1180.6, 1332.1, 1461.6, 2032.9, 2166., 2255.9, 2294.7,
    2587.2, 2396.5, 2122.4, 12553., 10784., 9832.5, 8827.3, 8029.1,
    7377.9, 7347.1, 6783.8, 6239.1, 5721.1, 5503., 4975.1, 4477.8,
    4021.3, 3676.8, 3275.3, 2914.9, 2597.4, 2328.2, 2075.4, 1857.6,
    1663.6, 1493.3, 1343.8, 1213.3, 1095.6, 1066.5, 958.91, 865.15,
    783.31, 714.35, 650.77, 593.98, 546.2, 499.9, 457.87, 421.75,
    387.61, 355.25, 326.62, 299.7, 275.21, 253.17, 232.83, 214.31,
    197.5, 182.08, 167.98, 155.12, 143.32, 132.5, 122.58, 113.48,
    105.11, 97.415, 90.182, 83.463, 77.281, 71.587, 66.341, 61.493,
    57.014, 53.062, 49.21, 45.663, 42.38, 39.348, 36.547, 33.967,
    31.573, 29.357, 27.314, 25.415, 23.658, 22.03, 20.524, 19.125,
```

```
17.829, 16.627, 15.511, 14.476, 13.514, 12.618, 11.786, 11.013,
10.294, 9.6246, 9.0018, 8.4218, 7.8816, 7.3783, 6.9092, 6.4719,
6.0641, 5.6838, 5.3289, 4.998, 4.6893, 4.4014, 4.1325, 3.8813,
3.6469, 3.4283, 3.2241, 3.035, 2.8576, 2.6922, 2.5348, 2.3896,
2.2535, 2.1258, 2.0059, 1.8929, 1.7862, 1.6854, 1.5898, 1.4992,
1.4017, 1.3218, 1.2479, 1.1809, 1.1215, 1.0693, 1.0116, .96016,
.9105, .84859, .80105, .74381, .69982, .65127, .60899, .57843,
.54592, .51792, .49336, .47155, .45201, .43426, .41807, .40303,
.38876, .3863, .37098, .35492, .33801, .32032, .30341, .29874,
.29193, .28689, .29584, .29155, .29826, .29195, .29287, .2904,
.28199, .27709, .27162, .26622, .26133, .25676, .25235, .23137,
.22365, .21519, .20597, .19636, .18699, .16485, .16262, .16643,
.17542, .18198, .18631, .16759, .16338, .13505, .1267, .10053,
.092554, .074093, .062159, .055523, .054849, .05401, .05528,
.058982, .07952, .08647, .093244, .099285, .10393, .10661,
.12072, .11417, .10396, .093265, .089137, .088909, .10902,
.11277, .13625, .13565, .14907, .14167, .1428, .13744, .12768,
.11382, .10244, .091686, .08109, .071739, .063616, .056579,
.050504, .045251, .040689, .036715, .033237, .030181, .027488,
.025107, .022998, .021125, .01946, .017979, .016661, .015489,
.014448, .013526, .012712, .011998, .011375, .010839, .010384,
.010007, .0097053, .0094783, .0093257, .0092489, .0092504,
.0093346, .0095077, .0097676, .01012, .01058, .011157, .011844,
.012672, .013665, .014766, .015999, .017509, .018972, .020444,
.022311, .023742, .0249, .025599, .026981, .026462, .025143,
.025066, .022814, .020458, .020026, .019142, .020189, .022371,
.024163, .023728, .02199, .019506, .018591, .015576, .012784,
.011744, .0094777, .0079148, .0070652, .006986, .0071758,
.008086, .0098025, .01087, .013609, .016764, .018137, .021061,
.023498, .023576, .023965, .022828, .021519, .021283, .023364,
.026457, .029782, .030856, .033486, .035515, .035543, .036558,
.037198, .037472, .037045, .037284, .03777, .038085, .038366,
.038526, .038282, .038915, .037697, .035667, .032941, .031959,
.028692, .025918, .024596, .025592, .027873, .028935, .02984,
.028148, .025305, .021912, .020454, .016732, .013357, .01205,
.009731, .0079881, .0077704, .0074387, .0083895, .0096776,
.010326, .01293, .015955, .019247, .020145, .02267, .024231,
.024184, .022131, .019784, .01955, .01971, .022119, .025116,
.027978, .028107, .029808, .030701, .029164, .028551, .027286,
.024946, .023259, .020982, .019221, .017471, .015643, .014074,
.01261, .011301, .010116, .0090582, .0081036, .0072542, .0065034,
.0058436, .0052571, .0047321, .0042697, .0038607, .0034977,
.0031747, .0028864, .0026284, .002397, .002189, .0020017,
.0018326, .0016798, .0015414, .0014159, .0013019, .0011983,
.0011039, .0010177, 9.391e-4, 8.6717e-4, 8.0131e-4, 7.4093e-4,
6.8553e-4, 6.3464e-4, 5.8787e-4, 5.4487e-4, 5.0533e-4, 4.69e-4,
4.3556e-4, 4.0474e-4, 3.7629e-4, 3.5e-4, 3.2569e-4, 3.032e-4,
2.8239e-4, 2.6314e-4, 2.4535e-4, 2.2891e-4, 2.1374e-4, 1.9975e-4,
1.8685e-4, 1.7498e-4, 1.6406e-4, 1.5401e-4, 1.4479e-4, 1.3633e-4,
1.2858e-4, 1.2148e-4, 1.1499e-4, 1.0907e-4, 1.0369e-4, 9.8791e-5,
9.4359e-5, 9.0359e-5, 8.6766e-5, 8.3555e-5, 8.0703e-5, 7.8192e-5,
7.6003e-5, 7.4119e-5, 7.2528e-5, 7.1216e-5, 7.0171e-5, 6.9385e-5,
6.8848e-5, 6.8554e-5, 6.8496e-5, 6.8669e-5, 6.9069e-5, 6.9694e-5,
7.054e-5, 7.1608e-5, 7.2896e-5, 7.4406e-5, 7.6139e-5, 7.8097e-5,
8.0283e-5, 8.2702e-5, 8.5357e-5, 8.8255e-5, 9.1402e-5, 9.4806e-5,
9.8473e-5, 1.0241e-4, 1.0664e-4, 1.1115e-4, 1.1598e-4, 1.2112e-4,
1.2659e-4, 1.3241e-4, 1.3859e-4, 1.4515e-4, 1.521e-4, 1.5947e-4,
1.6728e-4, 1.7555e-4, 1.8429e-4, 1.9355e-4, 2.0334e-4, 2.1369e-4,
2.2463e-4, 2.3619e-4, 2.4841e-4, 2.6132e-4, 2.7497e-4, 2.8938e-4,
3.0462e-4, 3.2071e-4, 3.3771e-4, 3.5567e-4, 3.7465e-4, 3.947e-4,
4.1588e-4, 4.3828e-4, 4.6194e-4, 4.8695e-4, 5.1338e-4, 5.4133e-4,
5.7087e-4, 6.0211e-4, 6.3515e-4, 6.701e-4, 7.0706e-4, 7.4617e-4,
7.8756e-4, 8.3136e-4, 8.7772e-4, 9.2681e-4, 9.788e-4, .0010339,
.0010922, .001154, .0012195, .0012889, .0013626, .0014407,
.0015235, .0016114, .0017048, .0018038, .001909, .0020207,
.0021395, .0022657, .0023998, .0025426, .0026944, .002856,
.0030281, .0032114, .0034068, .003615, .0038371, .004074,
.004327, .0045971, .0048857, .0051942, .0055239, .0058766,
.0062538, .0066573, .0070891, .007551, .0080455, .0085747,
.0091412, .0097481, .010397, .011092, .011837, .012638, .013495,
.014415, .01541, .016475, .017621, .018857, .020175, .02162,
.023185, .024876, .02672, .028732, .030916, .033319, .035939,
.038736, .041847, .04524, .048715, .052678, .056977, .061203,
.066184, .07164, .076952, .083477, .090674, .098049, .10697,
.1169, .1277, .14011, .15323, .1684, .18601, .20626, .22831,
.25417, .28407, .31405, .34957, .38823, .41923, .46026, .50409,
.51227, .54805, .57976, .53818, .55056, .557, .46741, .46403,
.4636, .42265, .45166, .49852, .56663, .34306, .17779, .17697,
.18346, .19129, .20014, .21778, .23604, .25649, .28676, .31238,
.33856, .39998, .4288, .46568, .56654, .60786, .64473, .76466,
.7897, .80778, .86443, .85736, .84798, .84157, 1.1385, 1.2446,
1.1923, 1.1552, 1.1338, 1.1266, 1.1292, 1.1431, 1.1683, 1.2059,
1.2521, 1.3069, 1.3712, 1.4471, 1.5275, 1.6165, 1.7145, 1.8189,
1.9359, 2.065, 2.2007, 2.3591, 2.5362, 2.7346, 2.9515, 3.2021,
3.4851, 3.7935, 4.0694, 4.4463, 4.807, 5.2443, 5.7178, 6.2231,
6.4796, 6.9461, 7.4099, 7.3652, 7.7182, 8.048, 7.7373, 8.0363,
8.3855, 8.8044, 9.0257, 9.8574, 10.948, 10.563, 6.8979, 7.0744,
```

```
7.4121, 7.7663, 8.1768, 8.6243, 9.1437, 9.7847, 10.182, 10.849,
11.572, 12.602, 13.482, 14.431, 15.907, 16.983, 18.11, 19.884,
21.02, 22.18, 23.355, 24.848, 25.954, 27.13, 30.186, 34.893,
35.682, 36.755, 38.111, 39.703, 41.58, 43.606, 45.868, 48.573,
51.298, 54.291, 57.559, 61.116, 64.964, 69.124, 73.628, 78.471,
83.683, 89.307, 95.341, 101.84, 108.83, 116.36, 124.46, 133.18,
142.57, 152.79, 163.69, 175.43, 188.11, 201.79, 216.55, 232.51,
249.74, 268.38, 288.54, 310.35, 333.97, 359.55, 387.26, 417.3,
449.88, 485.2, 523.54, 565.14, 610.28, 659.31, 712.56, 770.43,
833.36, 901.82, 976.36, 1057.6, 1146.8, 1243.8, 1350., 1466.3,
1593.6, 1732.7, 1884.1, 2049.1, 2228.2, 2421.9, 2629.4, 2853.7,
3094.4, 3351.1, 3622.3, 3829.8, 4123.1, 4438.3, 4777.2, 5144.1,
5545.4, 5990.5, 6404.5, 6996.8, 7687.6, 8482.9, 9349.4, 10203.,
11223., 12358., 13493., 14916., 16416., 18236., 20222., 22501.,
25102., 28358., 31707., 35404., 39538., 43911., 48391., 53193.,
58028., 58082., 61276., 64193., 66294., 67480., 67921., 67423.,
66254., 64341., 51737., 51420., 53072., 58145., 66195., 65358.,
67377., 67869., 53509., 50553., 35737., 32425., 21704., 19974.,
14457., 12142., 16798., 19489., 23049., 27270., 31910., 36457.,
40877., 44748., 47876., 59793., 58626., 55454., 50337., 44893.,
50228., 52216., 54747., 69541., 70455., 81014., 77694., 80533.,
73953., 70927., 65539., 59002., 52281., 45953., 40292., 35360.,
31124., 27478., 24346., 21647., 19308., 17271., 15491., 13927.,
12550., 11331., 10250., 9288.8, 8431.4, 7664.9, 6978.3, 6361.8,
5807.4, 5307.7, 4856.8, 4449., 4079.8, 3744.9, 3440.8, 3164.2,
2912.3, 2682.7, 2473., 2281.4, 2106., 1945.3, 1797.9, 1662.5,
1538.1, 1423.6, 1318.1, 1221., 1131.5, 1049., 972.99, 902.87,
838.01, 777.95, 722.2, 670.44, 622.35, 577.68, 536.21, 497.76,
462.12, 429.13, 398.61, 370.39, 344.29, 320.16, 297.85, 277.2,
258.08, 240.38, 223.97, 208.77, 194.66, 181.58, 169.43, 158.15,
147.67, 137.92, 128.86, 120.44, 112.6, 105.3, 98.499, 92.166,
86.264, 80.763, 75.632, 70.846, 66.381, 62.213, 58.321, 54.685,
51.288, 48.114, 45.145, 42.368, 39.772, 37.341, 35.065, 32.937,
30.943, 29.077, 27.33, 25.693, 24.158, 22.717, 21.367, 20.099,
18.909, 17.792, 16.744, 15.761, 14.838, 13.971, 13.157, 12.393,
11.676, 11.003, 10.369, 9.775, 9.2165, 8.6902, 8.1963, 7.7314,
7.2923, 6.8794, 6.4898, 6.122, 5.7764, 5.4525, 5.1484, 4.8611,
4.5918, 4.3379, 4.0982, 3.8716, 3.6567, 3.4545, 3.2634, 3.0828,
2.9122, 2.7512, 2.5993, 2.4561, 2.3211, 2.1938, 2.0737, 1.9603,
1.8534, 1.7525, 1.6572, 1.5673, 1.4824, 1.4022, 1.3265, 1.2551,
1.1876, 1.1239, 1.0637, 1.0069, .9532, .90248, .85454, .80921,
.76631, .72569, .6872, .65072, .61635, .5836, .55261, .52336,
.49581, .46998, .44559, .42236, .40036, .37929, .35924, .34043,
.32238, .30547, .28931, .27405, .25975, .24616, .23341, .22133,
.20997, .19924, .18917, .17967, .17075, .16211, .15411, .14646,
.13912, .13201, .12509, .11857, .11261, .10698, .10186, .097039,
.092236, .087844, .083443, .07938, .075452, .071564, .067931,
.064389, .061078, .057901, .054921, .052061, .049364, .046789,
.04435, .042044, .039866, .037808, .035863, .034023, .032282,
.030634, .029073, .027595, .026194, .024866, .023608, .022415,
.021283, .02021, .019193, .018228, .017312, .016443, .015619,
.014837, .014094, .01339, .012721, .012086, .011483, .010911,
.010368, .009852, .0093623, .0088972, .0084556, .0080362,
.0076379, .0072596, .0069003, .006559, .0062349, .0059269,
.0056344, .0053565, .0050925, .0048417, .0046034, .004377,
.0041618, .0039575, .0037633, .0035788, .0034034, .0032368,
.0030785, .002928, .0027851, .0026492, .0025201, .0023975,
.0022809, .0021701, .0020649, .0019649, .0018699, .0017796,
.0016938, .0016122, .0015348, .0014612, .0013913, .001325,
.0012619, .0012021, .0011452, .0010913, .0010401, 9.9149e-4,
9.454e-4, 9.0169e-4, 8.6024e-4, 8.2097e-4, 7.8377e-4, 7.4854e-4,
7.1522e-4, 6.8371e-4, 6.5393e-4, 6.2582e-4, 5.9932e-4, 5.7435e-4,
5.5087e-4, 5.2882e-4, 5.0814e-4, 4.8881e-4, 4.7076e-4, 4.5398e-4,
4.3843e-4, 4.2407e-4, 4.109e-4, 3.9888e-4, 3.88e-4, 3.7826e-4,
3.6963e-4, 3.6213e-4, 3.5575e-4, 3.505e-4, 3.464e-4, 3.4346e-4,
3.4173e-4, 3.4125e-4, 3.4206e-4, 3.4424e-4, 3.4787e-4, 3.5303e-4,
3.5986e-4, 3.6847e-4, 3.7903e-4, 3.9174e-4, 4.0681e-4, 4.2455e-4,
4.4527e-4, 4.6942e-4, 4.9637e-4, 5.2698e-4, 5.5808e-4, 5.9514e-4,
6.2757e-4, 6.689e-4, 7.1298e-4, 7.3955e-4, 7.8403e-4, 8.0449e-4,
8.5131e-4, 9.0256e-4, 9.3692e-4, .0010051, .0010846, .0011678,
.001282, .0014016, .0015355, .0016764, .0018272, .0020055,
.0021455, .0023421, .0024615, .0026786, .0028787, .0031259,
.0034046, .0036985, .0040917, .0043902, .0048349, .0049531,
.0052989, .0056148, .0052452, .0053357, .005333, .0045069,
.0043851, .004253, .003738, .0038084, .0039013, .0041505,
.0045372, .0050569, .0054507, .0061267, .0066122, .0072449,
.0078012, .0082651, .0076538, .0076573, .0076806, .0075227,
.0076269, .0063758, .006254, .0067749, .0067909, .0068231,
.0072143, .0072762, .0072954, .007679, .0075107, .0073658,
.0072441, .0071074, .0070378, .007176, .0072472, .0075844,
.0079291, .008412, .0090165, .010688, .011535, .012375, .013166,
.013895, .015567, .016011, .016392, .016737, .017043, .017731,
.018031, .018419, .018877, .019474, .019868, .020604, .021538,
.022653, .023869, .025288, .026879, .028547, .030524, .03274,
.035132, .03769, .040567, .043793, .047188, .049962, .053542,
.057205, .060776, .061489, .064419, .067124, .065945, .068487,
.071209, .074783, .077039, .082444, .08902, .09692, .10617,
```

```
    .11687, .12952, .12362, .13498, .14412, .15492, .16519, .1744,
    .17096, .17714, .18208, .17363, .17813, .18564, .18295, .19045,
    .20252, .20815, .21844, .22929, .24229, .25321, .26588, .2797,
    .29465, .31136, .32961, .36529, .38486, .41027, .43694, .4667,
    .49943, .54542, .58348, .62303, .67633, .71755, .76054, .81371,
    .85934, .90841, .96438, 1.0207, 1.0821, 1.1491, 1.2226, 1.3018,
    1.388, 1.4818, 1.5835, 1.6939, 1.8137, 1.9435, 2.0843, 2.237,
    2.4026, 2.5818, 2.7767, 2.9885, 3.2182, 3.4679, 3.7391, 4.0349,
    4.3554, 4.7053, 5.0849, 5.4986, 5.9436, 6.4294, 6.9598, 7.5203,
    8.143, 8.8253, 9.5568, 10.371, 11.267, 12.233, 13.31, 14.357,
    15.598, 16.93, 18.358, 19.849, 21.408, 23.04, 24.706, 26.409,
    28.153, 28.795, 30.549, 32.43, 34.49, 36.027, 38.955, 42.465,
    46.565, 50.875, 55.378, 59.002, 63.882, 67.949, 73.693, 80.095,
    86.403, 94.264, 102.65, 112.37, 123.3, 135.54, 149.14, 163.83,
    179.17, 196.89, 217.91, 240.94, 264.13, 292.39, 324.83, 358.21,
    397.16, 440.5, 488.6, 541.04, 595.3, 650.43, 652.03, 688.74,
    719.47, 743.54, 757.68, 762.35, 756.43, 741.42, 595.43, 580.97,
    580.83, 605.68, 667.88, 764.49, 759.93, 789.12, 798.17, 645.66,
    615.65, 455.05, 421.09, 306.45, 289.14, 235.7, 215.52, 274.57,
    316.53, 357.73, 409.89, 465.06, 521.84, 579.02, 630.64, 794.46,
    813., 813.56, 796.25, 761.57, 727.97, 812.14, 866.75, 932.5,
    1132.8, 1194.8, 1362.2, 1387.2, 1482.3, 1479.7, 1517.9, 1533.1,
    1534.2, 1523.3, 1522.5, 1515.5, 1505.2, 1486.5, 1454., 1412.,
    1358.8, 1107.8, 1060.9, 1033.5, 1048.2, 1122.4, 1248.9, 1227.1,
    1255.4, 1058.9, 1020.7, 970.59, 715.24, 512.56, 468.47, 349.3,
    338.26, 299.22, 301.26, 332.38, 382.08, 445.49, 515.87, 590.85,
    662.3, 726.05, 955.59, 964.11, 945.17, 891.48, 807.11, 720.9,
    803.36, 834.46, 1073.9, 1107.1, 1123.6, 1296., 1393.7, 1303.1,
    1284.3, 1161.8, 1078.8, 976.13, 868.72, 767.4, 674.72, 593.73,
    523.12, 462.24, 409.75, 364.34, 325., 290.73, 260.76, 234.46,
    211.28, 190.78, 172.61, 156.44, 142.01, 129.12, 117.57, 107.2,
    97.877, 89.47, 81.882, 75.021, 68.807, 63.171, 58.052, 53.396,
    49.155, 45.288, 41.759, 38.531, 35.576, 32.868, 30.384, 28.102,
    26.003, 24.071, 22.293, 20.655, 19.147, 17.756, 16.476, 15.292,
    14.198, 13.183, 12.241, 11.367, 10.554, 9.7989, 9.0978, 8.4475,
    7.845, 7.2868, 6.7704, 6.2927, 5.8508, 5.4421, 5.064, 4.714,
    4.3902, 4.0902, 3.8121, 3.5543, 3.315, 3.093, 2.8869, 2.6953,
    2.5172, 2.3517, 2.1977, 2.0544, 1.9211, 1.7969, 1.6812, 1.5735,
    1.4731, 1.3794, 1.2921, 1.2107, 1.1346, 1.0637, .99744, .93554,
    .87771, .82368, .77313, .72587, .6816, .64014, .60134, .565,
    .53086, .49883, .46881, .44074, .4144, .38979, .36679, .34513,
    .32474, .30552, .28751, .27045, .25458, .23976, .22584, .21278,
    .20051, .18899, .17815, .16801, .15846, .14954, .14117, .13328,
    .12584
  };

  double xw, dw, ew, cw296, cw260, cw230, dt230, dt260, dt296, ctw, ctmpth;

  int iw;

  /* Get CO2 continuum absorption... */
  xw = nu / 2 + 1;
  if (xw >= 1 && xw < 2001) {
    iw = (int) xw;
    dw = xw - iw;
    ew = 1 - dw;
    cw296 = ew * co2296[iw - 1] + dw * co2296[iw];
    cw260 = ew * co2260[iw - 1] + dw * co2260[iw];
    cw230 = ew * co2230[iw - 1] + dw * co2230[iw];
    dt230 = t - 230;
    dt260 = t - 260;
    dt296 = t - 296;
    ctw = dt260 * 5.050505e-4 * dt296 * cw230 - dt230 * 9.259259e-4
      * dt296 * cw260 + dt230 * 4.208754e-4 * dt260 * cw296;
    ctmpth = u / GSL_CONST_NUM_AVOGADRO / 1000 * p / P0 * ctw;
  } else
    ctmpth = 0;
  return ctmpth;
}
```

5.33.2.7 double ctmh2o (double *nu*, double *p*, double *t*, double *q*, double *u*)

Compute water vapor continuum (optical depth).

Definition at line 1737 of file jurassic.c.

```
             {

  static double h2o296[2001] = { .17, .1695, .172, .168, .1687, .1624, .1606,
    .1508, .1447, .1344, .1214, .1133, .1009, .09217, .08297, .06989,
```

```
.06513, .05469, .05056, .04417, .03779, .03484, .02994, .0272,
.02325, .02063, .01818, .01592, .01405, .01251, .0108, .009647,
.008424, .007519, .006555, .00588, .005136, .004511, .003989,
.003509, .003114, .00274, .002446, .002144, .001895, .001676,
.001486, .001312, .001164, .001031, 9.129e-4, 8.106e-4, 7.213e-4,
6.4e-4, 5.687e-4, 5.063e-4, 4.511e-4, 4.029e-4, 3.596e-4,
3.22e-4, 2.889e-4, 2.597e-4, 2.337e-4, 2.108e-4, 1.907e-4,
1.728e-4, 1.57e-4, 1.43e-4, 1.305e-4, 1.195e-4, 1.097e-4,
1.009e-4, 9.307e-5, 8.604e-5, 7.971e-5, 7.407e-5, 6.896e-5,
6.433e-5, 6.013e-5, 5.631e-5, 5.283e-5, 4.963e-5, 4.669e-5,
4.398e-5, 4.148e-5, 3.917e-5, 3.702e-5, 3.502e-5, 3.316e-5,
3.142e-5, 2.978e-5, 2.825e-5, 2.681e-5, 2.546e-5, 2.419e-5,
2.299e-5, 2.186e-5, 2.079e-5, 1.979e-5, 1.884e-5, 1.795e-5,
1.711e-5, 1.633e-5, 1.559e-5, 1.49e-5, 1.426e-5, 1.367e-5,
1.312e-5, 1.263e-5, 1.218e-5, 1.178e-5, 1.143e-5, 1.112e-5,
1.088e-5, 1.07e-5, 1.057e-5, 1.05e-5, 1.051e-5, 1.059e-5,
1.076e-5, 1.1e-5, 1.133e-5, 1.18e-5, 1.237e-5, 1.308e-5,
1.393e-5, 1.483e-5, 1.614e-5, 1.758e-5, 1.93e-5, 2.123e-5,
2.346e-5, 2.647e-5, 2.93e-5, 3.279e-5, 3.745e-5, 4.152e-5,
4.813e-5, 5.477e-5, 6.203e-5, 7.331e-5, 8.056e-5, 9.882e-5,
1.05e-4, 1.21e-4, 1.341e-4, 1.572e-4, 1.698e-4, 1.968e-4,
2.175e-4, 2.431e-4, 2.735e-4, 2.867e-4, 3.19e-4, 3.371e-4,
3.554e-4, 3.726e-4, 3.837e-4, 3.878e-4, 3.864e-4, 3.858e-4,
3.841e-4, 3.852e-4, 3.815e-4, 3.762e-4, 3.618e-4, 3.579e-4,
3.45e-4, 3.202e-4, 3.018e-4, 2.785e-4, 2.602e-4, 2.416e-4,
2.097e-4, 1.939e-4, 1.689e-4, 1.498e-4, 1.308e-4, 1.17e-4,
1.011e-4, 9.237e-5, 7.909e-5, 7.006e-5, 6.112e-5, 5.401e-5,
4.914e-5, 4.266e-5, 3.963e-5, 3.316e-5, 3.037e-5, 2.598e-5,
2.294e-5, 2.066e-5, 1.813e-5, 1.583e-5, 1.423e-5, 1.247e-5,
1.116e-5, 9.76e-6, 8.596e-6, 7.72e-6, 6.825e-6, 6.108e-6,
5.366e-6, 4.733e-6, 4.229e-6, 3.731e-6, 3.346e-6, 2.972e-6,
2.628e-6, 2.356e-6, 2.102e-6, 1.878e-6, 1.678e-6, 1.507e-6,
1.348e-6, 1.21e-6, 1.089e-6, 9.806e-7, 8.857e-7, 8.004e-7,
7.261e-7, 6.599e-7, 6.005e-7, 5.479e-7, 5.011e-7, 4.595e-7,
4.219e-7, 3.885e-7, 3.583e-7, 3.314e-7, 3.071e-7, 2.852e-7,
2.654e-7, 2.474e-7, 2.311e-7, 2.162e-7, 2.026e-7, 1.902e-7,
1.788e-7, 1.683e-7, 1.587e-7, 1.497e-7, 1.415e-7, 1.338e-7,
1.266e-7, 1.2e-7, 1.138e-7, 1.08e-7, 1.027e-7, 9.764e-8,
9.296e-8, 8.862e-8, 8.458e-8, 8.087e-8, 7.744e-8, 7.429e-8,
7.145e-8, 6.893e-8, 6.664e-8, 6.468e-8, 6.322e-8, 6.162e-8,
6.07e-8, 5.992e-8, 5.913e-8, 5.841e-8, 5.796e-8, 5.757e-8,
5.746e-8, 5.731e-8, 5.679e-8, 5.577e-8, 5.671e-8, 5.656e-8,
5.594e-8, 5.593e-8, 5.602e-8, 5.62e-8, 5.693e-8, 5.725e-8,
5.858e-8, 6.037e-8, 6.249e-8, 6.535e-8, 6.899e-8, 7.356e-8,
7.918e-8, 8.618e-8, 9.385e-8, 1.039e-7, 1.158e-7, 1.29e-7,
1.437e-7, 1.65e-7, 1.871e-7, 2.121e-7, 2.427e-7, 2.773e-7,
3.247e-7, 3.677e-7, 4.037e-7, 4.776e-7, 5.101e-7, 6.214e-7,
6.936e-7, 7.581e-7, 8.486e-7, 9.355e-7, 9.942e-7, 1.063e-6,
1.123e-6, 1.191e-6, 1.215e-6, 1.247e-6, 1.26e-6, 1.271e-6,
1.284e-6, 1.317e-6, 1.323e-6, 1.349e-6, 1.353e-6, 1.362e-6,
1.344e-6, 1.329e-6, 1.336e-6, 1.327e-6, 1.325e-6, 1.359e-6,
1.374e-6, 1.415e-6, 1.462e-6, 1.526e-6, 1.619e-6, 1.735e-6,
1.863e-6, 2.034e-6, 2.265e-6, 2.482e-6, 2.756e-6, 3.103e-6,
3.466e-6, 3.832e-6, 4.378e-6, 4.913e-6, 5.651e-6, 6.311e-6,
7.169e-6, 8.057e-6, 9.253e-6, 1.047e-5, 1.212e-5, 1.36e-5,
1.569e-5, 1.776e-5, 2.02e-5, 2.281e-5, 2.683e-5, 2.994e-5,
3.488e-5, 3.896e-5, 4.499e-5, 5.175e-5, 6.035e-5, 6.34e-5,
7.281e-5, 7.923e-5, 8.348e-5, 9.631e-5, 1.044e-4, 1.102e-4,
1.176e-4, 1.244e-4, 1.283e-4, 1.326e-4, 1.4e-4, 1.395e-4,
1.387e-4, 1.363e-4, 1.314e-4, 1.241e-4, 1.228e-4, 1.148e-4,
1.086e-4, 1.018e-4, 8.89e-5, 8.316e-5, 7.292e-5, 6.452e-5,
5.625e-5, 5.045e-5, 4.38e-5, 3.762e-5, 3.29e-5, 2.836e-5,
2.485e-5, 2.168e-5, 1.895e-5, 1.659e-5, 1.453e-5, 1.282e-5,
1.132e-5, 1.001e-5, 8.836e-6, 7.804e-6, 6.922e-6, 6.116e-6,
5.429e-6, 4.824e-6, 4.278e-6, 3.788e-6, 3.371e-6, 2.985e-6,
2.649e-6, 2.357e-6, 2.09e-6, 1.858e-6, 1.647e-6, 1.462e-6,
1.299e-6, 1.155e-6, 1.028e-6, 9.142e-7, 8.132e-7, 7.246e-7,
6.451e-7, 5.764e-7, 5.151e-7, 4.603e-7, 4.121e-7, 3.694e-7,
3.318e-7, 2.985e-7, 2.69e-7, 2.428e-7, 2.197e-7, 1.992e-7,
1.81e-7, 1.649e-7, 1.506e-7, 1.378e-7, 1.265e-7, 1.163e-7,
1.073e-7, 9.918e-8, 9.191e-8, 8.538e-8, 7.949e-8, 7.419e-8,
6.94e-8, 6.508e-8, 6.114e-8, 5.761e-8, 5.437e-8, 5.146e-8,
4.89e-8, 4.636e-8, 4.406e-8, 4.201e-8, 4.015e-8, 3.84e-8,
3.661e-8, 3.51e-8, 3.377e-8, 3.242e-8, 3.13e-8, 3.015e-8,
2.918e-8, 2.83e-8, 2.758e-8, 2.707e-8, 2.656e-8, 2.619e-8,
2.609e-8, 2.615e-8, 2.63e-8, 2.675e-8, 2.745e-8, 2.842e-8,
2.966e-8, 3.125e-8, 3.318e-8, 3.565e-8, 3.85e-8, 4.191e-8,
4.59e-8, 5.059e-8, 5.607e-8, 6.239e-8, 6.958e-8, 7.796e-8,
8.773e-8, 9.88e-8, 1.114e-7, 1.258e-7, 1.422e-7, 1.61e-7,
1.822e-7, 2.06e-7, 2.337e-7, 2.645e-7, 2.996e-7, 3.393e-7,
3.843e-7, 4.363e-7, 4.935e-7, 5.607e-7, 6.363e-7, 7.242e-7,
8.23e-7, 9.411e-7, 1.071e-6, 1.232e-6, 1.402e-6, 1.6e-6, 1.82e-6,
2.128e-6, 2.386e-6, 2.781e-6, 3.242e-6, 3.653e-6, 4.323e-6,
4.747e-6, 5.321e-6, 5.919e-6, 6.681e-6, 7.101e-6, 7.983e-6,
8.342e-6, 8.741e-6, 9.431e-6, 9.952e-6, 1.026e-5, 1.055e-5,
1.095e-5, 1.095e-5, 1.087e-5, 1.056e-5, 1.026e-5, 9.715e-6,
9.252e-6, 8.452e-6, 7.958e-6, 7.268e-6, 6.295e-6, 6.003e-6, 5e-6,
```

```
4.591e-6, 3.983e-6, 3.479e-6, 3.058e-6, 2.667e-6, 2.293e-6,
1.995e-6, 1.747e-6, 1.517e-6, 1.335e-6, 1.165e-6, 1.028e-6,
9.007e-7, 7.956e-7, 7.015e-7, 6.192e-7, 5.491e-7, 4.859e-7,
4.297e-7, 3.799e-7, 3.38e-7, 3.002e-7, 2.659e-7, 2.366e-7,
2.103e-7, 1.861e-7, 1.655e-7, 1.469e-7, 1.309e-7, 1.162e-7,
1.032e-7, 9.198e-8, 8.181e-8, 7.294e-8, 6.516e-8, 5.787e-8,
5.163e-8, 4.612e-8, 4.119e-8, 3.695e-8, 3.308e-8, 2.976e-8,
2.67e-8, 2.407e-8, 2.171e-8, 1.965e-8, 1.78e-8, 1.617e-8,
1.47e-8, 1.341e-8, 1.227e-8, 1.125e-8, 1.033e-8, 9.524e-9,
8.797e-9, 8.162e-9, 7.565e-9, 7.04e-9, 6.56e-9, 6.129e-9,
5.733e-9, 5.376e-9, 5.043e-9, 4.75e-9, 4.466e-9, 4.211e-9,
3.977e-9, 3.759e-9, 3.558e-9, 3.373e-9, 3.201e-9, 3.043e-9,
2.895e-9, 2.76e-9, 2.635e-9, 2.518e-9, 2.411e-9, 2.314e-9,
2.23e-9, 2.151e-9, 2.087e-9, 2.035e-9, 1.988e-9, 1.946e-9,
1.927e-9, 1.916e-9, 1.916e-9, 1.933e-9, 1.966e-9, 2.018e-9,
2.09e-9, 2.182e-9, 2.299e-9, 2.442e-9, 2.623e-9, 2.832e-9,
3.079e-9, 3.368e-9, 3.714e-9, 4.104e-9, 4.567e-9, 5.091e-9,
5.701e-9, 6.398e-9, 7.194e-9, 8.127e-9, 9.141e-9, 1.035e-8,
1.177e-8, 1.338e-8, 1.508e-8, 1.711e-8, 1.955e-8, 2.216e-8,
2.534e-8, 2.871e-8, 3.291e-8, 3.711e-8, 4.285e-8, 4.868e-8,
5.509e-8, 6.276e-8, 7.262e-8, 8.252e-8, 9.4e-8, 1.064e-7,
1.247e-7, 1.411e-7, 1.626e-7, 1.827e-7, 2.044e-7, 2.284e-7,
2.452e-7, 2.854e-7, 3.026e-7, 3.278e-7, 3.474e-7, 3.693e-7,
3.93e-7, 4.104e-7, 4.22e-7, 4.439e-7, 4.545e-7, 4.778e-7,
4.812e-7, 5.018e-7, 4.899e-7, 5.075e-7, 5.073e-7, 5.171e-7,
5.131e-7, 5.25e-7, 5.617e-7, 5.846e-7, 6.239e-7, 6.696e-7,
7.398e-7, 8.073e-7, 9.15e-7, 1.009e-6, 1.116e-6, 1.264e-6,
1.439e-6, 1.644e-6, 1.856e-6, 2.147e-6, 2.317e-6, 2.713e-6,
2.882e-6, 2.99e-6, 3.489e-6, 3.581e-6, 4.033e-6, 4.26e-6,
4.543e-6, 4.84e-6, 4.826e-6, 5.013e-6, 5.252e-6, 5.277e-6,
5.306e-6, 5.236e-6, 5.123e-6, 5.171e-6, 4.843e-6, 4.615e-6,
4.385e-6, 3.97e-6, 3.693e-6, 3.231e-6, 2.915e-6, 2.495e-6,
2.144e-6, 1.91e-6, 1.639e-6, 1.417e-6, 1.226e-6, 1.065e-6,
9.29e-7, 8.142e-7, 7.161e-7, 6.318e-7, 5.581e-7, 4.943e-7,
4.376e-7, 3.884e-7, 3.449e-7, 3.06e-7, 2.712e-7, 2.412e-7,
2.139e-7, 1.903e-7, 1.689e-7, 1.499e-7, 1.331e-7, 1.183e-7,
1.05e-7, 9.362e-8, 8.306e-8, 7.403e-8, 6.578e-8, 5.853e-8,
5.216e-8, 4.632e-8, 4.127e-8, 3.678e-8, 3.279e-8, 2.923e-8,
2.612e-8, 2.339e-8, 2.094e-8, 1.877e-8, 1.686e-8, 1.516e-8,
1.366e-8, 1.234e-8, 1.114e-8, 1.012e-8, 9.182e-9, 8.362e-9,
7.634e-9, 6.981e-9, 6.406e-9, 5.888e-9, 5.428e-9, 5.021e-9,
4.65e-9, 4.326e-9, 4.033e-9, 3.77e-9, 3.536e-9, 3.327e-9,
3.141e-9, 2.974e-9, 2.825e-9, 2.697e-9, 2.584e-9, 2.488e-9,
2.406e-9, 2.34e-9, 2.292e-9, 2.259e-9, 2.244e-9, 2.243e-9,
2.272e-9, 2.31e-9, 2.378e-9, 2.454e-9, 2.618e-9, 2.672e-9,
2.831e-9, 3.05e-9, 3.225e-9, 3.425e-9, 3.677e-9, 3.968e-9,
4.221e-9, 4.639e-9, 4.96e-9, 5.359e-9, 5.649e-9, 6.23e-9,
6.716e-9, 7.218e-9, 7.746e-9, 7.988e-9, 8.627e-9, 8.999e-9,
9.442e-9, 9.82e-9, 1.015e-8, 1.06e-8, 1.079e-8, 1.109e-8,
1.137e-8, 1.186e-8, 1.18e-8, 1.187e-8, 1.194e-8, 1.192e-8,
1.224e-8, 1.245e-8, 1.246e-8, 1.318e-8, 1.377e-8, 1.471e-8,
1.582e-8, 1.713e-8, 1.853e-8, 2.063e-8, 2.27e-8, 2.567e-8,
2.891e-8, 3.264e-8, 3.744e-8, 4.286e-8, 4.915e-8, 5.623e-8,
6.336e-8, 7.293e-8, 8.309e-8, 9.319e-8, 1.091e-7, 1.243e-7,
1.348e-7, 1.449e-7, 1.62e-7, 1.846e-7, 1.937e-7, 2.04e-7,
2.179e-7, 2.298e-7, 2.433e-7, 2.439e-7, 2.464e-7, 2.611e-7,
2.617e-7, 2.582e-7, 2.453e-7, 2.401e-7, 2.349e-7, 2.203e-7,
2.066e-7, 1.939e-7, 1.78e-7, 1.558e-7, 1.391e-7, 1.203e-7,
1.048e-7, 9.464e-8, 8.306e-8, 7.239e-8, 6.317e-8, 5.52e-8,
4.847e-8, 4.282e-8, 3.796e-8, 3.377e-8, 2.996e-8, 2.678e-8,
2.4e-8, 2.134e-8, 1.904e-8, 1.705e-8, 1.523e-8, 1.35e-8,
1.204e-8, 1.07e-8, 9.408e-9, 8.476e-9, 7.47e-9, 6.679e-9,
5.929e-9, 5.267e-9, 4.711e-9, 4.172e-9, 3.761e-9, 3.288e-9,
2.929e-9, 2.609e-9, 2.315e-9, 2.042e-9, 1.844e-9, 1.64e-9,
1.47e-9, 1.31e-9, 1.176e-9, 1.049e-9, 9.377e-10, 8.462e-10,
7.616e-10, 6.854e-10, 6.191e-10, 5.596e-10, 5.078e-10, 4.611e-10,
4.197e-10, 3.83e-10, 3.505e-10, 3.215e-10, 2.956e-10, 2.726e-10,
2.521e-10, 2.338e-10, 2.173e-10, 2.026e-10, 1.895e-10, 1.777e-10,
1.672e-10, 1.579e-10, 1.496e-10, 1.423e-10, 1.358e-10, 1.302e-10,
1.254e-10, 1.216e-10, 1.187e-10, 1.163e-10, 1.147e-10, 1.145e-10,
1.15e-10, 1.17e-10, 1.192e-10, 1.25e-10, 1.298e-10, 1.345e-10,
1.405e-10, 1.538e-10, 1.648e-10, 1.721e-10, 1.872e-10, 1.968e-10,
2.089e-10, 2.172e-10, 2.317e-10, 2.389e-10, 2.503e-10, 2.585e-10,
2.686e-10, 2.8e-10, 2.895e-10, 3.019e-10, 3.037e-10, 3.076e-10,
3.146e-10, 3.198e-10, 3.332e-10, 3.397e-10, 3.54e-10, 3.667e-10,
3.895e-10, 4.071e-10, 4.565e-10, 4.983e-10, 5.439e-10, 5.968e-10,
6.676e-10, 7.456e-10, 8.405e-10, 9.478e-10, 1.064e-9, 1.218e-9,
1.386e-9, 1.581e-9, 1.787e-9, 2.032e-9, 2.347e-9, 2.677e-9,
3.008e-9, 3.544e-9, 4.056e-9, 4.687e-9, 5.331e-9, 6.227e-9,
6.854e-9, 8.139e-9, 8.945e-9, 9.865e-9, 1.125e-8, 1.178e-8,
1.364e-8, 1.436e-8, 1.54e-8, 1.672e-8, 1.793e-8, 1.906e-8,
2.036e-8, 2.144e-8, 2.292e-8, 2.371e-8, 2.493e-8, 2.606e-8,
2.706e-8, 2.866e-8, 3.036e-8, 3.136e-8, 3.405e-8, 3.665e-8,
3.837e-8, 4.229e-8, 4.748e-8, 5.32e-8, 5.763e-8, 6.677e-8,
7.216e-8, 7.716e-8, 8.958e-8, 9.419e-8, 1.036e-7, 1.108e-7,
1.189e-7, 1.246e-7, 1.348e-7, 1.31e-7, 1.361e-7, 1.364e-7,
1.363e-7, 1.343e-7, 1.293e-7, 1.254e-7, 1.235e-7, 1.158e-7,
```

```
1.107e-7, 9.961e-8, 9.011e-8, 7.91e-8, 6.916e-8, 6.338e-8,
5.564e-8, 4.827e-8, 4.198e-8, 3.695e-8, 3.276e-8, 2.929e-8,
2.633e-8, 2.391e-8, 2.192e-8, 2.021e-8, 1.89e-8, 1.772e-8,
1.667e-8, 1.603e-8, 1.547e-8, 1.537e-8, 1.492e-8, 1.515e-8,
1.479e-8, 1.45e-8, 1.513e-8, 1.495e-8, 1.529e-8, 1.565e-8,
1.564e-8, 1.553e-8, 1.569e-8, 1.584e-8, 1.57e-8, 1.538e-8,
1.513e-8, 1.472e-8, 1.425e-8, 1.349e-8, 1.328e-8, 1.249e-8,
1.17e-8, 1.077e-8, 9.514e-9, 8.614e-9, 7.46e-9, 6.621e-9,
5.775e-9, 5.006e-9, 4.308e-9, 3.747e-9, 3.24e-9, 2.84e-9,
2.481e-9, 2.184e-9, 1.923e-9, 1.71e-9, 1.504e-9, 1.334e-9,
1.187e-9, 1.053e-9, 9.367e-10, 8.306e-10, 7.419e-10, 6.63e-10,
5.918e-10, 5.277e-10, 4.717e-10, 4.222e-10, 3.783e-10, 3.39e-10,
3.036e-10, 2.729e-10, 2.455e-10, 2.211e-10, 1.995e-10, 1.804e-10,
1.635e-10, 1.485e-10, 1.355e-10, 1.24e-10, 1.139e-10, 1.051e-10,
9.757e-11, 9.114e-11, 8.577e-11, 8.139e-11, 7.792e-11, 7.52e-11,
7.39e-11, 7.311e-11, 7.277e-11, 7.482e-11, 7.698e-11, 8.162e-11,
8.517e-11, 8.968e-11, 9.905e-11, 1.075e-10, 1.187e-10, 1.291e-10,
1.426e-10, 1.573e-10, 1.734e-10, 1.905e-10, 2.097e-10, 2.28e-10,
2.473e-10, 2.718e-10, 2.922e-10, 3.128e-10, 3.361e-10, 3.641e-10,
3.91e-10, 4.196e-10, 4.501e-10, 4.932e-10, 5.258e-10, 5.755e-10,
6.253e-10, 6.664e-10, 7.344e-10, 7.985e-10, 8.877e-10, 1.005e-9,
1.118e-9, 1.251e-9, 1.428e-9, 1.61e-9, 1.888e-9, 2.077e-9,
2.331e-9, 2.751e-9, 3.061e-9, 3.522e-9, 3.805e-9, 4.181e-9,
4.575e-9, 5.167e-9, 5.634e-9, 6.007e-9, 6.501e-9, 6.829e-9,
7.211e-9, 7.262e-9, 7.696e-9, 7.832e-9, 7.799e-9, 7.651e-9,
7.304e-9, 7.15e-9, 6.977e-9, 6.603e-9, 6.209e-9, 5.69e-9,
5.432e-9, 4.764e-9, 4.189e-9, 3.64e-9, 3.203e-9, 2.848e-9,
2.51e-9, 2.194e-9, 1.946e-9, 1.75e-9, 1.567e-9, 1.426e-9,
1.302e-9, 1.197e-9, 1.109e-9, 1.035e-9, 9.719e-10, 9.207e-10,
8.957e-10, 8.578e-10, 8.262e-10, 8.117e-10, 7.987e-10, 7.875e-10,
7.741e-10, 7.762e-10, 7.537e-10, 7.424e-10, 7.474e-10, 7.294e-10,
7.216e-10, 7.233e-10, 7.075e-10, 6.892e-10, 6.618e-10, 6.314e-10,
6.208e-10, 5.689e-10, 5.55e-10, 4.984e-10, 4.6e-10, 4.078e-10,
3.879e-10, 3.459e-10, 2.982e-10, 2.626e-10, 2.329e-10, 1.988e-10,
1.735e-10, 1.487e-10, 1.297e-10, 1.133e-10, 9.943e-11, 8.736e-11,
7.726e-11, 6.836e-11, 6.053e-11, 5.384e-11, 4.789e-11, 4.267e-11,
3.804e-11, 3.398e-11, 3.034e-11, 2.71e-11, 2.425e-11, 2.173e-11,
1.95e-11, 1.752e-11, 1.574e-11, 1.418e-11, 1.278e-11, 1.154e-11,
1.044e-11, 9.463e-12, 8.602e-12, 7.841e-12, 7.171e-12, 6.584e-12,
6.073e-12, 5.631e-12, 5.254e-12, 4.937e-12, 4.679e-12, 4.476e-12,
4.328e-12, 4.233e-12, 4.194e-12, 4.211e-12, 4.286e-12, 4.424e-12,
4.628e-12, 4.906e-12, 5.262e-12, 5.708e-12, 6.254e-12, 6.914e-12,
7.714e-12, 8.677e-12, 9.747e-12, 1.101e-11, 1.256e-11, 1.409e-11,
1.597e-11, 1.807e-11, 2.034e-11, 2.316e-11, 2.622e-11, 2.962e-11,
3.369e-11, 3.819e-11, 4.329e-11, 4.932e-11, 5.589e-11, 6.364e-11,
7.284e-11, 8.236e-11, 9.447e-11, 1.078e-10, 1.229e-10, 1.417e-10,
1.614e-10, 1.843e-10, 2.107e-10, 2.406e-10, 2.728e-10, 3.195e-10,
3.595e-10, 4.153e-10, 4.736e-10, 5.41e-10, 6.088e-10, 6.769e-10,
7.691e-10, 8.545e-10, 9.621e-10, 1.047e-9, 1.161e-9, 1.296e-9,
1.424e-9, 1.576e-9, 1.739e-9, 1.893e-9, 2.08e-9, 2.336e-9,
2.604e-9, 2.76e-9, 3.001e-9, 3.365e-9, 3.55e-9, 3.895e-9,
4.183e-9, 4.614e-9, 4.846e-9, 5.068e-9, 5.427e-9, 5.541e-9,
5.864e-9, 5.997e-9, 5.997e-9, 6.061e-9, 5.944e-9, 5.855e-9,
5.661e-9, 5.523e-9, 5.374e-9, 4.94e-9, 4.688e-9, 4.17e-9,
3.913e-9, 3.423e-9, 2.997e-9, 2.598e-9, 2.253e-9, 1.946e-9,
1.71e-9, 1.507e-9, 1.336e-9, 1.19e-9, 1.068e-9, 9.623e-10,
8.772e-10, 8.007e-10, 7.42e-10, 6.884e-10, 6.483e-10, 6.162e-10,
5.922e-10, 5.688e-10, 5.654e-10, 5.637e-10, 5.701e-10, 5.781e-10,
5.874e-10, 6.268e-10, 6.357e-10, 6.525e-10, 7.137e-10, 7.441e-10,
8.024e-10, 8.485e-10, 9.143e-10, 9.536e-10, 9.717e-10, 1.018e-9,
1.042e-9, 1.054e-9, 1.092e-9, 1.079e-9, 1.064e-9, 1.043e-9,
1.02e-9, 9.687e-10, 9.273e-10, 9.208e-10, 9.068e-10, 7.687e-10,
7.385e-10, 6.595e-10, 5.87e-10, 5.144e-10, 4.417e-10, 3.804e-10,
3.301e-10, 2.866e-10, 2.509e-10, 2.202e-10, 1.947e-10, 1.719e-10,
1.525e-10, 1.361e-10, 1.21e-10, 1.084e-10, 9.8e-11, 8.801e-11,
7.954e-11, 7.124e-11, 6.335e-11, 5.76e-11, 5.132e-11, 4.601e-11,
4.096e-11, 3.657e-11, 3.25e-11, 2.909e-11, 2.587e-11, 2.297e-11,
2.05e-11, 1.828e-11, 1.632e-11, 1.462e-11, 1.314e-11, 1.185e-11,
1.073e-11, 9.76e-12, 8.922e-12, 8.206e-12, 7.602e-12, 7.1e-12,
6.694e-12, 6.378e-12, 6.149e-12, 6.004e-12, 5.941e-12, 5.962e-12,
6.069e-12, 6.265e-12, 6.551e-12, 6.935e-12, 7.457e-12, 8.074e-12,
8.811e-12, 9.852e-12, 1.086e-11, 1.207e-11, 1.361e-11, 1.553e-11,
1.737e-11, 1.93e-11, 2.175e-11, 2.41e-11, 2.706e-11, 3.023e-11,
3.313e-11, 3.657e-11, 4.118e-11, 4.569e-11, 5.025e-11, 5.66e-11,
6.231e-11, 6.881e-11, 7.996e-11, 8.526e-11, 9.694e-11, 1.106e-10,
1.222e-10, 1.355e-10, 1.525e-10, 1.775e-10, 1.924e-10, 2.181e-10,
2.379e-10, 2.662e-10, 2.907e-10, 3.154e-10, 3.366e-10, 3.579e-10,
3.858e-10, 4.046e-10, 4.196e-10, 4.166e-10, 4.457e-10, 4.466e-10,
4.404e-10, 4.337e-10, 4.15e-10, 4.083e-10, 3.91e-10, 3.723e-10,
3.514e-10, 3.303e-10, 2.847e-10, 2.546e-10, 2.23e-10, 1.994e-10,
1.733e-10, 1.488e-10, 1.297e-10, 1.144e-10, 1.004e-10, 8.741e-11,
7.928e-11, 7.034e-11, 6.323e-11, 5.754e-11, 5.25e-11, 4.85e-11,
4.502e-11, 4.286e-11, 4.028e-11, 3.899e-11, 3.824e-11, 3.761e-11,
3.804e-11, 3.839e-11, 3.845e-11, 4.244e-11, 4.382e-11, 4.582e-11,
4.847e-11, 5.209e-11, 5.384e-11, 5.887e-11, 6.371e-11, 6.737e-11,
7.168e-11, 7.415e-11, 7.827e-11, 8.037e-11, 8.12e-11, 8.071e-11,
8.008e-11, 7.851e-11, 7.544e-11, 7.377e-11, 7.173e-11, 6.801e-11,
```

```
  6.267e-11, 5.727e-11, 5.288e-11, 4.853e-11, 4.082e-11, 3.645e-11,
  3.136e-11, 2.672e-11, 2.304e-11, 1.986e-11, 1.725e-11, 1.503e-11,
  1.315e-11, 1.153e-11, 1.014e-11, 8.942e-12, 7.901e-12, 6.993e-12,
  6.199e-12, 5.502e-12, 4.89e-12, 4.351e-12, 3.878e-12, 3.461e-12,
  3.094e-12, 2.771e-12, 2.488e-12, 2.241e-12, 2.025e-12, 1.838e-12,
  1.677e-12, 1.541e-12, 1.427e-12, 1.335e-12, 1.262e-12, 1.209e-12,
  1.176e-12, 1.161e-12, 1.165e-12, 1.189e-12, 1.234e-12, 1.3e-12,
  1.389e-12, 1.503e-12, 1.644e-12, 1.814e-12, 2.017e-12, 2.255e-12,
  2.534e-12, 2.858e-12, 3.231e-12, 3.661e-12, 4.153e-12, 4.717e-12,
  5.36e-12, 6.094e-12, 6.93e-12, 7.882e-12, 8.966e-12, 1.02e-11,
  1.162e-11, 1.324e-11, 1.51e-11, 1.72e-11, 1.965e-11, 2.237e-11,
  2.56e-11, 2.927e-11, 3.371e-11, 3.842e-11, 4.429e-11, 5.139e-11,
  5.798e-11, 6.697e-11, 7.626e-11, 8.647e-11, 1.022e-10, 1.136e-10,
  1.3e-10, 1.481e-10, 1.672e-10, 1.871e-10, 2.126e-10, 2.357e-10,
  2.583e-10, 2.997e-10, 3.289e-10, 3.702e-10, 4.012e-10, 4.319e-10,
  4.527e-10, 5.001e-10, 5.448e-10, 5.611e-10, 5.76e-10, 5.965e-10,
  6.079e-10, 6.207e-10, 6.276e-10, 6.222e-10, 6.137e-10, 6e-10,
  5.814e-10, 5.393e-10, 5.35e-10, 4.947e-10, 4.629e-10, 4.117e-10,
  3.712e-10, 3.372e-10, 2.923e-10, 2.55e-10, 2.232e-10, 1.929e-10,
  1.679e-10, 1.46e-10, 1.289e-10, 1.13e-10, 9.953e-11, 8.763e-11,
  7.76e-11, 6.9e-11, 6.16e-11, 5.525e-11, 4.958e-11, 4.489e-11,
  4.072e-11, 3.728e-11, 3.438e-11, 3.205e-11, 3.006e-11, 2.848e-11,
  2.766e-11, 2.688e-11, 2.664e-11, 2.67e-11, 2.696e-11, 2.786e-11,
  2.861e-11, 3.009e-11, 3.178e-11, 3.389e-11, 3.587e-11, 3.819e-11,
  4.054e-11, 4.417e-11, 4.703e-11, 5.137e-11, 5.46e-11, 6.055e-11,
  6.333e-11, 6.773e-11, 7.219e-11, 7.717e-11, 8.131e-11, 8.491e-11,
  8.574e-11, 9.01e-11, 9.017e-11, 8.999e-11, 8.959e-11, 8.838e-11,
  8.579e-11, 8.162e-11, 8.098e-11, 7.472e-11, 7.108e-11, 6.559e-11,
  5.994e-11, 5.172e-11, 4.424e-11, 3.951e-11, 3.34e-11, 2.902e-11,
  2.541e-11, 2.215e-11, 1.945e-11, 1.716e-11, 1.503e-11, 1.339e-11,
  1.185e-11, 1.05e-11, 9.336e-12, 8.307e-12, 7.312e-12, 6.55e-12,
  5.836e-12, 5.178e-12, 4.6e-12, 4.086e-12, 3.639e-12, 3.247e-12,
  2.904e-12, 2.604e-12, 2.341e-12, 2.112e-12, 1.914e-12, 1.744e-12,
  1.598e-12, 1.476e-12, 1.374e-12, 1.293e-12, 1.23e-12, 1.185e-12,
  1.158e-12, 1.147e-12, 1.154e-12, 1.177e-12, 1.219e-12, 1.28e-12,
  1.36e-12, 1.463e-12, 1.591e-12, 1.75e-12, 1.94e-12, 2.156e-12,
  2.43e-12, 2.748e-12, 3.052e-12, 3.533e-12, 3.967e-12, 4.471e-12,
  5.041e-12, 5.86e-12, 6.664e-12, 7.522e-12, 8.342e-12, 9.412e-12,
  1.072e-11, 1.213e-11, 1.343e-11, 1.496e-11, 1.664e-11, 1.822e-11,
  2.029e-11, 2.233e-11, 2.457e-11, 2.709e-11, 2.928e-11, 3.115e-11,
  3.356e-11, 3.592e-11, 3.818e-11, 3.936e-11, 4.061e-11, 4.149e-11,
  4.299e-11, 4.223e-11, 4.251e-11, 4.287e-11, 4.177e-11, 4.094e-11,
  3.942e-11, 3.772e-11, 3.614e-11, 3.394e-11, 3.222e-11, 2.791e-11,
  2.665e-11, 2.309e-11, 2.032e-11, 1.74e-11, 1.535e-11, 1.323e-11,
  1.151e-11, 9.803e-12, 8.65e-12, 7.54e-12, 6.619e-12, 5.832e-12,
  5.113e-12, 4.503e-12, 3.975e-12, 3.52e-12, 3.112e-12, 2.797e-12,
  2.5e-12, 2.24e-12, 2.013e-12, 1.819e-12, 1.653e-12, 1.513e-12,
  1.395e-12, 1.299e-12, 1.225e-12, 1.168e-12, 1.124e-12, 1.148e-12,
  1.107e-12, 1.128e-12, 1.169e-12, 1.233e-12, 1.307e-12, 1.359e-12,
  1.543e-12, 1.686e-12, 1.794e-12, 2.028e-12, 2.21e-12, 2.441e-12,
  2.653e-12, 2.828e-12, 3.093e-12, 3.28e-12, 3.551e-12, 3.677e-12,
  3.803e-12, 3.844e-12, 4.068e-12, 4.093e-12, 4.002e-12, 3.904e-12,
  3.624e-12, 3.633e-12, 3.622e-12, 3.443e-12, 3.184e-12, 2.934e-12,
  2.476e-12, 2.212e-12, 1.867e-12, 1.594e-12, 1.37e-12, 1.192e-12,
  1.045e-12, 9.211e-13, 8.17e-13, 7.29e-13, 6.55e-13, 5.929e-13,
  5.415e-13, 4.995e-13, 4.661e-13, 4.406e-13, 4.225e-13, 4.116e-13,
  4.075e-13, 4.102e-13, 4.198e-13, 4.365e-13, 4.606e-13, 4.925e-13,
  5.326e-13, 5.818e-13, 6.407e-13, 7.104e-13, 7.92e-13, 8.868e-13,
  9.964e-13, 1.123e-12, 1.268e-12, 1.434e-12, 1.626e-12, 1.848e-12,
  2.107e-12, 2.422e-12, 2.772e-12, 3.145e-12, 3.704e-12, 4.27e-12,
  4.721e-12, 5.361e-12, 6.083e-12, 7.095e-12, 7.968e-12, 9.228e-12,
  1.048e-11, 1.187e-11, 1.336e-11, 1.577e-11, 1.772e-11, 2.017e-11,
  2.25e-11, 2.63e-11, 2.911e-11, 3.356e-11, 3.82e-11, 4.173e-11,
  4.811e-11, 5.254e-11, 5.839e-11, 6.187e-11, 6.805e-11, 7.118e-11,
  7.369e-11, 7.664e-11, 7.794e-11, 7.947e-11, 8.036e-11, 7.954e-11,
  7.849e-11, 7.518e-11, 7.462e-11, 6.926e-11, 6.531e-11, 6.197e-11,
  5.421e-11, 4.777e-11, 4.111e-11, 3.679e-11, 3.166e-11, 2.786e-11,
  2.436e-11, 2.144e-11, 1.859e-11, 1.628e-11, 1.414e-11, 1.237e-11,
  1.093e-11, 9.558e-12
};

static double h2o260[2001] = { .2752, .2732, .2749, .2676, .2667, .2545,
  .2497, .2327, .2218, .2036, .1825, .1694, .1497, .1353, .121,
  .1014, .09405, .07848, .07195, .06246, .05306, .04853, .04138,
  .03735, .03171, .02785, .02431, .02111, .01845, .0164, .01405,
  .01255, .01098, .009797, .008646, .007779, .006898, .006099,
  .005453, .004909, .004413, .003959, .003581, .003199, .002871,
  .002583, .00233, .002086, .001874, .001684, .001512, .001361,
  .001225, .0011, 9.89e-4, 8.916e-4, 8.039e-4, 7.256e-4, 6.545e-4,
  5.918e-4, 5.359e-4, 4.867e-4, 4.426e-4, 4.033e-4, 3.682e-4,
  3.366e-4, 3.085e-4, 2.833e-4, 2.605e-4, 2.403e-4, 2.221e-4,
  2.055e-4, 1.908e-4, 1.774e-4, 1.653e-4, 1.544e-4, 1.443e-4,
  1.351e-4, 1.267e-4, 1.19e-4, 1.119e-4, 1.053e-4, 9.922e-5,
  9.355e-5, 8.831e-5, 8.339e-5, 7.878e-5, 7.449e-5, 7.043e-5,
  6.664e-5, 6.307e-5, 5.969e-5, 5.654e-5, 5.357e-5, 5.075e-5,
  4.81e-5, 4.56e-5, 4.322e-5, 4.102e-5, 3.892e-5, 3.696e-5,
  3.511e-5, 3.339e-5, 3.177e-5, 3.026e-5, 2.886e-5, 2.756e-5,
```

```
2.636e-5, 2.527e-5, 2.427e-5, 2.337e-5, 2.257e-5, 2.185e-5,
2.127e-5, 2.08e-5, 2.041e-5, 2.013e-5, 2e-5, 1.997e-5, 2.009e-5,
2.031e-5, 2.068e-5, 2.124e-5, 2.189e-5, 2.267e-5, 2.364e-5,
2.463e-5, 2.618e-5, 2.774e-5, 2.937e-5, 3.144e-5, 3.359e-5,
3.695e-5, 4.002e-5, 4.374e-5, 4.947e-5, 5.431e-5, 6.281e-5,
7.169e-5, 8.157e-5, 9.728e-5, 1.079e-4, 1.337e-4, 1.442e-4,
1.683e-4, 1.879e-4, 2.223e-4, 2.425e-4, 2.838e-4, 3.143e-4,
3.527e-4, 4.012e-4, 4.237e-4, 4.747e-4, 5.057e-4, 5.409e-4,
5.734e-4, 5.944e-4, 6.077e-4, 6.175e-4, 6.238e-4, 6.226e-4,
6.248e-4, 6.192e-4, 6.098e-4, 5.818e-4, 5.709e-4, 5.465e-4,
5.043e-4, 4.699e-4, 4.294e-4, 3.984e-4, 3.672e-4, 3.152e-4,
2.883e-4, 2.503e-4, 2.211e-4, 1.92e-4, 1.714e-4, 1.485e-4,
1.358e-4, 1.156e-4, 1.021e-4, 8.887e-5, 7.842e-5, 7.12e-5,
6.186e-5, 5.73e-5, 4.792e-5, 4.364e-5, 3.72e-5, 3.28e-5,
2.946e-5, 2.591e-5, 2.261e-5, 2.048e-5, 1.813e-5, 1.63e-5,
1.447e-5, 1.282e-5, 1.167e-5, 1.041e-5, 9.449e-6, 8.51e-6,
7.596e-6, 6.961e-6, 6.272e-6, 5.728e-6, 5.198e-6, 4.667e-6,
4.288e-6, 3.897e-6, 3.551e-6, 3.235e-6, 2.952e-6, 2.688e-6,
2.449e-6, 2.241e-6, 2.05e-6, 1.879e-6, 1.722e-6, 1.582e-6,
1.456e-6, 1.339e-6, 1.236e-6, 1.144e-6, 1.06e-6, 9.83e-7,
9.149e-7, 8.535e-7, 7.973e-7, 7.466e-7, 6.999e-7, 6.574e-7,
6.18e-7, 5.821e-7, 5.487e-7, 5.18e-7, 4.896e-7, 4.631e-7,
4.386e-7, 4.16e-7, 3.945e-7, 3.748e-7, 3.562e-7, 3.385e-7,
3.222e-7, 3.068e-7, 2.922e-7, 2.788e-7, 2.659e-7, 2.539e-7,
2.425e-7, 2.318e-7, 2.219e-7, 2.127e-7, 2.039e-7, 1.958e-7,
1.885e-7, 1.818e-7, 1.758e-7, 1.711e-7, 1.662e-7, 1.63e-7,
1.605e-7, 1.58e-7, 1.559e-7, 1.545e-7, 1.532e-7, 1.522e-7,
1.51e-7, 1.495e-7, 1.465e-7, 1.483e-7, 1.469e-7, 1.448e-7,
1.444e-7, 1.436e-7, 1.426e-7, 1.431e-7, 1.425e-7, 1.445e-7,
1.477e-7, 1.515e-7, 1.567e-7, 1.634e-7, 1.712e-7, 1.802e-7,
1.914e-7, 2.024e-7, 2.159e-7, 2.295e-7, 2.461e-7, 2.621e-7,
2.868e-7, 3.102e-7, 3.394e-7, 3.784e-7, 4.223e-7, 4.864e-7,
5.501e-7, 6.039e-7, 7.193e-7, 7.728e-7, 9.514e-7, 1.073e-6,
1.18e-6, 1.333e-6, 1.472e-6, 1.566e-6, 1.677e-6, 1.784e-6,
1.904e-6, 1.953e-6, 2.02e-6, 2.074e-6, 2.128e-6, 2.162e-6,
2.219e-6, 2.221e-6, 2.249e-6, 2.239e-6, 2.235e-6, 2.185e-6,
2.141e-6, 2.124e-6, 2.09e-6, 2.068e-6, 2.1e-6, 2.104e-6,
2.142e-6, 2.181e-6, 2.257e-6, 2.362e-6, 2.5e-6, 2.664e-6,
2.884e-6, 3.189e-6, 3.48e-6, 3.847e-6, 4.313e-6, 4.79e-6,
5.25e-6, 5.989e-6, 6.692e-6, 7.668e-6, 8.52e-6, 9.606e-6,
1.073e-5, 1.225e-5, 1.377e-5, 1.582e-5, 1.761e-5, 2.029e-5,
2.284e-5, 2.602e-5, 2.94e-5, 3.483e-5, 3.928e-5, 4.618e-5,
5.24e-5, 6.132e-5, 7.183e-5, 8.521e-5, 9.111e-5, 1.07e-4,
1.184e-4, 1.264e-4, 1.475e-4, 1.612e-4, 1.704e-4, 1.818e-4,
1.924e-4, 1.994e-4, 2.061e-4, 2.18e-4, 2.187e-4, 2.2e-4,
2.196e-4, 2.131e-4, 2.015e-4, 1.988e-4, 1.847e-4, 1.729e-4,
1.597e-4, 1.373e-4, 1.262e-4, 1.087e-4, 9.439e-5, 8.061e-5,
7.093e-5, 6.049e-5, 5.12e-5, 4.435e-5, 3.817e-5, 3.34e-5,
2.927e-5, 2.573e-5, 2.291e-5, 2.04e-5, 1.827e-5, 1.636e-5,
1.463e-5, 1.309e-5, 1.17e-5, 1.047e-5, 9.315e-6, 8.328e-6,
7.458e-6, 6.665e-6, 5.94e-6, 5.316e-6, 4.752e-6, 4.252e-6,
3.825e-6, 3.421e-6, 3.064e-6, 2.746e-6, 2.465e-6, 2.216e-6,
1.99e-6, 1.79e-6, 1.609e-6, 1.449e-6, 1.306e-6, 1.177e-6,
1.063e-6, 9.607e-7, 8.672e-7, 7.855e-7, 7.118e-7, 6.46e-7,
5.871e-7, 5.34e-7, 4.868e-7, 4.447e-7, 4.068e-7, 3.729e-7,
3.423e-7, 3.151e-7, 2.905e-7, 2.686e-7, 2.484e-7, 2.306e-7,
2.142e-7, 1.995e-7, 1.86e-7, 1.738e-7, 1.626e-7, 1.522e-7,
1.427e-7, 1.338e-7, 1.258e-7, 1.183e-7, 1.116e-7, 1.056e-7,
9.972e-8, 9.46e-8, 9.007e-8, 8.592e-8, 8.195e-8, 7.816e-8,
7.483e-8, 7.193e-8, 6.892e-8, 6.642e-8, 6.386e-8, 6.154e-8,
5.949e-8, 5.764e-8, 5.622e-8, 5.479e-8, 5.364e-8, 5.301e-8,
5.267e-8, 5.263e-8, 5.313e-8, 5.41e-8, 5.55e-8, 5.745e-8,
6.003e-8, 6.311e-8, 6.713e-8, 7.173e-8, 7.724e-8, 8.368e-8,
9.121e-8, 9.986e-8, 1.097e-7, 1.209e-7, 1.338e-7, 1.486e-7,
1.651e-7, 1.837e-7, 2.048e-7, 2.289e-7, 2.557e-7, 2.857e-7,
3.195e-7, 3.587e-7, 4.015e-7, 4.497e-7, 5.049e-7, 5.665e-7,
6.366e-7, 7.121e-7, 7.996e-7, 8.946e-7, 1.002e-6, 1.117e-6,
1.262e-6, 1.416e-6, 1.611e-6, 1.807e-6, 2.056e-6, 2.351e-6,
2.769e-6, 3.138e-6, 3.699e-6, 4.386e-6, 5.041e-6, 6.074e-6,
6.812e-6, 7.79e-6, 8.855e-6, 1.014e-5, 1.095e-5, 1.245e-5,
1.316e-5, 1.39e-5, 1.504e-5, 1.583e-5, 1.617e-5, 1.652e-5,
1.713e-5, 1.724e-5, 1.715e-5, 1.668e-5, 1.629e-5, 1.552e-5,
1.478e-5, 1.34e-5, 1.245e-5, 1.121e-5, 9.575e-6, 8.956e-6,
7.345e-6, 6.597e-6, 5.612e-6, 4.818e-6, 4.165e-6, 3.579e-6,
3.041e-6, 2.623e-6, 2.29e-6, 1.984e-6, 1.748e-6, 1.534e-6,
1.369e-6, 1.219e-6, 1.092e-6, 9.8e-7, 8.762e-7, 7.896e-7,
7.104e-7, 6.364e-7, 5.691e-7, 5.107e-7, 4.575e-7, 4.09e-7,
3.667e-7, 3.287e-7, 2.931e-7, 2.633e-7, 2.356e-7, 2.111e-7,
1.895e-7, 1.697e-7, 1.525e-7, 1.369e-7, 1.233e-7, 1.114e-7,
9.988e-8, 9.004e-8, 8.149e-8, 7.352e-8, 6.662e-8, 6.03e-8,
5.479e-8, 4.974e-8, 4.532e-8, 4.129e-8, 3.781e-8, 3.462e-8,
3.176e-8, 2.919e-8, 2.687e-8, 2.481e-8, 2.292e-8, 2.119e-8,
1.967e-8, 1.828e-8, 1.706e-8, 1.589e-8, 1.487e-8, 1.393e-8,
1.307e-8, 1.228e-8, 1.156e-8, 1.089e-8, 1.028e-8, 9.696e-9,
9.159e-9, 8.658e-9, 8.187e-9, 7.746e-9, 7.34e-9, 6.953e-9,
6.594e-9, 6.259e-9, 5.948e-9, 5.66e-9, 5.386e-9, 5.135e-9,
4.903e-9, 4.703e-9, 4.515e-9, 4.362e-9, 4.233e-9, 4.117e-9,
```

```
4.017e-9, 3.962e-9, 3.924e-9, 3.905e-9, 3.922e-9, 3.967e-9,
4.046e-9, 4.165e-9, 4.32e-9, 4.522e-9, 4.769e-9, 5.083e-9,
5.443e-9, 5.872e-9, 6.366e-9, 6.949e-9, 7.601e-9, 8.371e-9,
9.22e-9, 1.02e-8, 1.129e-8, 1.251e-8, 1.393e-8, 1.542e-8,
1.72e-8, 1.926e-8, 2.152e-8, 2.392e-8, 2.678e-8, 3.028e-8,
3.39e-8, 3.836e-8, 4.309e-8, 4.9e-8, 5.481e-8, 6.252e-8,
7.039e-8, 7.883e-8, 8.849e-8, 1.012e-7, 1.142e-7, 1.3e-7,
1.475e-7, 1.732e-7, 1.978e-7, 2.304e-7, 2.631e-7, 2.988e-7,
3.392e-7, 3.69e-7, 4.355e-7, 4.672e-7, 5.11e-7, 5.461e-7,
5.828e-7, 6.233e-7, 6.509e-7, 6.672e-7, 6.969e-7, 7.104e-7,
7.439e-7, 7.463e-7, 7.708e-7, 7.466e-7, 7.668e-7, 7.549e-7,
7.586e-7, 7.384e-7, 7.439e-7, 7.785e-7, 7.915e-7, 8.31e-7,
8.745e-7, 9.558e-7, 1.038e-6, 1.173e-6, 1.304e-6, 1.452e-6,
1.671e-6, 1.931e-6, 2.239e-6, 2.578e-6, 3.032e-6, 3.334e-6,
3.98e-6, 4.3e-6, 4.518e-6, 5.321e-6, 5.508e-6, 6.211e-6, 6.59e-6,
7.046e-6, 7.555e-6, 7.558e-6, 7.875e-6, 8.319e-6, 8.433e-6,
8.59e-6, 8.503e-6, 8.304e-6, 8.336e-6, 7.739e-6, 7.301e-6,
6.827e-6, 6.078e-6, 5.551e-6, 4.762e-6, 4.224e-6, 3.538e-6,
2.984e-6, 2.619e-6, 2.227e-6, 1.923e-6, 1.669e-6, 1.462e-6,
1.294e-6, 1.155e-6, 1.033e-6, 9.231e-7, 8.238e-7, 7.36e-7,
6.564e-7, 5.869e-7, 5.236e-7, 4.673e-7, 4.174e-7, 3.736e-7,
3.33e-7, 2.976e-7, 2.657e-7, 2.367e-7, 2.106e-7, 1.877e-7,
1.671e-7, 1.494e-7, 1.332e-7, 1.192e-7, 1.065e-7, 9.558e-8,
8.586e-8, 7.717e-8, 6.958e-8, 6.278e-8, 5.666e-8, 5.121e-8,
4.647e-8, 4.213e-8, 3.815e-8, 3.459e-8, 3.146e-8, 2.862e-8,
2.604e-8, 2.375e-8, 2.162e-8, 1.981e-8, 1.817e-8, 1.67e-8,
1.537e-8, 1.417e-8, 1.31e-8, 1.215e-8, 1.128e-8, 1.05e-8,
9.793e-9, 9.158e-9, 8.586e-9, 8.068e-9, 7.595e-9, 7.166e-9,
6.778e-9, 6.427e-9, 6.108e-9, 5.826e-9, 5.571e-9, 5.347e-9,
5.144e-9, 4.968e-9, 4.822e-9, 4.692e-9, 4.589e-9, 4.506e-9,
4.467e-9, 4.44e-9, 4.466e-9, 4.515e-9, 4.718e-9, 4.729e-9,
4.937e-9, 5.249e-9, 5.466e-9, 5.713e-9, 6.03e-9, 6.436e-9,
6.741e-9, 7.33e-9, 7.787e-9, 8.414e-9, 8.908e-9, 9.868e-9,
1.069e-8, 1.158e-8, 1.253e-8, 1.3e-8, 1.409e-8, 1.47e-8,
1.548e-8, 1.612e-8, 1.666e-8, 1.736e-8, 1.763e-8, 1.812e-8,
1.852e-8, 1.923e-8, 1.897e-8, 1.893e-8, 1.888e-8, 1.868e-8,
1.895e-8, 1.899e-8, 1.876e-8, 1.96e-8, 2.02e-8, 2.121e-8,
2.239e-8, 2.379e-8, 2.526e-8, 2.766e-8, 2.994e-8, 3.332e-8,
3.703e-8, 4.158e-8, 4.774e-8, 5.499e-8, 6.355e-8, 7.349e-8,
8.414e-8, 9.846e-8, 1.143e-7, 1.307e-7, 1.562e-7, 1.817e-7,
2.011e-7, 2.192e-7, 2.485e-7, 2.867e-7, 3.035e-7, 3.223e-7,
3.443e-7, 3.617e-7, 3.793e-7, 3.793e-7, 3.839e-7, 4.081e-7,
4.117e-7, 4.085e-7, 3.92e-7, 3.851e-7, 3.754e-7, 3.49e-7,
3.229e-7, 2.978e-7, 2.691e-7, 2.312e-7, 2.029e-7, 1.721e-7,
1.472e-7, 1.308e-7, 1.132e-7, 9.736e-8, 8.458e-8, 7.402e-8,
6.534e-8, 5.811e-8, 5.235e-8, 4.762e-8, 4.293e-8, 3.896e-8,
3.526e-8, 3.165e-8, 2.833e-8, 2.551e-8, 2.288e-8, 2.036e-8,
1.82e-8, 1.626e-8, 1.438e-8, 1.299e-8, 1.149e-8, 1.03e-8,
9.148e-9, 8.122e-9, 7.264e-9, 6.425e-9, 5.777e-9, 5.06e-9,
4.502e-9, 4.013e-9, 3.567e-9, 3.145e-9, 2.864e-9, 2.553e-9,
2.311e-9, 2.087e-9, 1.886e-9, 1.716e-9, 1.556e-9, 1.432e-9,
1.311e-9, 1.202e-9, 1.104e-9, 1.013e-9, 9.293e-10, 8.493e-10,
7.79e-10, 7.185e-10, 6.642e-10, 6.141e-10, 5.684e-10, 5.346e-10,
5.032e-10, 4.725e-10, 4.439e-10, 4.176e-10, 3.93e-10, 3.714e-10,
3.515e-10, 3.332e-10, 3.167e-10, 3.02e-10, 2.887e-10, 2.769e-10,
2.665e-10, 2.578e-10, 2.503e-10, 2.436e-10, 2.377e-10, 2.342e-10,
2.305e-10, 2.296e-10, 2.278e-10, 2.321e-10, 2.355e-10, 2.402e-10,
2.478e-10, 2.67e-10, 2.848e-10, 2.982e-10, 3.263e-10, 3.438e-10,
3.649e-10, 3.829e-10, 4.115e-10, 4.264e-10, 4.473e-10, 4.63e-10,
4.808e-10, 4.995e-10, 5.142e-10, 5.313e-10, 5.318e-10, 5.358e-10,
5.452e-10, 5.507e-10, 5.698e-10, 5.782e-10, 5.983e-10, 6.164e-10,
6.532e-10, 6.811e-10, 7.624e-10, 8.302e-10, 9.067e-10, 9.937e-10,
1.104e-9, 1.221e-9, 1.361e-9, 1.516e-9, 1.675e-9, 1.883e-9,
2.101e-9, 2.349e-9, 2.614e-9, 2.92e-9, 3.305e-9, 3.724e-9,
4.142e-9, 4.887e-9, 5.614e-9, 6.506e-9, 7.463e-9, 8.817e-9,
9.849e-9, 1.187e-8, 1.321e-8, 1.474e-8, 1.698e-8, 1.794e-8,
2.09e-8, 2.211e-8, 2.362e-8, 2.556e-8, 2.729e-8, 2.88e-8,
3.046e-8, 3.167e-8, 3.367e-8, 3.457e-8, 3.59e-8, 3.711e-8,
3.826e-8, 4.001e-8, 4.211e-8, 4.315e-8, 4.661e-8, 5.01e-8,
5.249e-8, 5.84e-8, 6.628e-8, 7.512e-8, 8.253e-8, 9.722e-8,
1.067e-7, 1.153e-7, 1.347e-7, 1.428e-7, 1.577e-7, 1.694e-7,
1.833e-7, 1.938e-7, 2.108e-7, 2.059e-7, 2.157e-7, 2.185e-7,
2.208e-7, 2.182e-7, 2.093e-7, 2.014e-7, 1.962e-7, 1.819e-7,
1.713e-7, 1.51e-7, 1.34e-7, 1.154e-7, 9.89e-8, 8.88e-8, 7.673e-8,
6.599e-8, 5.73e-8, 5.081e-8, 4.567e-8, 4.147e-8, 3.773e-8,
3.46e-8, 3.194e-8, 2.953e-8, 2.759e-8, 2.594e-8, 2.442e-8,
2.355e-8, 2.283e-8, 2.279e-8, 2.231e-8, 2.279e-8, 2.239e-8,
2.21e-8, 2.309e-8, 2.293e-8, 2.352e-8, 2.415e-8, 2.43e-8,
2.426e-8, 2.465e-8, 2.5e-8, 2.496e-8, 2.465e-8, 2.445e-8,
2.383e-8, 2.299e-8, 2.165e-8, 2.113e-8, 1.968e-8, 1.819e-8,
1.644e-8, 1.427e-8, 1.27e-8, 1.082e-8, 9.428e-9, 8.091e-9,
6.958e-9, 5.988e-9, 5.246e-9, 4.601e-9, 4.098e-9, 3.664e-9,
3.287e-9, 2.942e-9, 2.656e-9, 2.364e-9, 2.118e-9, 1.903e-9,
1.703e-9, 1.525e-9, 1.365e-9, 1.229e-9, 1.107e-9, 9.96e-10,
8.945e-10, 8.08e-10, 7.308e-10, 6.616e-10, 5.994e-10, 5.422e-10,
4.929e-10, 4.478e-10, 4.07e-10, 3.707e-10, 3.379e-10, 3.087e-10,
2.823e-10, 2.592e-10, 2.385e-10, 2.201e-10, 2.038e-10, 1.897e-10,
```

```
1.774e-10, 1.667e-10, 1.577e-10, 1.502e-10, 1.437e-10, 1.394e-10,
1.358e-10, 1.324e-10, 1.329e-10, 1.324e-10, 1.36e-10, 1.39e-10,
1.424e-10, 1.544e-10, 1.651e-10, 1.817e-10, 1.984e-10, 2.195e-10,
2.438e-10, 2.7e-10, 2.991e-10, 3.322e-10, 3.632e-10, 3.957e-10,
4.36e-10, 4.701e-10, 5.03e-10, 5.381e-10, 5.793e-10, 6.19e-10,
6.596e-10, 7.004e-10, 7.561e-10, 7.934e-10, 8.552e-10, 9.142e-10,
9.57e-10, 1.027e-9, 1.097e-9, 1.193e-9, 1.334e-9, 1.47e-9,
1.636e-9, 1.871e-9, 2.122e-9, 2.519e-9, 2.806e-9, 3.203e-9,
3.846e-9, 4.362e-9, 5.114e-9, 5.643e-9, 6.305e-9, 6.981e-9,
7.983e-9, 8.783e-9, 9.419e-9, 1.017e-8, 1.063e-8, 1.121e-8,
1.13e-8, 1.201e-8, 1.225e-8, 1.232e-8, 1.223e-8, 1.177e-8,
1.151e-8, 1.116e-8, 1.047e-8, 9.698e-9, 8.734e-9, 8.202e-9,
7.041e-9, 6.074e-9, 5.172e-9, 4.468e-9, 3.913e-9, 3.414e-9,
2.975e-9, 2.65e-9, 2.406e-9, 2.173e-9, 2.009e-9, 1.861e-9,
1.727e-9, 1.612e-9, 1.514e-9, 1.43e-9, 1.362e-9, 1.333e-9,
1.288e-9, 1.249e-9, 1.238e-9, 1.228e-9, 1.217e-9, 1.202e-9,
1.209e-9, 1.177e-9, 1.157e-9, 1.165e-9, 1.142e-9, 1.131e-9,
1.138e-9, 1.117e-9, 1.1e-9, 1.069e-9, 1.023e-9, 1.005e-9,
9.159e-10, 8.863e-10, 7.865e-10, 7.153e-10, 6.247e-10, 5.846e-10,
5.133e-10, 4.36e-10, 3.789e-10, 3.335e-10, 2.833e-10, 2.483e-10,
2.155e-10, 1.918e-10, 1.709e-10, 1.529e-10, 1.374e-10, 1.235e-10,
1.108e-10, 9.933e-11, 8.932e-11, 8.022e-11, 7.224e-11, 6.52e-11,
5.896e-11, 5.328e-11, 4.813e-11, 4.365e-11, 3.961e-11, 3.594e-11,
3.266e-11, 2.967e-11, 2.701e-11, 2.464e-11, 2.248e-11, 2.054e-11,
1.878e-11, 1.721e-11, 1.579e-11, 1.453e-11, 1.341e-11, 1.241e-11,
1.154e-11, 1.078e-11, 1.014e-11, 9.601e-12, 9.167e-12, 8.838e-12,
8.614e-12, 8.493e-12, 8.481e-12, 8.581e-12, 8.795e-12, 9.131e-12,
9.601e-12, 1.021e-11, 1.097e-11, 1.191e-11, 1.303e-11, 1.439e-11,
1.601e-11, 1.778e-11, 1.984e-11, 2.234e-11, 2.474e-11, 2.766e-11,
3.085e-11, 3.415e-11, 3.821e-11, 4.261e-11, 4.748e-11, 5.323e-11,
5.935e-11, 6.619e-11, 7.418e-11, 8.294e-11, 9.26e-11, 1.039e-10,
1.156e-10, 1.297e-10, 1.46e-10, 1.641e-10, 1.858e-10, 2.1e-10,
2.383e-10, 2.724e-10, 3.116e-10, 3.538e-10, 4.173e-10, 4.727e-10,
5.503e-10, 6.337e-10, 7.32e-10, 8.298e-10, 9.328e-10, 1.059e-9,
1.176e-9, 1.328e-9, 1.445e-9, 1.593e-9, 1.77e-9, 1.954e-9,
2.175e-9, 2.405e-9, 2.622e-9, 2.906e-9, 3.294e-9, 3.713e-9,
3.98e-9, 4.384e-9, 4.987e-9, 5.311e-9, 5.874e-9, 6.337e-9,
7.027e-9, 7.39e-9, 7.769e-9, 8.374e-9, 8.605e-9, 9.165e-9,
9.415e-9, 9.511e-9, 9.704e-9, 9.588e-9, 9.45e-9, 9.086e-9,
8.798e-9, 8.469e-9, 7.697e-9, 7.168e-9, 6.255e-9, 5.772e-9,
4.97e-9, 4.271e-9, 3.653e-9, 3.154e-9, 2.742e-9, 2.435e-9,
2.166e-9, 1.936e-9, 1.731e-9, 1.556e-9, 1.399e-9, 1.272e-9,
1.157e-9, 1.066e-9, 9.844e-10, 9.258e-10, 8.787e-10, 8.421e-10,
8.083e-10, 8.046e-10, 8.067e-10, 8.181e-10, 8.325e-10, 8.517e-10,
9.151e-10, 9.351e-10, 9.677e-10, 1.071e-9, 1.126e-9, 1.219e-9,
1.297e-9, 1.408e-9, 1.476e-9, 1.517e-9, 1.6e-9, 1.649e-9,
1.678e-9, 1.746e-9, 1.742e-9, 1.728e-9, 1.699e-9, 1.655e-9,
1.561e-9, 1.48e-9, 1.451e-9, 1.411e-9, 1.171e-9, 1.106e-9,
9.714e-10, 8.523e-10, 7.346e-10, 6.241e-10, 5.371e-10, 4.704e-10,
4.144e-10, 3.683e-10, 3.292e-10, 2.942e-10, 2.62e-10, 2.341e-10,
2.104e-10, 1.884e-10, 1.7e-10, 1.546e-10, 1.394e-10, 1.265e-10,
1.14e-10, 1.019e-10, 9.279e-11, 8.283e-11, 7.458e-11, 6.668e-11,
5.976e-11, 5.33e-11, 4.794e-11, 4.289e-11, 3.841e-11, 3.467e-11,
3.13e-11, 2.832e-11, 2.582e-11, 2.356e-11, 2.152e-11, 1.97e-11,
1.808e-11, 1.664e-11, 1.539e-11, 1.434e-11, 1.344e-11, 1.269e-11,
1.209e-11, 1.162e-11, 1.129e-11, 1.108e-11, 1.099e-11, 1.103e-11,
1.119e-11, 1.148e-11, 1.193e-11, 1.252e-11, 1.329e-11, 1.421e-11,
1.555e-11, 1.685e-11, 1.839e-11, 2.054e-11, 2.317e-11, 2.571e-11,
2.839e-11, 3.171e-11, 3.49e-11, 3.886e-11, 4.287e-11, 4.645e-11,
5.047e-11, 5.592e-11, 6.109e-11, 6.628e-11, 7.381e-11, 8.088e-11,
8.966e-11, 1.045e-10, 1.12e-10, 1.287e-10, 1.486e-10, 1.662e-10,
1.866e-10, 2.133e-10, 2.524e-10, 2.776e-10, 3.204e-10, 3.559e-10,
4.028e-10, 4.448e-10, 4.882e-10, 5.244e-10, 5.605e-10, 6.018e-10,
6.328e-10, 6.579e-10, 6.541e-10, 7.024e-10, 7.074e-10, 7.068e-10,
7.009e-10, 6.698e-10, 6.545e-10, 6.209e-10, 5.834e-10, 5.412e-10,
5.001e-10, 4.231e-10, 3.727e-10, 3.211e-10, 2.833e-10, 2.447e-10,
2.097e-10, 1.843e-10, 1.639e-10, 1.449e-10, 1.27e-10, 1.161e-10,
1.033e-10, 9.282e-11, 8.407e-11, 7.639e-11, 7.023e-11, 6.474e-11,
6.142e-11, 5.76e-11, 5.568e-11, 5.472e-11, 5.39e-11, 5.455e-11,
5.54e-11, 5.587e-11, 6.23e-11, 6.49e-11, 6.868e-11, 7.382e-11,
8.022e-11, 8.372e-11, 9.243e-11, 1.004e-10, 1.062e-10, 1.13e-10,
1.176e-10, 1.244e-10, 1.279e-10, 1.298e-10, 1.302e-10, 1.312e-10,
1.295e-10, 1.244e-10, 1.211e-10, 1.167e-10, 1.098e-10, 9.927e-11,
8.854e-11, 8.011e-11, 7.182e-11, 5.923e-11, 5.212e-11, 4.453e-11,
3.832e-11, 3.371e-11, 2.987e-11, 2.651e-11, 2.354e-11, 2.093e-11,
1.863e-11, 1.662e-11, 1.486e-11, 1.331e-11, 1.193e-11, 1.071e-11,
9.628e-12, 8.66e-12, 7.801e-12, 7.031e-12, 6.347e-12, 5.733e-12,
5.182e-12, 4.695e-12, 4.26e-12, 3.874e-12, 3.533e-12, 3.235e-12,
2.979e-12, 2.76e-12, 2.579e-12, 2.432e-12, 2.321e-12, 2.246e-12,
2.205e-12, 2.196e-12, 2.223e-12, 2.288e-12, 2.387e-12, 2.525e-12,
2.704e-12, 2.925e-12, 3.191e-12, 3.508e-12, 3.876e-12, 4.303e-12,
4.793e-12, 5.347e-12, 5.978e-12, 6.682e-12, 7.467e-12, 8.34e-12,
9.293e-12, 1.035e-11, 1.152e-11, 1.285e-11, 1.428e-11, 1.586e-11,
1.764e-11, 1.972e-11, 2.214e-11, 2.478e-11, 2.776e-11, 3.151e-11,
3.591e-11, 4.103e-11, 4.66e-11, 5.395e-11, 6.306e-11, 7.172e-11,
8.358e-11, 9.67e-11, 1.11e-10, 1.325e-10, 1.494e-10, 1.736e-10,
2.007e-10, 2.296e-10, 2.608e-10, 3.004e-10, 3.361e-10, 3.727e-10,
```

```
  4.373e-10, 4.838e-10, 5.483e-10, 6.006e-10, 6.535e-10, 6.899e-10,
  7.687e-10, 8.444e-10, 8.798e-10, 9.135e-10, 9.532e-10, 9.757e-10,
  9.968e-10, 1.006e-9, 9.949e-10, 9.789e-10, 9.564e-10, 9.215e-10,
  8.51e-10, 8.394e-10, 7.707e-10, 7.152e-10, 6.274e-10, 5.598e-10,
  5.028e-10, 4.3e-10, 3.71e-10, 3.245e-10, 2.809e-10, 2.461e-10,
  2.154e-10, 1.91e-10, 1.685e-10, 1.487e-10, 1.313e-10, 1.163e-10,
  1.031e-10, 9.172e-11, 8.221e-11, 7.382e-11, 6.693e-11, 6.079e-11,
  5.581e-11, 5.167e-11, 4.811e-11, 4.506e-11, 4.255e-11, 4.083e-11,
  3.949e-11, 3.881e-11, 3.861e-11, 3.858e-11, 3.951e-11, 4.045e-11,
  4.24e-11, 4.487e-11, 4.806e-11, 5.133e-11, 5.518e-11, 5.919e-11,
  6.533e-11, 7.031e-11, 7.762e-11, 8.305e-11, 9.252e-11, 9.727e-11,
  1.045e-10, 1.117e-10, 1.2e-10, 1.275e-10, 1.341e-10, 1.362e-10,
  1.438e-10, 1.45e-10, 1.455e-10, 1.455e-10, 1.434e-10, 1.381e-10,
  1.301e-10, 1.276e-10, 1.163e-10, 1.089e-10, 9.911e-11, 8.943e-11,
  7.618e-11, 6.424e-11, 5.717e-11, 4.866e-11, 4.257e-11, 3.773e-11,
  3.331e-11, 2.958e-11, 2.629e-11, 2.316e-11, 2.073e-11, 1.841e-11,
  1.635e-11, 1.464e-11, 1.31e-11, 1.16e-11, 1.047e-11, 9.408e-12,
  8.414e-12, 7.521e-12, 6.705e-12, 5.993e-12, 5.371e-12, 4.815e-12,
  4.338e-12, 3.921e-12, 3.567e-12, 3.265e-12, 3.01e-12, 2.795e-12,
  2.613e-12, 2.464e-12, 2.346e-12, 2.256e-12, 2.195e-12, 2.165e-12,
  2.166e-12, 2.198e-12, 2.262e-12, 2.364e-12, 2.502e-12, 2.682e-12,
  2.908e-12, 3.187e-12, 3.533e-12, 3.946e-12, 4.418e-12, 5.013e-12,
  5.708e-12, 6.379e-12, 7.43e-12, 8.39e-12, 9.51e-12, 1.078e-11,
  1.259e-11, 1.438e-11, 1.63e-11, 1.814e-11, 2.055e-11, 2.348e-11,
  2.664e-11, 2.956e-11, 3.3e-11, 3.677e-11, 4.032e-11, 4.494e-11,
  4.951e-11, 5.452e-11, 6.014e-11, 6.5e-11, 6.915e-11, 7.45e-11,
  7.971e-11, 8.468e-11, 8.726e-11, 8.995e-11, 9.182e-11, 9.509e-11,
  9.333e-11, 9.386e-11, 9.457e-11, 9.21e-11, 9.019e-11, 8.68e-11,
  8.298e-11, 7.947e-11, 7.46e-11, 7.082e-11, 6.132e-11, 5.855e-11,
  5.073e-11, 4.464e-11, 3.825e-11, 3.375e-11, 2.911e-11, 2.535e-11,
  2.16e-11, 1.907e-11, 1.665e-11, 1.463e-11, 1.291e-11, 1.133e-11,
  9.997e-12, 8.836e-12, 7.839e-12, 6.943e-12, 6.254e-12, 5.6e-12,
  5.029e-12, 4.529e-12, 4.102e-12, 3.737e-12, 3.428e-12, 3.169e-12,
  2.959e-12, 2.798e-12, 2.675e-12, 2.582e-12, 2.644e-12, 2.557e-12,
  2.614e-12, 2.717e-12, 2.874e-12, 3.056e-12, 3.187e-12, 3.631e-12,
  3.979e-12, 4.248e-12, 4.817e-12, 5.266e-12, 5.836e-12, 6.365e-12,
  6.807e-12, 7.47e-12, 7.951e-12, 8.636e-12, 8.972e-12, 9.314e-12,
  9.445e-12, 1.003e-11, 1.013e-11, 9.937e-12, 9.729e-12, 9.064e-12,
  9.119e-12, 9.124e-12, 8.704e-12, 8.078e-12, 7.47e-12, 6.329e-12,
  5.674e-12, 4.808e-12, 4.119e-12, 3.554e-12, 3.103e-12, 2.731e-12,
  2.415e-12, 2.15e-12, 1.926e-12, 1.737e-12, 1.578e-12, 1.447e-12,
  1.34e-12, 1.255e-12, 1.191e-12, 1.146e-12, 1.121e-12, 1.114e-12,
  1.126e-12, 1.156e-12, 1.207e-12, 1.278e-12, 1.372e-12, 1.49e-12,
  1.633e-12, 1.805e-12, 2.01e-12, 2.249e-12, 2.528e-12, 2.852e-12,
  3.228e-12, 3.658e-12, 4.153e-12, 4.728e-12, 5.394e-12, 6.176e-12,
  7.126e-12, 8.188e-12, 9.328e-12, 1.103e-11, 1.276e-11, 1.417e-11,
  1.615e-11, 1.84e-11, 2.155e-11, 2.429e-11, 2.826e-11, 3.222e-11,
  3.664e-11, 4.14e-11, 4.906e-11, 5.536e-11, 6.327e-11, 7.088e-11,
  8.316e-11, 9.242e-11, 1.07e-10, 1.223e-10, 1.341e-10, 1.553e-10,
  1.703e-10, 1.9e-10, 2.022e-10, 2.233e-10, 2.345e-10, 2.438e-10,
  2.546e-10, 2.599e-10, 2.661e-10, 2.703e-10, 2.686e-10, 2.662e-10,
  2.56e-10, 2.552e-10, 2.378e-10, 2.252e-10, 2.146e-10, 1.885e-10,
  1.668e-10, 1.441e-10, 1.295e-10, 1.119e-10, 9.893e-11, 8.687e-11,
  7.678e-11, 6.685e-11, 5.879e-11, 5.127e-11, 4.505e-11, 3.997e-11,
  3.511e-11
};

static double h2ofrn[2001] = { .01095, .01126, .01205, .01322, .0143,
  .01506, .01548, .01534, .01486, .01373, .01262, .01134, .01001,
  .008702, .007475, .006481, .00548, .0046, .003833, .00311,
  .002543, .002049, .00168, .001374, .001046, 8.193e-4, 6.267e-4,
  4.968e-4, 3.924e-4, 2.983e-4, 2.477e-4, 1.997e-4, 1.596e-4,
  1.331e-4, 1.061e-4, 8.942e-5, 7.168e-5, 5.887e-5, 4.848e-5,
  3.817e-5, 3.17e-5, 2.579e-5, 2.162e-5, 1.768e-5, 1.49e-5,
  1.231e-5, 1.013e-5, 8.555e-6, 7.328e-6, 6.148e-6, 5.207e-6,
  4.387e-6, 3.741e-6, 3.22e-6, 2.753e-6, 2.346e-6, 1.985e-6,
  1.716e-6, 1.475e-6, 1.286e-6, 1.122e-6, 9.661e-7, 8.284e-7,
  7.057e-7, 6.119e-7, 5.29e-7, 4.571e-7, 3.948e-7, 3.432e-7,
  2.983e-7, 2.589e-7, 2.265e-7, 1.976e-7, 1.704e-7, 1.456e-7,
  1.26e-7, 1.101e-7, 9.648e-8, 8.415e-8, 7.34e-8, 6.441e-8,
  5.643e-8, 4.94e-8, 4.276e-8, 3.703e-8, 3.227e-8, 2.825e-8,
  2.478e-8, 2.174e-8, 1.898e-8, 1.664e-8, 1.458e-8, 1.278e-8,
  1.126e-8, 9.891e-9, 8.709e-9, 7.652e-9, 6.759e-9, 5.975e-9,
  5.31e-9, 4.728e-9, 4.214e-9, 3.792e-9, 3.463e-9, 3.226e-9,
  2.992e-9, 2.813e-9, 2.749e-9, 2.809e-9, 2.913e-9, 3.037e-9,
  3.413e-9, 3.738e-9, 4.189e-9, 4.808e-9, 5.978e-9, 7.088e-9,
  8.071e-9, 9.61e-9, 1.21e-8, 1.5e-8, 1.764e-8, 2.221e-8, 2.898e-8,
  3.948e-8, 5.068e-8, 6.227e-8, 7.898e-8, 1.033e-7, 1.437e-7,
  1.889e-7, 2.589e-7, 3.59e-7, 4.971e-7, 7.156e-7, 9.983e-7,
  1.381e-6, 1.929e-6, 2.591e-6, 3.453e-6, 4.57e-6, 5.93e-6,
  7.552e-6, 9.556e-6, 1.183e-5, 1.425e-5, 1.681e-5, 1.978e-5,
  2.335e-5, 2.668e-5, 3.022e-5, 3.371e-5, 3.715e-5, 3.967e-5,
  4.06e-5, 4.01e-5, 3.809e-5, 3.491e-5, 3.155e-5, 2.848e-5,
  2.678e-5, 2.66e-5, 2.811e-5, 3.071e-5, 3.294e-5, 3.459e-5,
  3.569e-5, 3.56e-5, 3.434e-5, 3.186e-5, 2.916e-5, 2.622e-5,
  2.275e-5, 1.918e-5, 1.62e-5, 1.373e-5, 1.182e-5, 1.006e-5,
  8.556e-6, 7.26e-6, 6.107e-6, 5.034e-6, 4.211e-6, 3.426e-6,
```

```
2.865e-6, 2.446e-6, 1.998e-6, 1.628e-6, 1.242e-6, 1.005e-6,
7.853e-7, 6.21e-7, 5.071e-7, 4.156e-7, 3.548e-7, 2.825e-7,
2.261e-7, 1.916e-7, 1.51e-7, 1.279e-7, 1.059e-7, 9.14e-8,
7.707e-8, 6.17e-8, 5.311e-8, 4.263e-8, 3.518e-8, 2.961e-8,
2.457e-8, 2.119e-8, 1.712e-8, 1.439e-8, 1.201e-8, 1.003e-8,
8.564e-9, 7.199e-9, 6.184e-9, 5.206e-9, 4.376e-9, 3.708e-9,
3.157e-9, 2.725e-9, 2.361e-9, 2.074e-9, 1.797e-9, 1.562e-9,
1.364e-9, 1.196e-9, 1.042e-9, 8.862e-10, 7.648e-10, 6.544e-10,
5.609e-10, 4.791e-10, 4.108e-10, 3.531e-10, 3.038e-10, 2.618e-10,
2.268e-10, 1.969e-10, 1.715e-10, 1.496e-10, 1.308e-10, 1.147e-10,
1.008e-10, 8.894e-11, 7.885e-11, 7.031e-11, 6.355e-11, 5.854e-11,
5.534e-11, 5.466e-11, 5.725e-11, 6.447e-11, 7.943e-11, 1.038e-10,
1.437e-10, 2.04e-10, 2.901e-10, 4.051e-10, 5.556e-10, 7.314e-10,
9.291e-10, 1.134e-9, 1.321e-9, 1.482e-9, 1.596e-9, 1.669e-9,
1.715e-9, 1.762e-9, 1.817e-9, 1.828e-9, 1.848e-9, 1.873e-9,
1.902e-9, 1.894e-9, 1.864e-9, 1.841e-9, 1.797e-9, 1.704e-9,
1.559e-9, 1.382e-9, 1.187e-9, 1.001e-9, 8.468e-10, 7.265e-10,
6.521e-10, 6.381e-10, 6.66e-10, 7.637e-10, 9.705e-10, 1.368e-9,
1.856e-9, 2.656e-9, 3.954e-9, 5.96e-9, 8.72e-9, 1.247e-8,
1.781e-8, 2.491e-8, 3.311e-8, 4.272e-8, 5.205e-8, 6.268e-8,
7.337e-8, 8.277e-8, 9.185e-8, 1.004e-7, 1.091e-7, 1.159e-7,
1.188e-7, 1.175e-7, 1.124e-7, 1.033e-7, 9.381e-8, 8.501e-8,
7.956e-8, 7.894e-8, 8.331e-8, 9.102e-8, 9.836e-8, 1.035e-7,
1.064e-7, 1.06e-7, 1.032e-7, 9.808e-8, 9.139e-8, 8.442e-8,
7.641e-8, 6.881e-8, 6.161e-8, 5.404e-8, 4.804e-8, 4.446e-8,
4.328e-8, 4.259e-8, 4.421e-8, 4.673e-8, 4.985e-8, 5.335e-8,
5.796e-8, 6.542e-8, 7.714e-8, 8.827e-8, 1.04e-7, 1.238e-7,
1.499e-7, 1.829e-7, 2.222e-7, 2.689e-7, 3.303e-7, 3.981e-7,
4.84e-7, 5.91e-7, 7.363e-7, 9.087e-7, 1.139e-6, 1.455e-6,
1.866e-6, 2.44e-6, 3.115e-6, 3.941e-6, 4.891e-6, 5.992e-6,
7.111e-6, 8.296e-6, 9.21e-6, 9.987e-6, 1.044e-5, 1.073e-5,
1.092e-5, 1.106e-5, 1.138e-5, 1.171e-5, 1.186e-5, 1.186e-5,
1.179e-5, 1.166e-5, 1.151e-5, 1.16e-5, 1.197e-5, 1.241e-5,
1.268e-5, 1.26e-5, 1.184e-5, 1.063e-5, 9.204e-6, 7.584e-6,
6.053e-6, 4.482e-6, 3.252e-6, 2.337e-6, 1.662e-6, 1.18e-6,
8.15e-7, 5.95e-7, 4.354e-7, 3.302e-7, 2.494e-7, 1.93e-7,
1.545e-7, 1.25e-7, 1.039e-7, 8.602e-8, 7.127e-8, 5.897e-8,
4.838e-8, 4.018e-8, 3.28e-8, 2.72e-8, 2.307e-8, 1.972e-8,
1.654e-8, 1.421e-8, 1.174e-8, 1.004e-8, 8.739e-9, 7.358e-9,
6.242e-9, 5.303e-9, 4.567e-9, 3.94e-9, 3.375e-9, 2.864e-9,
2.422e-9, 2.057e-9, 1.75e-9, 1.505e-9, 1.294e-9, 1.101e-9,
9.401e-10, 8.018e-10, 6.903e-10, 5.965e-10, 5.087e-10, 4.364e-10,
3.759e-10, 3.247e-10, 2.809e-10, 2.438e-10, 2.123e-10, 1.853e-10,
1.622e-10, 1.426e-10, 1.26e-10, 1.125e-10, 1.022e-10, 9.582e-11,
9.388e-11, 9.801e-11, 1.08e-10, 1.276e-10, 1.551e-10, 1.903e-10,
2.291e-10, 2.724e-10, 3.117e-10, 3.4e-10, 3.562e-10, 3.625e-10,
3.619e-10, 3.429e-10, 3.221e-10, 2.943e-10, 2.645e-10, 2.338e-10,
2.062e-10, 1.901e-10, 1.814e-10, 1.827e-10, 1.906e-10, 1.984e-10,
2.04e-10, 2.068e-10, 2.075e-10, 2.018e-10, 1.959e-10, 1.897e-10,
1.852e-10, 1.791e-10, 1.696e-10, 1.634e-10, 1.598e-10, 1.561e-10,
1.518e-10, 1.443e-10, 1.377e-10, 1.346e-10, 1.342e-10, 1.375e-10,
1.525e-10, 1.767e-10, 2.108e-10, 2.524e-10, 2.981e-10, 3.477e-10,
4.262e-10, 5.326e-10, 6.646e-10, 8.321e-10, 1.069e-9, 1.386e-9,
1.743e-9, 2.216e-9, 2.808e-9, 3.585e-9, 4.552e-9, 5.907e-9,
7.611e-9, 9.774e-9, 1.255e-8, 1.666e-8, 2.279e-8, 3.221e-8,
4.531e-8, 6.4e-8, 9.187e-8, 1.295e-7, 1.825e-7, 2.431e-7,
3.181e-7, 4.009e-7, 4.941e-7, 5.88e-7, 6.623e-7, 7.155e-7,
7.451e-7, 7.594e-7, 7.541e-7, 7.467e-7, 7.527e-7, 7.935e-7,
8.461e-7, 8.954e-7, 9.364e-7, 9.843e-7, 1.024e-6, 1.05e-6,
1.059e-6, 1.074e-6, 1.072e-6, 1.043e-6, 9.789e-7, 8.803e-7,
7.662e-7, 6.378e-7, 5.133e-7, 3.958e-7, 2.914e-7, 2.144e-7,
1.57e-7, 1.14e-7, 8.47e-8, 6.2e-8, 4.657e-8, 3.559e-8, 2.813e-8,
2.222e-8, 1.769e-8, 1.391e-8, 1.125e-8, 9.186e-9, 7.704e-9,
6.447e-9, 5.381e-9, 4.442e-9, 3.669e-9, 3.057e-9, 2.564e-9,
2.153e-9, 1.784e-9, 1.499e-9, 1.281e-9, 1.082e-9, 9.304e-10,
8.169e-10, 6.856e-10, 5.866e-10, 5.043e-10, 4.336e-10, 3.731e-10,
3.175e-10, 2.745e-10, 2.374e-10, 2.007e-10, 1.737e-10, 1.508e-10,
1.302e-10, 1.13e-10, 9.672e-11, 8.375e-11, 7.265e-11, 6.244e-11,
5.343e-11, 4.654e-11, 3.975e-11, 3.488e-11, 3.097e-11, 2.834e-11,
2.649e-11, 2.519e-11, 2.462e-11, 2.443e-11, 2.44e-11, 2.398e-11,
2.306e-11, 2.183e-11, 2.021e-11, 1.821e-11, 1.599e-11, 1.403e-11,
1.196e-11, 1.023e-11, 8.728e-12, 7.606e-12, 6.941e-12, 6.545e-12,
6.484e-12, 6.6e-12, 6.718e-12, 6.785e-12, 6.746e-12, 6.724e-12,
6.764e-12, 6.995e-12, 7.144e-12, 7.32e-12, 7.33e-12, 7.208e-12,
6.789e-12, 6.09e-12, 5.337e-12, 4.62e-12, 4.037e-12, 3.574e-12,
3.311e-12, 3.346e-12, 3.566e-12, 3.836e-12, 4.076e-12, 4.351e-12,
4.691e-12, 5.114e-12, 5.427e-12, 6.167e-12, 7.436e-12, 8.842e-12,
1.038e-11, 1.249e-11, 1.54e-11, 1.915e-11, 2.48e-11, 3.256e-11,
4.339e-11, 5.611e-11, 7.519e-11, 1.037e-10, 1.409e-10, 1.883e-10,
2.503e-10, 3.38e-10, 4.468e-10, 5.801e-10, 7.335e-10, 8.98e-10,
1.11e-9, 1.363e-9, 1.677e-9, 2.104e-9, 2.681e-9, 3.531e-9,
4.621e-9, 6.106e-9, 8.154e-9, 1.046e-8, 1.312e-8, 1.607e-8,
1.948e-8, 2.266e-8, 2.495e-8, 2.655e-8, 2.739e-8, 2.739e-8,
2.662e-8, 2.589e-8, 2.59e-8, 2.664e-8, 2.833e-8, 3.023e-8,
3.305e-8, 3.558e-8, 3.793e-8, 3.961e-8, 4.056e-8, 4.102e-8,
4.025e-8, 3.917e-8, 3.706e-8, 3.493e-8, 3.249e-8, 3.096e-8,
3.011e-8, 3.111e-8, 3.395e-8, 3.958e-8, 4.875e-8, 6.066e-8,
```

```
7.915e-8, 1.011e-7, 1.3e-7, 1.622e-7, 2.003e-7, 2.448e-7,
2.863e-7, 3.317e-7, 3.655e-7, 3.96e-7, 4.098e-7, 4.168e-7,
4.198e-7, 4.207e-7, 4.289e-7, 4.384e-7, 4.471e-7, 4.524e-7,
4.574e-7, 4.633e-7, 4.785e-7, 5.028e-7, 5.371e-7, 5.727e-7,
5.955e-7, 5.998e-7, 5.669e-7, 5.082e-7, 4.397e-7, 3.596e-7,
2.814e-7, 2.074e-7, 1.486e-7, 1.057e-7, 7.25e-8, 4.946e-8,
3.43e-8, 2.447e-8, 1.793e-8, 1.375e-8, 1.096e-8, 9.091e-9,
7.709e-9, 6.631e-9, 5.714e-9, 4.886e-9, 4.205e-9, 3.575e-9,
3.07e-9, 2.631e-9, 2.284e-9, 2.002e-9, 1.745e-9, 1.509e-9,
1.284e-9, 1.084e-9, 9.163e-10, 7.663e-10, 6.346e-10, 5.283e-10,
4.354e-10, 3.59e-10, 2.982e-10, 2.455e-10, 2.033e-10, 1.696e-10,
1.432e-10, 1.211e-10, 1.02e-10, 8.702e-11, 7.38e-11, 6.293e-11,
5.343e-11, 4.532e-11, 3.907e-11, 3.365e-11, 2.945e-11, 2.558e-11,
2.192e-11, 1.895e-11, 1.636e-11, 1.42e-11, 1.228e-11, 1.063e-11,
9.348e-12, 8.2e-12, 7.231e-12, 6.43e-12, 5.702e-12, 5.052e-12,
4.469e-12, 4e-12, 3.679e-12, 3.387e-12, 3.197e-12, 3.158e-12,
3.327e-12, 3.675e-12, 4.292e-12, 5.437e-12, 7.197e-12, 1.008e-11,
1.437e-11, 2.035e-11, 2.905e-11, 4.062e-11, 5.528e-11, 7.177e-11,
9.064e-11, 1.109e-10, 1.297e-10, 1.473e-10, 1.652e-10, 1.851e-10,
2.079e-10, 2.313e-10, 2.619e-10, 2.958e-10, 3.352e-10, 3.796e-10,
4.295e-10, 4.923e-10, 5.49e-10, 5.998e-10, 6.388e-10, 6.645e-10,
6.712e-10, 6.549e-10, 6.38e-10, 6.255e-10, 6.253e-10, 6.459e-10,
6.977e-10, 7.59e-10, 8.242e-10, 8.92e-10, 9.403e-10, 9.701e-10,
9.483e-10, 9.135e-10, 8.617e-10, 7.921e-10, 7.168e-10, 6.382e-10,
5.677e-10, 5.045e-10, 4.572e-10, 4.312e-10, 4.145e-10, 4.192e-10,
4.541e-10, 5.368e-10, 6.771e-10, 8.962e-10, 1.21e-9, 1.659e-9,
2.33e-9, 3.249e-9, 4.495e-9, 5.923e-9, 7.642e-9, 9.607e-9,
1.178e-8, 1.399e-8, 1.584e-8, 1.73e-8, 1.816e-8, 1.87e-8,
1.868e-8, 1.87e-8, 1.884e-8, 1.99e-8, 2.15e-8, 2.258e-8,
2.364e-8, 2.473e-8, 2.602e-8, 2.689e-8, 2.731e-8, 2.816e-8,
2.859e-8, 2.839e-8, 2.703e-8, 2.451e-8, 2.149e-8, 1.787e-8,
1.449e-8, 1.111e-8, 8.282e-9, 6.121e-9, 4.494e-9, 3.367e-9,
2.487e-9, 1.885e-9, 1.503e-9, 1.249e-9, 1.074e-9, 9.427e-10,
8.439e-10, 7.563e-10, 6.772e-10, 6.002e-10, 5.254e-10, 4.588e-10,
3.977e-10, 3.449e-10, 3.003e-10, 2.624e-10, 2.335e-10, 2.04e-10,
1.771e-10, 1.534e-10, 1.296e-10, 1.097e-10, 9.173e-11, 7.73e-11,
6.547e-11, 5.191e-11, 4.198e-11, 3.361e-11, 2.732e-11, 2.244e-11,
1.791e-11, 1.509e-11, 1.243e-11, 1.035e-11, 8.969e-12, 7.394e-12,
6.323e-12, 5.282e-12, 4.543e-12, 3.752e-12, 3.14e-12, 2.6e-12,
2.194e-12, 1.825e-12, 1.511e-12, 1.245e-12, 1.024e-12, 8.539e-13,
7.227e-13, 6.102e-13, 5.189e-13, 4.43e-13, 3.774e-13, 3.236e-13,
2.8e-13, 2.444e-13, 2.156e-13, 1.932e-13, 1.775e-13, 1.695e-13,
1.672e-13, 1.704e-13, 1.825e-13, 2.087e-13, 2.614e-13, 3.377e-13,
4.817e-13, 6.989e-13, 1.062e-12, 1.562e-12, 2.288e-12, 3.295e-12,
4.55e-12, 5.965e-12, 7.546e-12, 9.395e-12, 1.103e-11, 1.228e-11,
1.318e-11, 1.38e-11, 1.421e-11, 1.39e-11, 1.358e-11, 1.336e-11,
1.342e-11, 1.356e-11, 1.424e-11, 1.552e-11, 1.73e-11, 1.951e-11,
2.128e-11, 2.249e-11, 2.277e-11, 2.226e-11, 2.111e-11, 1.922e-11,
1.775e-11, 1.661e-11, 1.547e-11, 1.446e-11, 1.323e-11, 1.21e-11,
1.054e-11, 9.283e-12, 8.671e-12, 8.67e-12, 9.429e-12, 1.062e-11,
1.255e-11, 1.506e-11, 1.818e-11, 2.26e-11, 2.831e-11, 3.723e-11,
5.092e-11, 6.968e-11, 9.826e-11, 1.349e-10, 1.87e-10, 2.58e-10,
3.43e-10, 4.424e-10, 5.521e-10, 6.812e-10, 8.064e-10, 9.109e-10,
9.839e-10, 1.028e-9, 1.044e-9, 1.029e-9, 1.005e-9, 1.002e-9,
1.038e-9, 1.122e-9, 1.233e-9, 1.372e-9, 1.524e-9, 1.665e-9,
1.804e-9, 1.908e-9, 2.015e-9, 2.117e-9, 2.219e-9, 2.336e-9,
2.531e-9, 2.805e-9, 3.189e-9, 3.617e-9, 4.208e-9, 4.911e-9,
5.619e-9, 6.469e-9, 7.188e-9, 7.957e-9, 8.503e-9, 9.028e-9,
9.571e-9, 9.99e-9, 1.055e-8, 1.102e-8, 1.132e-8, 1.141e-8,
1.145e-8, 1.145e-8, 1.176e-8, 1.224e-8, 1.304e-8, 1.388e-8,
1.445e-8, 1.453e-8, 1.368e-8, 1.22e-8, 1.042e-8, 8.404e-9,
6.403e-9, 4.643e-9, 3.325e-9, 2.335e-9, 1.638e-9, 1.19e-9,
9.161e-10, 7.412e-10, 6.226e-10, 5.516e-10, 5.068e-10, 4.831e-10,
4.856e-10, 5.162e-10, 5.785e-10, 6.539e-10, 7.485e-10, 8.565e-10,
9.534e-10, 1.052e-9, 1.115e-9, 1.173e-9, 1.203e-9, 1.224e-9,
1.243e-9, 1.248e-9, 1.261e-9, 1.265e-9, 1.25e-9, 1.217e-9,
1.176e-9, 1.145e-9, 1.153e-9, 1.199e-9, 1.278e-9, 1.366e-9,
1.426e-9, 1.444e-9, 1.365e-9, 1.224e-9, 1.051e-9, 8.539e-10,
6.564e-10, 4.751e-10, 3.404e-10, 2.377e-10, 1.631e-10, 1.114e-10,
7.87e-11, 5.793e-11, 4.284e-11, 3.3e-11, 2.62e-11, 2.152e-11,
1.777e-11, 1.496e-11, 1.242e-11, 1.037e-11, 8.725e-12, 7.004e-12,
5.718e-12, 4.769e-12, 3.952e-12, 3.336e-12, 2.712e-12, 2.213e-12,
1.803e-12, 1.492e-12, 1.236e-12, 1.006e-12, 8.384e-13, 7.063e-13,
5.879e-13, 4.93e-13, 4.171e-13, 3.569e-13, 3.083e-13, 2.688e-13,
2.333e-13, 2.035e-13, 1.82e-13, 1.682e-13, 1.635e-13, 1.674e-13,
1.769e-13, 2.022e-13, 2.485e-13, 3.127e-13, 4.25e-13, 5.928e-13,
8.514e-13, 1.236e-12, 1.701e-12, 2.392e-12, 3.231e-12, 4.35e-12,
5.559e-12, 6.915e-12, 8.519e-12, 1.013e-11, 1.146e-11, 1.24e-11,
1.305e-11, 1.333e-11, 1.318e-11, 1.263e-11, 1.238e-11, 1.244e-11,
1.305e-11, 1.432e-11, 1.623e-11, 1.846e-11, 2.09e-11, 2.328e-11,
2.526e-11, 2.637e-11, 2.702e-11, 2.794e-11, 2.889e-11, 2.989e-11,
3.231e-11, 3.68e-11, 4.375e-11, 5.504e-11, 7.159e-11, 9.502e-11,
1.279e-10, 1.645e-10, 2.098e-10, 2.618e-10, 3.189e-10, 3.79e-10,
4.303e-10, 4.753e-10, 5.027e-10, 5.221e-10, 5.293e-10, 5.346e-10,
5.467e-10, 5.796e-10, 6.2e-10, 6.454e-10, 6.705e-10, 6.925e-10,
7.233e-10, 7.35e-10, 7.538e-10, 7.861e-10, 8.077e-10, 8.132e-10,
7.749e-10, 7.036e-10, 6.143e-10, 5.093e-10, 4.089e-10, 3.092e-10,
```

```
2.299e-10, 1.705e-10, 1.277e-10, 9.723e-11, 7.533e-11, 6.126e-11,
5.154e-11, 4.428e-11, 3.913e-11, 3.521e-11, 3.297e-11, 3.275e-11,
3.46e-11, 3.798e-11, 4.251e-11, 4.745e-11, 5.232e-11, 5.606e-11,
5.82e-11, 5.88e-11, 5.79e-11, 5.661e-11, 5.491e-11, 5.366e-11,
5.341e-11, 5.353e-11, 5.336e-11, 5.293e-11, 5.248e-11, 5.235e-11,
5.208e-11, 5.322e-11, 5.521e-11, 5.725e-11, 5.827e-11, 5.685e-11,
5.245e-11, 4.612e-11, 3.884e-11, 3.129e-11, 2.404e-11, 1.732e-11,
1.223e-11, 8.574e-12, 5.888e-12, 3.986e-12, 2.732e-12, 1.948e-12,
1.414e-12, 1.061e-12, 8.298e-13, 6.612e-13, 5.413e-13, 4.472e-13,
3.772e-13, 3.181e-13, 2.645e-13, 2.171e-13, 1.778e-13, 1.464e-13,
1.183e-13, 9.637e-14, 7.991e-14, 6.668e-14, 5.57e-14, 4.663e-14,
3.848e-14, 3.233e-14, 2.706e-14, 2.284e-14, 1.944e-14, 1.664e-14,
1.43e-14, 1.233e-14, 1.066e-14, 9.234e-15, 8.023e-15, 6.993e-15,
6.119e-15, 5.384e-15, 4.774e-15, 4.283e-15, 3.916e-15, 3.695e-15,
3.682e-15, 4.004e-15, 4.912e-15, 6.853e-15, 1.056e-14, 1.712e-14,
2.804e-14, 4.516e-14, 7.113e-14, 1.084e-13, 1.426e-13, 1.734e-13,
1.978e-13, 2.194e-13, 2.388e-13, 2.489e-13, 2.626e-13, 2.865e-13,
3.105e-13, 3.387e-13, 3.652e-13, 3.984e-13, 4.398e-13, 4.906e-13,
5.55e-13, 6.517e-13, 7.813e-13, 9.272e-13, 1.164e-12, 1.434e-12,
1.849e-12, 2.524e-12, 3.328e-12, 4.523e-12, 6.108e-12, 8.207e-12,
1.122e-11, 1.477e-11, 1.9e-11, 2.412e-11, 2.984e-11, 3.68e-11,
4.353e-11, 4.963e-11, 5.478e-11, 5.903e-11, 6.233e-11, 6.483e-11,
6.904e-11, 7.569e-11, 8.719e-11, 1.048e-10, 1.278e-10, 1.557e-10,
1.869e-10, 2.218e-10, 2.61e-10, 2.975e-10, 3.371e-10, 3.746e-10,
4.065e-10, 4.336e-10, 4.503e-10, 4.701e-10, 4.8e-10, 4.917e-10,
5.038e-10, 5.128e-10, 5.143e-10, 5.071e-10, 5.019e-10, 5.025e-10,
5.183e-10, 5.496e-10, 5.877e-10, 6.235e-10, 6.42e-10, 6.234e-10,
5.698e-10, 4.916e-10, 4.022e-10, 3.126e-10, 2.282e-10, 1.639e-10,
1.142e-10, 7.919e-11, 5.69e-11, 4.313e-11, 3.413e-11, 2.807e-11,
2.41e-11, 2.166e-11, 2.024e-11, 1.946e-11, 1.929e-11, 1.963e-11,
2.035e-11, 2.162e-11, 2.305e-11, 2.493e-11, 2.748e-11, 3.048e-11,
3.413e-11, 3.754e-11, 4.155e-11, 4.635e-11, 5.11e-11, 5.734e-11,
6.338e-11, 6.99e-11, 7.611e-11, 8.125e-11, 8.654e-11, 8.951e-11,
9.182e-11, 9.31e-11, 9.273e-11, 9.094e-11, 8.849e-11, 8.662e-11,
8.67e-11, 8.972e-11, 9.566e-11, 1.025e-10, 1.083e-10, 1.111e-10,
1.074e-10, 9.771e-11, 8.468e-11, 6.958e-11, 5.47e-11, 4.04e-11,
2.94e-11, 2.075e-11, 1.442e-11, 1.01e-11, 7.281e-12, 5.409e-12,
4.138e-12, 3.304e-12, 2.784e-12, 2.473e-12, 2.273e-12, 2.186e-12,
2.118e-12, 2.066e-12, 1.958e-12, 1.818e-12, 1.675e-12, 1.509e-12,
1.349e-12, 1.171e-12, 9.838e-13, 8.213e-13, 6.765e-13, 5.378e-13,
4.161e-13, 3.119e-13, 2.279e-13, 1.637e-13, 1.152e-13, 8.112e-14,
5.919e-14, 4.47e-14, 3.492e-14, 2.811e-14, 2.319e-14, 1.948e-14,
1.66e-14, 1.432e-14, 1.251e-14, 1.109e-14, 1.006e-14, 9.45e-15,
9.384e-15, 1.012e-14, 1.216e-14, 1.636e-14, 2.305e-14, 3.488e-14,
5.572e-14, 8.479e-14, 1.265e-13, 1.905e-13, 2.73e-13, 3.809e-13,
4.955e-13, 6.303e-13, 7.861e-13, 9.427e-13, 1.097e-12, 1.212e-12,
1.328e-12, 1.415e-12, 1.463e-12, 1.495e-12, 1.571e-12, 1.731e-12,
1.981e-12, 2.387e-12, 2.93e-12, 3.642e-12, 4.584e-12, 5.822e-12,
7.278e-12, 9.193e-12, 1.135e-11, 1.382e-11, 1.662e-11, 1.958e-11,
2.286e-11, 2.559e-11, 2.805e-11, 2.988e-11, 3.106e-11, 3.182e-11,
3.2e-11, 3.258e-11, 3.362e-11, 3.558e-11, 3.688e-11, 3.8e-11,
3.929e-11, 4.062e-11, 4.186e-11, 4.293e-11, 4.48e-11, 4.643e-11,
4.704e-11, 4.571e-11, 4.206e-11, 3.715e-11, 3.131e-11, 2.541e-11,
1.978e-11, 1.508e-11, 1.146e-11, 8.7e-12, 6.603e-12, 5.162e-12,
4.157e-12, 3.408e-12, 2.829e-12, 2.405e-12, 2.071e-12, 1.826e-12,
1.648e-12, 1.542e-12, 1.489e-12, 1.485e-12, 1.493e-12, 1.545e-12,
1.637e-12, 1.814e-12, 2.061e-12, 2.312e-12, 2.651e-12, 3.03e-12,
3.46e-12, 3.901e-12, 4.306e-12, 4.721e-12, 5.008e-12, 5.281e-12,
5.541e-12, 5.791e-12, 6.115e-12, 6.442e-12, 6.68e-12, 6.791e-12,
6.831e-12, 6.839e-12, 6.946e-12, 7.128e-12, 7.537e-12, 8.036e-12,
8.392e-12, 8.526e-12, 8.11e-12, 7.325e-12, 6.329e-12, 5.183e-12,
4.081e-12, 2.985e-12, 2.141e-12, 1.492e-12, 1.015e-12, 6.684e-13,
4.414e-13, 2.987e-13, 2.038e-13, 1.391e-13, 9.86e-14, 7.24e-14,
5.493e-14, 4.288e-14, 3.427e-14, 2.787e-14, 2.296e-14, 1.909e-14,
1.598e-14, 1.344e-14, 1.135e-14, 9.616e-15, 8.169e-15, 6.957e-15,
5.938e-15, 5.08e-15, 4.353e-15, 3.738e-15, 3.217e-15, 2.773e-15,
2.397e-15, 2.077e-15, 1.805e-15, 1.575e-15, 1.382e-15, 1.221e-15,
1.09e-15, 9.855e-16, 9.068e-16, 8.537e-16, 8.27e-16, 8.29e-16,
8.634e-16, 9.359e-16, 1.055e-15, 1.233e-15, 1.486e-15, 1.839e-15,
2.326e-15, 2.998e-15, 3.934e-15, 5.256e-15, 7.164e-15, 9.984e-15,
1.427e-14, 2.099e-14, 3.196e-14, 5.121e-14, 7.908e-14, 1.131e-13,
1.602e-13, 2.239e-13, 3.075e-13, 4.134e-13, 5.749e-13, 7.886e-13,
1.071e-12, 1.464e-12, 2.032e-12, 2.8e-12, 3.732e-12, 4.996e-12,
6.483e-12, 8.143e-12, 1.006e-11, 1.238e-11, 1.484e-11, 1.744e-11,
2.02e-11, 2.274e-11, 2.562e-11, 2.848e-11, 3.191e-11, 3.617e-11,
4.081e-11, 4.577e-11, 4.937e-11, 5.204e-11, 5.401e-11, 5.462e-11,
5.507e-11, 5.51e-11, 5.605e-11, 5.686e-11, 5.739e-11, 5.766e-11,
5.74e-11, 5.754e-11, 5.761e-11, 5.777e-11, 5.712e-11, 5.51e-11,
5.088e-11, 4.438e-11, 3.728e-11, 2.994e-11, 2.305e-11, 1.715e-11,
1.256e-11, 9.208e-12, 6.745e-12, 5.014e-12, 3.785e-12, 2.9e-12,
2.239e-12, 1.757e-12, 1.414e-12, 1.142e-12, 9.482e-13, 8.01e-13,
6.961e-13, 6.253e-13, 5.735e-13, 5.433e-13, 5.352e-13, 5.493e-13,
5.706e-13, 6.068e-13, 6.531e-13, 7.109e-13, 7.767e-13, 8.59e-13,
9.792e-13, 1.142e-12, 1.371e-12, 1.65e-12, 1.957e-12, 2.302e-12,
2.705e-12, 3.145e-12, 3.608e-12, 4.071e-12, 4.602e-12, 5.133e-12,
5.572e-12, 5.987e-12, 6.248e-12, 6.533e-12, 6.757e-12, 6.935e-12,
7.224e-12, 7.422e-12, 7.538e-12, 7.547e-12, 7.495e-12, 7.543e-12,
```

```
    7.725e-12, 8.139e-12, 8.627e-12, 9.146e-12, 9.443e-12, 9.318e-12,
    8.649e-12, 7.512e-12, 6.261e-12, 4.915e-12, 3.647e-12, 2.597e-12,
    1.785e-12, 1.242e-12, 8.66e-13, 6.207e-13, 4.61e-13, 3.444e-13,
    2.634e-13, 2.1e-13, 1.725e-13, 1.455e-13, 1.237e-13, 1.085e-13,
    9.513e-14, 7.978e-14, 6.603e-14, 5.288e-14, 4.084e-14, 2.952e-14,
    2.157e-14, 1.593e-14, 1.199e-14, 9.267e-15, 7.365e-15, 6.004e-15,
    4.995e-15, 4.218e-15, 3.601e-15, 3.101e-15, 2.692e-15, 2.36e-15,
    2.094e-15, 1.891e-15, 1.755e-15, 1.699e-15, 1.755e-15, 1.987e-15,
    2.506e-15, 3.506e-15, 5.289e-15, 8.311e-15, 1.325e-14, 2.129e-14,
    3.237e-14, 4.595e-14, 6.441e-14, 8.433e-14, 1.074e-13, 1.383e-13,
    1.762e-13, 2.281e-13, 2.831e-13, 3.523e-13, 4.38e-13, 5.304e-13,
    6.29e-13, 7.142e-13, 8.032e-13, 8.934e-13, 9.888e-13, 1.109e-12,
    1.261e-12, 1.462e-12, 1.74e-12, 2.099e-12, 2.535e-12, 3.008e-12,
    3.462e-12, 3.856e-12, 4.098e-12, 4.239e-12, 4.234e-12, 4.132e-12,
    3.986e-12, 3.866e-12, 3.829e-12, 3.742e-12, 3.705e-12, 3.694e-12,
    3.765e-12, 3.849e-12, 3.929e-12, 4.056e-12, 4.092e-12, 4.047e-12,
    3.792e-12, 3.407e-12, 2.953e-12, 2.429e-12, 1.931e-12, 1.46e-12,
    1.099e-12, 8.199e-13, 6.077e-13, 4.449e-13, 3.359e-13, 2.524e-13,
    1.881e-13, 1.391e-13, 1.02e-13, 7.544e-14, 5.555e-14, 4.22e-14,
    3.321e-14, 2.686e-14, 2.212e-14, 1.78e-14, 1.369e-14, 1.094e-14,
    9.13e-15, 8.101e-15, 7.828e-15, 8.393e-15, 1.012e-14, 1.259e-14,
    1.538e-14, 1.961e-14, 2.619e-14, 3.679e-14, 5.049e-14, 6.917e-14,
    8.88e-14, 1.115e-13, 1.373e-13, 1.619e-13, 1.878e-13, 2.111e-13,
    2.33e-13, 2.503e-13, 2.613e-13, 2.743e-13, 2.826e-13, 2.976e-13,
    3.162e-13, 3.36e-13, 3.491e-13, 3.541e-13, 3.595e-13, 3.608e-13,
    3.709e-13, 3.869e-13, 4.12e-13, 4.366e-13, 4.504e-13, 4.379e-13,
    3.955e-13, 3.385e-13, 2.741e-13, 2.089e-13, 1.427e-13, 9.294e-14,
    5.775e-14, 3.565e-14, 2.21e-14, 1.398e-14, 9.194e-15, 6.363e-15,
    4.644e-15, 3.55e-15, 2.808e-15, 2.274e-15, 1.871e-15, 1.557e-15,
    1.308e-15, 1.108e-15, 9.488e-16, 8.222e-16, 7.238e-16, 6.506e-16,
    6.008e-16, 5.742e-16, 5.724e-16, 5.991e-16, 6.625e-16, 7.775e-16,
    9.734e-16, 1.306e-15, 1.88e-15, 2.879e-15, 4.616e-15, 7.579e-15,
    1.248e-14, 2.03e-14, 3.244e-14, 5.171e-14, 7.394e-14, 9.676e-14,
    1.199e-13, 1.467e-13, 1.737e-13, 2.02e-13, 2.425e-13, 3.016e-13,
    3.7e-13, 4.617e-13, 5.949e-13, 7.473e-13, 9.378e-13, 1.191e-12,
    1.481e-12, 1.813e-12, 2.232e-12, 2.722e-12, 3.254e-12, 3.845e-12,
    4.458e-12, 5.048e-12, 5.511e-12, 5.898e-12, 6.204e-12, 6.293e-12,
    6.386e-12, 6.467e-12, 6.507e-12, 6.466e-12, 6.443e-12, 6.598e-12,
    6.873e-12, 7.3e-12, 7.816e-12, 8.368e-12, 8.643e-12, 8.466e-12,
    7.871e-12, 6.853e-12, 5.714e-12, 4.482e-12, 3.392e-12, 2.613e-12,
    2.008e-12, 1.562e-12, 1.228e-12, 9.888e-13, 7.646e-13, 5.769e-13,
    4.368e-13, 3.324e-13, 2.508e-13, 1.916e-13
  };

  static double xfcrev[15] =
    { 1.003, 1.009, 1.015, 1.023, 1.029, 1.033, 1.037,
    1.039, 1.04, 1.046, 1.036, 1.027, 1.01, 1.002, 1.
  };

  double a1, a2, a3, dw, ew, dx, xw, xx, vf2, vf6, cw260, cw296,
    sfac, fscal, cwfrn, ctmpth, ctwfrn, ctwslf;

  int iw, ix;

  /* Get H2O continuum absorption... */
  xw = nu / 10 + 1;
  if (xw >= 1 && xw < 2001) {
    iw = (int) xw;
    dw = xw - iw;
    ew = 1 - dw;
    cw296 = ew * h2o296[iw - 1] + dw * h2o296[iw];
    cw260 = ew * h2o260[iw - 1] + dw * h2o260[iw];
    cwfrn = ew * h2ofrn[iw - 1] + dw * h2ofrn[iw];
    if (nu <= 820 || nu >= 960) {
      sfac = 1;
    } else {
      xx = (nu - 820) / 10;
      ix = (int) xx;
      dx = xx - ix;
      sfac = (1 - dx) * xfcrev[ix] + dx * xfcrev[ix + 1];
    }
    ctwslf = sfac * cw296 * pow(cw260 / cw296, (296 - t) / (296 - 260));
    vf2 = gsl_pow_2(nu - 370);
    vf6 = gsl_pow_3(vf2);
    fscal = 36100 / (vf2 + vf6 * 1e-8 + 36100) * -.25 + 1;
    ctwfrn = cwfrn * fscal;
    a1 = nu * u * tanh(.7193876 / t * nu);
    a2 = 296 / t;
    a3 = p / P0 * (q * ctwslf + (1 - q) * ctwfrn) * 1e-20;
    ctmpth = a1 * a2 * a3;
  } else
    ctmpth = 0;
  return ctmpth;
}
```

5.33.2.8 double ctmn2 (double *nu,* double *p,* double *t*)

Compute nitrogen continuum (absorption coefficient).

Definition at line 2790 of file jurassic.c.

```
            {

static double ba[98] = { 0., 4.45e-8, 5.22e-8, 6.46e-8, 7.75e-8, 9.03e-8,
  1.06e-7, 1.21e-7, 1.37e-7, 1.57e-7, 1.75e-7, 2.01e-7, 2.3e-7,
  2.59e-7, 2.95e-7, 3.26e-7, 3.66e-7, 4.05e-7, 4.47e-7, 4.92e-7,
  5.34e-7, 5.84e-7, 6.24e-7, 6.67e-7, 7.14e-7, 7.26e-7, 7.54e-7,
  7.84e-7, 8.09e-7, 8.42e-7, 8.62e-7, 8.87e-7, 9.11e-7, 9.36e-7,
  9.76e-7, 1.03e-6, 1.11e-6, 1.23e-6, 1.39e-6, 1.61e-6, 1.76e-6,
  1.94e-6, 1.97e-6, 1.87e-6, 1.75e-6, 1.56e-6, 1.42e-6, 1.35e-6,
  1.32e-6, 1.29e-6, 1.29e-6, 1.29e-6, 1.3e-6, 1.32e-6, 1.33e-6,
  1.34e-6, 1.35e-6, 1.33e-6, 1.31e-6, 1.29e-6, 1.24e-6, 1.2e-6,
  1.16e-6, 1.1e-6, 1.04e-6, 9.96e-7, 9.38e-7, 8.63e-7, 7.98e-7,
  7.26e-7, 6.55e-7, 5.94e-7, 5.35e-7, 4.74e-7, 4.24e-7, 3.77e-7,
  3.33e-7, 2.96e-7, 2.63e-7, 2.34e-7, 2.08e-7, 1.85e-7, 1.67e-7,
  1.47e-7, 1.32e-7, 1.2e-7, 1.09e-7, 9.85e-8, 9.08e-8, 8.18e-8,
  7.56e-8, 6.85e-8, 6.14e-8, 5.83e-8, 5.77e-8, 5e-8, 4.32e-8, 0.
};

static double betaa[98] = { 802., 802., 761., 722., 679., 646., 609., 562.,
  511., 472., 436., 406., 377., 355., 338., 319., 299., 278., 255.,
  233., 208., 184., 149., 107., 66., 25., -13., -49., -82., -104.,
  -119., -130., -139., -144., -146., -146., -147., -148., -150.,
  -153., -160., -169., -181., -189., -195., -200., -205., -209.,
  -211., -210., -210., -209., -205., -199., -190., -180., -168.,
  -157., -143., -126., -108., -89., -63., -32., 1., 35., 65., 95.,
  121., 141., 152., 161., 164., 164., 161., 155., 148., 143., 137.,
  133., 131., 133., 139., 150., 165., 187., 213., 248., 284., 321.,
  372., 449., 514., 569., 609., 642., 673., 673.
};

static double nua[98] = { 2120., 2125., 2130., 2135., 2140., 2145., 2150.,
  2155., 2160., 2165., 2170., 2175., 2180., 2185., 2190., 2195.,
  2200., 2205., 2210., 2215., 2220., 2225., 2230., 2235., 2240.,
  2245., 2250., 2255., 2260., 2265., 2270., 2275., 2280., 2285.,
  2290., 2295., 2300., 2305., 2310., 2315., 2320., 2325., 2330.,
  2335., 2340., 2345., 2350., 2355., 2360., 2365., 2370., 2375.,
  2380., 2385., 2390., 2395., 2400., 2405., 2410., 2415., 2420.,
  2425., 2430., 2435., 2440., 2445., 2450., 2455., 2460., 2465.,
  2470., 2475., 2480., 2485., 2490., 2495., 2500., 2505., 2510.,
  2515., 2520., 2525., 2530., 2535., 2540., 2545., 2550., 2555.,
  2560., 2565., 2570., 2575., 2580., 2585., 2590., 2595., 2600., 2605.
};

double b, beta, q_n2 = 0.79, t0 = 273, tr = 296;

int idx;

/* Check wavenumber range... */
if (nu < nua[0] || nu > nua[97])
  return 0;

/* Interpolate B and beta... */
idx = locate(nua, 98, nu);
b = LIN(nua[idx], ba[idx], nua[idx + 1], ba[idx + 1], nu);
beta = LIN(nua[idx], betaa[idx], nua[idx + 1], betaa[idx + 1], nu);

/* Compute absorption coefficient... */
return 0.1 * gsl_pow_2(p / P0) * gsl_pow_2(t0 / t)
  * exp(beta * (1 / tr - 1 / t))
  * q_n2 * b * (q_n2 + (1 - q_n2) * (1.294 - 0.4545 * t / tr));
}
```

Here is the call graph for this function:

5.33.2.9 double ctmo2 (double *nu,* double *p,* double *t*)

Compute oxygen continuum (absorption coefficient).

Definition at line 2857 of file jurassic.c.

```
            {

  static double ba[90] = { 0., .061, .074, .084, .096, .12, .162, .208, .246,
    .285, .314, .38, .444, .5, .571, .673, .768, .853, .966, 1.097,
    1.214, 1.333, 1.466, 1.591, 1.693, 1.796, 1.922, 2.037, 2.154,
    2.264, 2.375, 2.508, 2.671, 2.847, 3.066, 3.417, 3.828, 4.204,
    4.453, 4.599, 4.528, 4.284, 3.955, 3.678, 3.477, 3.346, 3.29,
    3.251, 3.231, 3.226, 3.212, 3.192, 3.108, 3.033, 2.911, 2.798,
    2.646, 2.508, 2.322, 2.13, 1.928, 1.757, 1.588, 1.417, 1.253,
    1.109, .99, .888, .791, .678, .587, .524, .464, .403, .357, .32,
    .29, .267, .242, .215, .182, .16, .146, .128, .103, .087, .081,
    .071, .064, 0.
  };

  static double betaa[90] = { 467., 467., 400., 315., 379., 368., 475., 521.,
    531., 512., 442., 444., 430., 381., 335., 324., 296., 248., 215.,
    193., 158., 127., 101., 71., 31., -6., -26., -47., -63., -79.,
    -88., -88., -87., -90., -98., -99., -109., -134., -160., -167.,
    -164., -158., -153., -151., -156., -166., -168., -173., -170.,
    -161., -145., -126., -108., -84., -59., -29., 4., 41., 73., 97.,
    123., 159., 198., 220., 242., 256., 281., 311., 334., 319., 313.,
    321., 323., 310., 315., 320., 335., 361., 378., 373., 338., 319.,
    346., 322., 291., 290., 350., 371., 504., 504.
  };

  static double nua[90] = { 1360., 1365., 1370., 1375., 1380., 1385., 1390.,
    1395., 1400., 1405., 1410., 1415., 1420., 1425., 1430., 1435.,
    1440., 1445., 1450., 1455., 1460., 1465., 1470., 1475., 1480.,
    1485., 1490., 1495., 1500., 1505., 1510., 1515., 1520., 1525.,
    1530., 1535., 1540., 1545., 1550., 1555., 1560., 1565., 1570.,
    1575., 1580., 1585., 1590., 1595., 1600., 1605., 1610., 1615.,
    1620., 1625., 1630., 1635., 1640., 1645., 1650., 1655., 1660.,
    1665., 1670., 1675., 1680., 1685., 1690., 1695., 1700., 1705.,
    1710., 1715., 1720., 1725., 1730., 1735., 1740., 1745., 1750.,
    1755., 1760., 1765., 1770., 1775., 1780., 1785., 1790., 1795.,
    1800., 1805.
  };

  double b, beta, q_o2 = 0.21, t0 = 273, tr = 296;

  int idx;

  /* Check wavenumber range... */
  if (nu < nua[0] || nu > nua[89])
    return 0;

  /* Interpolate B and beta... */
  idx = locate(nua, 90, nu);
  b = LIN(nua[idx], ba[idx], nua[idx + 1], ba[idx + 1], nu);
  beta = LIN(nua[idx], betaa[idx], nua[idx + 1], betaa[idx + 1], nu);

  /* Compute absorption coefficient... */
  return 0.1 * gsl_pow_2(p / P0) * gsl_pow_2(t0 / t)
    * exp(beta * (1 / tr - 1 / t)) * q_o2 * b;
}
```

Here is the call graph for this function:

5.33.2.10 void copy_atm (ctl_t * *ctl*, atm_t * *atm_dest*, atm_t * *atm_src*, int *init*)

Copy and initialize atmospheric data.

Definition at line 2918 of file jurassic.c.

```
            {

  int ig, ip, iw;

  size_t s;

  /* Data size... */
  s = (size_t) atm_src->np * sizeof(double);

  /* Copy data... */
  atm_dest->np = atm_src->np;
  memcpy(atm_dest->time, atm_src->time, s);
  memcpy(atm_dest->z, atm_src->z, s);
  memcpy(atm_dest->lon, atm_src->lon, s);
  memcpy(atm_dest->lat, atm_src->lat, s);
```

```
  memcpy(atm_dest->p, atm_src->p, s);
  memcpy(atm_dest->t, atm_src->t, s);
  for (ig = 0; ig < ctl->ng; ig++)
    memcpy(atm_dest->q[ig], atm_src->q[ig], s);
  for (iw = 0; iw < ctl->nw; iw++)
    memcpy(atm_dest->k[iw], atm_src->k[iw], s);

  /* Initialize... */
  if (init)
    for (ip = 0; ip < atm_dest->np; ip++) {
      atm_dest->p[ip] = 0;
      atm_dest->t[ip] = 0;
      for (ig = 0; ig < ctl->ng; ig++)
        atm_dest->q[ig][ip] = 0;
      for (iw = 0; iw < ctl->nw; iw++)
        atm_dest->k[iw][ip] = 0;
    }
}
```

5.33.2.11 void copy_obs (ctl_t * *ctl*, obs_t * *obs_dest*, obs_t * *obs_src*, int *init*)

Copy and initialize observation data.

Definition at line 2958 of file jurassic.c.

```
             {

  int id, ir;

  size_t s;

  /* Data size... */
  s = (size_t) obs_src->nr * sizeof(double);

  /* Copy data... */
  obs_dest->nr = obs_src->nr;
  memcpy(obs_dest->time, obs_src->time, s);
  memcpy(obs_dest->obsz, obs_src->obsz, s);
  memcpy(obs_dest->obslon, obs_src->obslon, s);
  memcpy(obs_dest->obslat, obs_src->obslat, s);
  memcpy(obs_dest->vpz, obs_src->vpz, s);
  memcpy(obs_dest->vplon, obs_src->vplon, s);
  memcpy(obs_dest->vplat, obs_src->vplat, s);
  memcpy(obs_dest->tpz, obs_src->tpz, s);
  memcpy(obs_dest->tplon, obs_src->tplon, s);
  memcpy(obs_dest->tplat, obs_src->tplat, s);
  for (id = 0; id < ctl->nd; id++)
    memcpy(obs_dest->rad[id], obs_src->rad[id], s);
  for (id = 0; id < ctl->nd; id++)
    memcpy(obs_dest->tau[id], obs_src->tau[id], s);

  /* Initialize... */
  if (init)
    for (id = 0; id < ctl->nd; id++)
      for (ir = 0; ir < obs_dest->nr; ir++)
        if (gsl_finite(obs_dest->rad[id][ir])) {
          obs_dest->rad[id][ir] = 0;
          obs_dest->tau[id][ir] = 0;
        }
}
```

5.33.2.12 int find_emitter (ctl_t * *ctl*, const char * *emitter*)

Find index of an emitter.

Definition at line 3000 of file jurassic.c.

```
                          {

  int ig;

  for (ig = 0; ig < ctl->ng; ig++)
    if (strcasecmp(ctl->emitter[ig], emitter) == 0)
      return ig;

  return -1;
}
```

5.33.2.13 void formod (ctl_t * ctl, atm_t * atm, obs_t * obs)

Determine ray paths and compute radiative transfer.

Definition at line 3015 of file jurassic.c.

```
                 {

  int id, ir, *mask;

  /* Allocate... */
  ALLOC(mask, int,
        ND * NR);

  /* Save observation mask... */
  for (id = 0; id < ctl->nd; id++)
    for (ir = 0; ir < obs->nr; ir++)
      mask[id * NR + ir] = !gsl_finite(obs->rad[id][ir]);

  /* Hydrostatic equilibrium... */
  hydrostatic(ctl, atm);

  /* Claculate pencil beams... */
  for (ir = 0; ir < obs->nr; ir++)
    formod_pencil(ctl, atm, obs, ir);

  /* Apply field-of-view convolution... */
  formod_fov(ctl, obs);

  /* Convert radiance to brightness temperature... */
  if (ctl->write_bbt)
    for (id = 0; id < ctl->nd; id++)
      for (ir = 0; ir < obs->nr; ir++)
        obs->rad[id][ir] = brightness(obs->rad[id][ir], ctl->nu[id]);

  /* Apply observation mask... */
  for (id = 0; id < ctl->nd; id++)
    for (ir = 0; ir < obs->nr; ir++)
      if (mask[id * NR + ir])
        obs->rad[id][ir] = GSL_NAN;

  /* Free... */
  free(mask);
}
```

Here is the call graph for this function:

5.33.2.14 void formod_continua (ctl_t * ctl, los_t * los, int ip, double * beta)

Compute absorption coefficient of continua.

Definition at line 3059 of file jurassic.c.

```
                  {

  static int ig_co2 = -999, ig_h2o = -999;

  int id;

  /* Extinction... */
  for (id = 0; id < ctl->nd; id++)
    beta[id] = los->k[ctl->window[id]][ip];

  /* CO2 continuum... */
  if (ctl->ctm_co2) {
    if (ig_co2 == -999)
      ig_co2 = find_emitter(ctl, "CO2");
    if (ig_co2 >= 0)
      for (id = 0; id < ctl->nd; id++)
        beta[id] += ctmco2(ctl->nu[id], los->p[ip], los->t[ip],
                           los->u[ig_co2][ip]) / los->ds[ip];
  }

  /* H2O continuum... */
  if (ctl->ctm_h2o) {
```

```
    if (ig_h2o == -999)
      ig_h2o = find_emitter(ctl, "H2O");
    if (ig_h2o >= 0)
      for (id = 0; id < ctl->nd; id++)
        beta[id] += ctmh2o(ctl->nu[id], los->p[ip], los->t[ip],
                           los->q[ig_h2o][ip],
                           los->u[ig_h2o][ip]) / los->ds[ip];
  }

  /* N2 continuum... */
  if (ctl->ctm_n2)
    for (id = 0; id < ctl->nd; id++)
      beta[id] += ctmn2(ctl->nu[id], los->p[ip], los->t[ip]);

  /* O2 continuum... */
  if (ctl->ctm_o2)
    for (id = 0; id < ctl->nd; id++)
      beta[id] += ctmo2(ctl->nu[id], los->p[ip], los->t[ip]);
}
```

Here is the call graph for this function:

5.33.2.15 void formod_fov (ctl_t * *ctl*, obs_t * *obs*)

Apply field of view convolution.

Definition at line 3107 of file jurassic.c.

```
                 {

  static double dz[NSHAPE], w[NSHAPE];

  static int init = 0, n;

  obs_t *obs2;

  double rad[ND][NR], tau[ND][NR], wsum, z[NR], zfov;

  int i, id, idx, ir, ir2, nz;

  /* Do not take into account FOV... */
  if (ctl->fov[0] == '-')
    return;

  /* Initialize FOV data... */
  if (!init) {
    init = 1;
    read_shape(ctl->fov, dz, w, &n);
  }

  /* Allocate... */
  ALLOC(obs2, obs_t, 1);

  /* Copy observation data... */
  copy_obs(ctl, obs2, obs, 0);

  /* Loop over ray paths... */
  for (ir = 0; ir < obs->nr; ir++) {

    /* Get radiance and transmittance profiles... */
    nz = 0;
    for (ir2 = GSL_MAX(ir - NFOV, 0); ir2 < GSL_MIN(ir + 1 + NFOV, obs->nr);
         ir2++)
      if (obs->time[ir2] == obs->time[ir]) {
        z[nz] = obs2->vpz[ir2];
        for (id = 0; id < ctl->nd; id++) {
          rad[id][nz] = obs2->rad[id][ir2];
          tau[id][nz] = obs2->tau[id][ir2];
        }
        nz++;
      }
    if (nz < 2)
      ERRMSG("Cannot apply FOV convolution!");

    /* Convolute profiles with FOV... */
    wsum = 0;
    for (id = 0; id < ctl->nd; id++) {
      obs->rad[id][ir] = 0;
      obs->tau[id][ir] = 0;
```

```
    }
    for (i = 0; i < n; i++) {
      zfov = obs->vpz[ir] + dz[i];
      idx = locate(z, nz, zfov);
      for (id = 0; id < ctl->nd; id++) {
        obs->rad[id][ir] += w[i]
          * LIN(z[idx], rad[id][idx], z[idx + 1], rad[id][idx + 1], zfov);
        obs->tau[id][ir] += w[i]
          * LIN(z[idx], tau[id][idx], z[idx + 1], tau[id][idx + 1], zfov);
      }
      wsum += w[i];
    }
    for (id = 0; id < ctl->nd; id++) {
      obs->rad[id][ir] /= wsum;
      obs->tau[id][ir] /= wsum;
    }
  }

  /* Free... */
  free(obs2);
}
```

Here is the call graph for this function:

5.33.2.16 void formod_pencil (ctl_t ∗ *ctl,* atm_t ∗ *atm,* obs_t ∗ *obs,* int *ir*)

Compute radiative transfer for a pencil beam.

Definition at line 3184 of file jurassic.c.

```
          {

  static tbl_t *tbl;

  static int init = 0;

  los_t *los;

  double beta_ctm[ND], eps, src_planck[ND], tau_path[NG][ND], tau_gas[ND];

  int id, ip;

  /* Initialize look-up tables... */
  if (!init) {
    init = 1;
    ALLOC(tbl, tbl_t, 1);
    init_tbl(ctl, tbl);
  }

  /* Allocate... */
  ALLOC(los, los_t, 1);

  /* Initialize... */
  for (id = 0; id < ctl->nd; id++) {
    obs->rad[id][ir] = 0;
    obs->tau[id][ir] = 1;
  }

  /* Raytracing... */
  raytrace(ctl, atm, obs, los, ir);

  /* Loop over LOS points... */
  for (ip = 0; ip < los->np; ip++) {

    /* Get trace gas transmittance... */
    intpol_tbl(ctl, tbl, los, ip, tau_path, tau_gas);

    /* Get continuum absorption... */
    formod_continua(ctl, los, ip, beta_ctm);

    /* Compute Planck function... */
    formod_srcfunc(ctl, tbl, los->t[ip], src_planck);

    /* Loop over channels... */
    for (id = 0; id < ctl->nd; id++)
      if (tau_gas[id] > 0) {

        /* Get segment emissivity... */
        eps = 1 - tau_gas[id] * exp(-beta_ctm[id] * los->ds[ip]);
```

```
        /* Compute radiance... */
        obs->rad[id][ir] += src_planck[id] * eps * obs->tau[id][ir];

        /* Compute path transmittance... */
        obs->tau[id][ir] *= (1 - eps);
      }
  }

  /* Add surface... */
  if (los->tsurf > 0) {
    formod_srcfunc(ctl, tbl, los->tsurf, src_planck);
    for (id = 0; id < ctl->nd; id++)
      obs->rad[id][ir] += src_planck[id] * obs->tau[id][ir];
  }

  /* Free... */
  free(los);
}
```

Here is the call graph for this function:

5.33.2.17 void formod_srcfunc (ctl_t * *ctl*, tbl_t * *tbl*, double *t*, double * *src*)

Compute Planck source function.

Definition at line 3259 of file jurassic.c.

```
                  {

  int id, it;

  /* Determine index in temperature array... */
  it = locate(tbl->st, TBLNS, t);

  /* Interpolate Planck function value... */
  for (id = 0; id < ctl->nd; id++)
    src[id] = LIN(tbl->st[it], tbl->sr[id][it],
                  tbl->st[it + 1], tbl->sr[id][it + 1], t);
}
```

Here is the call graph for this function:

5.33.2.18 void geo2cart (double *z*, double *lon*, double *lat*, double * *x*)

Convert geolocation to Cartesian coordinates.

Definition at line 3278 of file jurassic.c.

```
                {

  double radius;

  radius = z + RE;
  x[0] = radius * cos(lat / 180 * M_PI) * cos(lon / 180 * M_PI);
  x[1] = radius * cos(lat / 180 * M_PI) * sin(lon / 180 * M_PI);
  x[2] = radius * sin(lat / 180 * M_PI);
}
```

5.33.2.19 double gravity (double *z*, double *lat*)

Determine gravity of Earth.

Definition at line 3294 of file jurassic.c.

```
                 {

  /* Compute gravity according to 1967 Geodetic Reference System... */
  return 9.780318 * (1 + 0.0053024 * gsl_pow_2(sin(lat / 180 * M_PI))
                     - 0.0000058 * gsl_pow_2(sin(2 * lat / 180 * M_PI))) -
    3.086e-3 * z;
}
```

5.33.2.20 void hydrostatic (ctl_t * *ctl*, atm_t * *atm*)

Set hydrostatic equilibrium.

Definition at line 3306 of file jurassic.c.

```
                  {

  static int ig_h2o = -999;

  double dzmin = 1e99, e = 0, mean, mmair = 28.96456e-3, mmh2o =
    18.0153e-3, z;

  int i, ip, ipref = 0, ipts = 20;

  /* Check reference height... */
  if (ctl->hydz < 0)
    return;

  /* Determine emitter index of H2O... */
  if (ig_h2o == -999)
    ig_h2o = find_emitter(ctl, "H2O");

  /* Find air parcel next to reference height... */
  for (ip = 0; ip < atm->np; ip++)
    if (fabs(atm->z[ip] - ctl->hydz) < dzmin) {
      dzmin = fabs(atm->z[ip] - ctl->hydz);
      ipref = ip;
    }

  /* Upper part of profile... */
  for (ip = ipref + 1; ip < atm->np; ip++) {
    mean = 0;
    for (i = 0; i < ipts; i++) {
      z = LIN(0.0, atm->z[ip - 1], ipts - 1.0, atm->z[ip], (double) i);
      if (ig_h2o >= 0)
        e = LIN(0.0, atm->q[ig_h2o][ip - 1],
                ipts - 1.0, atm->q[ig_h2o][ip], (double) i);
      mean += (e * mmh2o + (1 - e) * mmair)
        * gravity(z, atm->lat[ipref]) / GSL_CONST_MKSA_MOLAR_GAS
        / LIN(0.0, atm->t[ip - 1], ipts - 1.0, atm->t[ip], (double) i) / ipts;
    }

    /* Compute p(z,T)... */
    atm->p[ip] =
      exp(log(atm->p[ip - 1]) - mean * 1000 * (atm->z[ip] - atm->z[ip - 1]));
  }

  /* Lower part of profile... */
  for (ip = ipref - 1; ip >= 0; ip--) {
    mean = 0;
    for (i = 0; i < ipts; i++) {
      z = LIN(0.0, atm->z[ip + 1], ipts - 1.0, atm->z[ip], (double) i);
      if (ig_h2o >= 0)
        e = LIN(0.0, atm->q[ig_h2o][ip + 1],
                ipts - 1.0, atm->q[ig_h2o][ip], (double) i);
      mean += (e * mmh2o + (1 - e) * mmair)
        * gravity(z, atm->lat[ipref]) / GSL_CONST_MKSA_MOLAR_GAS
        / LIN(0.0, atm->t[ip + 1], ipts - 1.0, atm->t[ip], (double) i) / ipts;
    }

    /* Compute p(z,T)... */
    atm->p[ip] =
      exp(log(atm->p[ip + 1]) - mean * 1000 * (atm->z[ip] - atm->z[ip + 1]));
  }
}
```

Here is the call graph for this function:

5.33.2.21 void idx2name (ctl_t * *ctl*, int *idx*, char * *quantity*)

Determine name of state vector quantity for given index.

Definition at line 3371 of file jurassic.c.

```
                   {

  int ig, iw;

  if (idx == IDXP)
    sprintf(quantity, "PRESSURE");

  if (idx == IDXT)
    sprintf(quantity, "TEMPERATURE");

  for (ig = 0; ig < ctl->ng; ig++)
    if (idx == IDXQ(ig))
      sprintf(quantity, "%s", ctl->emitter[ig]);

  for (iw = 0; iw < ctl->nw; iw++)
    if (idx == IDXK(iw))
      sprintf(quantity, "EXTINCT_WINDOW%d", iw);
}
```

5.33.2.22 void init_tbl (ctl_t * *ctl*, tbl_t * *tbl*)

Initialize look-up tables.

Definition at line 3395 of file jurassic.c.

```
                {

  FILE *in;

  char filename[LEN], line[LEN];

  double eps, eps_old, press, press_old, temp, temp_old, u, u_old,
    f[NSHAPE], fsum, nu[NSHAPE];

  int i, id, ig, ip, it, n;

  /* Loop over trace gases and channels... */
  for (ig = 0; ig < ctl->ng; ig++)
#pragma omp parallel for default(none) shared(ctl,tbl,ig) private(in,filename,line,eps,eps_old,press,
      press_old,temp,temp_old,u,u_old,id,ip,it)
    for (id = 0; id < ctl->nd; id++) {

      /* Initialize... */
      tbl->np[ig][id] = -1;
      eps_old = -999;
      press_old = -999;
      temp_old = -999;
      u_old = -999;

      /* Try to open file... */
      sprintf(filename, "%s_%.4f_%s.tab",
              ctl->tblbase, ctl->nu[id], ctl->emitter[ig]);
      if (!(in = fopen(filename, "r"))) {
        printf("Missing emissivity table: %s\n", filename);
        continue;
      }
      printf("Read emissivity table: %s\n", filename);

      /* Read data... */
      while (fgets(line, LEN, in)) {

        /* Parse line... */
        if (sscanf(line, "%lg %lg %lg %lg", &press, &temp, &u, &eps) != 4)
          continue;

        /* Determine pressure index... */
        if (press != press_old) {
          press_old = press;
          if ((++tbl->np[ig][id]) >= TBLNP)
            ERRMSG("Too many pressure levels!");
          tbl->nt[ig][id][tbl->np[ig][id]] = -1;
        }

        /* Determine temperature index... */
        if (temp != temp_old) {
          temp_old = temp;
          if ((++tbl->nt[ig][id][tbl->np[ig][id]]) >= TBLNT)
            ERRMSG("Too many temperatures!");
          tbl->nu[ig][id][tbl->np[ig][id]]
            [tbl->nt[ig][id][tbl->np[ig][id]]] = -1;
```

```
        }

        /* Determine column density index... */
        if ((eps > eps_old && u > u_old) || tbl->nu[ig][id][tbl->np[ig][id]]
            [tbl->nt[ig][id][tbl->np[ig][id]]] < 0) {
          eps_old = eps;
          u_old = u;
          if ((++tbl->nu[ig][id][tbl->np[ig][id]]
               [tbl->nt[ig][id][tbl->np[ig][id]]]) >= TBLNU) {
            tbl->nu[ig][id][tbl->np[ig][id]]
              [tbl->nt[ig][id][tbl->np[ig][id]]]--;
            continue;
          }
        }

        /* Store data... */
        tbl->p[ig][id][tbl->np[ig][id]] = press;
        tbl->t[ig][id][tbl->np[ig][id]][tbl->nt[ig][id][tbl->np[ig][id]]]
          = temp;
        tbl->u[ig][id][tbl->np[ig][id]][tbl->nt[ig][id][tbl->np[ig][id]]]
          [tbl->nu[ig][id][tbl->np[ig][id]]
           [tbl->nt[ig][id][tbl->np[ig][id]]]] = (float) u;
        tbl->eps[ig][id][tbl->np[ig][id]][tbl->nt[ig][id][tbl->np[ig][id]]]
          [tbl->nu[ig][id][tbl->np[ig][id]]
           [tbl->nt[ig][id][tbl->np[ig][id]]]] = (float) eps;
      }

      /* Increment counters... */
      tbl->np[ig][id]++;
      for (ip = 0; ip < tbl->np[ig][id]; ip++) {
        tbl->nt[ig][id][ip]++;
        for (it = 0; it < tbl->nt[ig][id][ip]; it++)
          tbl->nu[ig][id][ip][it]++;
      }

      /* Close file... */
      fclose(in);
    }

  /* Write info... */
  printf("Initialize source function table...\n");

  /* Loop over channels... */
#pragma omp parallel for default(none) shared(ctl,tbl,ig) private(filename,it,i,n,f,fsum,nu)
  for (id = 0; id < ctl->nd; id++) {

    /* Read filter function... */
    sprintf(filename, "%s_%.4f.filt", ctl->tblbase, ctl->nu[id]);
    read_shape(filename, nu, f, &n);

    /* Compute source function table... */
    for (it = 0; it < TBLNS; it++) {

      /* Set temperature... */
      tbl->st[it] = LIN(0.0, TMIN, TBLNS - 1.0, TMAX, (double) it);

      /* Integrate Planck function... */
      fsum = 0;
      tbl->sr[id][it] = 0;
      for (i = 0; i < n; i++) {
        fsum += f[i];
        tbl->sr[id][it] += f[i] * planck(tbl->st[it], nu[i]);
      }
      tbl->sr[id][it] /= fsum;
    }
  }
}
```

Here is the call graph for this function:

5.33.2.23 void intpol_atm (**ctl_t** * *ctl*, **atm_t** * *atm*, double *z*, double * *p*, double * *t*, double * *q*, double * *k*)

Interpolate atmospheric data.

Definition at line 3521 of file jurassic.c.

```
               {

  int ig, ip, iw;
```

```

  /* Get array index... */
  ip = locate(atm->z, atm->np, z);

  /* Interpolate... */
  *p = EXP(atm->z[ip], atm->p[ip], atm->z[ip + 1], atm->p[ip + 1], z);
  *t = LIN(atm->z[ip], atm->t[ip], atm->z[ip + 1], atm->t[ip + 1], z);
  for (ig = 0; ig < ctl->ng; ig++)
    q[ig] =
      LIN(atm->z[ip], atm->q[ig][ip], atm->z[ip + 1], atm->q[ig][ip + 1], z);
  for (iw = 0; iw < ctl->nw; iw++)
    k[iw] =
      LIN(atm->z[ip], atm->k[iw][ip], atm->z[ip + 1], atm->k[iw][ip + 1], z);
}
```

Here is the call graph for this function:

5.33.2.24 void intpol_tbl (ctl_t * *ctl*, tbl_t * *tbl*, los_t * *los*, int *ip*, double *tau_path[NG][ND]*, double *tau_seg[ND]*)

Get transmittance from look-up tables.

Definition at line 3548 of file jurassic.c.

```
                              {

  double eps, eps00, eps01, eps10, eps11, u;

  int id, ig, ipr, it0, it1;

  /* Initialize... */
  if (ip <= 0)
    for (ig = 0; ig < ctl->ng; ig++)
      for (id = 0; id < ctl->nd; id++)
        tau_path[ig][id] = 1;

  /* Loop over channels... */
  for (id = 0; id < ctl->nd; id++) {

    /* Initialize... */
    tau_seg[id] = 1;

    /* Loop over emitters.... */
    for (ig = 0; ig < ctl->ng; ig++) {

      /* Check size of table (pressure)... */
      if (tbl->np[ig][id] < 2)
        eps = 0;

      /* Check transmittance... */
      else if (tau_path[ig][id] < 1e-9)
        eps = 1;

      /* Interpolate... */
      else {

        /* Determine pressure and temperature indices... */
        ipr = locate(tbl->p[ig][id], tbl->np[ig][id], los->p[ip]);
        it0 = locate(tbl->t[ig][id][ipr], tbl->nt[ig][id][ipr], los->t[ip]);
        it1 =
          locate(tbl->t[ig][id][ipr + 1], tbl->nt[ig][id][ipr + 1],
                 los->t[ip]);

        /* Check size of table (temperature and column density)... */
        if (tbl->nt[ig][id][ipr] < 2 || tbl->nt[ig][id][ipr + 1] < 2
            || tbl->nu[ig][id][ipr][it0] < 2
            || tbl->nu[ig][id][ipr][it0 + 1] < 2
            || tbl->nu[ig][id][ipr + 1][it1] < 2
            || tbl->nu[ig][id][ipr + 1][it1 + 1] < 2)
          eps = 0;

        else {

          /* Get emissivities of extended path... */
          u = intpol_tbl_u(tbl, ig, id, ipr, it0, 1 - tau_path[ig][id]);
          eps00 = intpol_tbl_eps(tbl, ig, id, ipr, it0, u + los->u[ig][ip]);

          u = intpol_tbl_u(tbl, ig, id, ipr, it0 + 1, 1 - tau_path[ig][id]);
          eps01 =
            intpol_tbl_eps(tbl, ig, id, ipr, it0 + 1, u + los->u[ig][ip]);
```

```
          u = intpol_tbl_u(tbl, ig, id, ipr + 1, it1, 1 - tau_path[ig][id]);
          eps10 =
            intpol_tbl_eps(tbl, ig, id, ipr + 1, it1, u + los->u[ig][ip]);

          u =
            intpol_tbl_u(tbl, ig, id, ipr + 1, it1 + 1, 1 - tau_path[ig][id]);
          eps11 =
            intpol_tbl_eps(tbl, ig, id, ipr + 1, it1 + 1, u + los->
      u[ig][ip]);

          /* Interpolate with respect to temperature... */
          eps00 = LIN(tbl->t[ig][id][ipr][it0], eps00,
                      tbl->t[ig][id][ipr][it0 + 1], eps01, los->t[ip]);
          eps11 = LIN(tbl->t[ig][id][ipr + 1][it1], eps10,
                      tbl->t[ig][id][ipr + 1][it1 + 1], eps11, los->t[ip]);

          /* Interpolate with respect to pressure... */
          eps00 = LIN(tbl->p[ig][id][ipr], eps00,
                      tbl->p[ig][id][ipr + 1], eps11, los->p[ip]);

          /* Check emssivity range... */
          eps00 = GSL_MAX(GSL_MIN(eps00, 1), 0);

          /* Determine segment emissivity... */
          eps = 1 - (1 - eps00) / tau_path[ig][id];
        }
      }

      /* Get transmittance of extended path... */
      tau_path[ig][id] *= (1 - eps);

      /* Get segment transmittance... */
      tau_seg[id] *= (1 - eps);
    }
  }
}
```

Here is the call graph for this function:

5.33.2.25 double intpol_tbl_eps (tbl_t * *tbl,* int *ig,* int *id,* int *ip,* int *it,* double *u*)

Interpolate emissivity from look-up tables.

Definition at line 3649 of file jurassic.c.

```
               {

  int idx;

  /* Lower boundary... */
  if (u < tbl->u[ig][id][ip][it][0])
    return LIN(0, 0, tbl->u[ig][id][ip][it][0], tbl->eps[ig][id][ip][it][0],
               u);

  /* Upper boundary... */
  else if (u > tbl->u[ig][id][ip][it][tbl->nu[ig][id][ip][it] - 1])
    return LIN(tbl->u[ig][id][ip][it][tbl->nu[ig][id][ip][it] - 1],
               tbl->eps[ig][id][ip][it][tbl->nu[ig][id][ip][it] - 1],
               1e30, 1, u);

  /* Interpolation... */
  else {

    /* Get index... */
    idx = locate_tbl(tbl->u[ig][id][ip][it], tbl->nu[ig][id][ip][it], u);

    /* Interpolate... */
    return
      LIN(tbl->u[ig][id][ip][it][idx], tbl->eps[ig][id][ip][it][idx],
          tbl->u[ig][id][ip][it][idx + 1], tbl->eps[ig][id][ip][it][idx + 1],
          u);
  }
}
```

Here is the call graph for this function:

5.33.2.26 double intpol_tbl_u (**tbl_t** ∗ *tbl,* int *ig,* int *id,* int *ip,* int *it,* double *eps*)

Interpolate column density from look-up tables.

Definition at line 3686 of file jurassic.c.

```
               {

  int idx;

  /* Lower boundary... */
  if (eps < tbl->eps[ig][id][ip][it][0])
    return LIN(0, 0, tbl->eps[ig][id][ip][it][0], tbl->u[ig][id][ip][it][0],
               eps);

  /* Upper boundary... */
  else if (eps > tbl->eps[ig][id][ip][it][tbl->nu[ig][id][ip][it] - 1])
    return LIN(tbl->eps[ig][id][ip][it][tbl->nu[ig][id][ip][it] - 1],
               tbl->u[ig][id][ip][it][tbl->nu[ig][id][ip][it] - 1],
               1, 1e30, eps);

  /* Interpolation... */
  else {

    /* Get index... */
    idx = locate_tbl(tbl->eps[ig][id][ip][it], tbl->nu[ig][id][ip][it], eps);

    /* Interpolate... */
    return
      LIN(tbl->eps[ig][id][ip][it][idx], tbl->u[ig][id][ip][it][idx],
          tbl->eps[ig][id][ip][it][idx + 1], tbl->u[ig][id][ip][it][idx + 1],
          eps);
  }
}
```

Here is the call graph for this function:

5.33.2.27 void jsec2time (double *jsec,* int ∗ *year,* int ∗ *mon,* int ∗ *day,* int ∗ *hour,* int ∗ *min,* int ∗ *sec,* double ∗ *remain*)

Convert seconds to date.

Definition at line 3723 of file jurassic.c.

```
                   {

  struct tm t0, *t1;

  time_t jsec0;

  t0.tm_year = 100;
  t0.tm_mon = 0;
  t0.tm_mday = 1;
  t0.tm_hour = 0;
  t0.tm_min = 0;
  t0.tm_sec = 0;

  jsec0 = (time_t) jsec + timegm(&t0);
  t1 = gmtime(&jsec0);

  *year = t1->tm_year + 1900;
  *mon = t1->tm_mon + 1;
  *day = t1->tm_mday;
  *hour = t1->tm_hour;
  *min = t1->tm_min;
  *sec = t1->tm_sec;
  *remain = jsec - floor(jsec);
}
```

5.33.2.28 void kernel (ctl_t ∗ *ctl*, atm_t ∗ *atm*, obs_t ∗ *obs*, gsl_matrix ∗ *k*)

Compute Jacobians.

Definition at line 3758 of file jurassic.c.

```
                  {

  atm_t *atm1;
  obs_t *obs1;

  gsl_vector *x0, *x1, *yy0, *yy1;

  int *iqa, j;

  double h;

  size_t i, n, m;

  /* Get sizes... */
  m = k->size1;
  n = k->size2;

  /* Allocate... */
  x0 = gsl_vector_alloc(n);
  yy0 = gsl_vector_alloc(m);
  ALLOC(iqa, int,
        N);

  /* Compute radiance for undisturbed atmospheric data... */
  formod(ctl, atm, obs);

  /* Compose vectors... */
  atm2x(ctl, atm, x0, iqa, NULL);
  obs2y(ctl, obs, yy0, NULL, NULL);

  /* Initialize kernel matrix... */
  gsl_matrix_set_zero(k);

  /* Loop over state vector elements... */
#pragma omp parallel for default(none) shared(ctl,atm,obs,k,x0,yy0,n,m,iqa) private(i, j, h, x1, yy1, atm1,
 obs1)
  for (j = 0; j < (int) n; j++) {

    /* Allocate... */
    x1 = gsl_vector_alloc(n);
    yy1 = gsl_vector_alloc(m);
    ALLOC(atm1, atm_t, 1);
    ALLOC(obs1, obs_t, 1);

    /* Set perturbation size... */
    if (iqa[j] == IDXP)
      h = GSL_MAX(fabs(0.01 * gsl_vector_get(x0, (size_t) j)), 1e-7);
    else if (iqa[j] == IDXT)
      h = 1;
    else if (iqa[j] >= IDXQ(0) && iqa[j] < IDXQ(ctl->ng))
      h = GSL_MAX(fabs(0.01 * gsl_vector_get(x0, (size_t) j)), 1e-15);
    else if (iqa[j] >= IDXK(0) && iqa[j] < IDXK(ctl->nw))
      h = 1e-4;
    else
      ERRMSG("Cannot set perturbation size!");

    /* Disturb state vector element... */
    gsl_vector_memcpy(x1, x0);
    gsl_vector_set(x1, (size_t) j, gsl_vector_get(x1, (size_t) j) + h);
    copy_atm(ctl, atm1, atm, 0);
    copy_obs(ctl, obs1, obs, 0);
    x2atm(ctl, x1, atm1);

    /* Compute radiance for disturbed atmospheric data... */
    formod(ctl, atm1, obs1);

    /* Compose measurement vector for disturbed radiance data... */
    obs2y(ctl, obs1, yy1, NULL, NULL);

    /* Compute derivatives... */
    for (i = 0; i < m; i++)
      gsl_matrix_set(k, i, (size_t) j,
                     (gsl_vector_get(yy1, i) - gsl_vector_get(yy0, i)) / h);

    /* Free... */
    gsl_vector_free(x1);
    gsl_vector_free(yy1);
```

```
    free(atm1);
    free(obs1);
  }

  /* Free... */
  gsl_vector_free(x0);
  gsl_vector_free(yy0);
  free(iqa);
}
```

Here is the call graph for this function:

5.33.2.29 int locate (double ∗ *xx*, int *n*, double *x*)

Find array index.

Definition at line 3850 of file jurassic.c.

```
              {

  int i, ilo, ihi;

  ilo = 0;
  ihi = n - 1;
  i = (ihi + ilo) >> 1;

  if (xx[i] < xx[i + 1])
    while (ihi > ilo + 1) {
      i = (ihi + ilo) >> 1;
      if (xx[i] > x)
        ihi = i;
      else
        ilo = i;
  } else
    while (ihi > ilo + 1) {
      i = (ihi + ilo) >> 1;
      if (xx[i] <= x)
        ihi = i;
      else
        ilo = i;
    }

  return ilo;
}
```

5.33.2.30 int locate_tbl (float ∗ *xx*, int *n*, double *x*)

Find array index in float array.

Definition at line 3882 of file jurassic.c.

```
              {

  int i, ilo, ihi;

  ilo = 0;
  ihi = n - 1;
  i = (ihi + ilo) >> 1;

  while (ihi > ilo + 1) {
    i = (ihi + ilo) >> 1;
    if (xx[i] > x)
      ihi = i;
    else
      ilo = i;
  }

  return ilo;
}
```

5.33.2.31 size_t obs2y (ctl_t * *ctl*, obs_t * *obs*, gsl_vector * *y*, int * *ida*, int * *ira*)

Compose measurement vector.

Definition at line 3906 of file jurassic.c.

```
            {

  int id, ir;

  size_t m = 0;

  /* Determine measurement vector... */
  for (ir = 0; ir < obs->nr; ir++)
    for (id = 0; id < ctl->nd; id++)
      if (gsl_finite(obs->rad[id][ir])) {
        if (y != NULL)
          gsl_vector_set(y, m, obs->rad[id][ir]);
        if (ida != NULL)
          ida[m] = id;
        if (ira != NULL)
          ira[m] = ir;
        m++;
      }

  return m;
}
```

5.33.2.32 double planck (double *t*, double *nu*)

Compute Planck function.

Definition at line 3935 of file jurassic.c.

```
             {

  return C1 * gsl_pow_3(nu) / gsl_expm1(C2 * nu / t);
}
```

5.33.2.33 void raytrace (ctl_t * *ctl*, atm_t * *atm*, obs_t * *obs*, los_t * *los*, int *ir*)

Do ray-tracing to determine LOS.

Definition at line 3944 of file jurassic.c.

```
          {

  double cosa, d, dmax, dmin = 0, ds, ex0[3], ex1[3], frac, h = 0.02, k[NW],
    lat, lon, n, naux, ng[3], norm, p, q[NG], t, x[3], xh[3],
    xobs[3], xvp[3], z = 1e99, zmax, zmin, zrefrac = 60;

  int i, ig, ip, iw, stop = 0;

  /* Initialize... */
  los->np = 0;
  los->tsurf = -999;
  obs->tpz[ir] = obs->vpz[ir];
  obs->tplon[ir] = obs->vplon[ir];
  obs->tplat[ir] = obs->vplat[ir];

  /* Get altitude range of atmospheric data... */
  gsl_stats_minmax(&zmin, &zmax, atm->z, 1, (size_t) atm->np);

  /* Check observer altitude... */
  if (obs->obsz[ir] < zmin)
    ERRMSG("Observer below surface!");

  /* Check view point altitude... */
  if (obs->vpz[ir] > zmax)
    return;
```

```

  /* Determine Cartesian coordinates for observer and view point... */
  geo2cart(obs->obsz[ir], obs->obslon[ir], obs->obslat[ir], xobs);
  geo2cart(obs->vpz[ir], obs->vplon[ir], obs->vplat[ir], xvp);

  /* Determine initial tangent vector... */
  for (i = 0; i < 3; i++)
    ex0[i] = xvp[i] - xobs[i];
  norm = NORM(ex0);
  for (i = 0; i < 3; i++)
    ex0[i] /= norm;

  /* Observer within atmosphere... */
  for (i = 0; i < 3; i++)
    x[i] = xobs[i];

  /* Observer above atmosphere (search entry point)... */
  if (obs->obsz[ir] > zmax) {
    dmax = norm;
    while (fabs(dmin - dmax) > 0.001) {
      d = (dmax + dmin) / 2;
      for (i = 0; i < 3; i++)
        x[i] = xobs[i] + d * ex0[i];
      cart2geo(x, &z, &lon, &lat);
      if (z <= zmax && z > zmax - 0.001)
        break;
      if (z < zmax - 0.0005)
        dmax = d;
      else
        dmin = d;
    }
  }

  /* Ray-tracing... */
  while (1) {

    /* Set step length... */
    ds = ctl->rayds;
    if (ctl->raydz > 0) {
      norm = NORM(x);
      for (i = 0; i < 3; i++)
        xh[i] = x[i] / norm;
      cosa = fabs(DOTP(ex0, xh));
      if (cosa != 0)
        ds = GSL_MIN(ctl->rayds, ctl->raydz / cosa);
    }

    /* Determine geolocation... */
    cart2geo(x, &z, &lon, &lat);

    /* Check if LOS hits the ground or has left atmosphere... */
    if (z < zmin || z > zmax) {
      stop = (z < zmin ? 2 : 1);
      frac =
        ((z <
          zmin ? zmin : zmax) - los->z[los->np - 1]) / (z - los->z[los->np -
                                                                   1]);
      geo2cart(los->z[los->np - 1], los->lon[los->np - 1],
               los->lat[los->np - 1], xh);
      for (i = 0; i < 3; i++)
        x[i] = xh[i] + frac * (x[i] - xh[i]);
      cart2geo(x, &z, &lon, &lat);
      los->ds[los->np - 1] = ds * frac;
      ds = 0;
    }

    /* Interpolate atmospheric data... */
    intpol_atm(ctl, atm, z, &p, &t, q, k);

    /* Save data... */
    los->lon[los->np] = lon;
    los->lat[los->np] = lat;
    los->z[los->np] = z;
    los->p[los->np] = p;
    los->t[los->np] = t;
    for (ig = 0; ig < ctl->ng; ig++)
      los->q[ig][los->np] = q[ig];
    for (iw = 0; iw < ctl->nw; iw++)
      los->k[iw][los->np] = k[iw];
    los->ds[los->np] = ds;

    /* Increment and check number of LOS points... */
    if ((++los->np) > NLOS)
      ERRMSG("Too many LOS points!");

    /* Check stop flag... */
    if (stop) {
```

```
      los->tsurf = (stop == 2 ? t : -999);
      break;
    }

    /* Determine refractivity... */
    if (ctl->refrac && z <= zrefrac)
      n = 1 + refractivity(p, t);
    else
      n = 1;

    /* Construct new tangent vector (first term)... */
    for (i = 0; i < 3; i++)
      ex1[i] = ex0[i] * n;

    /* Compute gradient of refractivity... */
    if (ctl->refrac && z <= zrefrac) {
      for (i = 0; i < 3; i++)
        xh[i] = x[i] + 0.5 * ds * ex0[i];
      cart2geo(xh, &z, &lon, &lat);
      intpol_atm(ctl, atm, z, &p, &t, q, k);
      n = refractivity(p, t);
      for (i = 0; i < 3; i++) {
        xh[i] += h;
        cart2geo(xh, &z, &lon, &lat);
        intpol_atm(ctl, atm, z, &p, &t, q, k);
        naux = refractivity(p, t);
        ng[i] = (naux - n) / h;
        xh[i] -= h;
      }
    } else
      for (i = 0; i < 3; i++)
        ng[i] = 0;

    /* Construct new tangent vector (second term)... */
    for (i = 0; i < 3; i++)
      ex1[i] += ds * ng[i];

    /* Normalize new tangent vector... */
    norm = NORM(ex1);
    for (i = 0; i < 3; i++)
      ex1[i] /= norm;

    /* Determine next point of LOS... */
    for (i = 0; i < 3; i++)
      x[i] += 0.5 * ds * (ex0[i] + ex1[i]);

    /* Copy tangent vector... */
    for (i = 0; i < 3; i++)
      ex0[i] = ex1[i];
  }

  /* Get tangent point (to be done before changing segment lengths!)... */
  tangent_point(los, &obs->tpz[ir], &obs->tplon[ir], &obs->tplat[ir]);

  /* Change segment lengths according to trapezoid rule... */
  for (ip = los->np - 1; ip >= 1; ip--)
    los->ds[ip] = 0.5 * (los->ds[ip - 1] + los->ds[ip]);
  los->ds[0] *= 0.5;

  /* Compute column density... */
  for (ip = 0; ip < los->np; ip++)
    for (ig = 0; ig < ctl->ng; ig++)
      los->u[ig][ip] = 10 * los->q[ig][ip] * los->p[ip]
        / (GSL_CONST_MKSA_BOLTZMANN * los->t[ip]) * los->ds[ip];
}
```

Here is the call graph for this function:

5.33.2.34 void read_atm (const char * *dirname*, const char * *filename*, ctl_t * *ctl*, atm_t * *atm*)

Read atmospheric data.

Definition at line 4129 of file jurassic.c.

```
                  {

  FILE *in;

```

```
  char file[LEN], line[LEN], *tok;

  int ig, iw;

  /* Init... */
  atm->np = 0;

  /* Set filename... */
  if (dirname != NULL)
    sprintf(file, "%s/%s", dirname, filename);
  else
    sprintf(file, "%s", filename);

  /* Write info... */
  printf("Read atmospheric data: %s\n", file);

  /* Open file... */
  if (!(in = fopen(file, "r")))
    ERRMSG("Cannot open file!");

  /* Read line... */
  while (fgets(line, LEN, in)) {

    /* Read data... */
    TOK(line, tok, "%lg", atm->time[atm->np]);
    TOK(NULL, tok, "%lg", atm->z[atm->np]);
    TOK(NULL, tok, "%lg", atm->lon[atm->np]);
    TOK(NULL, tok, "%lg", atm->lat[atm->np]);
    TOK(NULL, tok, "%lg", atm->p[atm->np]);
    TOK(NULL, tok, "%lg", atm->t[atm->np]);
    for (ig = 0; ig < ctl->ng; ig++)
      TOK(NULL, tok, "%lg", atm->q[ig][atm->np]);
    for (iw = 0; iw < ctl->nw; iw++)
      TOK(NULL, tok, "%lg", atm->k[iw][atm->np]);

    /* Increment data point counter... */
    if ((++atm->np) > NP)
      ERRMSG("Too many data points!");
  }

  /* Close file... */
  fclose(in);

  /* Check number of points... */
  if (atm->np < 1)
    ERRMSG("Could not read any data!");
}
```

5.33.2.35 void read_ctl (int *argc*, char ∗ *argv[ ]*, **ctl_t** ∗ *ctl*)

Read forward model control parameters.

Definition at line 4187 of file jurassic.c.

```
                 {

  int id, ig, iw;

  /* Write info... */
  printf("\nJuelich Rapid Spectral Simulation Code (JURASSIC)\n"
         "(executable: %s | compiled: %s, %s)\n\n",
         argv[0], __DATE__, __TIME__);

  /* Emitters... */
  ctl->ng = (int) scan_ctl(argc, argv, "NG", -1, "0", NULL);
  if (ctl->ng < 0 || ctl->ng > NG)
    ERRMSG("Set 0 <= NG <= MAX!");
  for (ig = 0; ig < ctl->ng; ig++)
    scan_ctl(argc, argv, "EMITTER", ig, "", ctl->emitter[ig]);

  /* Radiance channels... */
  ctl->nd = (int) scan_ctl(argc, argv, "ND", -1, "0", NULL);
  if (ctl->nd < 0 || ctl->nd > ND)
    ERRMSG("Set 0 <= ND <= MAX!");
  for (id = 0; id < ctl->nd; id++)
    ctl->nu[id] = scan_ctl(argc, argv, "NU", id, "", NULL);

  /* Spectral windows... */
  ctl->nw = (int) scan_ctl(argc, argv, "NW", -1, "1", NULL);
  if (ctl->nw < 0 || ctl->nw > NW)
```

```
    ERRMSG("Set 0 <= NW <= MAX!");
  for (id = 0; id < ctl->nd; id++)
    ctl->window[id] = (int) scan_ctl(argc, argv, "WINDOW", id, "0", NULL);

  /* Emissivity look-up tables... */
  scan_ctl(argc, argv, "TBLBASE", -1, "-", ctl->tblbase);

  /* Hydrostatic equilibrium... */
  ctl->hydz = scan_ctl(argc, argv, "HYDZ", -1, "-999", NULL);

  /* Continua... */
  ctl->ctm_co2 = (int) scan_ctl(argc, argv, "CTM_CO2", -1, "1", NULL);
  ctl->ctm_h2o = (int) scan_ctl(argc, argv, "CTM_H2O", -1, "1", NULL);
  ctl->ctm_n2 = (int) scan_ctl(argc, argv, "CTM_N2", -1, "1", NULL);
  ctl->ctm_o2 = (int) scan_ctl(argc, argv, "CTM_O2", -1, "1", NULL);

  /* Ray-tracing... */
  ctl->refrac = (int) scan_ctl(argc, argv, "REFRAC", -1, "1", NULL);
  ctl->rayds = scan_ctl(argc, argv, "RAYDS", -1, "10", NULL);
  ctl->raydz = scan_ctl(argc, argv, "RAYDZ", -1, "0.5", NULL);

  /* Field of view... */
  scan_ctl(argc, argv, "FOV", -1, "-", ctl->fov);

  /* Retrieval interface... */
  ctl->retp_zmin = scan_ctl(argc, argv, "RETP_ZMIN", -1, "-999", NULL);
  ctl->retp_zmax = scan_ctl(argc, argv, "RETP_ZMAX", -1, "-999", NULL);
  ctl->rett_zmin = scan_ctl(argc, argv, "RETT_ZMIN", -1, "-999", NULL);
  ctl->rett_zmax = scan_ctl(argc, argv, "RETT_ZMAX", -1, "-999", NULL);
  for (ig = 0; ig < ctl->ng; ig++) {
    ctl->retq_zmin[ig] = scan_ctl(argc, argv, "RETQ_ZMIN", ig, "-999", NULL);
    ctl->retq_zmax[ig] = scan_ctl(argc, argv, "RETQ_ZMAX", ig, "-999", NULL);
  }
  for (iw = 0; iw < ctl->nw; iw++) {
    ctl->retk_zmin[iw] = scan_ctl(argc, argv, "RETK_ZMIN", iw, "-999", NULL);
    ctl->retk_zmax[iw] = scan_ctl(argc, argv, "RETK_ZMAX", iw, "-999", NULL);
  }

  /* Output flags... */
  ctl->write_bbt = (int) scan_ctl(argc, argv, "WRITE_BBT", -1, "0", NULL);
  ctl->write_matrix =
    (int) scan_ctl(argc, argv, "WRITE_MATRIX", -1, "0", NULL);
}
```

Here is the call graph for this function:

5.33.2.36 void read_matrix (const char ∗ *dirname*, const char ∗ *filename*, gsl_matrix ∗ *matrix*)

Read matrix.

Definition at line 4262 of file jurassic.c.

```
                       {

  FILE *in;

  char dum[LEN], file[LEN], line[LEN];

  double value;

  int i, j;

  /* Set filename... */
  if (dirname != NULL)
    sprintf(file, "%s/%s", dirname, filename);
  else
    sprintf(file, "%s", filename);

  /* Write info... */
  printf("Read matrix: %s\n", file);

  /* Open file... */
  if (!(in = fopen(file, "r")))
    ERRMSG("Cannot open file!");

  /* Read data... */
  gsl_matrix_set_zero(matrix);
  while (fgets(line, LEN, in))
    if (sscanf(line, "%d %s %s %s %s %s %d %s %s %s %s %s %lg",
```

```
                 &i, dum, dum, dum, dum, dum,
                 &j, dum, dum, dum, dum, dum, &value) == 13)
        gsl_matrix_set(matrix, (size_t) i, (size_t) j, value);

    /* Close file... */
    fclose(in);
}
```

5.33.2.37 void read_obs (const char ∗ *dirname*, const char ∗ *filename*, ctl_t ∗ *ctl*, obs_t ∗ *obs*)

Read observation data.

Definition at line 4302 of file jurassic.c.

```
                 {

  FILE *in;

  char file[LEN], line[LEN], *tok;

  int id;

  /* Init... */
  obs->nr = 0;

  /* Set filename... */
  if (dirname != NULL)
    sprintf(file, "%s/%s", dirname, filename);
  else
    sprintf(file, "%s", filename);

  /* Write info... */
  printf("Read observation data: %s\n", file);

  /* Open file... */
  if (!(in = fopen(file, "r")))
    ERRMSG("Cannot open file!");

  /* Read line... */
  while (fgets(line, LEN, in)) {

    /* Read data... */
    TOK(line, tok, "%lg", obs->time[obs->nr]);
    TOK(NULL, tok, "%lg", obs->obsz[obs->nr]);
    TOK(NULL, tok, "%lg", obs->obslon[obs->nr]);
    TOK(NULL, tok, "%lg", obs->obslat[obs->nr]);
    TOK(NULL, tok, "%lg", obs->vpz[obs->nr]);
    TOK(NULL, tok, "%lg", obs->vplon[obs->nr]);
    TOK(NULL, tok, "%lg", obs->vplat[obs->nr]);
    TOK(NULL, tok, "%lg", obs->tpz[obs->nr]);
    TOK(NULL, tok, "%lg", obs->tplon[obs->nr]);
    TOK(NULL, tok, "%lg", obs->tplat[obs->nr]);
    for (id = 0; id < ctl->nd; id++)
      TOK(NULL, tok, "%lg", obs->rad[id][obs->nr]);
    for (id = 0; id < ctl->nd; id++)
      TOK(NULL, tok, "%lg", obs->tau[id][obs->nr]);

    /* Increment counter... */
    if ((++obs->nr) > NR)
      ERRMSG("Too many rays!");
  }

  /* Close file... */
  fclose(in);

  /* Check number of points... */
  if (obs->nr < 1)
    ERRMSG("Could not read any data!");
}
```

5.33.2.38 void read_shape (const char ∗ *filename*, double ∗ *x*, double ∗ *y*, int ∗ *n*)

Read shape function.

Definition at line 4364 of file jurassic.c.

```
        {

  FILE *in;

  char line[LEN];

  /* Write info... */
  printf("Read shape function: %s\n", filename);

  /* Open file... */
  if (!(in = fopen(filename, "r")))
    ERRMSG("Cannot open file!");

  /* Read data... */
  *n = 0;
  while (fgets(line, LEN, in))
    if (sscanf(line, "%lg %lg", &x[*n], &y[*n]) == 2)
      if ((++(*n)) > NSHAPE)
        ERRMSG("Too many data points!");

  /* Check number of points... */
  if (*n < 1)
    ERRMSG("Could not read any data!");

  /* Close file... */
  fclose(in);
}
```

5.33.2.39 double refractivity (double *p*, double *t*)

Compute refractivity (return value is n - 1).

Definition at line 4398 of file jurassic.c.

```
          {

  /* Refractivity of air at 4 to 15 micron... */
  return 7.753e-05 * p / t;
}
```

5.33.2.40 double scan_ctl (int *argc*, char ∗ *argv[ ]*, const char ∗ *varname*, int *arridx*, const char ∗ *defvalue*, char ∗ *value*)

Search control parameter file for variable entry.

Definition at line 4408 of file jurassic.c.

```
               {

  FILE *in = NULL;

  char dummy[LEN], fullname1[LEN], fullname2[LEN], line[LEN],
    msg[LEN], rvarname[LEN], rval[LEN];

  int contain = 0, i;

  /* Open file... */
  if (argv[1][0] != '-')
    if (!(in = fopen(argv[1], "r")))
      ERRMSG("Cannot open file!");

  /* Set full variable name... */
  if (arridx >= 0) {
    sprintf(fullname1, "%s[%d]", varname, arridx);
    sprintf(fullname2, "%s[*]", varname);
  } else {
    sprintf(fullname1, "%s", varname);
    sprintf(fullname2, "%s", varname);
  }

  /* Read data... */
  if (in != NULL)
    while (fgets(line, LEN, in))
      if (sscanf(line, "%s %s %s", rvarname, dummy, rval) == 3)
        if (strcasecmp(rvarname, fullname1) == 0 ||
```

```
          strcasecmp(rvarname, fullname2) == 0) {
        contain = 1;
        break;
      }
  for (i = 1; i < argc - 1; i++)
    if (strcasecmp(argv[i], fullname1) == 0 ||
        strcasecmp(argv[i], fullname2) == 0) {
      sprintf(rval, "%s", argv[i + 1]);
      contain = 1;
      break;
    }

  /* Close file... */
  if (in != NULL)
    fclose(in);

  /* Check for missing variables... */
  if (!contain) {
    if (strlen(defvalue) > 0)
      sprintf(rval, "%s", defvalue);
    else {
      sprintf(msg, "Missing variable %s!\n", fullname1);
      ERRMSG(msg);
    }
  }

  /* Write info... */
  printf("%s = %s\n", fullname1, rval);

  /* Return values... */
  if (value != NULL)
    sprintf(value, "%s", rval);
  return atof(rval);
}
```

5.33.2.41 void tangent_point (los_t * los, double * tpz, double * tplon, double * tplat)

Find tangent point of a given LOS.

Definition at line 4479 of file jurassic.c.

```
                  {

  double a, b, c, dummy, v[3], v0[3], v2[3], x, x1, x2, yy0, yy1, yy2;

  size_t i, ip;

  /* Find minimum altitude... */
  ip = gsl_stats_min_index(los->z, 1, (size_t) los->np);

  /* Nadir or zenith... */
  if (ip <= 0 || ip >= (size_t) los->np - 1) {
    *tpz = los->z[los->np - 1];
    *tplon = los->lon[los->np - 1];
    *tplat = los->lat[los->np - 1];
  }

  /* Limb... */
  else {

    /* Determine interpolating polynomial y=a*x^2+b*x+c... */
    yy0 = los->z[ip - 1];
    yy1 = los->z[ip];
    yy2 = los->z[ip + 1];
    x1 = sqrt(gsl_pow_2(los->ds[ip]) - gsl_pow_2(yy1 - yy0));
    x2 = x1 + sqrt(gsl_pow_2(los->ds[ip + 1]) - gsl_pow_2(yy2 - yy1));
    a = 1 / (x1 - x2) * (-(yy0 - yy1) / x1 + (yy0 - yy2) / x2);
    b = -(yy0 - yy1) / x1 - a * x1;
    c = yy0;

    /* Get tangent point location... */
    x = -b / (2 * a);
    *tpz = a * x * x + b * x + c;
    geo2cart(los->z[ip - 1], los->lon[ip - 1], los->lat[ip - 1], v0);
    geo2cart(los->z[ip + 1], los->lon[ip + 1], los->lat[ip + 1], v2);
    for (i = 0; i < 3; i++)
      v[i] = LIN(0.0, v0[i], x2, v2[i], x);
    cart2geo(v, &dummy, tplon, tplat);
  }
}
```

Here is the call graph for this function:

5.33.2.42 void time2jsec (int *year,* int *mon,* int *day,* int *hour,* int *min,* int *sec,* double *remain,* double ∗ *jsec*)

Convert date to seconds.

Definition at line 4525 of file jurassic.c.

```
                  {

  struct tm t0, t1;

  t0.tm_year = 100;
  t0.tm_mon = 0;
  t0.tm_mday = 1;
  t0.tm_hour = 0;
  t0.tm_min = 0;
  t0.tm_sec = 0;

  t1.tm_year = year - 1900;
  t1.tm_mon = mon - 1;
  t1.tm_mday = day;
  t1.tm_hour = hour;
  t1.tm_min = min;
  t1.tm_sec = sec;

  *jsec = (double) timegm(&t1) - (double) timegm(&t0) + remain;
}
```

5.33.2.43 void timer (const char ∗ *name,* const char ∗ *file,* const char ∗ *func,* int *line,* int *mode*)

Measure wall-clock time.

Definition at line 4556 of file jurassic.c.

```
              {

  static double dt_w, w0[10];

  static int l0[10], nt;

  struct timeval tim;

  /* Start new timer... */
  if (mode == 1) {
    gettimeofday(&tim, NULL);
    w0[nt] = (double) tim.tv_sec + (double) tim.tv_usec / 1e6;
    l0[nt] = line;
    if ((++nt) >= 10)
      ERRMSG("Too many timers!");
  }

  /* Write elapsed time... */
  else {

    /* Check timer index... */
    if (nt - 1 < 0)
      ERRMSG("Coding error!");

    /* Get time differences... */
    gettimeofday(&tim, NULL);
    dt_w = (double) tim.tv_sec + (double) tim.tv_usec / 1e6 - w0[nt - 1];

    /* Write elapsed time... */
    printf("Timer '%s' (%s, %s, l%d-%d): %.3f sec\n",
           name, file, func, l0[nt - 1], line, dt_w);
  }

  /* Stop timer... */
  if (mode == 3)
    nt--;
}
```

5.33.2.44 void write_atm (const char ∗ *dirname*, const char ∗ *filename*, ctl_t ∗ *ctl*, atm_t ∗ *atm*)

Write atmospheric data.

Definition at line 4601 of file jurassic.c.

```
                {

  FILE *out;

  char file[LEN];

  int ig, ip, iw, n = 6;

  /* Set filename... */
  if (dirname != NULL)
    sprintf(file, "%s/%s", dirname, filename);
  else
    sprintf(file, "%s", filename);

  /* Write info... */
  printf("Write atmospheric data: %s\n", file);

  /* Create file... */
  if (!(out = fopen(file, "w")))
    ERRMSG("Cannot create file!");

  /* Write header... */
  fprintf(out,
          "# $1 = time (seconds since 2000-01-01T00:00Z)\n"
          "# $2 = altitude [km]\n"
          "# $3 = longitude [deg]\n"
          "# $4 = latitude [deg]\n"
          "# $5 = pressure [hPa]\n" "# $6 = temperature [K]\n");
  for (ig = 0; ig < ctl->ng; ig++)
    fprintf(out, "# $%d = %s volume mixing ratio\n", ++n, ctl->emitter[ig]);
  for (iw = 0; iw < ctl->nw; iw++)
    fprintf(out, "# $%d = window %d: extinction [1/km]\n", ++n, iw);

  /* Write data... */
  for (ip = 0; ip < atm->np; ip++) {
    if (ip == 0 || atm->lat[ip] != atm->lat[ip - 1]
        || atm->lon[ip] != atm->lon[ip - 1])
      fprintf(out, "\n");
    fprintf(out, "%.2f %g %g %g %g %g", atm->time[ip], atm->z[ip],
            atm->lon[ip], atm->lat[ip], atm->p[ip], atm->t[ip]);
    for (ig = 0; ig < ctl->ng; ig++)
      fprintf(out, " %g", atm->q[ig][ip]);
    for (iw = 0; iw < ctl->nw; iw++)
      fprintf(out, " %g", atm->k[iw][ip]);
    fprintf(out, "\n");
  }

  /* Close file... */
  fclose(out);
}
```

5.33.2.45 void write_matrix (const char ∗ *dirname*, const char ∗ *filename*, ctl_t ∗ *ctl*, gsl_matrix ∗ *matrix*, atm_t ∗ *atm*, obs_t ∗ *obs*, const char ∗ *rowspace*, const char ∗ *colspace*, const char ∗ *sort*)

Write matrix.

Definition at line 4658 of file jurassic.c.

```
                     {

  FILE *out;

  char file[LEN], quantity[LEN];

  int *cida, *ciqa, *cipa, *cira, *rida, *riqa, *ripa, *rira;

  size_t i, j, nc, nr;

  /* Check output flag... */
  if (!ctl->write_matrix)
```

```
    return;

  /* Allocate... */
  ALLOC(cida, int, M);
  ALLOC(ciqa, int,
        N);
  ALLOC(cipa, int,
        N);
  ALLOC(cira, int,
        M);
  ALLOC(rida, int,
        M);
  ALLOC(riqa, int,
        N);
  ALLOC(ripa, int,
        N);
  ALLOC(rira, int,
        M);

  /* Set filename... */
  if (dirname != NULL)
    sprintf(file, "%s/%s", dirname, filename);
  else
    sprintf(file, "%s", filename);

  /* Write info... */
  printf("Write matrix: %s\n", file);

  /* Create file... */
  if (!(out = fopen(file, "w")))
    ERRMSG("Cannot create file!");

  /* Write header (row space)... */
  if (rowspace[0] == 'y') {

    fprintf(out,
            "# $1 = Row: index (measurement space)\n"
            "# $2 = Row: channel wavenumber [cm^-1]\n"
            "# $3 = Row: time (seconds since 2000-01-01T00:00Z)\n"
            "# $4 = Row: view point altitude [km]\n"
            "# $5 = Row: view point longitude [deg]\n"
            "# $6 = Row: view point latitude [deg]\n");

    /* Get number of rows... */
    nr = obs2y(ctl, obs, NULL, rida, rira);

  } else {

    fprintf(out,
            "# $1 = Row: index (state space)\n"
            "# $2 = Row: name of quantity\n"
            "# $3 = Row: time (seconds since 2000-01-01T00:00Z)\n"
            "# $4 = Row: altitude [km]\n"
            "# $5 = Row: longitude [deg]\n" "# $6 = Row: latitude [deg]\n");

    /* Get number of rows... */
    nr = atm2x(ctl, atm, NULL, riqa, ripa);
  }

  /* Write header (column space)... */
  if (colspace[0] == 'y') {

    fprintf(out,
            "# $7 = Col: index (measurement space)\n"
            "# $8 = Col: channel wavenumber [cm^-1]\n"
            "# $9 = Col: time (seconds since 2000-01-01T00:00Z)\n"
            "# $10 = Col: view point altitude [km]\n"
            "# $11 = Col: view point longitude [deg]\n"
            "# $12 = Col: view point latitude [deg]\n");

    /* Get number of columns... */
    nc = obs2y(ctl, obs, NULL, cida, cira);

  } else {

    fprintf(out,
            "# $7 = Col: index (state space)\n"
            "# $8 = Col: name of quantity\n"
            "# $9 = Col: time (seconds since 2000-01-01T00:00Z)\n"
            "# $10 = Col: altitude [km]\n"
            "# $11 = Col: longitude [deg]\n" "# $12 = Col: latitude [deg]\n");

    /* Get number of columns... */
    nc = atm2x(ctl, atm, NULL, ciqa, cipa);
  }

  /* Write header entry... */
```

```
  fprintf(out, "# $13 = Matrix element\n\n");

  /* Write matrix data... */
  i = j = 0;
  while (i < nr && j < nc) {

    /* Write info about the row... */
    if (rowspace[0] == 'y')
      fprintf(out, "%d %g %.2f %g %g %g",
              (int) i, ctl->nu[rida[i]],
              obs->time[rira[i]], obs->vpz[rira[i]],
              obs->vplon[rira[i]], obs->vplat[rira[i]]);
    else {
      idx2name(ctl, riqa[i], quantity);
      fprintf(out, "%d %s %.2f %g %g %g", (int) i, quantity,
              atm->time[ripa[i]], atm->z[ripa[i]],
              atm->lon[ripa[i]], atm->lat[ripa[i]]);
    }

    /* Write info about the column... */
    if (colspace[0] == 'y')
      fprintf(out, " %d %g %.2f %g %g %g",
              (int) j, ctl->nu[cida[j]],
              obs->time[cira[j]], obs->vpz[cira[j]],
              obs->vplon[cira[j]], obs->vplat[cira[j]]);
    else {
      idx2name(ctl, ciqa[j], quantity);
      fprintf(out, " %d %s %.2f %g %g %g", (int) j, quantity,
              atm->time[cipa[j]], atm->z[cipa[j]],
              atm->lon[cipa[j]], atm->lat[cipa[j]]);
    }

    /* Write matrix entry... */
    fprintf(out, " %g\n", gsl_matrix_get(matrix, i, j));

    /* Set matrix indices... */
    if (sort[0] == 'r') {
      j++;
      if (j >= nc) {
        j = 0;
        i++;
        fprintf(out, "\n");
      }
    } else {
      i++;
      if (i >= nr) {
        i = 0;
        j++;
        fprintf(out, "\n");
      }
    }
  }

  /* Close file... */
  fclose(out);

  /* Free... */
  free(cida);
  free(ciqa);
  free(cipa);
  free(cira);
  free(rida);
  free(riqa);
  free(ripa);
  free(rira);
}
```

Here is the call graph for this function:

5.33.2.46 void write_obs (const char ∗ *dirname*, const char ∗ *filename*, ctl_t ∗ *ctl*, obs_t ∗ *obs*)

Write observation data.

Definition at line 4835 of file jurassic.c.

```
                  {

  FILE *out;

```

```
04843   char file[LEN];
04844
04845   int id, ir, n = 10;
04846
04847   /* Set filename... */
04848   if (dirname != NULL)
04849     sprintf(file, "%s/%s", dirname, filename);
04850   else
04851     sprintf(file, "%s", filename);
04852
04853   /* Write info... */
04854   printf("Write observation data: %s\n", file);
04855
04856   /* Create file... */
04857   if (!(out = fopen(file, "w")))
04858     ERRMSG("Cannot create file!");
04859
04860   /* Write header... */
04861   fprintf(out,
04862           "# $1 = time (seconds since 2000-01-01T00:00Z)\n"
04863           "# $2 = observer altitude [km]\n"
04864           "# $3 = observer longitude [deg]\n"
04865           "# $4 = observer latitude [deg]\n"
04866           "# $5 = view point altitude [km]\n"
04867           "# $6 = view point longitude [deg]\n"
04868           "# $7 = view point latitude [deg]\n"
04869           "# $8 = tangent point altitude [km]\n"
04870           "# $9 = tangent point longitude [deg]\n"
04871           "# $10 = tangent point latitude [deg]\n");
04872   for (id = 0; id < ctl->nd; id++)
04873     fprintf(out, "# $%d = channel %g: radiance [W/(m^2 sr cm^-1)]\n",
04874             ++n, ctl->nu[id]);
04875   for (id = 0; id < ctl->nd; id++)
04876     fprintf(out, "# $%d = channel %g: transmittance\n", ++n, ctl->nu[id]);
04877
04878   /* Write data... */
04879   for (ir = 0; ir < obs->nr; ir++) {
04880     if (ir == 0 || obs->time[ir] != obs->time[ir - 1])
04881       fprintf(out, "\n");
04882     fprintf(out, "%.2f %g %g %g %g %g %g %g %g %g", obs->time[ir],
04883             obs->obsz[ir], obs->obslon[ir], obs->obslat[ir],
04884             obs->vpz[ir], obs->vplon[ir], obs->vplat[ir],
04885             obs->tpz[ir], obs->tplon[ir], obs->tplat[ir]);
04886     for (id = 0; id < ctl->nd; id++)
04887       fprintf(out, " %g", obs->rad[id][ir]);
04888     for (id = 0; id < ctl->nd; id++)
04889       fprintf(out, " %g", obs->tau[id][ir]);
04890     fprintf(out, "\n");
04891   }
04892
04893   /* Close file... */
04894   fclose(out);
04895 }
```

5.33.2.47 void x2atm (ctl_t * *ctl*, gsl_vector * *x*, atm_t * *atm*)

Decompose parameter vector or state vector.

Definition at line 4899 of file jurassic.c.

```
04902                 {
04903
04904   int ig, iw;
04905
04906   size_t n = 0;
04907
04908   /* Set pressure... */
04909   x2atm_help(atm, ctl->retp_zmin, ctl->retp_zmax, atm->
      p, x, &n);
04910
04911   /* Set temperature... */
04912   x2atm_help(atm, ctl->rett_zmin, ctl->rett_zmax, atm->
      t, x, &n);
04913
04914   /* Set volume mixing ratio... */
04915   for (ig = 0; ig < ctl->ng; ig++)
04916     x2atm_help(atm, ctl->retq_zmin[ig], ctl->retq_zmax[ig],
04917                atm->q[ig], x, &n);
04918
04919   /* Set extinction... */
04920   for (iw = 0; iw < ctl->nw; iw++)
04921     x2atm_help(atm, ctl->retk_zmin[iw], ctl->retk_zmax[iw],
04922                atm->k[iw], x, &n);
04923 }
```

Here is the call graph for this function:

5.33.2.48 void x2atm_help (atm_t * *atm*, double *zmin*, double *zmax*, double * *value*, gsl_vector * *x*, size_t * *n*)

Extract elements from state vector.

Definition at line 4927 of file jurassic.c.

```
                   {

  int ip;

  /* Extract state vector elements... */
  for (ip = 0; ip < atm->np; ip++)
    if (atm->z[ip] >= zmin && atm->z[ip] <= zmax) {
      value[ip] = gsl_vector_get(x, *n);
      (*n)++;
    }
}
```

5.33.2.49 void y2obs (ctl_t * *ctl*, gsl_vector * *y*, obs_t * *obs*)

Decompose measurement vector.

Definition at line 4947 of file jurassic.c.

```
                    {

  int id, ir;

  size_t m = 0;

  /* Decompose measurement vector... */
  for (ir = 0; ir < obs->nr; ir++)
    for (id = 0; id < ctl->nd; id++)
      if (gsl_finite(obs->rad[id][ir])) {
        obs->rad[id][ir] = gsl_vector_get(y, m);
        m++;
      }
}
```

5.34 jurassic.h

```
/*
  This file is part of JURASSIC.

  JURASSIC is free software: you can redistribute it and/or modify
  it under the terms of the GNU General Public License as published by
  the Free Software Foundation, either version 3 of the License, or
  (at your option) any later version.

  JURASSIC is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
  GNU General Public License for more details.

  You should have received a copy of the GNU General Public License
  along with JURASSIC. If not, see <http://www.gnu.org/licenses/>.

  Copright (C) 2003-2015 Forschungszentrum Juelich GmbH
*/

#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <sys/time.h>
#include <gsl/gsl_math.h>
#include <gsl/gsl_blas.h>
```

```
#include <gsl/gsl_const_mksa.h>
#include <gsl/gsl_const_num.h>
#include <gsl/gsl_linalg.h>
#include <gsl/gsl_statistics.h>

/* ------------------------------------------------------------
   Macros...
   ------------------------------------------------------------ */

#define ALLOC(ptr, type, n)                                  \
  if((ptr=malloc((size_t)(n)*sizeof(type)))==NULL)          \
    ERRMSG("Out of memory!");

#define DIST(a, b) sqrt(DIST2(a, b))

#define DIST2(a, b)                                                        \
  ((a[0]-b[0])*(a[0]-b[0])+(a[1]-b[1])*(a[1]-b[1])+(a[2]-b[2])*(a[2]-b[2]))

#define DOTP(a, b) (a[0]*b[0]+a[1]*b[1]+a[2]*b[2])

#define ERRMSG(msg) {                                                      \
    printf("\nError (%s, %s, l%d): %s\n\n",                                \
           __FILE__, __func__, __LINE__, msg);                              \
    exit(EXIT_FAILURE);                                                     \
  }

#define EXP(x0, y0, x1, y1, x)                                     \
  (((y0)>0 && (y1)>0)                                               \
   ? ((y0)*exp(log((y1)/(y0))/((x1)-(x0))*((x)-(x0))))             \
   : LIN(x0, y0, x1, y1, x))

#define LIN(x0, y0, x1, y1, x)                         \
  ((y0)+((y1)-(y0))/((x1)-(x0))*((x)-(x0)))

#define NORM(a) sqrt(DOTP(a, a))

#define PRINT(format, var)                                                 \
  printf("Print (%s, %s, l%d): %s= "format"\n",                           \
         __FILE__, __func__, __LINE__, #var, var);

#define TIMER(name, mode)                                    \
  {timer(name, __FILE__, __func__, __LINE__, mode);}

#define TOK(line, tok, format, var) {                        \
    if(((tok)=strtok((line), " \t"))) {                     \
      if(sscanf(tok, format, &(var))!=1) continue;          \
    } else ERRMSG("Error while reading!");                  \
  }

/* ------------------------------------------------------------
   Constants...
   ------------------------------------------------------------ */

#define C1 1.19104259e-8

#define C2 1.43877506

#define TMIN 100.

#define TMAX 400.

#define G0 9.80665

#define P0 1013.25

#define T0 273.15

#define RE 6367.421

#define ME 5.976e24

/* ------------------------------------------------------------
   Dimensions...
   ------------------------------------------------------------ */

#define ND 50

#define NG 20

#define NP 1000

#define NR 1000

#define NW 5

#define LEN 5000

```

```
#define M (NR*ND)

#define N (NQ*NP)

#define NQ (2+NG+NW)

#define NLOS 1000

#define NSHAPE 10000

#define NFOV 5

#define TBLNP 41

#define TBLNT 30

#define TBLNU 320

#define TBLNS 1200

/* ------------------------------------------------------------
   Quantity indices...
   ------------------------------------------------------------ */

#define IDXP 0

#define IDXT 1

#define IDXQ(ig) (2+ig)

#define IDXK(iw) (2+ctl->ng+iw)

/* ------------------------------------------------------------
   Structs...
   ------------------------------------------------------------ */

typedef struct {

  int np;

  double time[NP];

  double z[NP];

  double lon[NP];

  double lat[NP];

  double p[NP];

  double t[NP];

  double q[NG][NP];

  double k[NW][NP];

} atm_t;

typedef struct {

  int ng;

  char emitter[NG][LEN];

  int nd;

  int nw;

  double nu[ND];

  int window[ND];

  char tblbase[LEN];

  double hydz;

  int ctm_co2;

  int ctm_h2o;

  int ctm_n2;

  int ctm_o2;

  int refrac;

  double rayds;
```

```
  double raydz;
  char fov[LEN];
  double retp_zmin;
  double retp_zmax;
  double rett_zmin;
  double rett_zmax;
  double retq_zmin[NG];
  double retq_zmax[NG];
  double retk_zmin[NW];
  double retk_zmax[NW];
  int write_bbt;
  int write_matrix;
} ctl_t;
typedef struct {
  int np;
  double z[NLOS];
  double lon[NLOS];
  double lat[NLOS];
  double p[NLOS];
  double t[NLOS];
  double q[NG][NLOS];
  double k[NW][NLOS];
  double tsurf;
  double ds[NLOS];
  double u[NG][NLOS];
} los_t;
typedef struct {
  int nr;
  double time[NR];
  double obsz[NR];
  double obslon[NR];
  double obslat[NR];
  double vpz[NR];
  double vplon[NR];
  double vplat[NR];
  double tpz[NR];
  double tplon[NR];
  double tplat[NR];
  double tau[ND][NR];
  double rad[ND][NR];
} obs_t;
typedef struct {
  int np[NG][ND];
```

```
  int nt[NG][ND][TBLNP];

  int nu[NG][ND][TBLNP][TBLNT];

  double p[NG][ND][TBLNP];

  double t[NG][ND][TBLNP][TBLNT];

  float u[NG][ND][TBLNP][TBLNT][TBLNU];

  float eps[NG][ND][TBLNP][TBLNT][TBLNU];

  double st[TBLNS];

  double sr[ND][TBLNS];

} tbl_t;

/* ------------------------------------------------------------
   Functions...
   ------------------------------------------------------------ */

size_t atm2x(
  ctl_t * ctl,
  atm_t * atm,
  gsl_vector * x,
  int *iqa,
  int *ipa);

void atm2x_help(
  atm_t * atm,
  double zmin,
  double zmax,
  double *value,
  int val_iqa,
  gsl_vector * x,
  int *iqa,
  int *ipa,
  size_t * n);

double brightness(
  double rad,
  double nu);

void cart2geo(
  double *x,
  double *z,
  double *lon,
  double *lat);

void climatology(
  ctl_t * ctl,
  atm_t * atm_mean);

double ctmco2(
  double nu,
  double p,
  double t,
  double u);

double ctmh2o(
  double nu,
  double p,
  double t,
  double q,
  double u);

double ctmn2(
  double nu,
  double p,
  double t);

double ctmo2(
  double nu,
  double p,
  double t);

void copy_atm(
  ctl_t * ctl,
  atm_t * atm_dest,
  atm_t * atm_src,
  int init);

void copy_obs(
  ctl_t * ctl,
  obs_t * obs_dest,
  obs_t * obs_src,
```

```
  int init);

int find_emitter(
  ctl_t * ctl,
  const char *emitter);

void formod(
  ctl_t * ctl,
  atm_t * atm,
  obs_t * obs);

void formod_continua(
  ctl_t * ctl,
  los_t * los,
  int ip,
  double *beta);

void formod_fov(
  ctl_t * ctl,
  obs_t * obs);

void formod_pencil(
  ctl_t * ctl,
  atm_t * atm,
  obs_t * obs,
  int ir);

void formod_srcfunc(
  ctl_t * ctl,
  tbl_t * tbl,
  double t,
  double *src);

void geo2cart(
  double z,
  double lon,
  double lat,
  double *x);

double gravity(
  double z,
  double lat);

void hydrostatic(
  ctl_t * ctl,
  atm_t * atm);

void idx2name(
  ctl_t * ctl,
  int idx,
  char *quantity);

void init_tbl(
  ctl_t * ctl,
  tbl_t * tbl);

void intpol_atm(
  ctl_t * ctl,
  atm_t * atm,
  double z,
  double *p,
  double *t,
  double *q,
  double *k);

void intpol_tbl(
  ctl_t * ctl,
  tbl_t * tbl,
  los_t * los,
  int ip,
  double tau_path[NG][ND],
  double tau_seg[ND]);

double intpol_tbl_eps(
  tbl_t * tbl,
  int ig,
  int id,
  int ip,
  int it,
  double u);

double intpol_tbl_u(
  tbl_t * tbl,
  int ig,
  int id,
  int ip,
  int it,
```

```
  double eps);

void jsec2time(
  double jsec,
  int *year,
  int *mon,
  int *day,
  int *hour,
  int *min,
  int *sec,
  double *remain);

void kernel(
  ctl_t * ctl,
  atm_t * atm,
  obs_t * obs,
  gsl_matrix * k);

int locate(
  double *xx,
  int n,
  double x);

int locate_tbl(
  float *xx,
  int n,
  double x);

size_t obs2y(
  ctl_t * ctl,
  obs_t * obs,
  gsl_vector * y,
  int *ida,
  int *ira);

double planck(
  double t,
  double nu);

void raytrace(
  ctl_t * ctl,
  atm_t * atm,
  obs_t * obs,
  los_t * los,
  int ir);

void read_atm(
  const char *dirname,
  const char *filename,
  ctl_t * ctl,
  atm_t * atm);

void read_ctl(
  int argc,
  char *argv[],
  ctl_t * ctl);

void read_matrix(
  const char *dirname,
  const char *filename,
  gsl_matrix * matrix);

void read_obs(
  const char *dirname,
  const char *filename,
  ctl_t * ctl,
  obs_t * obs);

void read_shape(
  const char *filename,
  double *x,
  double *y,
  int *n);

double refractivity(
  double p,
  double t);

double scan_ctl(
  int argc,
  char *argv[],
  const char *varname,
  int arridx,
  const char *defvalue,
  char *value);

void tangent_point(
```

```
  los_t * los,
  double *tpz,
  double *tplon,
  double *tplat);

void time2jsec(
  int year,
  int mon,
  int day,
  int hour,
  int min,
  int sec,
  double remain,
  double *jsec);

void timer(
  const char *name,
  const char *file,
  const char *func,
  int line,
  int mode);

void write_atm(
  const char *dirname,
  const char *filename,
  ctl_t * ctl,
  atm_t * atm);

void write_matrix(
  const char *dirname,
  const char *filename,
  ctl_t * ctl,
  gsl_matrix * matrix,
  atm_t * atm,
  obs_t * obs,
  const char *rowspace,
  const char *colspace,
  const char *sort);

void write_obs(
  const char *dirname,
  const char *filename,
  ctl_t * ctl,
  obs_t * obs);

void x2atm(
  ctl_t * ctl,
  gsl_vector * x,
  atm_t * atm);

void x2atm_help(
  atm_t * atm,
  double zmin,
  double zmax,
  double *value,
  gsl_vector * x,
  size_t * n);

void y2obs(
  ctl_t * ctl,
  gsl_vector * y,
  obs_t * obs);
```

5.35 libairs.c File Reference

Functions

- void add_var (int ncid, const char *varname, const char *unit, const char *longname, int type, int dimid[ ], int *varid, int ndims)

  Add variable to netCDF file.
- void background_poly_help (double *xx, double *yy, int n, int dim)

  Get background based on polynomial fits.
- void background_poly (wave_t *wave, int dim_x, int dim_y)

  Get background based on polynomial fits.
- void background_smooth (wave_t *wave, int npts_x, int npts_y)

  Smooth background.

- void create_background (wave_t *wave)

  Set background...
- void create_noise (wave_t *wave, double nedt)

  Add noise to perturbations and temperatures...
- void create_wave (wave_t *wave, double amp, double lx, double ly, double phi, double fwhm)

  Add linear wave pattern...
- void day2doy (int year, int mon, int day, int *doy)

  Get day of year from date.
- void doy2day (int year, int doy, int *mon, int *day)

  Get date from day of year.
- void fft_help (double *fcReal, double *fcImag, int n)

  Calculate 1-D FFT...
- void fft (wave_t *wave, double *Amax, double *phimax, double *lhmax, double *alphamax, double *betamax, char *filename)

  Calculate 2-D FFT...
- void gauss (wave_t *wave, double fwhm)

  Apply Gaussian filter to perturbations...
- void hamming (wave_t *wave, int niter)

  Apply Hamming filter to perturbations...
- void intpol_x (wave_t *wave, int n)

  Interpolate to regular grid in x-direction.
- void median (wave_t *wave, int dx)

  Apply median filter to perturbations...
- void merge_y (wave_t *wave1, wave_t *wave2)

  Merge wave structs in y-direction.
- void noise (wave_t *wave, double *mu, double *sig)

  Estimate noise.
- void period (wave_t *wave, double *Amax, double *phimax, double *lhmax, double *alphamax, double *betamax, char *filename)

  Compute periodogram.
- void pert2wave (pert_t *pert, wave_t *wave, int track0, int track1, int xtrack0, int xtrack1)

  Convert radiance perturbation data to wave analysis struct.
- void read_l1 (char *filename, airs_l1_t *l1)

  Read AIRS Level-1 data.
- void read_l2 (char *filename, airs_l2_t *l2)

  Read AIRS Level-2 data.
- void read_pert (char *filename, char *pertname, pert_t *pert)

  Read radiance perturbation data.
- void read_retr (char *filename, ret_t *ret)

  Read AIRS retrieval data.
- void read_retr_help (double *help, int nds, int np, double mat[NDS][NPG])

  Convert array.
- void read_wave (char *filename, wave_t *wave)

  Read wave analysis data.
- void rad2wave (airs_rad_gran_t *gran, double *nu, int nd, wave_t *wave)

  Convert AIRS radiance data to wave analysis struct.
- void ret2wave (ret_t *ret, wave_t *wave, int dataset, int ip)

  Convert AIRS retrieval results to wave analysis struct.
- double sza (double sec, double lon, double lat)

  Calculate solar zenith angle.
- void variance (wave_t *wave, double dh)

Compute local variance.

- void write_l1 (char ∗filename, airs_l1_t ∗l1)

  Write AIRS Level-1 data.

- void write_l2 (char ∗filename, airs_l2_t ∗l2)

  Write AIRS Level-2 data.

- void write_wave (char ∗filename, wave_t ∗wave)

  Write wave analysis data.

5.35.1 Function Documentation

5.35.1.1 void add_var (int *ncid,* const char ∗ *varname,* const char ∗ *unit,* const char ∗ *longname,* int *type,* int *dimid[ ],* int ∗ *varid,* int *ndims*)

Add variable to netCDF file.

Definition at line 5 of file libairs.c.

```
                 {

  /* Check if variable exists... */
  if (nc_inq_varid(ncid, varname, varid) != NC_NOERR) {

    /* Define variable... */
    NC(nc_def_var(ncid, varname, type, ndims, dimid, varid));

    /* Set long name... */
    NC(nc_put_att_text
       (ncid, *varid, "long_name", strlen(longname), longname));

    /* Set units... */
    NC(nc_put_att_text(ncid, *varid, "units", strlen(unit), unit));
  }
}
```

5.35.1.2 void background_poly_help (double ∗ *xx,* double ∗ *yy,* int *n,* int *dim*)

Get background based on polynomial fits.

Definition at line 32 of file libairs.c.

```
               {

  gsl_multifit_linear_workspace *work;
  gsl_matrix *cov, *X;
  gsl_vector *c, *x, *y;

  double chisq, xx2[WX > WY ? WX : WY], yy2[WX > WY ? WX : WY];

  size_t i, i2, n2 = 0;

  /* Check for nan... */
  for (i = 0; i < (size_t) n; i++)
    if (gsl_finite(yy[i])) {
      xx2[n2] = xx[i];
      yy2[n2] = yy[i];
      n2++;
    }
  if ((int) n2 < dim || n2 < 0.9 * n) {
    for (i = 0; i < (size_t) n; i++)
      yy[i] = GSL_NAN;
    return;
  }

  /* Allocate... */
  work = gsl_multifit_linear_alloc((size_t) n2, (size_t) dim);
  cov = gsl_matrix_alloc((size_t) dim, (size_t) dim);
  X = gsl_matrix_alloc((size_t) n2, (size_t) dim);
```

```
  c = gsl_vector_alloc((size_t) dim);
  x = gsl_vector_alloc((size_t) n2);
  y = gsl_vector_alloc((size_t) n2);

  /* Compute polynomial fit... */
  for (i = 0; i < (size_t) n2; i++) {
    gsl_vector_set(x, i, xx2[i]);
    gsl_vector_set(y, i, yy2[i]);
    for (i2 = 0; i2 < (size_t) dim; i2++)
      gsl_matrix_set(X, i, i2, pow(gsl_vector_get(x, i), (double) i2));
  }
  gsl_multifit_linear(X, y, c, cov, &chisq, work);
  for (i = 0; i < (size_t) n; i++)
    yy[i] = gsl_poly_eval(c->data, (int) dim, xx[i]);

  /* Free... */
  gsl_multifit_linear_free(work);
  gsl_matrix_free(cov);
  gsl_matrix_free(X);
  gsl_vector_free(c);
  gsl_vector_free(x);
  gsl_vector_free(y);
}
```

5.35.1.3 void background_poly (wave_t * *wave*, int *dim_x*, int *dim_y*)

Get background based on polynomial fits.

Definition at line 89 of file libairs.c.

```
              {

  double x[WX], x2[WY], y[WX], y2[WY];

  int ix, iy;

  /* Copy temperatures to background... */
  for (ix = 0; ix < wave->nx; ix++)
    for (iy = 0; iy < wave->ny; iy++) {
      wave->bg[ix][iy] = wave->temp[ix][iy];
      wave->pt[ix][iy] = 0;
    }

  /* Check parameters... */
  if (dim_x <= 0 && dim_y <= 0)
    return;

  /* Compute fit in x-direction... */
  if (dim_x > 0)
    for (iy = 0; iy < wave->ny; iy++) {
      for (ix = 0; ix < wave->nx; ix++) {
        x[ix] = (double) ix;
        y[ix] = wave->bg[ix][iy];
      }
      background_poly_help(x, y, wave->nx, dim_x);
      for (ix = 0; ix < wave->nx; ix++)
        wave->bg[ix][iy] = y[ix];
    }

  /* Compute fit in y-direction... */
  if (dim_y > 0)
    for (ix = 0; ix < wave->nx; ix++) {
      for (iy = 0; iy < wave->ny; iy++) {
        x2[iy] = (int) iy;
        y2[iy] = wave->bg[ix][iy];
      }
      background_poly_help(x2, y2, wave->ny, dim_y);
      for (iy = 0; iy < wave->ny; iy++)
        wave->bg[ix][iy] = y2[iy];
    }

  /* Recompute perturbations... */
  for (ix = 0; ix < wave->nx; ix++)
    for (iy = 0; iy < wave->ny; iy++)
      wave->pt[ix][iy] = wave->temp[ix][iy] - wave->bg[ix][iy];
}
```

Here is the call graph for this function:

5.35.1.4 void background_smooth (wave_t ∗ *wave*, int *npts_x*, int *npts_y*)

Smooth background.

Definition at line 141 of file libairs.c.

```
              {

  static double help[WX][WY], dmax = 2500.;

  int dx, dy, i, j, ix, iy, n;

  /* Check parameters... */
  if (npts_x <= 0 && npts_y <= 0)
    return;

  /* Smooth background... */
  for (ix = 0; ix < wave->nx; ix++)
    for (iy = 0; iy < wave->ny; iy++) {

      /* Init... */
      n = 0;
      help[ix][iy] = 0;

      /* Set maximum range... */
      dx = GSL_MIN(GSL_MIN(npts_x, ix), wave->nx - 1 - ix);
      dy = GSL_MIN(GSL_MIN(npts_y, iy), wave->ny - 1 - iy);

      /* Average... */
      for (i = ix - dx; i <= ix + dx; i++)
        for (j = iy - dy; j <= iy + dy; j++)
          if (fabs(wave->x[ix] - wave->x[i]) < dmax &&
              fabs(wave->y[iy] - wave->y[j]) < dmax) {
            help[ix][iy] += wave->bg[i][j];
            n++;
          }

      /* Normalize... */
      if (n > 0)
        help[ix][iy] /= n;
      else
        help[ix][iy] = GSL_NAN;
    }

  /* Recalculate perturbations... */
  for (ix = 0; ix < wave->nx; ix++)
    for (iy = 0; iy < wave->ny; iy++) {
      wave->bg[ix][iy] = help[ix][iy];
      wave->pt[ix][iy] = wave->temp[ix][iy] - wave->bg[ix][iy];
    }
}
```

5.35.1.5 void create_background (wave_t ∗ *wave*)

Set background...

Definition at line 192 of file libairs.c.

```
                 {

  int ix, iy;

  /* Loop over grid points... */
  for (ix = 0; ix < wave->nx; ix++)
    for (iy = 0; iy < wave->ny; iy++) {

      /* Set background for 4.3 micron BT measurements... */
      wave->bg[ix][iy] = 235.626 + 5.38165e-6 * gsl_pow_2(wave->x[ix]
                                                          -
                                                          0.5 * (wave->x[0] +
                                                                 wave->x
                                                                 [wave->nx -
                                                                  1]))
        - 1.78519e-12 * gsl_pow_4(wave->x[ix] -
                                  0.5 * (wave->x[0] + wave->x[wave->nx - 1]));

      /* Set temperature perturbation... */
      wave->pt[ix][iy] = 0;

      /* Set temperature... */
      wave->temp[ix][iy] = wave->bg[ix][iy];
    }
}
```

5.35.1.6 void create_noise (wave_t * *wave,* double *nedt*)

Add noise to perturbations and temperatures...

Definition at line 221 of file libairs.c.

```
                 {

  gsl_rng *r;

  int ix, iy;

  /* Initialize random number generator... */
  gsl_rng_env_setup();
  r = gsl_rng_alloc(gsl_rng_default);
  gsl_rng_set(r, (unsigned long int) time(NULL));

  /* Add noise to temperature... */
  if (nedt > 0)
    for (ix = 0; ix < wave->nx; ix++)
      for (iy = 0; iy < wave->ny; iy++)
        wave->temp[ix][iy] += gsl_ran_gaussian(r, nedt);

  /* Free... */
  gsl_rng_free(r);
}
```

5.35.1.7 void create_wave (wave_t * *wave,* double *amp,* double *lx,* double *ly,* double *phi,* double *fwhm*)

Add linear wave pattern...

Definition at line 246 of file libairs.c.

```
                 {

  int ix, iy;

  /* Loop over grid points... */
  for (ix = 0; ix < wave->nx; ix++)
    for (iy = 0; iy < wave->ny; iy++) {

      /* Set wave perturbation... */
      wave->pt[ix][iy] = amp * cos((lx != 0 ? 2 * M_PI / lx : 0) * wave->x[ix]
                                  + (ly !=
                                     0 ? 2 * M_PI / ly : 0) * wave->y[iy]
                                  - phi * M_PI / 180.)
        * (fwhm > 0 ? exp(-0.5 * gsl_pow_2((wave->x[ix]) / (lx * fwhm) * 2.35)
                         -
                         0.5 * gsl_pow_2((wave->y[iy]) / (ly * fwhm) *
                                         2.35)) : 1.0);

      /* Add perturbation to temperature... */
      wave->temp[ix][iy] += wave->pt[ix][iy];
    }
}
```

5.35.1.8 void day2doy (int *year,* int *mon,* int *day,* int * *doy*)

Get day of year from date.

Definition at line 277 of file libairs.c.

```
              {

  int d0[12] = { 1, 32, 60, 91, 121, 152, 182, 213, 244, 274, 305, 335 };
  int d01[12] = { 1, 32, 61, 92, 122, 153, 183, 214, 245, 275, 306, 336 };

  /* Get day of year... */
  if (year % 400 == 0 || (year % 100 != 0 && year % 4 == 0))
    *doy = d01[mon - 1] + day - 1;
  else
    *doy = d0[mon - 1] + day - 1;
}
```

5.35.1.9 void doy2day (int *year*, int *doy*, int ∗ *mon*, int ∗ *day*)

Get date from day of year.

Definition at line 295 of file libairs.c.

```
            {

  int d0[12] = { 1, 32, 60, 91, 121, 152, 182, 213, 244, 274, 305, 335 };
  int d01[12] = { 1, 32, 61, 92, 122, 153, 183, 214, 245, 275, 306, 336 };
  int i;

  /* Get month and day... */
  if (year % 400 == 0 || (year % 100 != 0 && year % 4 == 0)) {
    for (i = 11; i >= 0; i--)
      if (d01[i] <= doy)
        break;
    *mon = i + 1;
    *day = doy - d01[i] + 1;
  } else {
    for (i = 11; i >= 0; i--)
      if (d0[i] <= doy)
        break;
    *mon = i + 1;
    *day = doy - d0[i] + 1;
  }
}
```

5.35.1.10 void fft_help (double ∗ *fcReal*, double ∗ *fcImag*, int *n*)

Calculate 1-D FFT...

Definition at line 323 of file libairs.c.

```
         {

  gsl_fft_complex_wavetable *wavetable;
  gsl_fft_complex_workspace *workspace;

  double data[2 * PMAX];

  int i;

  /* Check size... */
  if (n > PMAX)
    ERRMSG("Too many data points!");

  /* Allocate... */
  wavetable = gsl_fft_complex_wavetable_alloc((size_t) n);
  workspace = gsl_fft_complex_workspace_alloc((size_t) n);

  /* Set data (real, complex)... */
  for (i = 0; i < n; i++) {
    data[2 * i] = fcReal[i];
    data[2 * i + 1] = fcImag[i];
  }

  /* Calculate FFT... */
  gsl_fft_complex_forward(data, 1, (size_t) n, wavetable, workspace);

  /* Copy data... */
  for (i = 0; i < n; i++) {
    fcReal[i] = data[2 * i];
    fcImag[i] = data[2 * i + 1];
  }

  /* Free... */
  gsl_fft_complex_wavetable_free(wavetable);
  gsl_fft_complex_workspace_free(workspace);
}
```

5.35.1.11 void fft (wave_t * *wave*, double * *Amax*, double * *phimax*, double * *lhmax*, double * *alphamax*, double * *betamax*, char * *filename*)

Calculate 2-D FFT...

Definition at line 365 of file libairs.c.

```
                 {

  static double A[PMAX][PMAX], phi[PMAX][PMAX], kx[PMAX], ky[PMAX],
    kxmax, kymax, cutReal[PMAX], cutImag[PMAX],
    boxImag[PMAX][PMAX], boxReal[PMAX][PMAX];

  FILE *out;

  int i, i2, imin, imax, j, j2, jmin, jmax, nx, ny;

  /* Find box... */
  imin = jmin = 9999;
  imax = jmax = -9999;
  for (i = 0; i < wave->nx; i++)
    for (j = 0; j < wave->ny; j++)
      if (gsl_finite(wave->var[i][j])) {
        imin = GSL_MIN(imin, i);
        imax = GSL_MAX(imax, i);
        jmin = GSL_MIN(jmin, j);
        jmax = GSL_MAX(jmax, j);
      }
  nx = imax - imin + 1;
  ny = jmax - jmin + 1;

  /* Copy data... */
  for (i = imin; i <= imax; i++)
    for (j = jmin; j <= jmax; j++) {
      if (gsl_finite(wave->pt[i][j]))
        boxReal[i - imin][j - jmin] = wave->pt[i][j];
      else
        boxReal[i - imin][j - jmin] = 0.0;
      boxImag[i - imin][j - jmin] = 0.0;
    }

  /* FFT of the rows... */
  for (i = 0; i < nx; i++) {
    for (j = 0; j < ny; j++) {
      cutReal[j] = boxReal[i][j];
      cutImag[j] = boxImag[i][j];
    }
    fft_help(cutReal, cutImag, ny);
    for (j = 0; j < ny; j++) {
      boxReal[i][j] = cutReal[j];
      boxImag[i][j] = cutImag[j];
    }
  }

  /* FFT of the columns... */
  for (j = 0; j < ny; j++) {
    for (i = 0; i < nx; i++) {
      cutReal[i] = boxReal[i][j];
      cutImag[i] = boxImag[i][j];
    }
    fft_help(cutReal, cutImag, nx);
    for (i = 0; i < nx; i++) {
      boxReal[i][j] = cutReal[i];
      boxImag[i][j] = cutImag[i];
    }
  }

  /* Get frequencies, amplitude, and phase... */
  for (i = 0; i < nx; i++)
    kx[i] = 2. * M_PI * ((i < nx / 2) ? (double) i : -(double) (nx - i))
      / (nx * fabs(wave->x[imax] - wave->x[imin]) / (nx - 1.0));
  for (j = 0; j < ny; j++)
    ky[j] = 2. * M_PI * ((j < ny / 2) ? (double) j : -(double) (ny - j))
      / (ny * fabs(wave->y[jmax] - wave->y[jmin]) / (ny - 1.0));
  for (i = 0; i < nx; i++)
    for (j = 0; j < ny; j++) {
      A[i][j]
        = (i == 0 && j == 0 ? 1.0 : 2.0) / (nx * ny)
        * sqrt(gsl_pow_2(boxReal[i][j]) + gsl_pow_2(boxImag[i][j]));
      phi[i][j]
        = 180. / M_PI * atan2(boxImag[i][j], boxReal[i][j]);
    }

```

```
  /* Check frequencies... */
  for (i = 0; i < nx; i++)
    for (j = 0; j < ny; j++)
      if (kx[i] == 0 || ky[j] == 0) {
        A[i][j] = GSL_NAN;
        phi[i][j] = GSL_NAN;
      }

  /* Find maximum... */
  *Amax = 0;
  for (i = 0; i < nx; i++)
    for (j = 0; j < ny / 2; j++)
      if (gsl_finite(A[i][j]) && A[i][j] > *Amax) {
        *Amax = A[i][j];
        *phimax = phi[i][j];
        kxmax = kx[i];
        kymax = ky[j];
        imax = i;
        jmax = j;
      }

  /* Get horizontal wavelength... */
  *lhmax = 2 * M_PI / sqrt(gsl_pow_2(kxmax) + gsl_pow_2(kymax));

  /* Get propagation direction in xy-plane... */
  *alphamax = 90. - 180. / M_PI * atan2(kxmax, kymax);

  /* Get propagation direction in lon,lat-plane... */
  *betamax = *alphamax
    +
    180. / M_PI *
    atan2(wave->lat[wave->nx / 2 >
                    0 ? wave->nx / 2 - 1 : wave->nx / 2][wave->ny / 2]
          - wave->lat[wave->nx / 2 <
                      wave->nx - 1 ? wave->nx / 2 +
                      1 : wave->nx / 2][wave->ny / 2],
          wave->lon[wave->nx / 2 >
                    0 ? wave->nx / 2 - 1 : wave->nx / 2][wave->ny / 2]
          - wave->lon[wave->nx / 2 <
                      wave->nx - 1 ? wave->nx / 2 +
                      1 : wave->nx / 2][wave->ny / 2]);

  /* Save FFT data... */
  if (filename != NULL) {

    /* Write info... */
    printf("Write FFT data: %s\n", filename);

    /* Create file... */
    if (!(out = fopen(filename, "w")))
      ERRMSG("Cannot create file!");

    /* Write header... */
    fprintf(out,
            "# $1 = altitude [km]\n"
            "# $2 = wavelength in x-direction [km]\n"
            "# $3 = wavelength in y-direction [km]\n"
            "# $4 = wavenumber in x-direction [1/km]\n"
            "# $5 = wavenumber in y-direction [1/km]\n"
            "# $6 = amplitude [K]\n" "# $7 = phase [rad]\n");

    /* Write data... */
    for (i = nx - 1; i > 0; i--) {
      fprintf(out, "\n");
      for (j = ny / 2; j > 0; j--) {
        i2 = (i == nx / 2 ? 0 : i);
        j2 = (j == ny / 2 ? 0 : j);
        fprintf(out, "%g %g %g %g %g %g %g\n", wave->z,
                (kx[i2] != 0 ? 2 * M_PI / kx[i2] : 0),
                (ky[j2] != 0 ? 2 * M_PI / ky[j2] : 0),
                kx[i2], ky[j2], A[i2][j2], phi[i2][j2]);
      }
    }

    /* Close file... */
    fclose(out);
  }
}
```

Here is the call graph for this function:

5.35.1.12 void gauss (wave_t * *wave*, double *fwhm*)

Apply Gaussian filter to perturbations...

Definition at line 529 of file libairs.c.

```
                 {

  static double d2, help[WX][WY], sigma2, w, wsum;

  int ix, ix2, iy, iy2;

  /* Check parameters... */
  if (fwhm <= 0)
    return;

  /* Compute sigma^2... */
  sigma2 = gsl_pow_2(fwhm / 2.3548);

  /* Loop over data points... */
  for (ix = 0; ix < wave->nx; ix++)
    for (iy = 0; iy < wave->ny; iy++) {

      /* Init... */
      wsum = 0;
      help[ix][iy] = 0;

      /* Average... */
      for (ix2 = 0; ix2 < wave->nx; ix2++)
        for (iy2 = 0; iy2 < wave->ny; iy2++) {
          d2 = gsl_pow_2(wave->x[ix] - wave->x[ix2])
            + gsl_pow_2(wave->y[iy] - wave->y[iy2]);
          if (d2 <= 9 * sigma2) {
            w = exp(-d2 / (2 * sigma2));
            wsum += w;
            help[ix][iy] += w * wave->pt[ix2][iy2];
          }
        }

      /* Normalize... */
      wave->pt[ix][iy] = help[ix][iy] / wsum;
    }
}
```

5.35.1.13 void hamming (wave_t * *wave,* int *nit*)

Apply Hamming filter to perturbations...

Definition at line 571 of file libairs.c.

```
                {

  static double help[WX][WY];

  int iter, ix, iy;

  /* Iterations... */
  for (iter = 0; iter < niter; iter++) {

    /* Filter in x direction... */
    for (ix = 0; ix < wave->nx; ix++)
      for (iy = 0; iy < wave->ny; iy++)
        help[ix][iy]
          = 0.23 * wave->pt[ix > 0 ? ix - 1 : ix][iy]
          + 0.54 * wave->pt[ix][iy]
          + 0.23 * wave->pt[ix < wave->nx - 1 ? ix + 1 : ix][iy];

    /* Filter in y direction... */
    for (ix = 0; ix < wave->nx; ix++)
      for (iy = 0; iy < wave->ny; iy++)
        wave->pt[ix][iy]
          = 0.23 * help[ix][iy > 0 ? iy - 1 : iy]
          + 0.54 * help[ix][iy]
          + 0.23 * help[ix][iy < wave->ny - 1 ? iy + 1 : iy];
  }
}
```

5.35.1.14 void intpol_x (wave_t * *wave*, int *n*)

Interpolate to regular grid in x-direction.

Definition at line 602 of file libairs.c.

```
        {

  gsl_interp_accel *acc;
  gsl_spline *spline;

  double dummy, x[WX], xc[WX][3], xc2[WX][3], y[WX];

  int i, ic, ix, iy;

  /* Check parameters... */
  if (n <= 0)
    return;
  if (n > WX)
    ERRMSG("Too many data points!");

  /* Set new x-coordinates... */
  for (i = 0; i < n; i++)
    x[i] = LIN(0.0, wave->x[0], n - 1.0, wave->x[wave->nx - 1], i);

  /* Allocate... */
  acc = gsl_interp_accel_alloc();
  spline = gsl_spline_alloc(gsl_interp_cspline, (size_t) wave->nx);

  /* Loop over scans... */
  for (iy = 0; iy < wave->ny; iy++) {

    /* Interpolate Cartesian coordinates... */
    for (ix = 0; ix < wave->nx; ix++)
      geo2cart(0, wave->lon[ix][iy], wave->lat[ix][iy], xc[ix]);
    for (ic = 0; ic < 3; ic++) {
      for (ix = 0; ix < wave->nx; ix++)
        y[ix] = xc[ix][ic];
      gsl_spline_init(spline, wave->x, y, (size_t) wave->nx);
      for (i = 0; i < n; i++)
        xc2[i][ic] = gsl_spline_eval(spline, x[i], acc);
    }
    for (i = 0; i < n; i++)
      cart2geo(xc2[i], &dummy, &wave->lon[i][iy], &wave->lat[i][iy]);

    /* Interpolate temperature... */
    for (ix = 0; ix < wave->nx; ix++)
      y[ix] = wave->temp[ix][iy];
    gsl_spline_init(spline, wave->x, y, (size_t) wave->nx);
    for (i = 0; i < n; i++)
      wave->temp[i][iy] = gsl_spline_eval(spline, x[i], acc);

    /* Interpolate background... */
    for (ix = 0; ix < wave->nx; ix++)
      y[ix] = wave->bg[ix][iy];
    gsl_spline_init(spline, wave->x, y, (size_t) wave->nx);
    for (i = 0; i < n; i++)
      wave->bg[i][iy] = gsl_spline_eval(spline, x[i], acc);

    /* Interpolate perturbations... */
    for (ix = 0; ix < wave->nx; ix++)
      y[ix] = wave->pt[ix][iy];
    gsl_spline_init(spline, wave->x, y, (size_t) wave->nx);
    for (i = 0; i < n; i++)
      wave->pt[i][iy] = gsl_spline_eval(spline, x[i], acc);

    /* Interpolate variance... */
    for (ix = 0; ix < wave->nx; ix++)
      y[ix] = wave->var[ix][iy];
    gsl_spline_init(spline, wave->x, y, (size_t) wave->nx);
    for (i = 0; i < n; i++)
      wave->var[i][iy] = gsl_spline_eval(spline, x[i], acc);
  }

  /* Free... */
  gsl_spline_free(spline);
  gsl_interp_accel_free(acc);

  /* Set new x-coordinates... */
  for (i = 0; i < n; i++)
    wave->x[i] = x[i];
  wave->nx = n;
}
```

Here is the call graph for this function:

5.35.1.15 void median (wave_t * wave, int dx)

Apply median filter to perturbations...

Definition at line 684 of file libairs.c.

```
           {

  static double data[WX * WY], help[WX][WY];

  int ix, ix2, iy, iy2;

  size_t n;

  /* Check parameters... */
  if (dx <= 0)
    return;

  /* Loop over data points... */
  for (ix = 0; ix < wave->nx; ix++)
    for (iy = 0; iy < wave->ny; iy++) {

      /* Init... */
      n = 0;

      /* Get data... */
      for (ix2 = GSL_MAX(ix - dx, 0); ix2 < GSL_MIN(ix + dx, wave->nx - 1);
           ix2++)
        for (iy2 = GSL_MAX(iy - dx, 0); iy2 < GSL_MIN(iy + dx, wave->ny - 1);
             iy2++) {
          data[n] = wave->pt[ix2][iy2];
          n++;
        }

      /* Normalize... */
      gsl_sort(data, 1, n);
      help[ix][iy] = gsl_stats_median_from_sorted_data(data, 1, n);
    }

  /* Loop over data points... */
  for (ix = 0; ix < wave->nx; ix++)
    for (iy = 0; iy < wave->ny; iy++)
      wave->pt[ix][iy] = help[ix][iy];
}
```

5.35.1.16 void merge_y (wave_t * wave1, wave_t * wave2)

Merge wave structs in y-direction.

Definition at line 727 of file libairs.c.

```
                   {

  double y;

  int ix, iy;

  /* Check data... */
  if (wave1->nx != wave2->nx)
    ERRMSG("Across-track sizes do not match!");
  if (wave1->ny + wave2->ny > WY)
    ERRMSG("Too many data points!");

  /* Get offset in y direction... */
  y =
    wave1->y[wave1->ny - 1] + (wave1->y[wave1->ny - 1] -
                               wave1->y[0]) / (wave1->ny - 1);

  /* Merge data... */
  for (ix = 0; ix < wave2->nx; ix++)
    for (iy = 0; iy < wave2->ny; iy++) {
      wave1->y[wave1->ny + iy] = y + wave2->y[iy];
      wave1->lon[ix][wave1->ny + iy] = wave2->lon[ix][iy];
      wave1->lat[ix][wave1->ny + iy] = wave2->lat[ix][iy];
      wave1->temp[ix][wave1->ny + iy] = wave2->temp[ix][iy];
      wave1->bg[ix][wave1->ny + iy] = wave2->bg[ix][iy];
      wave1->pt[ix][wave1->ny + iy] = wave2->pt[ix][iy];
      wave1->var[ix][wave1->ny + iy] = wave2->var[ix][iy];
    }

  /* Increment counter... */
  wave1->ny += wave2->ny;
}
```

5.35.1.17 void noise (wave_t ∗ *wave,* double ∗ *mu,* double ∗ *sig*)

Estimate noise.

Definition at line 764 of file libairs.c.

```
                {

  int ix, ix2, iy, iy2, n = 0, okay;

  /* Init... */
  *mu = 0;
  *sig = 0;

  /* Estimate noise (Immerkaer, 1996)... */
  for (ix = 1; ix < wave->nx - 1; ix++)
    for (iy = 1; iy < wave->ny - 1; iy++) {

      /* Check data... */
      okay = 1;
      for (ix2 = ix - 1; ix2 <= ix + 1; ix2++)
        for (iy2 = iy - 1; iy2 <= iy + 1; iy2++)
          if (!gsl_finite(wave->temp[ix2][iy2]))
            okay = 0;
      if (!okay)
        continue;

      /* Get mean noise... */
      n++;
      *mu += wave->temp[ix][iy];
      *sig += gsl_pow_2(+4. / 6. * wave->temp[ix][iy]
                        - 2. / 6. * (wave->temp[ix - 1][iy]
                                     + wave->temp[ix + 1][iy]
                                     + wave->temp[ix][iy - 1]
                                     + wave->temp[ix][iy + 1])
                        + 1. / 6. * (wave->temp[ix - 1][iy - 1]
                                     + wave->temp[ix + 1][iy - 1]
                                     + wave->temp[ix - 1][iy + 1]
                                     + wave->temp[ix + 1][iy + 1]));
    }

  /* Normalize... */
  *mu /= (double) n;
  *sig = sqrt(*sig / (double) n);
}
```

5.35.1.18 void period (wave_t ∗ *wave,* double ∗ *Amax,* double ∗ *phimax,* double ∗ *lhmax,* double ∗ *alphamax,* double ∗ *betamax,* char ∗ *filename*)

Compute periodogram.

Definition at line 809 of file libairs.c.

```
                    {

  FILE *out;

  static double kx[PMAX], ky[PMAX], kx_ny, ky_ny, kxmax, kymax, A[PMAX][PMAX],
    phi[PMAX][PMAX], cx[PMAX][WX], cy[PMAX][WY], sx[PMAX][WX], sy[PMAX][WY],
    a, b, c, lx, ly, lxymax = 1000, dlxy = 10;

  int i, imin, imax, j, jmin, jmax, l, lmax = 0, m, mmax = 0;

  /* Compute wavenumbers and periodogram coefficients... */
  for (lx = -lxymax; lx <= lxymax; lx += dlxy) {
    kx[lmax] = (lx != 0 ? 2 * M_PI / lx : 0);
    for (i = 0; i < wave->nx; i++) {
      cx[lmax][i] = cos(kx[lmax] * wave->x[i]);
      sx[lmax][i] = sin(kx[lmax] * wave->x[i]);
    }
    if ((++lmax) > PMAX)
      ERRMSG("Too many wavenumbers for periodogram!");
  }
  for (ly = 0; ly <= lxymax; ly += dlxy) {
    ky[mmax] = (ly != 0 ? 2 * M_PI / ly : 0);
    for (j = 0; j < wave->ny; j++) {
```

```
      cy[mmax][j] = cos(ky[mmax] * wave->y[j]);
      sy[mmax][j] = sin(ky[mmax] * wave->y[j]);
    }
    if ((++mmax) > PMAX)
      ERRMSG("Too many wavenumbers for periodogram!");
  }

  /* Find area... */
  imin = jmin = 9999;
  imax = jmax = -9999;
  for (i = 0; i < wave->nx; i++)
    for (j = 0; j < wave->ny; j++)
      if (gsl_finite(wave->var[i][j])) {
        imin = GSL_MIN(imin, i);
        imax = GSL_MAX(imax, i);
        jmin = GSL_MIN(jmin, j);
        jmax = GSL_MAX(jmax, j);
      }

  /* Get Nyquist frequencies... */
  kx_ny =
    M_PI / fabs((wave->x[imax] - wave->x[imin]) /
                ((double) imax - (double) imin));
  ky_ny =
    M_PI / fabs((wave->y[jmax] - wave->y[jmin]) /
                ((double) jmax - (double) jmin));

  /* Loop over wavelengths... */
  for (l = 0; l < lmax; l++)
    for (m = 0; m < mmax; m++) {

      /* Check frequencies... */
      if (kx[l] == 0 || fabs(kx[l]) > kx_ny ||
          ky[m] == 0 || fabs(ky[m]) > ky_ny) {
        A[l][m] = GSL_NAN;
        phi[l][m] = GSL_NAN;
        continue;
      }

      /* Compute periodogram... */
      a = b = c = 0;
      for (i = imin; i <= imax; i++)
        for (j = jmin; j <= jmax; j++)
          if (gsl_finite(wave->var[i][j])) {
            a += wave->pt[i][j] * (cx[l][i] * cy[m][j] - sx[l][i] * sy[m][j]);
            b += wave->pt[i][j] * (sx[l][i] * cy[m][j] + cx[l][i] * sy[m][j]);
            c++;
          }
      a *= 2. / c;
      b *= 2. / c;

      /* Get amplitude and phase... */
      A[l][m] = sqrt(gsl_pow_2(a) + gsl_pow_2(b));
      phi[l][m] = atan2(b, a) * 180. / M_PI;
    }

  /* Find maximum... */
  *Amax = 0;
  for (l = 0; l < lmax; l++)
    for (m = 0; m < mmax; m++)
      if (gsl_finite(A[l][m]) && A[l][m] > *Amax) {
        *Amax = A[l][m];
        *phimax = phi[l][m];
        kxmax = kx[l];
        kymax = ky[m];
        imax = i;
        jmax = j;
      }

  /* Get horizontal wavelength... */
  *lhmax = 2 * M_PI / sqrt(gsl_pow_2(kxmax) + gsl_pow_2(kymax));

  /* Get propagation direction in xy-plane... */
  *alphamax = 90. - 180. / M_PI * atan2(kxmax, kymax);

  /* Get propagation direction in lon,lat-plane... */
  *betamax = *alphamax
    +
    180. / M_PI *
    atan2(wave->lat[wave->nx / 2 >
                    0 ? wave->nx / 2 - 1 : wave->nx / 2][wave->ny / 2]
          - wave->lat[wave->nx / 2 <
                      wave->nx - 1 ? wave->nx / 2 +
                      1 : wave->nx / 2][wave->ny / 2],
          wave->lon[wave->nx / 2 >
                    0 ? wave->nx / 2 - 1 : wave->nx / 2][wave->ny / 2]
          - wave->lon[wave->nx / 2 <
```

```
                            wave->nx - 1 ? wave->nx / 2 +
                            1 : wave->nx / 2][wave->ny / 2]);

  /* Save periodogram data... */
  if (filename != NULL) {

    /* Write info... */
    printf("Write periodogram data: %s\n", filename);

    /* Create file... */
    if (!(out = fopen(filename, "w")))
      ERRMSG("Cannot create file!");

    /* Write header... */
    fprintf(out,
            "# $1 = altitude [km]\n"
            "# $2 = wavelength in x-direction [km]\n"
            "# $3 = wavelength in y-direction [km]\n"
            "# $4 = wavenumber in x-direction [1/km]\n"
            "# $5 = wavenumber in y-direction [1/km]\n"
            "# $6 = amplitude [K]\n" "# $7 = phase [rad]\n");

    /* Write data... */
    for (l = 0; l < lmax; l++) {
      fprintf(out, "\n");
      for (m = 0; m < mmax; m++)
        fprintf(out, "%g %g %g %g %g %g %g\n", wave->z,
                (kx[l] != 0 ? 2 * M_PI / kx[l] : 0),
                (ky[m] != 0 ? 2 * M_PI / ky[m] : 0),
                kx[l], ky[m], A[l][m], phi[l][m]);
    }

    /* Close file... */
    fclose(out);
  }
}
```

5.35.1.19 void pert2wave (pert_t * *pert*, wave_t * *wave*, int *track0*, int *track1*, int *xtrack0*, int *xtrack1*)

Convert radiance perturbation data to wave analysis struct.

Definition at line 965 of file libairs.c.

```
                 {

  double x0[3], x1[3];

  int itrack, ixtrack;

  /* Check ranges... */
  track0 = GSL_MIN(GSL_MAX(track0, 0), pert->ntrack - 1);
  track1 = GSL_MIN(GSL_MAX(track1, 0), pert->ntrack - 1);
  xtrack0 = GSL_MIN(GSL_MAX(xtrack0, 0), pert->nxtrack - 1);
  xtrack1 = GSL_MIN(GSL_MAX(xtrack1, 0), pert->nxtrack - 1);

  /* Set size... */
  wave->nx = xtrack1 - xtrack0 + 1;
  if (wave->nx > WX)
    ERRMSG("Too many across-track values!");
  wave->ny = track1 - track0 + 1;
  if (wave->ny > WY)
    ERRMSG("Too many along-track values!");

  /* Loop over footprints... */
  for (itrack = track0; itrack <= track1; itrack++)
    for (ixtrack = xtrack0; ixtrack <= xtrack1; ixtrack++) {

      /* Get distances... */
      if (itrack == track0) {
        wave->x[0] = 0;
        if (ixtrack > xtrack0) {
          geo2cart(0, pert->lon[itrack][ixtrack - 1],
                   pert->lat[itrack][ixtrack - 1], x0);
          geo2cart(0, pert->lon[itrack][ixtrack],
                   pert->lat[itrack][ixtrack], x1);
          wave->x[ixtrack - xtrack0] =
            wave->x[ixtrack - xtrack0 - 1] + DIST(x0, x1);
        }
      }
      if (ixtrack == xtrack0) {
```

```
        wave->y[0] = 0;
        if (itrack > track0) {
          geo2cart(0, pert->lon[itrack - 1][ixtrack],
                   pert->lat[itrack - 1][ixtrack], x0);
          geo2cart(0, pert->lon[itrack][ixtrack],
                   pert->lat[itrack][ixtrack], x1);
          wave->y[itrack - track0] =
            wave->y[itrack - track0 - 1] + DIST(x0, x1);
        }
      }

      /* Save geolocation... */
      wave->time = pert->time[(track0 + track1) / 2][(xtrack0 + xtrack1) / 2];
      wave->z = 0;
      wave->lon[ixtrack - xtrack0][itrack - track0] =
        pert->lon[itrack][ixtrack];
      wave->lat[ixtrack - xtrack0][itrack - track0] =
        pert->lat[itrack][ixtrack];

      /* Save temperature data... */
      wave->temp[ixtrack - xtrack0][itrack - track0]
        = pert->bt[itrack][ixtrack];
      wave->bg[ixtrack - xtrack0][itrack - track0]
        = pert->bt[itrack][ixtrack] - pert->pt[itrack][ixtrack];
      wave->pt[ixtrack - xtrack0][itrack - track0]
        = pert->pt[itrack][ixtrack];
      wave->var[ixtrack - xtrack0][itrack - track0]
        = pert->var[itrack][ixtrack];
    }
}
```

Here is the call graph for this function:

5.35.1.20 void read_l1 (char ∗ *filename*, airs_l1_t ∗ *l1*)

Read AIRS Level-1 data.

Definition at line 1041 of file libairs.c.

```
                  {

  int ncid, varid;

  /* Open netCDF file... */
  printf("Read AIRS Level-1 file: %s\n", filename);
  NC(nc_open(filename, NC_NOWRITE, &ncid));

  /* Read data... */
  NC(nc_inq_varid(ncid, "l1_time", &varid));
  NC(nc_get_var_double(ncid, varid, l1->time[0]));
  NC(nc_inq_varid(ncid, "l1_lon", &varid));
  NC(nc_get_var_double(ncid, varid, l1->lon[0]));
  NC(nc_inq_varid(ncid, "l1_lat", &varid));
  NC(nc_get_var_double(ncid, varid, l1->lat[0]));
  NC(nc_inq_varid(ncid, "l1_sat_z", &varid));
  NC(nc_get_var_double(ncid, varid, l1->sat_z));
  NC(nc_inq_varid(ncid, "l1_sat_lon", &varid));
  NC(nc_get_var_double(ncid, varid, l1->sat_lon));
  NC(nc_inq_varid(ncid, "l1_sat_lat", &varid));
  NC(nc_get_var_double(ncid, varid, l1->sat_lat));
  NC(nc_inq_varid(ncid, "l1_nu", &varid));
  NC(nc_get_var_double(ncid, varid, l1->nu));
  NC(nc_inq_varid(ncid, "l1_rad", &varid));
  NC(nc_get_var_float(ncid, varid, l1->rad[0][0]));

  /* Close file... */
  NC(nc_close(ncid));
}
```

5.35.1.21 void read_l2 (char ∗ *filename*, airs_l2_t ∗ *l2*)

Read AIRS Level-2 data.

Definition at line 1075 of file libairs.c.

```
                {

int ncid, varid;

/* Open netCDF file... */
printf("Read AIRS Level-2 file: %s\n", filename);
NC(nc_open(filename, NC_NOWRITE, &ncid));

/* Read data... */
NC(nc_inq_varid(ncid, "l2_time", &varid));
NC(nc_get_var_double(ncid, varid, l2->time[0]));
NC(nc_inq_varid(ncid, "l2_z", &varid));
NC(nc_get_var_double(ncid, varid, l2->z[0][0]));
NC(nc_inq_varid(ncid, "l2_lon", &varid));
NC(nc_get_var_double(ncid, varid, l2->lon[0]));
NC(nc_inq_varid(ncid, "l2_lat", &varid));
NC(nc_get_var_double(ncid, varid, l2->lat[0]));
NC(nc_inq_varid(ncid, "l2_press", &varid));
NC(nc_get_var_double(ncid, varid, l2->p));
NC(nc_inq_varid(ncid, "l2_temp", &varid));
NC(nc_get_var_double(ncid, varid, l2->t[0][0]));

/* Close file... */
NC(nc_close(ncid));
}
```

5.35.1.22 void read_pert (char * *filename*, char * *pertname*, pert_t * *pert*)

Read radiance perturbation data.

Definition at line 1105 of file libairs.c.

```
               {

static char varname[LEN];

static int dimid[2], ncid, varid;

static size_t itrack, ntrack, nxtrack, start[2] = { 0, 0 }, count[2] = {
1, 1};

/* Write info... */
printf("Read perturbation data: %s\n", filename);

/* Open netCDF file... */
NC(nc_open(filename, NC_NOWRITE, &ncid));

/* Get dimensions... */
NC(nc_inq_dimid(ncid, "NTRACK", &dimid[0]));
NC(nc_inq_dimid(ncid, "NXTRACK", &dimid[1]));
NC(nc_inq_dimlen(ncid, dimid[0], &ntrack));
NC(nc_inq_dimlen(ncid, dimid[1], &nxtrack));
if (nxtrack > PERT_NXTRACK)
  ERRMSG("Too many tracks!");
if (ntrack > PERT_NTRACK)
  ERRMSG("Too many scans!");
pert->ntrack = (int) ntrack;
pert->nxtrack = (int) nxtrack;
count[1] = nxtrack;

/* Read data... */
NC(nc_inq_varid(ncid, "time", &varid));
for (itrack = 0; itrack < ntrack; itrack++) {
  start[0] = itrack;
  NC(nc_get_vara_double(ncid, varid, start, count, pert->time[itrack]));
}

NC(nc_inq_varid(ncid, "lon", &varid));
for (itrack = 0; itrack < ntrack; itrack++) {
  start[0] = itrack;
  NC(nc_get_vara_double(ncid, varid, start, count, pert->lon[itrack]));
}

NC(nc_inq_varid(ncid, "lat", &varid));
for (itrack = 0; itrack < ntrack; itrack++) {
  start[0] = itrack;
  NC(nc_get_vara_double(ncid, varid, start, count, pert->lat[itrack]));
}

NC(nc_inq_varid(ncid, "bt_8mu", &varid));
```

```
  for (itrack = 0; itrack < ntrack; itrack++) {
    start[0] = itrack;
    NC(nc_get_vara_double(ncid, varid, start, count, pert->dc[itrack]));
  }

  sprintf(varname, "bt_%s", pertname);
  NC(nc_inq_varid(ncid, varname, &varid));
  for (itrack = 0; itrack < ntrack; itrack++) {
    start[0] = itrack;
    NC(nc_get_vara_double(ncid, varid, start, count, pert->bt[itrack]));
  }

  sprintf(varname, "bt_%s_pt", pertname);
  NC(nc_inq_varid(ncid, varname, &varid));
  for (itrack = 0; itrack < ntrack; itrack++) {
    start[0] = itrack;
    NC(nc_get_vara_double(ncid, varid, start, count, pert->pt[itrack]));
  }

  sprintf(varname, "bt_%s_var", pertname);
  NC(nc_inq_varid(ncid, varname, &varid));
  for (itrack = 0; itrack < ntrack; itrack++) {
    start[0] = itrack;
    NC(nc_get_vara_double(ncid, varid, start, count, pert->var[itrack]));
  }

  /* Close file... */
  NC(nc_close(ncid));
}
```

5.35.1.23 void read_retr (char * *filename*, ret_t * *ret*)

Read AIRS retrieval data.

Definition at line 1188 of file libairs.c.

```
                {

  static double help[NDS * NPG];

  int dimid, ids = 0, ip, ncid, varid;

  size_t itrack, ixtrack, nds, np, ntrack, nxtrack;

  /* Write info... */
  printf("Read retrieval data: %s\n", filename);

  /* Open netCDF file... */
  NC(nc_open(filename, NC_NOWRITE, &ncid));

  /* Read new retrieval file format... */
  if (nc_inq_dimid(ncid, "L1_NTRACK", &dimid) == NC_NOERR) {

    /* Get dimensions... */
    NC(nc_inq_dimid(ncid, "RET_NP", &dimid));
    NC(nc_inq_dimlen(ncid, dimid, &np));
    ret->np = (int) np;
    if (ret->np > NPG)
      ERRMSG("Too many data points!");

    NC(nc_inq_dimid(ncid, "L1_NTRACK", &dimid));
    NC(nc_inq_dimlen(ncid, dimid, &ntrack));
    NC(nc_inq_dimid(ncid, "L1_NXTRACK", &dimid));
    NC(nc_inq_dimlen(ncid, dimid, &nxtrack));
    ret->nds = (int) (ntrack * nxtrack);
    if (ret->nds > NDS)
      ERRMSG("Too many data sets!");

    /* Read time... */
    NC(nc_inq_varid(ncid, "l1_time", &varid));
    NC(nc_get_var_double(ncid, varid, help));
    ids = 0;
    for (itrack = 0; itrack < ntrack; itrack++)
      for (ixtrack = 0; ixtrack < nxtrack; ixtrack++) {
        for (ip = 0; ip < ret->np; ip++)
          ret->time[ids][ip] = help[ids];
        ids++;
      }

    /* Read altitudes... */
```

```
    NC(nc_inq_varid(ncid, "ret_z", &varid));
    NC(nc_get_var_double(ncid, varid, help));
    ids = 0;
    for (itrack = 0; itrack < ntrack; itrack++)
      for (ixtrack = 0; ixtrack < nxtrack; ixtrack++) {
        for (ip = 0; ip < ret->np; ip++)
          ret->z[ids][ip] = help[ip];
        ids++;
      }

    /* Read longitudes... */
    NC(nc_inq_varid(ncid, "l1_lon", &varid));
    NC(nc_get_var_double(ncid, varid, help));
    ids = 0;
    for (itrack = 0; itrack < ntrack; itrack++)
      for (ixtrack = 0; ixtrack < nxtrack; ixtrack++) {
        for (ip = 0; ip < ret->np; ip++)
          ret->lon[ids][ip] = help[ids];
        ids++;
      }

    /* Read latitudes... */
    NC(nc_inq_varid(ncid, "l1_lat", &varid));
    NC(nc_get_var_double(ncid, varid, help));
    ids = 0;
    for (itrack = 0; itrack < ntrack; itrack++)
      for (ixtrack = 0; ixtrack < nxtrack; ixtrack++) {
        for (ip = 0; ip < ret->np; ip++)
          ret->lat[ids][ip] = help[ids];
        ids++;
      }

    /* Read temperatures... */
    NC(nc_inq_varid(ncid, "ret_temp", &varid));
    NC(nc_get_var_double(ncid, varid, help));
    ids = 0;
    for (itrack = 0; itrack < ntrack; itrack++)
      for (ixtrack = 0; ixtrack < nxtrack; ixtrack++) {
        for (ip = 0; ip < ret->np; ip++)
          ret->t[ids][ip] =
            help[(itrack * nxtrack + ixtrack) * (size_t) np + (size_t) ip];
        ids++;
      }
  }

  /* Read old retrieval file format... */
  if (nc_inq_dimid(ncid, "np", &dimid) == NC_NOERR) {

    /* Get dimensions... */
    NC(nc_inq_dimid(ncid, "np", &dimid));
    NC(nc_inq_dimlen(ncid, dimid, &np));
    ret->np = (int) np;
    if (ret->np > NPG)
      ERRMSG("Too many data points!");

    NC(nc_inq_dimid(ncid, "nds", &dimid));
    NC(nc_inq_dimlen(ncid, dimid, &nds));
    ret->nds = (int) nds;
    if (ret->nds > NDS)
      ERRMSG("Too many data sets!");

    /* Read data... */
    NC(nc_inq_varid(ncid, "time", &varid));
    NC(nc_get_var_double(ncid, varid, help));
    read_retr_help(help, ret->nds, ret->np, ret->time);

    NC(nc_inq_varid(ncid, "z", &varid));
    NC(nc_get_var_double(ncid, varid, help));
    read_retr_help(help, ret->nds, ret->np, ret->z);

    NC(nc_inq_varid(ncid, "lon", &varid));
    NC(nc_get_var_double(ncid, varid, help));
    read_retr_help(help, ret->nds, ret->np, ret->lon);

    NC(nc_inq_varid(ncid, "lat", &varid));
    NC(nc_get_var_double(ncid, varid, help));
    read_retr_help(help, ret->nds, ret->np, ret->lat);

    NC(nc_inq_varid(ncid, "press", &varid));
    NC(nc_get_var_double(ncid, varid, help));
    read_retr_help(help, ret->nds, ret->np, ret->p);

    NC(nc_inq_varid(ncid, "temp", &varid));
    NC(nc_get_var_double(ncid, varid, help));
    read_retr_help(help, ret->nds, ret->np, ret->t);

    NC(nc_inq_varid(ncid, "temp_apr", &varid));
```

```
    NC(nc_get_var_double(ncid, varid, help));
    read_retr_help(help, ret->nds, ret->np, ret->t_apr);

    NC(nc_inq_varid(ncid, "temp_total", &varid));
    NC(nc_get_var_double(ncid, varid, help));
    read_retr_help(help, ret->nds, ret->np, ret->t_tot);

    NC(nc_inq_varid(ncid, "temp_noise", &varid));
    NC(nc_get_var_double(ncid, varid, help));
    read_retr_help(help, ret->nds, ret->np, ret->t_noise);

    NC(nc_inq_varid(ncid, "temp_formod", &varid));
    NC(nc_get_var_double(ncid, varid, help));
    read_retr_help(help, ret->nds, ret->np, ret->t_fm);

    NC(nc_inq_varid(ncid, "temp_cont", &varid));
    NC(nc_get_var_double(ncid, varid, help));
    read_retr_help(help, ret->nds, ret->np, ret->t_cont);

    NC(nc_inq_varid(ncid, "temp_res", &varid));
    NC(nc_get_var_double(ncid, varid, help));
    read_retr_help(help, ret->nds, ret->np, ret->t_res);

    NC(nc_inq_varid(ncid, "chisq", &varid));
    NC(nc_get_var_double(ncid, varid, ret->chisq));
  }

  /* Close file... */
  NC(nc_close(ncid));
}
```

Here is the call graph for this function:

5.35.1.24 void read_retr_help (double ∗ *help,* int *nds,* int *np,* double *mat[NDS][NPG]*)

Convert array.

Definition at line 1354 of file libairs.c.

```
                              {

  int ids, ip, n = 0;

  for (ip = 0; ip < np; ip++)
    for (ids = 0; ids < nds; ids++)
      mat[ids][ip] = help[n++];
}
```

5.35.1.25 void read_wave (char ∗ *filename,* wave_t ∗ *wave*)

Read wave analysis data.

Definition at line 1369 of file libairs.c.

```
                     {

  FILE *in;

  char line[LEN];

  double rtime, rz, rlon, rlat, rx, ry, ryold = -1e10, rtemp, rbg, rpt, rvar;

  /* Init... */
  wave->nx = 0;
  wave->ny = 0;

  /* Write info... */
  printf("Read wave data: %s\n", filename);

  /* Open file... */
  if (!(in = fopen(filename, "r")))
    ERRMSG("Cannot open file!");

```

```
  /* Read data... */
  while (fgets(line, LEN, in))
    if (sscanf(line, "%lg %lg %lg %lg %lg %lg %lg %lg %lg %lg", &rtime,
               &rz, &rlon, &rlat, &rx, &ry, &rtemp, &rbg, &rpt,
               &rvar) == 10) {

      /* Set index... */
      if (ry != ryold) {
        if ((++wave->ny >= WY))
          ERRMSG("Too many y-values!");
        wave->nx = 0;
      } else if ((++wave->nx) >= WX)
        ERRMSG("Too many x-values!");
      ryold = ry;

      /* Save data... */
      wave->time = rtime;
      wave->z = rz;
      wave->lon[wave->nx][wave->ny] = rlon;
      wave->lat[wave->nx][wave->ny] = rlat;
      wave->x[wave->nx] = rx;
      wave->y[wave->ny] = ry;
      wave->temp[wave->nx][wave->ny] = rtemp;
      wave->bg[wave->nx][wave->ny] = rbg;
      wave->pt[wave->nx][wave->ny] = rpt;
      wave->var[wave->nx][wave->ny] = rvar;
    }

  /* Increment counters... */
  wave->nx++;
  wave->ny++;

  /* Close file... */
  fclose(in);
}
```

5.35.1.26 void rad2wave (airs_rad_gran_t * *airs_rad_gran*, double * *nu*, int *nd*, wave_t * *wave*)

Convert AIRS radiance data to wave analysis struct.

Definition at line 1428 of file libairs.c.

```
                    {

  double x0[3], x1[3];

  int ichan[AIRS_RAD_CHANNEL], id, track, xtrack;

  /* Get channel numbers... */
  for (id = 0; id < nd; id++) {
    for (ichan[id] = 0; ichan[id] < AIRS_RAD_CHANNEL; ichan[id]++)
      if (fabs(gran->nominal_freq[ichan[id]] - nu[id]) < 0.1)
        break;
    if (ichan[id] >= AIRS_RAD_CHANNEL)
      ERRMSG("Could not find channel!");
  }

  /* Set size... */
  wave->nx = AIRS_RAD_GEOXTRACK;
  wave->ny = AIRS_RAD_GEOTRACK;
  if (wave->nx > WX || wave->ny > WY)
    ERRMSG("Wave struct too small!");

  /* Set Cartesian coordinates... */
  geo2cart(0, gran->Longitude[0][0], gran->Latitude[0][0], x0);
  for (xtrack = 0; xtrack < AIRS_RAD_GEOXTRACK; xtrack++) {
    geo2cart(0, gran->Longitude[0][xtrack], gran->Latitude[0][xtrack], x1);
    wave->x[xtrack] = DIST(x0, x1);
  }
  for (track = 0; track < AIRS_RAD_GEOTRACK; track++) {
    geo2cart(0, gran->Longitude[track][0], gran->Latitude[track][0], x1);
    wave->y[track] = DIST(x0, x1);
  }

  /* Set geolocation... */
  wave->time =
    gran->Time[AIRS_RAD_GEOTRACK / 2][AIRS_RAD_GEOXTRACK / 2] - 220838400;
  wave->z = 0;
  for (track = 0; track < AIRS_RAD_GEOTRACK; track++)
    for (xtrack = 0; xtrack < AIRS_RAD_GEOXTRACK; xtrack++) {
```

```
      wave->lon[xtrack][track] = gran->Longitude[track][xtrack];
      wave->lat[xtrack][track] = gran->Latitude[track][xtrack];
    }

  /* Set brightness temperature... */
  for (track = 0; track < AIRS_RAD_GEOTRACK; track++)
    for (xtrack = 0; xtrack < AIRS_RAD_GEOXTRACK; xtrack++) {
      wave->temp[xtrack][track] = 0;
      wave->bg[xtrack][track] = 0;
      wave->pt[xtrack][track] = 0;
      wave->var[xtrack][track] = 0;
      for (id = 0; id < nd; id++) {
        if ((gran->state[track][xtrack] != 0)
            || (gran->ExcludedChans[ichan[id]] > 2)
            || (gran->CalChanSummary[ichan[id]] & 8)
            || (gran->CalChanSummary[ichan[id]] & (32 + 64))
            || (gran->CalFlag[track][ichan[id]] & 16))
          wave->temp[xtrack][track] = GSL_NAN;
        else
          wave->temp[xtrack][track]
            += brightness(gran->radiances[track][xtrack][ichan[id]] * 1e-3,
                          gran->nominal_freq[ichan[id]]) / nd;
      }
    }
}
```

Here is the call graph for this function:

5.35.1.27 void ret2wave (ret_t * *ret*, wave_t * *wave*, int *dataset*, int *ip*)

Convert AIRS retrieval results to wave analysis struct.

Definition at line 1498 of file libairs.c.

```
          {

  double x0[3], x1[3];

  int ids, ix, iy;

  /* Initialize... */
  wave->nx = 90;
  if (wave->nx > WX)
    ERRMSG("Too many across-track values!");
  wave->ny = 135;
  if (wave->ny > WY)
    ERRMSG("Too many along-track values!");
  if (ip < 0 || ip >= ret->np)
    ERRMSG("Altitude index out of range!");

  /* Loop over data sets and data points... */
  for (ids = 0; ids < ret->nds; ids++) {

    /* Get horizontal indices... */
    ix = ids % 90;
    iy = ids / 90;

    /* Get distances... */
    if (iy == 0) {
      geo2cart(0.0, ret->lon[0][0], ret->lat[0][0], x0);
      geo2cart(0.0, ret->lon[ids][ip], ret->lat[ids][ip], x1);
      wave->x[ix] = DIST(x0, x1);
    }
    if (ix == 0) {
      geo2cart(0.0, ret->lon[0][0], ret->lat[0][0], x0);
      geo2cart(0.0, ret->lon[ids][ip], ret->lat[ids][ip], x1);
      wave->y[iy] = DIST(x0, x1);
    }

    /* Save geolocation... */
    wave->time = ret->time[0][0];
    if (ix == 0 && iy == 0)
      wave->z = ret->z[ids][ip];
    wave->lon[ix][iy] = ret->lon[ids][ip];
    wave->lat[ix][iy] = ret->lat[ids][ip];

    /* Save temperature... */
    if (dataset == 1)
      wave->temp[ix][iy] = ret->t[ids][ip];
    else if (dataset == 2)
      wave->temp[ix][iy] = ret->t_apr[ids][ip];
  }
}
```

Here is the call graph for this function:

5.35.1.28 double sza (double *sec*, double *lon*, double *lat*)

Calculate solar zenith angle.

Definition at line 1554 of file libairs.c.

```
            {

double D, dec, e, g, GMST, h, L, LST, q, ra;

/* Number of days and fraction with respect to 2000-01-01T12:00Z... */
D = sec / 86400 - 0.5;

/* Geocentric apparent ecliptic longitude [rad]... */
g = (357.529 + 0.98560028 * D) * M_PI / 180;
q = 280.459 + 0.98564736 * D;
L = (q + 1.915 * sin(g) + 0.020 * sin(2 * g)) * M_PI / 180;

/* Mean obliquity of the ecliptic [rad]... */
e = (23.439 - 0.00000036 * D) * M_PI / 180;

/* Declination [rad]... */
dec = asin(sin(e) * sin(L));

/* Right ascension [rad]... */
ra = atan2(cos(e) * sin(L), cos(L));

/* Greenwich Mean Sidereal Time [h]... */
GMST = 18.697374558 + 24.06570982441908 * D;

/* Local Sidereal Time [h]... */
LST = GMST + lon / 15;

/* Hour angle [rad]... */
h = LST / 12 * M_PI - ra;

/* Convert latitude... */
lat *= M_PI / 180;

/* Return solar zenith angle [deg]... */
return acos(sin(lat) * sin(dec) +
            cos(lat) * cos(dec) * cos(h)) * 180 / M_PI;
}
```

5.35.1.29 void variance (wave_t * *wave*, double *dh*)

Compute local variance.

Definition at line 1597 of file libairs.c.

```
           {

double dh2, mu, help;

int dx, dy, ix, ix2, iy, iy2, n;

/* Check parameters... */
if (dh <= 0)
  return;

/* Compute squared radius... */
dh2 = gsl_pow_2(dh);

/* Get sampling distances... */
dx =
  (int) (dh / fabs(wave->x[wave->nx - 1] - wave->x[0]) * (wave->nx - 1.0) +
         1);
dy =
  (int) (dh / fabs(wave->y[wave->ny - 1] - wave->y[0]) * (wave->ny - 1.0) +
         1);

/* Loop over data points... */
for (ix = 0; ix < wave->nx; ix++)
```

```
    for (iy = 0; iy < wave->ny; iy++) {

      /* Init... */
      mu = help = 0;
      n = 0;

      /* Get data... */
      for (ix2 = GSL_MAX(ix - dx, 0); ix2 <= GSL_MIN(ix + dx, wave->nx - 1);
           ix2++)
        for (iy2 = GSL_MAX(iy - dy, 0); iy2 <= GSL_MIN(iy + dy, wave->ny - 1);
             iy2++)
          if ((gsl_pow_2(wave->x[ix] - wave->x[ix2])
               + gsl_pow_2(wave->y[iy] - wave->y[iy2])) <= dh2)
            if (gsl_finite(wave->pt[ix2][iy2])) {
              mu += wave->pt[ix2][iy2];
              help += gsl_pow_2(wave->pt[ix2][iy2]);
              n++;
            }

      /* Compute local variance... */
      if (n > 1)
        wave->var[ix][iy] = help / n - gsl_pow_2(mu / n);
      else
        wave->var[ix][iy] = GSL_NAN;
    }
}
```

5.35.1.30 void write_l1 (char ∗ *filename*, airs_l1_t ∗ *l1*)

Write AIRS Level-1 data.

Definition at line 1651 of file libairs.c.

```
                  {

  int dimid[10], ncid, time_id, lon_id, lat_id,
    sat_z_id, sat_lon_id, sat_lat_id, nu_id, rad_id;

  /* Open or create netCDF file... */
  printf("Write AIRS Level-1 file: %s\n", filename);
  if (nc_open(filename, NC_WRITE, &ncid) != NC_NOERR) {
    NC(nc_create(filename, NC_CLOBBER, &ncid));
  } else {
    NC(nc_redef(ncid));
  }

  /* Set dimensions... */
  if (nc_inq_dimid(ncid, "L1_NTRACK", &dimid[0]) != NC_NOERR)
    NC(nc_def_dim(ncid, "L1_NTRACK", L1_NTRACK, &dimid[0]));
  if (nc_inq_dimid(ncid, "L1_NXTRACK", &dimid[1]) != NC_NOERR)
    NC(nc_def_dim(ncid, "L1_NXTRACK", L1_NXTRACK, &dimid[1]));
  if (nc_inq_dimid(ncid, "L1_NCHAN", &dimid[2]) != NC_NOERR)
    NC(nc_def_dim(ncid, "L1_NCHAN", L1_NCHAN, &dimid[2]));

  /* Add variables... */
  add_var(ncid, "l1_time", "s", "time (seconds since 2000-01-01T00:00Z)",
          NC_DOUBLE, dimid, &time_id, 2);
  add_var(ncid, "l1_lon", "deg", "longitude", NC_DOUBLE, dimid, &lon_id, 2);
  add_var(ncid, "l1_lat", "deg", "latitude", NC_DOUBLE, dimid, &lat_id, 2);
  add_var(ncid, "l1_sat_z", "km", "satellite altitude",
          NC_DOUBLE, dimid, &sat_z_id, 1);
  add_var(ncid, "l1_sat_lon", "deg", "satellite longitude",
          NC_DOUBLE, dimid, &sat_lon_id, 1);
  add_var(ncid, "l1_sat_lat", "deg", "satellite latitude",
          NC_DOUBLE, dimid, &sat_lat_id, 1);
  add_var(ncid, "l1_nu", "cm^-1", "channel wavenumber",
          NC_DOUBLE, &dimid[2], &nu_id, 1);
  add_var(ncid, "l1_rad", "W/(m^2 sr cm^-1)", "channel radiance",
          NC_FLOAT, dimid, &rad_id, 3);

  /* Leave define mode... */
  NC(nc_enddef(ncid));

  /* Write data... */
  NC(nc_put_var_double(ncid, time_id, l1->time[0]));
  NC(nc_put_var_double(ncid, lon_id, l1->lon[0]));
  NC(nc_put_var_double(ncid, lat_id, l1->lat[0]));
  NC(nc_put_var_double(ncid, sat_z_id, l1->sat_z));
  NC(nc_put_var_double(ncid, sat_lon_id, l1->sat_lon));
  NC(nc_put_var_double(ncid, sat_lat_id, l1->sat_lat));
```

```
  NC(nc_put_var_double(ncid, nu_id, l1->nu));
  NC(nc_put_var_float(ncid, rad_id, l1->rad[0][0]));

  /* Close file... */
  NC(nc_close(ncid));
}
```

Here is the call graph for this function:

5.35.1.31 void write_l2 (char * *filename*, airs_l2_t * *l2*)

Write AIRS Level-2 data.

Definition at line 1709 of file libairs.c.

```
                   {

  int dimid[10], ncid, time_id, z_id, lon_id, lat_id, p_id, t_id;

  /* Create netCDF file... */
  printf("Write AIRS Level-2 file: %s\n", filename);
  if (nc_open(filename, NC_WRITE, &ncid) != NC_NOERR) {
    NC(nc_create(filename, NC_CLOBBER, &ncid));
  } else {
    NC(nc_redef(ncid));
  }

  /* Set dimensions... */
  if (nc_inq_dimid(ncid, "L2_NTRACK", &dimid[0]) != NC_NOERR)
    NC(nc_def_dim(ncid, "L2_NTRACK", L2_NTRACK, &dimid[0]));
  if (nc_inq_dimid(ncid, "L2_NXTRACK", &dimid[1]) != NC_NOERR)
    NC(nc_def_dim(ncid, "L2_NXTRACK", L2_NXTRACK, &dimid[1]));
  if (nc_inq_dimid(ncid, "L2_NLAY", &dimid[2]) != NC_NOERR)
    NC(nc_def_dim(ncid, "L2_NLAY", L2_NLAY, &dimid[2]));

  /* Add variables... */
  add_var(ncid, "l2_time", "s", "time (seconds since 2000-01-01T00:00Z)",
          NC_DOUBLE, dimid, &time_id, 2);
  add_var(ncid, "l2_z", "km", "altitude", NC_DOUBLE, dimid, &z_id, 3);
  add_var(ncid, "l2_lon", "deg", "longitude", NC_DOUBLE, dimid, &lon_id, 2);
  add_var(ncid, "l2_lat", "deg", "latitude", NC_DOUBLE, dimid, &lat_id, 2);
  add_var(ncid, "l2_press", "hPa", "pressure",
          NC_DOUBLE, &dimid[2], &p_id, 1);
  add_var(ncid, "l2_temp", "K", "temperature", NC_DOUBLE, dimid, &t_id, 3);

  /* Leave define mode... */
  NC(nc_enddef(ncid));

  /* Write data... */
  NC(nc_put_var_double(ncid, time_id, l2->time[0]));
  NC(nc_put_var_double(ncid, z_id, l2->z[0][0]));
  NC(nc_put_var_double(ncid, lon_id, l2->lon[0]));
  NC(nc_put_var_double(ncid, lat_id, l2->lat[0]));
  NC(nc_put_var_double(ncid, p_id, l2->p));
  NC(nc_put_var_double(ncid, t_id, l2->t[0][0]));

  /* Close file... */
  NC(nc_close(ncid));
}
```

Here is the call graph for this function:

5.35.1.32 void write_wave (char * *filename*, wave_t * *wave*)

Write wave analysis data.

Definition at line 1758 of file libairs.c.

```
                  {

  FILE *out;

  int i, j;

  /* Write info... */
  printf("Write wave data: %s\n", filename);

  /* Create file... */
  if (!(out = fopen(filename, "w")))
    ERRMSG("Cannot create file!");

  /* Write header... */
  fprintf(out,
          "# $1  = time (seconds since 2000-01-01T00:00Z)\n"
          "# $2  = altitude [km]\n"
          "# $3  = longitude [deg]\n"
          "# $4  = latitude [deg]\n"
          "# $5  = across-track distance [km]\n"
          "# $6  = along-track distance [km]\n"
          "# $7  = temperature [K]\n"
          "# $8  = background [K]\n"
          "# $9  = perturbation [K]\n" "# $10 = variance [K^2]\n");

  /* Write data... */
  for (j = 0; j < wave->ny; j++) {
    fprintf(out, "\n");
    for (i = 0; i < wave->nx; i++)
      fprintf(out, "%.2f %g %g %g %g %g %g %g %g %g\n",
              wave->time, wave->z, wave->lon[i][j], wave->lat[i][j],
              wave->x[i], wave->y[j], wave->temp[i][j], wave->bg[i][j],
              wave->pt[i][j], wave->var[i][j]);
  }

  /* Close file... */
  fclose(out);
}
```

5.36 libairs.c

```
#include "libairs.h"

/*****************************************************************************/

void add_var(
  int ncid,
  const char *varname,
  const char *unit,
  const char *longname,
  int type,
  int dimid[],
  int *varid,
  int ndims) {

  /* Check if variable exists... */
  if (nc_inq_varid(ncid, varname, varid) != NC_NOERR) {

    /* Define variable... */
    NC(nc_def_var(ncid, varname, type, ndims, dimid, varid));

    /* Set long name... */
    NC(nc_put_att_text
       (ncid, *varid, "long_name", strlen(longname), longname));

    /* Set units... */
    NC(nc_put_att_text(ncid, *varid, "units", strlen(unit), unit));
  }
}

/*****************************************************************************/

void background_poly_help(
  double *xx,
  double *yy,
  int n,
  int dim) {

  gsl_multifit_linear_workspace *work;
  gsl_matrix *cov, *X;
  gsl_vector *c, *x, *y;

  double chisq, xx2[WX > WY ? WX : WY], yy2[WX > WY ? WX : WY];
```

```

  size_t i, i2, n2 = 0;

  /* Check for nan... */
  for (i = 0; i < (size_t) n; i++)
    if (gsl_finite(yy[i])) {
      xx2[n2] = xx[i];
      yy2[n2] = yy[i];
      n2++;
    }
  if ((int) n2 < dim || n2 < 0.9 * n) {
    for (i = 0; i < (size_t) n; i++)
      yy[i] = GSL_NAN;
    return;
  }

  /* Allocate... */
  work = gsl_multifit_linear_alloc((size_t) n2, (size_t) dim);
  cov = gsl_matrix_alloc((size_t) dim, (size_t) dim);
  X = gsl_matrix_alloc((size_t) n2, (size_t) dim);
  c = gsl_vector_alloc((size_t) dim);
  x = gsl_vector_alloc((size_t) n2);
  y = gsl_vector_alloc((size_t) n2);

  /* Compute polynomial fit... */
  for (i = 0; i < (size_t) n2; i++) {
    gsl_vector_set(x, i, xx2[i]);
    gsl_vector_set(y, i, yy2[i]);
    for (i2 = 0; i2 < (size_t) dim; i2++)
      gsl_matrix_set(X, i, i2, pow(gsl_vector_get(x, i), (double) i2));
  }
  gsl_multifit_linear(X, y, c, cov, &chisq, work);
  for (i = 0; i < (size_t) n; i++)
    yy[i] = gsl_poly_eval(c->data, (int) dim, xx[i]);

  /* Free... */
  gsl_multifit_linear_free(work);
  gsl_matrix_free(cov);
  gsl_matrix_free(X);
  gsl_vector_free(c);
  gsl_vector_free(x);
  gsl_vector_free(y);
}

/*****************************************************************************/

void background_poly(
  wave_t * wave,
  int dim_x,
  int dim_y) {

  double x[WX], x2[WY], y[WX], y2[WY];

  int ix, iy;

  /* Copy temperatures to background... */
  for (ix = 0; ix < wave->nx; ix++)
    for (iy = 0; iy < wave->ny; iy++) {
      wave->bg[ix][iy] = wave->temp[ix][iy];
      wave->pt[ix][iy] = 0;
    }

  /* Check parameters... */
  if (dim_x <= 0 && dim_y <= 0)
    return;

  /* Compute fit in x-direction... */
  if (dim_x > 0)
    for (iy = 0; iy < wave->ny; iy++) {
      for (ix = 0; ix < wave->nx; ix++) {
        x[ix] = (double) ix;
        y[ix] = wave->bg[ix][iy];
      }
      background_poly_help(x, y, wave->nx, dim_x);
      for (ix = 0; ix < wave->nx; ix++)
        wave->bg[ix][iy] = y[ix];
    }

  /* Compute fit in y-direction... */
  if (dim_y > 0)
    for (ix = 0; ix < wave->nx; ix++) {
      for (iy = 0; iy < wave->ny; iy++) {
        x2[iy] = (int) iy;
        y2[iy] = wave->bg[ix][iy];
      }
      background_poly_help(x2, y2, wave->ny, dim_y);
      for (iy = 0; iy < wave->ny; iy++)
```

```
        wave->bg[ix][iy] = y2[iy];
    }

  /* Recompute perturbations... */
  for (ix = 0; ix < wave->nx; ix++)
    for (iy = 0; iy < wave->ny; iy++)
      wave->pt[ix][iy] = wave->temp[ix][iy] - wave->bg[ix][iy];
}

/*****************************************************************************/

void background_smooth(
  wave_t * wave,
  int npts_x,
  int npts_y) {

  static double help[WX][WY], dmax = 2500.;

  int dx, dy, i, j, ix, iy, n;

  /* Check parameters... */
  if (npts_x <= 0 && npts_y <= 0)
    return;

  /* Smooth background... */
  for (ix = 0; ix < wave->nx; ix++)
    for (iy = 0; iy < wave->ny; iy++) {

      /* Init... */
      n = 0;
      help[ix][iy] = 0;

      /* Set maximum range... */
      dx = GSL_MIN(GSL_MIN(npts_x, ix), wave->nx - 1 - ix);
      dy = GSL_MIN(GSL_MIN(npts_y, iy), wave->ny - 1 - iy);

      /* Average... */
      for (i = ix - dx; i <= ix + dx; i++)
        for (j = iy - dy; j <= iy + dy; j++)
          if (fabs(wave->x[ix] - wave->x[i]) < dmax &&
              fabs(wave->y[iy] - wave->y[j]) < dmax) {
            help[ix][iy] += wave->bg[i][j];
            n++;
          }

      /* Normalize... */
      if (n > 0)
        help[ix][iy] /= n;
      else
        help[ix][iy] = GSL_NAN;
    }

  /* Recalculate perturbations... */
  for (ix = 0; ix < wave->nx; ix++)
    for (iy = 0; iy < wave->ny; iy++) {
      wave->bg[ix][iy] = help[ix][iy];
      wave->pt[ix][iy] = wave->temp[ix][iy] - wave->bg[ix][iy];
    }
}

/*****************************************************************************/

void create_background(
  wave_t * wave) {

  int ix, iy;

  /* Loop over grid points... */
  for (ix = 0; ix < wave->nx; ix++)
    for (iy = 0; iy < wave->ny; iy++) {

      /* Set background for 4.3 micron BT measurements... */
      wave->bg[ix][iy] = 235.626 + 5.38165e-6 * gsl_pow_2(wave->x[ix]
                                                           -
                                                           0.5 * (wave->x[0] +
                                                                  wave->x
                                                                  [wave->nx -
                                                                   1]))
        - 1.78519e-12 * gsl_pow_4(wave->x[ix] -
                                  0.5 * (wave->x[0] + wave->x[wave->nx - 1]));

      /* Set temperature perturbation... */
      wave->pt[ix][iy] = 0;

      /* Set temperature... */
      wave->temp[ix][iy] = wave->bg[ix][iy];
    }
```

```
}

/*****************************************************************************/

void create_noise(
  wave_t * wave,
  double nedt) {

  gsl_rng *r;

  int ix, iy;

  /* Initialize random number generator... */
  gsl_rng_env_setup();
  r = gsl_rng_alloc(gsl_rng_default);
  gsl_rng_set(r, (unsigned long int) time(NULL));

  /* Add noise to temperature... */
  if (nedt > 0)
    for (ix = 0; ix < wave->nx; ix++)
      for (iy = 0; iy < wave->ny; iy++)
        wave->temp[ix][iy] += gsl_ran_gaussian(r, nedt);

  /* Free... */
  gsl_rng_free(r);
}

/*****************************************************************************/

void create_wave(
  wave_t * wave,
  double amp,
  double lx,
  double ly,
  double phi,
  double fwhm) {

  int ix, iy;

  /* Loop over grid points... */
  for (ix = 0; ix < wave->nx; ix++)
    for (iy = 0; iy < wave->ny; iy++) {

      /* Set wave perturbation... */
      wave->pt[ix][iy] = amp * cos((lx != 0 ? 2 * M_PI / lx : 0) * wave->x[ix]
                                   + (ly !=
                                      0 ? 2 * M_PI / ly : 0) * wave->y[iy]
                                   - phi * M_PI / 180.)
        * (fwhm > 0 ? exp(-0.5 * gsl_pow_2((wave->x[ix]) / (lx * fwhm) * 2.35)
                          -
                          0.5 * gsl_pow_2((wave->y[iy]) / (ly * fwhm) *
                                          2.35)) : 1.0);

      /* Add perturbation to temperature... */
      wave->temp[ix][iy] += wave->pt[ix][iy];
    }
}

/*****************************************************************************/

void day2doy(
  int year,
  int mon,
  int day,
  int *doy) {

  int d0[12] = { 1, 32, 60, 91, 121, 152, 182, 213, 244, 274, 305, 335 };
  int d0l[12] = { 1, 32, 61, 92, 122, 153, 183, 214, 245, 275, 306, 336 };

  /* Get day of year... */
  if (year % 400 == 0 || (year % 100 != 0 && year % 4 == 0))
    *doy = d0l[mon - 1] + day - 1;
  else
    *doy = d0[mon - 1] + day - 1;
}

/*****************************************************************************/

void doy2day(
  int year,
  int doy,
  int *mon,
  int *day) {

  int d0[12] = { 1, 32, 60, 91, 121, 152, 182, 213, 244, 274, 305, 335 };
  int d0l[12] = { 1, 32, 61, 92, 122, 153, 183, 214, 245, 275, 306, 336 };
  int i;
```

```
  /* Get month and day... */
  if (year % 400 == 0 || (year % 100 != 0 && year % 4 == 0)) {
    for (i = 11; i >= 0; i--)
      if (d01[i] <= doy)
        break;
    *mon = i + 1;
    *day = doy - d01[i] + 1;
  } else {
    for (i = 11; i >= 0; i--)
      if (d0[i] <= doy)
        break;
    *mon = i + 1;
    *day = doy - d0[i] + 1;
  }
}

/*****************************************************************************/

void fft_help(
  double *fcReal,
  double *fcImag,
  int n) {

  gsl_fft_complex_wavetable *wavetable;
  gsl_fft_complex_workspace *workspace;

  double data[2 * PMAX];

  int i;

  /* Check size... */
  if (n > PMAX)
    ERRMSG("Too many data points!");

  /* Allocate... */
  wavetable = gsl_fft_complex_wavetable_alloc((size_t) n);
  workspace = gsl_fft_complex_workspace_alloc((size_t) n);

  /* Set data (real, complex)... */
  for (i = 0; i < n; i++) {
    data[2 * i] = fcReal[i];
    data[2 * i + 1] = fcImag[i];
  }

  /* Calculate FFT... */
  gsl_fft_complex_forward(data, 1, (size_t) n, wavetable, workspace);

  /* Copy data... */
  for (i = 0; i < n; i++) {
    fcReal[i] = data[2 * i];
    fcImag[i] = data[2 * i + 1];
  }

  /* Free... */
  gsl_fft_complex_wavetable_free(wavetable);
  gsl_fft_complex_workspace_free(workspace);
}

/*****************************************************************************/

void fft(
  wave_t * wave,
  double *Amax,
  double *phimax,
  double *lhmax,
  double *alphamax,
  double *betamax,
  char *filename) {

  static double A[PMAX][PMAX], phi[PMAX][PMAX], kx[PMAX], ky[PMAX],
    kxmax, kymax, cutReal[PMAX], cutImag[PMAX],
    boxImag[PMAX][PMAX], boxReal[PMAX][PMAX];

  FILE *out;

  int i, i2, imin, imax, j, j2, jmin, jmax, nx, ny;

  /* Find box... */
  imin = jmin = 9999;
  imax = jmax = -9999;
  for (i = 0; i < wave->nx; i++)
    for (j = 0; j < wave->ny; j++)
      if (gsl_finite(wave->var[i][j])) {
        imin = GSL_MIN(imin, i);
        imax = GSL_MAX(imax, i);
        jmin = GSL_MIN(jmin, j);
```

```
      jmax = GSL_MAX(jmax, j);
    }
  nx = imax - imin + 1;
  ny = jmax - jmin + 1;

  /* Copy data... */
  for (i = imin; i <= imax; i++)
    for (j = jmin; j <= jmax; j++) {
      if (gsl_finite(wave->pt[i][j]))
        boxReal[i - imin][j - jmin] = wave->pt[i][j];
      else
        boxReal[i - imin][j - jmin] = 0.0;
      boxImag[i - imin][j - jmin] = 0.0;
    }

  /* FFT of the rows... */
  for (i = 0; i < nx; i++) {
    for (j = 0; j < ny; j++) {
      cutReal[j] = boxReal[i][j];
      cutImag[j] = boxImag[i][j];
    }
    fft_help(cutReal, cutImag, ny);
    for (j = 0; j < ny; j++) {
      boxReal[i][j] = cutReal[j];
      boxImag[i][j] = cutImag[j];
    }
  }

  /* FFT of the columns... */
  for (j = 0; j < ny; j++) {
    for (i = 0; i < nx; i++) {
      cutReal[i] = boxReal[i][j];
      cutImag[i] = boxImag[i][j];
    }
    fft_help(cutReal, cutImag, nx);
    for (i = 0; i < nx; i++) {
      boxReal[i][j] = cutReal[i];
      boxImag[i][j] = cutImag[i];
    }
  }

  /* Get frequencies, amplitude, and phase... */
  for (i = 0; i < nx; i++)
    kx[i] = 2. * M_PI * ((i < nx / 2) ? (double) i : -(double) (nx - i))
      / (nx * fabs(wave->x[imax] - wave->x[imin]) / (nx - 1.0));
  for (j = 0; j < ny; j++)
    ky[j] = 2. * M_PI * ((j < ny / 2) ? (double) j : -(double) (ny - j))
      / (ny * fabs(wave->y[jmax] - wave->y[jmin]) / (ny - 1.0));
  for (i = 0; i < nx; i++)
    for (j = 0; j < ny; j++) {
      A[i][j]
        = (i == 0 && j == 0 ? 1.0 : 2.0) / (nx * ny)
        * sqrt(gsl_pow_2(boxReal[i][j]) + gsl_pow_2(boxImag[i][j]));
      phi[i][j]
        = 180. / M_PI * atan2(boxImag[i][j], boxReal[i][j]);
    }

  /* Check frequencies... */
  for (i = 0; i < nx; i++)
    for (j = 0; j < ny; j++)
      if (kx[i] == 0 || ky[j] == 0) {
        A[i][j] = GSL_NAN;
        phi[i][j] = GSL_NAN;
      }

  /* Find maximum... */
  *Amax = 0;
  for (i = 0; i < nx; i++)
    for (j = 0; j < ny / 2; j++)
      if (gsl_finite(A[i][j]) && A[i][j] > *Amax) {
        *Amax = A[i][j];
        *phimax = phi[i][j];
        kxmax = kx[i];
        kymax = ky[j];
        imax = i;
        jmax = j;
      }

  /* Get horizontal wavelength... */
  *lhmax = 2 * M_PI / sqrt(gsl_pow_2(kxmax) + gsl_pow_2(kymax));

  /* Get propagation direction in xy-plane... */
  *alphamax = 90. - 180. / M_PI * atan2(kxmax, kymax);

  /* Get propagation direction in lon,lat-plane... */
  *betamax = *alphamax
    +
```

```
    180. / M_PI *
    atan2(wave->lat[wave->nx / 2 >
                    0 ? wave->nx / 2 - 1 : wave->nx / 2][wave->ny / 2]
          - wave->lat[wave->nx / 2 <
                      wave->nx - 1 ? wave->nx / 2 +
                      1 : wave->nx / 2][wave->ny / 2],
          wave->lon[wave->nx / 2 >
                    0 ? wave->nx / 2 - 1 : wave->nx / 2][wave->ny / 2]
          - wave->lon[wave->nx / 2 <
                      wave->nx - 1 ? wave->nx / 2 +
                      1 : wave->nx / 2][wave->ny / 2]);

  /* Save FFT data... */
  if (filename != NULL) {

    /* Write info... */
    printf("Write FFT data: %s\n", filename);

    /* Create file... */
    if (!(out = fopen(filename, "w")))
      ERRMSG("Cannot create file!");

    /* Write header... */
    fprintf(out,
            "# $1 = altitude [km]\n"
            "# $2 = wavelength in x-direction [km]\n"
            "# $3 = wavelength in y-direction [km]\n"
            "# $4 = wavenumber in x-direction [1/km]\n"
            "# $5 = wavenumber in y-direction [1/km]\n"
            "# $6 = amplitude [K]\n" "# $7 = phase [rad]\n");

    /* Write data... */
    for (i = nx - 1; i > 0; i--) {
      fprintf(out, "\n");
      for (j = ny / 2; j > 0; j--) {
        i2 = (i == nx / 2 ? 0 : i);
        j2 = (j == ny / 2 ? 0 : j);
        fprintf(out, "%g %g %g %g %g %g %g\n", wave->z,
                (kx[i2] != 0 ? 2 * M_PI / kx[i2] : 0),
                (ky[j2] != 0 ? 2 * M_PI / ky[j2] : 0),
                kx[i2], ky[j2], A[i2][j2], phi[i2][j2]);
      }
    }

    /* Close file... */
    fclose(out);
  }
}

/*****************************************************************************/

void gauss(
  wave_t * wave,
  double fwhm) {

  static double d2, help[WX][WY], sigma2, w, wsum;

  int ix, ix2, iy, iy2;

  /* Check parameters... */
  if (fwhm <= 0)
    return;

  /* Compute sigma^2... */
  sigma2 = gsl_pow_2(fwhm / 2.3548);

  /* Loop over data points... */
  for (ix = 0; ix < wave->nx; ix++)
    for (iy = 0; iy < wave->ny; iy++) {

      /* Init... */
      wsum = 0;
      help[ix][iy] = 0;

      /* Average... */
      for (ix2 = 0; ix2 < wave->nx; ix2++)
        for (iy2 = 0; iy2 < wave->ny; iy2++) {
          d2 = gsl_pow_2(wave->x[ix] - wave->x[ix2])
            + gsl_pow_2(wave->y[iy] - wave->y[iy2]);
          if (d2 <= 9 * sigma2) {
            w = exp(-d2 / (2 * sigma2));
            wsum += w;
            help[ix][iy] += w * wave->pt[ix2][iy2];
          }
        }

      /* Normalize... */
```

```
      wave->pt[ix][iy] = help[ix][iy] / wsum;
    }
}

/*****************************************************************************/

void hamming(
  wave_t * wave,
  int niter) {

  static double help[WX][WY];

  int iter, ix, iy;

  /* Iterations... */
  for (iter = 0; iter < niter; iter++) {

    /* Filter in x direction... */
    for (ix = 0; ix < wave->nx; ix++)
      for (iy = 0; iy < wave->ny; iy++)
        help[ix][iy]
          = 0.23 * wave->pt[ix > 0 ? ix - 1 : ix][iy]
          + 0.54 * wave->pt[ix][iy]
          + 0.23 * wave->pt[ix < wave->nx - 1 ? ix + 1 : ix][iy];

    /* Filter in y direction... */
    for (ix = 0; ix < wave->nx; ix++)
      for (iy = 0; iy < wave->ny; iy++)
        wave->pt[ix][iy]
          = 0.23 * help[ix][iy > 0 ? iy - 1 : iy]
          + 0.54 * help[ix][iy]
          + 0.23 * help[ix][iy < wave->ny - 1 ? iy + 1 : iy];
  }
}

/*****************************************************************************/

void intpol_x(
  wave_t * wave,
  int n) {

  gsl_interp_accel *acc;
  gsl_spline *spline;

  double dummy, x[WX], xc[WX][3], xc2[WX][3], y[WX];

  int i, ic, ix, iy;

  /* Check parameters... */
  if (n <= 0)
    return;
  if (n > WX)
    ERRMSG("Too many data points!");

  /* Set new x-coordinates... */
  for (i = 0; i < n; i++)
    x[i] = LIN(0.0, wave->x[0], n - 1.0, wave->x[wave->nx - 1], i);

  /* Allocate... */
  acc = gsl_interp_accel_alloc();
  spline = gsl_spline_alloc(gsl_interp_cspline, (size_t) wave->nx);

  /* Loop over scans... */
  for (iy = 0; iy < wave->ny; iy++) {

    /* Interpolate Cartesian coordinates... */
    for (ix = 0; ix < wave->nx; ix++)
      geo2cart(0, wave->lon[ix][iy], wave->lat[ix][iy], xc[ix]);
    for (ic = 0; ic < 3; ic++) {
      for (ix = 0; ix < wave->nx; ix++)
        y[ix] = xc[ix][ic];
      gsl_spline_init(spline, wave->x, y, (size_t) wave->nx);
      for (i = 0; i < n; i++)
        xc2[i][ic] = gsl_spline_eval(spline, x[i], acc);
    }
    for (i = 0; i < n; i++)
      cart2geo(xc2[i], &dummy, &wave->lon[i][iy], &wave->lat[i][iy]);

    /* Interpolate temperature... */
    for (ix = 0; ix < wave->nx; ix++)
      y[ix] = wave->temp[ix][iy];
    gsl_spline_init(spline, wave->x, y, (size_t) wave->nx);
    for (i = 0; i < n; i++)
      wave->temp[i][iy] = gsl_spline_eval(spline, x[i], acc);

    /* Interpolate background... */
    for (ix = 0; ix < wave->nx; ix++)
```

```
      y[ix] = wave->bg[ix][iy];
    gsl_spline_init(spline, wave->x, y, (size_t) wave->nx);
    for (i = 0; i < n; i++)
      wave->bg[i][iy] = gsl_spline_eval(spline, x[i], acc);

    /* Interpolate perturbations... */
    for (ix = 0; ix < wave->nx; ix++)
      y[ix] = wave->pt[ix][iy];
    gsl_spline_init(spline, wave->x, y, (size_t) wave->nx);
    for (i = 0; i < n; i++)
      wave->pt[i][iy] = gsl_spline_eval(spline, x[i], acc);

    /* Interpolate variance... */
    for (ix = 0; ix < wave->nx; ix++)
      y[ix] = wave->var[ix][iy];
    gsl_spline_init(spline, wave->x, y, (size_t) wave->nx);
    for (i = 0; i < n; i++)
      wave->var[i][iy] = gsl_spline_eval(spline, x[i], acc);
  }

  /* Free... */
  gsl_spline_free(spline);
  gsl_interp_accel_free(acc);

  /* Set new x-coordinates... */
  for (i = 0; i < n; i++)
    wave->x[i] = x[i];
  wave->nx = n;
}

/*****************************************************************************/

void median(
  wave_t * wave,
  int dx) {

  static double data[WX * WY], help[WX][WY];

  int ix, ix2, iy, iy2;

  size_t n;

  /* Check parameters... */
  if (dx <= 0)
    return;

  /* Loop over data points... */
  for (ix = 0; ix < wave->nx; ix++)
    for (iy = 0; iy < wave->ny; iy++) {

      /* Init... */
      n = 0;

      /* Get data... */
      for (ix2 = GSL_MAX(ix - dx, 0); ix2 < GSL_MIN(ix + dx, wave->nx - 1);
           ix2++)
        for (iy2 = GSL_MAX(iy - dx, 0); iy2 < GSL_MIN(iy + dx, wave->ny - 1);
             iy2++) {
          data[n] = wave->pt[ix2][iy2];
          n++;
        }

      /* Normalize... */
      gsl_sort(data, 1, n);
      help[ix][iy] = gsl_stats_median_from_sorted_data(data, 1, n);
    }

  /* Loop over data points... */
  for (ix = 0; ix < wave->nx; ix++)
    for (iy = 0; iy < wave->ny; iy++)
      wave->pt[ix][iy] = help[ix][iy];
}

/*****************************************************************************/

void merge_y(
  wave_t * wave1,
  wave_t * wave2) {

  double y;

  int ix, iy;

  /* Check data... */
  if (wave1->nx != wave2->nx)
    ERRMSG("Across-track sizes do not match!");
  if (wave1->ny + wave2->ny > WY)
```

```
    ERRMSG("Too many data points!");

  /* Get offset in y direction... */
  y =
    wave1->y[wave1->ny - 1] + (wave1->y[wave1->ny - 1] -
                               wave1->y[0]) / (wave1->ny - 1);

  /* Merge data... */
  for (ix = 0; ix < wave2->nx; ix++)
    for (iy = 0; iy < wave2->ny; iy++) {
      wave1->y[wave1->ny + iy] = y + wave2->y[iy];
      wave1->lon[ix][wave1->ny + iy] = wave2->lon[ix][iy];
      wave1->lat[ix][wave1->ny + iy] = wave2->lat[ix][iy];
      wave1->temp[ix][wave1->ny + iy] = wave2->temp[ix][iy];
      wave1->bg[ix][wave1->ny + iy] = wave2->bg[ix][iy];
      wave1->pt[ix][wave1->ny + iy] = wave2->pt[ix][iy];
      wave1->var[ix][wave1->ny + iy] = wave2->var[ix][iy];
    }

  /* Increment counter... */
  wave1->ny += wave2->ny;
}

/*****************************************************************************/

void noise(
  wave_t * wave,
  double *mu,
  double *sig) {

  int ix, ix2, iy, iy2, n = 0, okay;

  /* Init... */
  *mu = 0;
  *sig = 0;

  /* Estimate noise (Immerkaer, 1996)... */
  for (ix = 1; ix < wave->nx - 1; ix++)
    for (iy = 1; iy < wave->ny - 1; iy++) {

      /* Check data... */
      okay = 1;
      for (ix2 = ix - 1; ix2 <= ix + 1; ix2++)
        for (iy2 = iy - 1; iy2 <= iy + 1; iy2++)
          if (!gsl_finite(wave->temp[ix2][iy2]))
            okay = 0;
      if (!okay)
        continue;

      /* Get mean noise... */
      n++;
      *mu += wave->temp[ix][iy];
      *sig += gsl_pow_2(+4. / 6. * wave->temp[ix][iy]
                        - 2. / 6. * (wave->temp[ix - 1][iy]
                                     + wave->temp[ix + 1][iy]
                                     + wave->temp[ix][iy - 1]
                                     + wave->temp[ix][iy + 1])
                        + 1. / 6. * (wave->temp[ix - 1][iy - 1]
                                     + wave->temp[ix + 1][iy - 1]
                                     + wave->temp[ix - 1][iy + 1]
                                     + wave->temp[ix + 1][iy + 1]));
    }

  /* Normalize... */
  *mu /= (double) n;
  *sig = sqrt(*sig / (double) n);
}

/*****************************************************************************/

void period(
  wave_t * wave,
  double *Amax,
  double *phimax,
  double *lhmax,
  double *alphamax,
  double *betamax,
  char *filename) {

  FILE *out;

  static double kx[PMAX], ky[PMAX], kx_ny, ky_ny, kxmax, kymax, A[PMAX][PMAX],
    phi[PMAX][PMAX], cx[PMAX][WX], cy[PMAX][WY], sx[PMAX][WX], sy[PMAX][WY],
    a, b, c, lx, ly, lxymax = 1000, dlxy = 10;

  int i, imin, imax, j, jmin, jmax, l, lmax = 0, m, mmax = 0;

```

```
  /* Compute wavenumbers and periodogram coefficients... */
  for (lx = -lxymax; lx <= lxymax; lx += dlxy) {
    kx[lmax] = (lx != 0 ? 2 * M_PI / lx : 0);
    for (i = 0; i < wave->nx; i++) {
      cx[lmax][i] = cos(kx[lmax] * wave->x[i]);
      sx[lmax][i] = sin(kx[lmax] * wave->x[i]);
    }
    if ((++lmax) > PMAX)
      ERRMSG("Too many wavenumbers for periodogram!");
  }
  for (ly = 0; ly <= lxymax; ly += dlxy) {
    ky[mmax] = (ly != 0 ? 2 * M_PI / ly : 0);
    for (j = 0; j < wave->ny; j++) {
      cy[mmax][j] = cos(ky[mmax] * wave->y[j]);
      sy[mmax][j] = sin(ky[mmax] * wave->y[j]);
    }
    if ((++mmax) > PMAX)
      ERRMSG("Too many wavenumbers for periodogram!");
  }

  /* Find area... */
  imin = jmin = 9999;
  imax = jmax = -9999;
  for (i = 0; i < wave->nx; i++)
    for (j = 0; j < wave->ny; j++)
      if (gsl_finite(wave->var[i][j])) {
        imin = GSL_MIN(imin, i);
        imax = GSL_MAX(imax, i);
        jmin = GSL_MIN(jmin, j);
        jmax = GSL_MAX(jmax, j);
      }

  /* Get Nyquist frequencies... */
  kx_ny =
    M_PI / fabs((wave->x[imax] - wave->x[imin]) /
                ((double) imax - (double) imin));
  ky_ny =
    M_PI / fabs((wave->y[jmax] - wave->y[jmin]) /
                ((double) jmax - (double) jmin));

  /* Loop over wavelengths... */
  for (l = 0; l < lmax; l++)
    for (m = 0; m < mmax; m++) {

      /* Check frequencies... */
      if (kx[l] == 0 || fabs(kx[l]) > kx_ny ||
          ky[m] == 0 || fabs(ky[m]) > ky_ny) {
        A[l][m] = GSL_NAN;
        phi[l][m] = GSL_NAN;
        continue;
      }

      /* Compute periodogram... */
      a = b = c = 0;
      for (i = imin; i <= imax; i++)
        for (j = jmin; j <= jmax; j++)
          if (gsl_finite(wave->var[i][j])) {
            a += wave->pt[i][j] * (cx[l][i] * cy[m][j] - sx[l][i] * sy[m][j]);
            b += wave->pt[i][j] * (sx[l][i] * cy[m][j] + cx[l][i] * sy[m][j]);
            c++;
          }
      a *= 2. / c;
      b *= 2. / c;

      /* Get amplitude and phase... */
      A[l][m] = sqrt(gsl_pow_2(a) + gsl_pow_2(b));
      phi[l][m] = atan2(b, a) * 180. / M_PI;
    }

  /* Find maximum... */
  *Amax = 0;
  for (l = 0; l < lmax; l++)
    for (m = 0; m < mmax; m++)
      if (gsl_finite(A[l][m]) && A[l][m] > *Amax) {
        *Amax = A[l][m];
        *phimax = phi[l][m];
        kxmax = kx[l];
        kymax = ky[m];
        imax = i;
        jmax = j;
      }

  /* Get horizontal wavelength... */
  *lhmax = 2 * M_PI / sqrt(gsl_pow_2(kxmax) + gsl_pow_2(kymax));

  /* Get propagation direction in xy-plane... */
  *alphamax = 90. - 180. / M_PI * atan2(kxmax, kymax);
```

```
  /* Get propagation direction in lon,lat-plane... */
  *betamax = *alphamax
    +
    180. / M_PI *
    atan2(wave->lat[wave->nx / 2 >
                    0 ? wave->nx / 2 - 1 : wave->nx / 2][wave->ny / 2]
          - wave->lat[wave->nx / 2 <
                      wave->nx - 1 ? wave->nx / 2 +
                      1 : wave->nx / 2][wave->ny / 2],
          wave->lon[wave->nx / 2 >
                    0 ? wave->nx / 2 - 1 : wave->nx / 2][wave->ny / 2]
          - wave->lon[wave->nx / 2 <
                      wave->nx - 1 ? wave->nx / 2 +
                      1 : wave->nx / 2][wave->ny / 2]);

  /* Save periodogram data... */
  if (filename != NULL) {

    /* Write info... */
    printf("Write periodogram data: %s\n", filename);

    /* Create file... */
    if (!(out = fopen(filename, "w")))
      ERRMSG("Cannot create file!");

    /* Write header... */
    fprintf(out,
            "# $1 = altitude [km]\n"
            "# $2 = wavelength in x-direction [km]\n"
            "# $3 = wavelength in y-direction [km]\n"
            "# $4 = wavenumber in x-direction [1/km]\n"
            "# $5 = wavenumber in y-direction [1/km]\n"
            "# $6 = amplitude [K]\n" "# $7 = phase [rad]\n");

    /* Write data... */
    for (l = 0; l < lmax; l++) {
      fprintf(out, "\n");
      for (m = 0; m < mmax; m++)
        fprintf(out, "%g %g %g %g %g %g %g\n", wave->z,
                (kx[l] != 0 ? 2 * M_PI / kx[l] : 0),
                (ky[m] != 0 ? 2 * M_PI / ky[m] : 0),
                kx[l], ky[m], A[l][m], phi[l][m]);
    }

    /* Close file... */
    fclose(out);
  }
}

/*****************************************************************************/

void pert2wave(
  pert_t * pert,
  wave_t * wave,
  int track0,
  int track1,
  int xtrack0,
  int xtrack1) {

  double x0[3], x1[3];

  int itrack, ixtrack;

  /* Check ranges... */
  track0 = GSL_MIN(GSL_MAX(track0, 0), pert->ntrack - 1);
  track1 = GSL_MIN(GSL_MAX(track1, 0), pert->ntrack - 1);
  xtrack0 = GSL_MIN(GSL_MAX(xtrack0, 0), pert->nxtrack - 1);
  xtrack1 = GSL_MIN(GSL_MAX(xtrack1, 0), pert->nxtrack - 1);

  /* Set size... */
  wave->nx = xtrack1 - xtrack0 + 1;
  if (wave->nx > WX)
    ERRMSG("Too many across-track values!");
  wave->ny = track1 - track0 + 1;
  if (wave->ny > WY)
    ERRMSG("Too many along-track values!");

  /* Loop over footprints... */
  for (itrack = track0; itrack <= track1; itrack++)
    for (ixtrack = xtrack0; ixtrack <= xtrack1; ixtrack++) {

      /* Get distances... */
      if (itrack == track0) {
        wave->x[0] = 0;
        if (ixtrack > xtrack0) {
          geo2cart(0, pert->lon[itrack][ixtrack - 1],
```

```
                    pert->lat[itrack][ixtrack - 1], x0);
          geo2cart(0, pert->lon[itrack][ixtrack],
                    pert->lat[itrack][ixtrack], x1);
          wave->x[ixtrack - xtrack0] =
            wave->x[ixtrack - xtrack0 - 1] + DIST(x0, x1);
        }
      }
      if (ixtrack == xtrack0) {
        wave->y[0] = 0;
        if (itrack > track0) {
          geo2cart(0, pert->lon[itrack - 1][ixtrack],
                    pert->lat[itrack - 1][ixtrack], x0);
          geo2cart(0, pert->lon[itrack][ixtrack],
                    pert->lat[itrack][ixtrack], x1);
          wave->y[itrack - track0] =
            wave->y[itrack - track0 - 1] + DIST(x0, x1);
        }
      }

      /* Save geolocation... */
      wave->time = pert->time[(track0 + track1) / 2][(xtrack0 + xtrack1) / 2];
      wave->z = 0;
      wave->lon[ixtrack - xtrack0][itrack - track0] =
        pert->lon[itrack][ixtrack];
      wave->lat[ixtrack - xtrack0][itrack - track0] =
        pert->lat[itrack][ixtrack];

      /* Save temperature data... */
      wave->temp[ixtrack - xtrack0][itrack - track0]
        = pert->bt[itrack][ixtrack];
      wave->bg[ixtrack - xtrack0][itrack - track0]
        = pert->bt[itrack][ixtrack] - pert->pt[itrack][ixtrack];
      wave->pt[ixtrack - xtrack0][itrack - track0]
        = pert->pt[itrack][ixtrack];
      wave->var[ixtrack - xtrack0][itrack - track0]
        = pert->var[itrack][ixtrack];
    }
}

/*****************************************************************************/

void read_l1(
  char *filename,
  airs_l1_t * l1) {

  int ncid, varid;

  /* Open netCDF file... */
  printf("Read AIRS Level-1 file: %s\n", filename);
  NC(nc_open(filename, NC_NOWRITE, &ncid));

  /* Read data... */
  NC(nc_inq_varid(ncid, "l1_time", &varid));
  NC(nc_get_var_double(ncid, varid, l1->time[0]));
  NC(nc_inq_varid(ncid, "l1_lon", &varid));
  NC(nc_get_var_double(ncid, varid, l1->lon[0]));
  NC(nc_inq_varid(ncid, "l1_lat", &varid));
  NC(nc_get_var_double(ncid, varid, l1->lat[0]));
  NC(nc_inq_varid(ncid, "l1_sat_z", &varid));
  NC(nc_get_var_double(ncid, varid, l1->sat_z));
  NC(nc_inq_varid(ncid, "l1_sat_lon", &varid));
  NC(nc_get_var_double(ncid, varid, l1->sat_lon));
  NC(nc_inq_varid(ncid, "l1_sat_lat", &varid));
  NC(nc_get_var_double(ncid, varid, l1->sat_lat));
  NC(nc_inq_varid(ncid, "l1_nu", &varid));
  NC(nc_get_var_double(ncid, varid, l1->nu));
  NC(nc_inq_varid(ncid, "l1_rad", &varid));
  NC(nc_get_var_float(ncid, varid, l1->rad[0][0]));

  /* Close file... */
  NC(nc_close(ncid));
}

/*****************************************************************************/

void read_l2(
  char *filename,
  airs_l2_t * l2) {

  int ncid, varid;

  /* Open netCDF file... */
  printf("Read AIRS Level-2 file: %s\n", filename);
  NC(nc_open(filename, NC_NOWRITE, &ncid));

  /* Read data... */
  NC(nc_inq_varid(ncid, "l2_time", &varid));
```

```
  NC(nc_get_var_double(ncid, varid, l2->time[0]));
  NC(nc_inq_varid(ncid, "l2_z", &varid));
  NC(nc_get_var_double(ncid, varid, l2->z[0][0]));
  NC(nc_inq_varid(ncid, "l2_lon", &varid));
  NC(nc_get_var_double(ncid, varid, l2->lon[0]));
  NC(nc_inq_varid(ncid, "l2_lat", &varid));
  NC(nc_get_var_double(ncid, varid, l2->lat[0]));
  NC(nc_inq_varid(ncid, "l2_press", &varid));
  NC(nc_get_var_double(ncid, varid, l2->p));
  NC(nc_inq_varid(ncid, "l2_temp", &varid));
  NC(nc_get_var_double(ncid, varid, l2->t[0][0]));

  /* Close file... */
  NC(nc_close(ncid));
}

/*****************************************************************************/

void read_pert(
  char *filename,
  char *pertname,
  pert_t * pert) {

  static char varname[LEN];

  static int dimid[2], ncid, varid;

  static size_t itrack, ntrack, nxtrack, start[2] = { 0, 0 }, count[2] = {
  1, 1};

  /* Write info... */
  printf("Read perturbation data: %s\n", filename);

  /* Open netCDF file... */
  NC(nc_open(filename, NC_NOWRITE, &ncid));

  /* Get dimensions... */
  NC(nc_inq_dimid(ncid, "NTRACK", &dimid[0]));
  NC(nc_inq_dimid(ncid, "NXTRACK", &dimid[1]));
  NC(nc_inq_dimlen(ncid, dimid[0], &ntrack));
  NC(nc_inq_dimlen(ncid, dimid[1], &nxtrack));
  if (nxtrack > PERT_NXTRACK)
    ERRMSG("Too many tracks!");
  if (ntrack > PERT_NTRACK)
    ERRMSG("Too many scans!");
  pert->ntrack = (int) ntrack;
  pert->nxtrack = (int) nxtrack;
  count[1] = nxtrack;

  /* Read data... */
  NC(nc_inq_varid(ncid, "time", &varid));
  for (itrack = 0; itrack < ntrack; itrack++) {
    start[0] = itrack;
    NC(nc_get_vara_double(ncid, varid, start, count, pert->time[itrack]));
  }

  NC(nc_inq_varid(ncid, "lon", &varid));
  for (itrack = 0; itrack < ntrack; itrack++) {
    start[0] = itrack;
    NC(nc_get_vara_double(ncid, varid, start, count, pert->lon[itrack]));
  }

  NC(nc_inq_varid(ncid, "lat", &varid));
  for (itrack = 0; itrack < ntrack; itrack++) {
    start[0] = itrack;
    NC(nc_get_vara_double(ncid, varid, start, count, pert->lat[itrack]));
  }

  NC(nc_inq_varid(ncid, "bt_8mu", &varid));
  for (itrack = 0; itrack < ntrack; itrack++) {
    start[0] = itrack;
    NC(nc_get_vara_double(ncid, varid, start, count, pert->dc[itrack]));
  }

  sprintf(varname, "bt_%s", pertname);
  NC(nc_inq_varid(ncid, varname, &varid));
  for (itrack = 0; itrack < ntrack; itrack++) {
    start[0] = itrack;
    NC(nc_get_vara_double(ncid, varid, start, count, pert->bt[itrack]));
  }

  sprintf(varname, "bt_%s_pt", pertname);
  NC(nc_inq_varid(ncid, varname, &varid));
  for (itrack = 0; itrack < ntrack; itrack++) {
    start[0] = itrack;
    NC(nc_get_vara_double(ncid, varid, start, count, pert->pt[itrack]));
  }
```

```
  sprintf(varname, "bt_%s_var", pertname);
  NC(nc_inq_varid(ncid, varname, &varid));
  for (itrack = 0; itrack < ntrack; itrack++) {
    start[0] = itrack;
    NC(nc_get_vara_double(ncid, varid, start, count, pert->var[itrack]));
  }

  /* Close file... */
  NC(nc_close(ncid));
}

/*****************************************************************************/

void read_retr(
  char *filename,
  ret_t * ret) {

  static double help[NDS * NPG];

  int dimid, ids = 0, ip, ncid, varid;

  size_t itrack, ixtrack, nds, np, ntrack, nxtrack;

  /* Write info... */
  printf("Read retrieval data: %s\n", filename);

  /* Open netCDF file... */
  NC(nc_open(filename, NC_NOWRITE, &ncid));

  /* Read new retrieval file format... */
  if (nc_inq_dimid(ncid, "L1_NTRACK", &dimid) == NC_NOERR) {

    /* Get dimensions... */
    NC(nc_inq_dimid(ncid, "RET_NP", &dimid));
    NC(nc_inq_dimlen(ncid, dimid, &np));
    ret->np = (int) np;
    if (ret->np > NPG)
      ERRMSG("Too many data points!");

    NC(nc_inq_dimid(ncid, "L1_NTRACK", &dimid));
    NC(nc_inq_dimlen(ncid, dimid, &ntrack));
    NC(nc_inq_dimid(ncid, "L1_NXTRACK", &dimid));
    NC(nc_inq_dimlen(ncid, dimid, &nxtrack));
    ret->nds = (int) (ntrack * nxtrack);
    if (ret->nds > NDS)
      ERRMSG("Too many data sets!");

    /* Read time... */
    NC(nc_inq_varid(ncid, "l1_time", &varid));
    NC(nc_get_var_double(ncid, varid, help));
    ids = 0;
    for (itrack = 0; itrack < ntrack; itrack++)
      for (ixtrack = 0; ixtrack < nxtrack; ixtrack++) {
        for (ip = 0; ip < ret->np; ip++)
          ret->time[ids][ip] = help[ids];
        ids++;
      }

    /* Read altitudes... */
    NC(nc_inq_varid(ncid, "ret_z", &varid));
    NC(nc_get_var_double(ncid, varid, help));
    ids = 0;
    for (itrack = 0; itrack < ntrack; itrack++)
      for (ixtrack = 0; ixtrack < nxtrack; ixtrack++) {
        for (ip = 0; ip < ret->np; ip++)
          ret->z[ids][ip] = help[ip];
        ids++;
      }

    /* Read longitudes... */
    NC(nc_inq_varid(ncid, "l1_lon", &varid));
    NC(nc_get_var_double(ncid, varid, help));
    ids = 0;
    for (itrack = 0; itrack < ntrack; itrack++)
      for (ixtrack = 0; ixtrack < nxtrack; ixtrack++) {
        for (ip = 0; ip < ret->np; ip++)
          ret->lon[ids][ip] = help[ids];
        ids++;
      }

    /* Read latitudes... */
    NC(nc_inq_varid(ncid, "l1_lat", &varid));
    NC(nc_get_var_double(ncid, varid, help));
    ids = 0;
    for (itrack = 0; itrack < ntrack; itrack++)
      for (ixtrack = 0; ixtrack < nxtrack; ixtrack++) {
```

```
        for (ip = 0; ip < ret->np; ip++)
          ret->lat[ids][ip] = help[ids];
        ids++;
      }

    /* Read temperatures... */
    NC(nc_inq_varid(ncid, "ret_temp", &varid));
    NC(nc_get_var_double(ncid, varid, help));
    ids = 0;
    for (itrack = 0; itrack < ntrack; itrack++)
      for (ixtrack = 0; ixtrack < nxtrack; ixtrack++) {
        for (ip = 0; ip < ret->np; ip++)
          ret->t[ids][ip] =
            help[(itrack * nxtrack + ixtrack) * (size_t) np + (size_t) ip];
        ids++;
      }
  }

  /* Read old retrieval file format... */
  if (nc_inq_dimid(ncid, "np", &dimid) == NC_NOERR) {

    /* Get dimensions... */
    NC(nc_inq_dimid(ncid, "np", &dimid));
    NC(nc_inq_dimlen(ncid, dimid, &np));
    ret->np = (int) np;
    if (ret->np > NPG)
      ERRMSG("Too many data points!");

    NC(nc_inq_dimid(ncid, "nds", &dimid));
    NC(nc_inq_dimlen(ncid, dimid, &nds));
    ret->nds = (int) nds;
    if (ret->nds > NDS)
      ERRMSG("Too many data sets!");

    /* Read data... */
    NC(nc_inq_varid(ncid, "time", &varid));
    NC(nc_get_var_double(ncid, varid, help));
    read_retr_help(help, ret->nds, ret->np, ret->time);

    NC(nc_inq_varid(ncid, "z", &varid));
    NC(nc_get_var_double(ncid, varid, help));
    read_retr_help(help, ret->nds, ret->np, ret->z);

    NC(nc_inq_varid(ncid, "lon", &varid));
    NC(nc_get_var_double(ncid, varid, help));
    read_retr_help(help, ret->nds, ret->np, ret->lon);

    NC(nc_inq_varid(ncid, "lat", &varid));
    NC(nc_get_var_double(ncid, varid, help));
    read_retr_help(help, ret->nds, ret->np, ret->lat);

    NC(nc_inq_varid(ncid, "press", &varid));
    NC(nc_get_var_double(ncid, varid, help));
    read_retr_help(help, ret->nds, ret->np, ret->p);

    NC(nc_inq_varid(ncid, "temp", &varid));
    NC(nc_get_var_double(ncid, varid, help));
    read_retr_help(help, ret->nds, ret->np, ret->t);

    NC(nc_inq_varid(ncid, "temp_apr", &varid));
    NC(nc_get_var_double(ncid, varid, help));
    read_retr_help(help, ret->nds, ret->np, ret->t_apr);

    NC(nc_inq_varid(ncid, "temp_total", &varid));
    NC(nc_get_var_double(ncid, varid, help));
    read_retr_help(help, ret->nds, ret->np, ret->t_tot);

    NC(nc_inq_varid(ncid, "temp_noise", &varid));
    NC(nc_get_var_double(ncid, varid, help));
    read_retr_help(help, ret->nds, ret->np, ret->t_noise);

    NC(nc_inq_varid(ncid, "temp_formod", &varid));
    NC(nc_get_var_double(ncid, varid, help));
    read_retr_help(help, ret->nds, ret->np, ret->t_fm);

    NC(nc_inq_varid(ncid, "temp_cont", &varid));
    NC(nc_get_var_double(ncid, varid, help));
    read_retr_help(help, ret->nds, ret->np, ret->t_cont);

    NC(nc_inq_varid(ncid, "temp_res", &varid));
    NC(nc_get_var_double(ncid, varid, help));
    read_retr_help(help, ret->nds, ret->np, ret->t_res);

    NC(nc_inq_varid(ncid, "chisq", &varid));
    NC(nc_get_var_double(ncid, varid, ret->chisq));
  }

```

```
  /* Close file... */
  NC(nc_close(ncid));
}

/*****************************************************************************/

void read_retr_help(
  double *help,
  int nds,
  int np,
  double mat[NDS][NPG]) {

  int ids, ip, n = 0;

  for (ip = 0; ip < np; ip++)
    for (ids = 0; ids < nds; ids++)
      mat[ids][ip] = help[n++];
}

/*****************************************************************************/

void read_wave(
  char *filename,
  wave_t * wave) {

  FILE *in;

  char line[LEN];

  double rtime, rz, rlon, rlat, rx, ry, ryold = -1e10, rtemp, rbg, rpt, rvar;

  /* Init... */
  wave->nx = 0;
  wave->ny = 0;

  /* Write info... */
  printf("Read wave data: %s\n", filename);

  /* Open file... */
  if (!(in = fopen(filename, "r")))
    ERRMSG("Cannot open file!");

  /* Read data... */
  while (fgets(line, LEN, in))
    if (sscanf(line, "%lg %lg %lg %lg %lg %lg %lg %lg %lg %lg", &rtime,
               &rz, &rlon, &rlat, &rx, &ry, &rtemp, &rbg, &rpt,
               &rvar) == 10) {

      /* Set index... */
      if (ry != ryold) {
        if ((++wave->ny >= WY))
          ERRMSG("Too many y-values!");
        wave->nx = 0;
      } else if ((++wave->nx) >= WX)
        ERRMSG("Too many x-values!");
      ryold = ry;

      /* Save data... */
      wave->time = rtime;
      wave->z = rz;
      wave->lon[wave->nx][wave->ny] = rlon;
      wave->lat[wave->nx][wave->ny] = rlat;
      wave->x[wave->nx] = rx;
      wave->y[wave->ny] = ry;
      wave->temp[wave->nx][wave->ny] = rtemp;
      wave->bg[wave->nx][wave->ny] = rbg;
      wave->pt[wave->nx][wave->ny] = rpt;
      wave->var[wave->nx][wave->ny] = rvar;
    }

  /* Increment counters... */
  wave->nx++;
  wave->ny++;

  /* Close file... */
  fclose(in);
}

/*****************************************************************************/

void rad2wave(
  airs_rad_gran_t * gran,
  double *nu,
  int nd,
  wave_t * wave) {

  double x0[3], x1[3];
```

```
  int ichan[AIRS_RAD_CHANNEL], id, track, xtrack;

  /* Get channel numbers... */
  for (id = 0; id < nd; id++) {
    for (ichan[id] = 0; ichan[id] < AIRS_RAD_CHANNEL; ichan[id]++)
      if (fabs(gran->nominal_freq[ichan[id]] - nu[id]) < 0.1)
        break;
    if (ichan[id] >= AIRS_RAD_CHANNEL)
      ERRMSG("Could not find channel!");
  }

  /* Set size... */
  wave->nx = AIRS_RAD_GEOXTRACK;
  wave->ny = AIRS_RAD_GEOTRACK;
  if (wave->nx > WX || wave->ny > WY)
    ERRMSG("Wave struct too small!");

  /* Set Cartesian coordinates... */
  geo2cart(0, gran->Longitude[0][0], gran->Latitude[0][0], x0);
  for (xtrack = 0; xtrack < AIRS_RAD_GEOXTRACK; xtrack++) {
    geo2cart(0, gran->Longitude[0][xtrack], gran->Latitude[0][xtrack], x1);
    wave->x[xtrack] = DIST(x0, x1);
  }
  for (track = 0; track < AIRS_RAD_GEOTRACK; track++) {
    geo2cart(0, gran->Longitude[track][0], gran->Latitude[track][0], x1);
    wave->y[track] = DIST(x0, x1);
  }

  /* Set geolocation... */
  wave->time =
    gran->Time[AIRS_RAD_GEOTRACK / 2][AIRS_RAD_GEOXTRACK / 2] - 220838400;
  wave->z = 0;
  for (track = 0; track < AIRS_RAD_GEOTRACK; track++)
    for (xtrack = 0; xtrack < AIRS_RAD_GEOXTRACK; xtrack++) {
      wave->lon[xtrack][track] = gran->Longitude[track][xtrack];
      wave->lat[xtrack][track] = gran->Latitude[track][xtrack];
    }

  /* Set brightness temperature... */
  for (track = 0; track < AIRS_RAD_GEOTRACK; track++)
    for (xtrack = 0; xtrack < AIRS_RAD_GEOXTRACK; xtrack++) {
      wave->temp[xtrack][track] = 0;
      wave->bg[xtrack][track] = 0;
      wave->pt[xtrack][track] = 0;
      wave->var[xtrack][track] = 0;
      for (id = 0; id < nd; id++) {
        if ((gran->state[track][xtrack] != 0)
            || (gran->ExcludedChans[ichan[id]] > 2)
            || (gran->CalChanSummary[ichan[id]] & 8)
            || (gran->CalChanSummary[ichan[id]] & (32 + 64))
            || (gran->CalFlag[track][ichan[id]] & 16))
          wave->temp[xtrack][track] = GSL_NAN;
        else
          wave->temp[xtrack][track]
            += brightness(gran->radiances[track][xtrack][ichan[id]] * 1e-3,
                          gran->nominal_freq[ichan[id]]) / nd;
      }
    }
}

/*****************************************************************************/

void ret2wave(
  ret_t * ret,
  wave_t * wave,
  int dataset,
  int ip) {

  double x0[3], x1[3];

  int ids, ix, iy;

  /* Initialize... */
  wave->nx = 90;
  if (wave->nx > WX)
    ERRMSG("Too many across-track values!");
  wave->ny = 135;
  if (wave->ny > WY)
    ERRMSG("Too many along-track values!");
  if (ip < 0 || ip >= ret->np)
    ERRMSG("Altitude index out of range!");

  /* Loop over data sets and data points... */
  for (ids = 0; ids < ret->nds; ids++) {

    /* Get horizontal indices... */
```

```
    ix = ids % 90;
    iy = ids / 90;

    /* Get distances... */
    if (iy == 0) {
      geo2cart(0.0, ret->lon[0][0], ret->lat[0][0], x0);
      geo2cart(0.0, ret->lon[ids][ip], ret->lat[ids][ip], x1);
      wave->x[ix] = DIST(x0, x1);
    }
    if (ix == 0) {
      geo2cart(0.0, ret->lon[0][0], ret->lat[0][0], x0);
      geo2cart(0.0, ret->lon[ids][ip], ret->lat[ids][ip], x1);
      wave->y[iy] = DIST(x0, x1);
    }

    /* Save geolocation... */
    wave->time = ret->time[0][0];
    if (ix == 0 && iy == 0)
      wave->z = ret->z[ids][ip];
    wave->lon[ix][iy] = ret->lon[ids][ip];
    wave->lat[ix][iy] = ret->lat[ids][ip];

    /* Save temperature... */
    if (dataset == 1)
      wave->temp[ix][iy] = ret->t[ids][ip];
    else if (dataset == 2)
      wave->temp[ix][iy] = ret->t_apr[ids][ip];
  }
}

/*****************************************************************************/

double sza(
  double sec,
  double lon,
  double lat) {

  double D, dec, e, g, GMST, h, L, LST, q, ra;

  /* Number of days and fraction with respect to 2000-01-01T12:00Z... */
  D = sec / 86400 - 0.5;

  /* Geocentric apparent ecliptic longitude [rad]... */
  g = (357.529 + 0.98560028 * D) * M_PI / 180;
  q = 280.459 + 0.98564736 * D;
  L = (q + 1.915 * sin(g) + 0.020 * sin(2 * g)) * M_PI / 180;

  /* Mean obliquity of the ecliptic [rad]... */
  e = (23.439 - 0.00000036 * D) * M_PI / 180;

  /* Declination [rad]... */
  dec = asin(sin(e) * sin(L));

  /* Right ascension [rad]... */
  ra = atan2(cos(e) * sin(L), cos(L));

  /* Greenwich Mean Sidereal Time [h]... */
  GMST = 18.697374558 + 24.06570982441908 * D;

  /* Local Sidereal Time [h]... */
  LST = GMST + lon / 15;

  /* Hour angle [rad]... */
  h = LST / 12 * M_PI - ra;

  /* Convert latitude... */
  lat *= M_PI / 180;

  /* Return solar zenith angle [deg]... */
  return acos(sin(lat) * sin(dec) +
              cos(lat) * cos(dec) * cos(h)) * 180 / M_PI;
}

/*****************************************************************************/

void variance(
  wave_t * wave,
  double dh) {

  double dh2, mu, help;

  int dx, dy, ix, ix2, iy, iy2, n;

  /* Check parameters... */
  if (dh <= 0)
    return;

```

```
  /* Compute squared radius... */
  dh2 = gsl_pow_2(dh);

  /* Get sampling distances... */
  dx =
    (int) (dh / fabs(wave->x[wave->nx - 1] - wave->x[0]) * (wave->nx - 1.0) +
           1);
  dy =
    (int) (dh / fabs(wave->y[wave->ny - 1] - wave->y[0]) * (wave->ny - 1.0) +
           1);

  /* Loop over data points... */
  for (ix = 0; ix < wave->nx; ix++)
    for (iy = 0; iy < wave->ny; iy++) {

      /* Init... */
      mu = help = 0;
      n = 0;

      /* Get data... */
      for (ix2 = GSL_MAX(ix - dx, 0); ix2 <= GSL_MIN(ix + dx, wave->nx - 1);
           ix2++)
        for (iy2 = GSL_MAX(iy - dy, 0); iy2 <= GSL_MIN(iy + dy, wave->ny - 1);
             iy2++)
          if ((gsl_pow_2(wave->x[ix] - wave->x[ix2])
               + gsl_pow_2(wave->y[iy] - wave->y[iy2])) <= dh2)
            if (gsl_finite(wave->pt[ix2][iy2])) {
              mu += wave->pt[ix2][iy2];
              help += gsl_pow_2(wave->pt[ix2][iy2]);
              n++;
            }

      /* Compute local variance... */
      if (n > 1)
        wave->var[ix][iy] = help / n - gsl_pow_2(mu / n);
      else
        wave->var[ix][iy] = GSL_NAN;
    }
}

/*****************************************************************************/

void write_l1(
  char *filename,
  airs_l1_t * l1) {

  int dimid[10], ncid, time_id, lon_id, lat_id,
    sat_z_id, sat_lon_id, sat_lat_id, nu_id, rad_id;

  /* Open or create netCDF file... */
  printf("Write AIRS Level-1 file: %s\n", filename);
  if (nc_open(filename, NC_WRITE, &ncid) != NC_NOERR) {
    NC(nc_create(filename, NC_CLOBBER, &ncid));
  } else {
    NC(nc_redef(ncid));
  }

  /* Set dimensions... */
  if (nc_inq_dimid(ncid, "L1_NTRACK", &dimid[0]) != NC_NOERR)
    NC(nc_def_dim(ncid, "L1_NTRACK", L1_NTRACK, &dimid[0]));
  if (nc_inq_dimid(ncid, "L1_NXTRACK", &dimid[1]) != NC_NOERR)
    NC(nc_def_dim(ncid, "L1_NXTRACK", L1_NXTRACK, &dimid[1]));
  if (nc_inq_dimid(ncid, "L1_NCHAN", &dimid[2]) != NC_NOERR)
    NC(nc_def_dim(ncid, "L1_NCHAN", L1_NCHAN, &dimid[2]));

  /* Add variables... */
  add_var(ncid, "l1_time", "s", "time (seconds since 2000-01-01T00:00Z)",
          NC_DOUBLE, dimid, &time_id, 2);
  add_var(ncid, "l1_lon", "deg", "longitude", NC_DOUBLE, dimid, &lon_id, 2);
  add_var(ncid, "l1_lat", "deg", "latitude", NC_DOUBLE, dimid, &lat_id, 2);
  add_var(ncid, "l1_sat_z", "km", "satellite altitude",
          NC_DOUBLE, dimid, &sat_z_id, 1);
  add_var(ncid, "l1_sat_lon", "deg", "satellite longitude",
          NC_DOUBLE, dimid, &sat_lon_id, 1);
  add_var(ncid, "l1_sat_lat", "deg", "satellite latitude",
          NC_DOUBLE, dimid, &sat_lat_id, 1);
  add_var(ncid, "l1_nu", "cm^-1", "channel wavenumber",
          NC_DOUBLE, &dimid[2], &nu_id, 1);
  add_var(ncid, "l1_rad", "W/(m^2 sr cm^-1)", "channel radiance",
          NC_FLOAT, dimid, &rad_id, 3);

  /* Leave define mode... */
  NC(nc_enddef(ncid));

  /* Write data... */
  NC(nc_put_var_double(ncid, time_id, l1->time[0]));
  NC(nc_put_var_double(ncid, lon_id, l1->lon[0]));
```

```
  NC(nc_put_var_double(ncid, lat_id, l1->lat[0]));
  NC(nc_put_var_double(ncid, sat_z_id, l1->sat_z));
  NC(nc_put_var_double(ncid, sat_lon_id, l1->sat_lon));
  NC(nc_put_var_double(ncid, sat_lat_id, l1->sat_lat));
  NC(nc_put_var_double(ncid, nu_id, l1->nu));
  NC(nc_put_var_float(ncid, rad_id, l1->rad[0][0]));

  /* Close file... */
  NC(nc_close(ncid));
}

/*****************************************************************************/

void write_l2(
  char *filename,
  airs_l2_t * l2) {

  int dimid[10], ncid, time_id, z_id, lon_id, lat_id, p_id, t_id;

  /* Create netCDF file... */
  printf("Write AIRS Level-2 file: %s\n", filename);
  if (nc_open(filename, NC_WRITE, &ncid) != NC_NOERR) {
    NC(nc_create(filename, NC_CLOBBER, &ncid));
  } else {
    NC(nc_redef(ncid));
  }

  /* Set dimensions... */
  if (nc_inq_dimid(ncid, "L2_NTRACK", &dimid[0]) != NC_NOERR)
    NC(nc_def_dim(ncid, "L2_NTRACK", L2_NTRACK, &dimid[0]));
  if (nc_inq_dimid(ncid, "L2_NXTRACK", &dimid[1]) != NC_NOERR)
    NC(nc_def_dim(ncid, "L2_NXTRACK", L2_NXTRACK, &dimid[1]));
  if (nc_inq_dimid(ncid, "L2_NLAY", &dimid[2]) != NC_NOERR)
    NC(nc_def_dim(ncid, "L2_NLAY", L2_NLAY, &dimid[2]));

  /* Add variables... */
  add_var(ncid, "l2_time", "s", "time (seconds since 2000-01-01T00:00Z)",
          NC_DOUBLE, dimid, &time_id, 2);
  add_var(ncid, "l2_z", "km", "altitude", NC_DOUBLE, dimid, &z_id, 3);
  add_var(ncid, "l2_lon", "deg", "longitude", NC_DOUBLE, dimid, &lon_id, 2);
  add_var(ncid, "l2_lat", "deg", "latitude", NC_DOUBLE, dimid, &lat_id, 2);
  add_var(ncid, "l2_press", "hPa", "pressure",
          NC_DOUBLE, &dimid[2], &p_id, 1);
  add_var(ncid, "l2_temp", "K", "temperature", NC_DOUBLE, dimid, &t_id, 3);

  /* Leave define mode... */
  NC(nc_enddef(ncid));

  /* Write data... */
  NC(nc_put_var_double(ncid, time_id, l2->time[0]));
  NC(nc_put_var_double(ncid, z_id, l2->z[0][0]));
  NC(nc_put_var_double(ncid, lon_id, l2->lon[0]));
  NC(nc_put_var_double(ncid, lat_id, l2->lat[0]));
  NC(nc_put_var_double(ncid, p_id, l2->p));
  NC(nc_put_var_double(ncid, t_id, l2->t[0][0]));

  /* Close file... */
  NC(nc_close(ncid));
}

/*****************************************************************************/

void write_wave(
  char *filename,
  wave_t * wave) {

  FILE *out;

  int i, j;

  /* Write info... */
  printf("Write wave data: %s\n", filename);

  /* Create file... */
  if (!(out = fopen(filename, "w")))
    ERRMSG("Cannot create file!");

  /* Write header... */
  fprintf(out,
          "# $1  = time (seconds since 2000-01-01T00:00Z)\n"
          "# $2  = altitude [km]\n"
          "# $3  = longitude [deg]\n"
          "# $4  = latitude [deg]\n"
          "# $5  = across-track distance [km]\n"
          "# $6  = along-track distance [km]\n"
          "# $7  = temperature [K]\n"
          "# $8  = background [K]\n"
```

```
           "# $9  = perturbation [K]\n" "# $10 = variance [K^2]\n");

  /* Write data... */
  for (j = 0; j < wave->ny; j++) {
    fprintf(out, "\n");
    for (i = 0; i < wave->nx; i++)
      fprintf(out, "%.2f %g %g %g %g %g %g %g %g %g\n",
              wave->time, wave->z, wave->lon[i][j], wave->lat[i][j],
              wave->x[i], wave->y[j], wave->temp[i][j], wave->bg[i][j],
              wave->pt[i][j], wave->var[i][j]);
  }

  /* Close file... */
  fclose(out);
}
```

5.37 libairs.h File Reference

Data Structures

- struct airs_l1_t

  AIRS Level-1 data.
- struct airs_l2_t

  AIRS Level-2 data.
- struct pert_t

  Perturbation data.
- struct ret_t

  Retrieval results.
- struct wave_t

  Wave analysis data.

Functions

- void add_var (int ncid, const char ∗varname, const char ∗unit, const char ∗longname, int type, int dimid[ ], int ∗varid, int ndims)

  Add variable to netCDF file.
- void background_poly (wave_t ∗wave, int dim_x, int dim_y)

  Get background based on polynomial fits.
- void background_poly_help (double ∗xx, double ∗yy, int n, int dim)

  Get background based on polynomial fits.
- void background_smooth (wave_t ∗wave, int npts_x, int npts_y)

  Smooth background.
- void create_background (wave_t ∗wave)

  Set background...
- void create_noise (wave_t ∗wave, double nedt)

  Add noise to perturbations and temperatures...
- void create_wave (wave_t ∗wave, double amp, double lx, double ly, double phi, double fwhm)

  Add linear wave pattern...
- void day2doy (int year, int mon, int day, int ∗doy)

  Get day of year from date.
- void doy2day (int year, int doy, int ∗mon, int ∗day)

  Get date from day of year.
- void fft_help (double ∗fcReal, double ∗fcImag, int n)

  Calculate 1-D FFT...

- void fft (wave_t *wave, double *Amax, double *phimax, double *lhmax, double *alphamax, double *betamax, char *filename)

  Calculate 2-D FFT...
- void gauss (wave_t *wave, double fwhm)

  Apply Gaussian filter to perturbations...
- void hamming (wave_t *wave, int nit)

  Apply Hamming filter to perturbations...
- void intpol_x (wave_t *wave, int n)

  Interpolate to regular grid in x-direction.
- void median (wave_t *wave, int dx)

  Apply median filter to perturbations...
- void merge_y (wave_t *wave1, wave_t *wave2)

  Merge wave structs in y-direction.
- void noise (wave_t *wave, double *mu, double *sig)

  Estimate noise.
- void period (wave_t *wave, double *Amax, double *phimax, double *lhmax, double *alphamax, double *betamax, char *filename)

  Compute periodogram.
- void pert2wave (pert_t *pert, wave_t *wave, int track0, int track1, int xtrack0, int xtrack1)

  Convert radiance perturbation data to wave analysis struct.
- void read_l1 (char *filename, airs_l1_t *l1)

  Read AIRS Level-1 data.
- void read_l2 (char *filename, airs_l2_t *l2)

  Read AIRS Level-2 data.
- void read_pert (char *filename, char *pertname, pert_t *pert)

  Read radiance perturbation data.
- void read_retr (char *filename, ret_t *ret)

  Read AIRS retrieval data.
- void read_retr_help (double *help, int nds, int np, double mat[NDS][NPG])

  Convert array.
- void read_wave (char *filename, wave_t *wave)

  Read wave analysis data.
- void rad2wave (airs_rad_gran_t *airs_rad_gran, double *nu, int nd, wave_t *wave)

  Convert AIRS radiance data to wave analysis struct.
- void ret2wave (ret_t *ret, wave_t *wave, int dataset, int ip)

  Convert AIRS retrieval results to wave analysis struct.
- double sza (double sec, double lon, double lat)

  Calculate solar zenith angle.
- void variance (wave_t *wave, double dh)

  Compute local variance.
- void write_l1 (char *filename, airs_l1_t *l1)

  Write AIRS Level-1 data.
- void write_l2 (char *filename, airs_l2_t *l2)

  Write AIRS Level-2 data.
- void write_wave (char *filename, wave_t *wave)

  Write wave analysis data.

5.37.1 Function Documentation

5.37.1.1 void add_var (int *ncid*, const char ∗ *varname*, const char ∗ *unit*, const char ∗ *longname*, int *type*, int *dimid[ ]*, int ∗ *varid*, int *ndims*)

Add variable to netCDF file.

Definition at line 5 of file libairs.c.

```
                  {

  /* Check if variable exists... */
  if (nc_inq_varid(ncid, varname, varid) != NC_NOERR) {

    /* Define variable... */
    NC(nc_def_var(ncid, varname, type, ndims, dimid, varid));

    /* Set long name... */
    NC(nc_put_att_text
       (ncid, *varid, "long_name", strlen(longname), longname));

    /* Set units... */
    NC(nc_put_att_text(ncid, *varid, "units", strlen(unit), unit));
  }
}
```

5.37.1.2 void background_poly (wave_t ∗ *wave*, int *dim_x*, int *dim_y*)

Get background based on polynomial fits.

Definition at line 89 of file libairs.c.

```
                  {

  double x[WX], x2[WY], y[WX], y2[WY];

  int ix, iy;

  /* Copy temperatures to background... */
  for (ix = 0; ix < wave->nx; ix++)
    for (iy = 0; iy < wave->ny; iy++) {
      wave->bg[ix][iy] = wave->temp[ix][iy];
      wave->pt[ix][iy] = 0;
    }

  /* Check parameters... */
  if (dim_x <= 0 && dim_y <= 0)
    return;

  /* Compute fit in x-direction... */
  if (dim_x > 0)
    for (iy = 0; iy < wave->ny; iy++) {
      for (ix = 0; ix < wave->nx; ix++) {
        x[ix] = (double) ix;
        y[ix] = wave->bg[ix][iy];
      }
      background_poly_help(x, y, wave->nx, dim_x);
      for (ix = 0; ix < wave->nx; ix++)
        wave->bg[ix][iy] = y[ix];
    }

  /* Compute fit in y-direction... */
  if (dim_y > 0)
    for (ix = 0; ix < wave->nx; ix++) {
      for (iy = 0; iy < wave->ny; iy++) {
        x2[iy] = (int) iy;
        y2[iy] = wave->bg[ix][iy];
      }
      background_poly_help(x2, y2, wave->ny, dim_y);
      for (iy = 0; iy < wave->ny; iy++)
        wave->bg[ix][iy] = y2[iy];
    }

  /* Recompute perturbations... */
  for (ix = 0; ix < wave->nx; ix++)
    for (iy = 0; iy < wave->ny; iy++)
      wave->pt[ix][iy] = wave->temp[ix][iy] - wave->bg[ix][iy];
}
```

Here is the call graph for this function:

5.37.1.3 void background_poly_help (double * *xx*, double * *yy*, int *n*, int *dim*)

Get background based on polynomial fits.

Definition at line 32 of file libairs.c.

```
{

gsl_multifit_linear_workspace *work;
gsl_matrix *cov, *X;
gsl_vector *c, *x, *y;

double chisq, xx2[WX > WY ? WX : WY], yy2[WX > WY ? WX : WY];

size_t i, i2, n2 = 0;

/* Check for nan... */
for (i = 0; i < (size_t) n; i++)
  if (gsl_finite(yy[i])) {
    xx2[n2] = xx[i];
    yy2[n2] = yy[i];
    n2++;
  }
if ((int) n2 < dim || n2 < 0.9 * n) {
  for (i = 0; i < (size_t) n; i++)
    yy[i] = GSL_NAN;
  return;
}

/* Allocate... */
work = gsl_multifit_linear_alloc((size_t) n2, (size_t) dim);
cov = gsl_matrix_alloc((size_t) dim, (size_t) dim);
X = gsl_matrix_alloc((size_t) n2, (size_t) dim);
c = gsl_vector_alloc((size_t) dim);
x = gsl_vector_alloc((size_t) n2);
y = gsl_vector_alloc((size_t) n2);

/* Compute polynomial fit... */
for (i = 0; i < (size_t) n2; i++) {
  gsl_vector_set(x, i, xx2[i]);
  gsl_vector_set(y, i, yy2[i]);
  for (i2 = 0; i2 < (size_t) dim; i2++)
    gsl_matrix_set(X, i, i2, pow(gsl_vector_get(x, i), (double) i2));
}
gsl_multifit_linear(X, y, c, cov, &chisq, work);
for (i = 0; i < (size_t) n; i++)
  yy[i] = gsl_poly_eval(c->data, (int) dim, xx[i]);

/* Free... */
gsl_multifit_linear_free(work);
gsl_matrix_free(cov);
gsl_matrix_free(X);
gsl_vector_free(c);
gsl_vector_free(x);
gsl_vector_free(y);
}
```

5.37.1.4 void background_smooth (wave_t * *wave*, int *npts_x*, int *npts_y*)

Smooth background.

Definition at line 141 of file libairs.c.

```
{

static double help[WX][WY], dmax = 2500.;

int dx, dy, i, j, ix, iy, n;

/* Check parameters... */
if (npts_x <= 0 && npts_y <= 0)
  return;

/* Smooth background... */
for (ix = 0; ix < wave->nx; ix++)
  for (iy = 0; iy < wave->ny; iy++) {
```

```
    /* Init... */
    n = 0;
    help[ix][iy] = 0;

    /* Set maximum range... */
    dx = GSL_MIN(GSL_MIN(npts_x, ix), wave->nx - 1 - ix);
    dy = GSL_MIN(GSL_MIN(npts_y, iy), wave->ny - 1 - iy);

    /* Average... */
    for (i = ix - dx; i <= ix + dx; i++)
      for (j = iy - dy; j <= iy + dy; j++)
        if (fabs(wave->x[ix] - wave->x[i]) < dmax &&
            fabs(wave->y[iy] - wave->y[j]) < dmax) {
          help[ix][iy] += wave->bg[i][j];
          n++;
        }

    /* Normalize... */
    if (n > 0)
      help[ix][iy] /= n;
    else
      help[ix][iy] = GSL_NAN;
  }

  /* Recalculate perturbations... */
  for (ix = 0; ix < wave->nx; ix++)
    for (iy = 0; iy < wave->ny; iy++) {
      wave->bg[ix][iy] = help[ix][iy];
      wave->pt[ix][iy] = wave->temp[ix][iy] - wave->bg[ix][iy];
    }
}
```

5.37.1.5 void create_background (wave_t * *wave*)

Set background...

Definition at line 192 of file libairs.c.

```
                  {

  int ix, iy;

  /* Loop over grid points... */
  for (ix = 0; ix < wave->nx; ix++)
    for (iy = 0; iy < wave->ny; iy++) {

      /* Set background for 4.3 micron BT measurements... */
      wave->bg[ix][iy] = 235.626 + 5.38165e-6 * gsl_pow_2(wave->x[ix]
                                                          -
                                                          0.5 * (wave->x[0] +
                                                                 wave->x
                                                                 [wave->nx -
                                                                  1]))
        - 1.78519e-12 * gsl_pow_4(wave->x[ix] -
                                  0.5 * (wave->x[0] + wave->x[wave->nx - 1]));

      /* Set temperature perturbation... */
      wave->pt[ix][iy] = 0;

      /* Set temperature... */
      wave->temp[ix][iy] = wave->bg[ix][iy];
    }
}
```

5.37.1.6 void create_noise (wave_t * *wave,* double *nedt*)

Add noise to perturbations and temperatures...

Definition at line 221 of file libairs.c.

```
{

gsl_rng *r;

int ix, iy;

/* Initialize random number generator... */
gsl_rng_env_setup();
r = gsl_rng_alloc(gsl_rng_default);
gsl_rng_set(r, (unsigned long int) time(NULL));

/* Add noise to temperature... */
if (nedt > 0)
  for (ix = 0; ix < wave->nx; ix++)
    for (iy = 0; iy < wave->ny; iy++)
      wave->temp[ix][iy] += gsl_ran_gaussian(r, nedt);

/* Free... */
gsl_rng_free(r);
}
```

5.37.1.7 void create_wave (wave_t * *wave,* double *amp,* double *lx,* double *ly,* double *phi,* double *fwhm*)

Add linear wave pattern...

Definition at line 246 of file libairs.c.

```
{

int ix, iy;

/* Loop over grid points... */
for (ix = 0; ix < wave->nx; ix++)
  for (iy = 0; iy < wave->ny; iy++) {

    /* Set wave perturbation... */
    wave->pt[ix][iy] = amp * cos((lx != 0 ? 2 * M_PI / lx : 0) * wave->x[ix]
                                 + (ly !=
                                    0 ? 2 * M_PI / ly : 0) * wave->y[iy]
                                 - phi * M_PI / 180.)
      * (fwhm > 0 ? exp(-0.5 * gsl_pow_2((wave->x[ix]) / (lx * fwhm) * 2.35)
                        -
                        0.5 * gsl_pow_2((wave->y[iy]) / (ly * fwhm) *
                                        2.35)) : 1.0);

    /* Add perturbation to temperature... */
    wave->temp[ix][iy] += wave->pt[ix][iy];
  }
}
```

5.37.1.8 void day2doy (int *year,* int *mon,* int *day,* int * *doy*)

Get day of year from date.

Definition at line 277 of file libairs.c.

```
{

int d0[12] = { 1, 32, 60, 91, 121, 152, 182, 213, 244, 274, 305, 335 };
int d0l[12] = { 1, 32, 61, 92, 122, 153, 183, 214, 245, 275, 306, 336 };

/* Get day of year... */
if (year % 400 == 0 || (year % 100 != 0 && year % 4 == 0))
  *doy = d0l[mon - 1] + day - 1;
else
  *doy = d0[mon - 1] + day - 1;
}
```

5.37.1.9 void doy2day (int *year,* int *doy,* int ∗ *mon,* int ∗ *day*)

Get date from day of year.

Definition at line 295 of file libairs.c.

```
            {

  int d0[12] = { 1, 32, 60, 91, 121, 152, 182, 213, 244, 274, 305, 335 };
  int d01[12] = { 1, 32, 61, 92, 122, 153, 183, 214, 245, 275, 306, 336 };
  int i;

  /* Get month and day... */
  if (year % 400 == 0 || (year % 100 != 0 && year % 4 == 0)) {
    for (i = 11; i >= 0; i--)
      if (d01[i] <= doy)
        break;
    *mon = i + 1;
    *day = doy - d01[i] + 1;
  } else {
    for (i = 11; i >= 0; i--)
      if (d0[i] <= doy)
        break;
    *mon = i + 1;
    *day = doy - d0[i] + 1;
  }
}
```

5.37.1.10 void fft_help (double ∗ *fcReal,* double ∗ *fcImag,* int *n*)

Calculate 1-D FFT...

Definition at line 323 of file libairs.c.

```
         {

  gsl_fft_complex_wavetable *wavetable;
  gsl_fft_complex_workspace *workspace;

  double data[2 * PMAX];

  int i;

  /* Check size... */
  if (n > PMAX)
    ERRMSG("Too many data points!");

  /* Allocate... */
  wavetable = gsl_fft_complex_wavetable_alloc((size_t) n);
  workspace = gsl_fft_complex_workspace_alloc((size_t) n);

  /* Set data (real, complex)... */
  for (i = 0; i < n; i++) {
    data[2 * i] = fcReal[i];
    data[2 * i + 1] = fcImag[i];
  }

  /* Calculate FFT... */
  gsl_fft_complex_forward(data, 1, (size_t) n, wavetable, workspace);

  /* Copy data... */
  for (i = 0; i < n; i++) {
    fcReal[i] = data[2 * i];
    fcImag[i] = data[2 * i + 1];
  }

  /* Free... */
  gsl_fft_complex_wavetable_free(wavetable);
  gsl_fft_complex_workspace_free(workspace);
}
```

5.37.1.11 void fft (wave_t * *wave*, double * *Amax*, double * *phimax*, double * *lhmax*, double * *alphamax*, double * *betamax*, char * *filename*)

Calculate 2-D FFT...

Definition at line 365 of file libairs.c.

```
                    {

  static double A[PMAX][PMAX], phi[PMAX][PMAX], kx[PMAX], ky[PMAX],
    kxmax, kymax, cutReal[PMAX], cutImag[PMAX],
    boxImag[PMAX][PMAX], boxReal[PMAX][PMAX];

  FILE *out;

  int i, i2, imin, imax, j, j2, jmin, jmax, nx, ny;

  /* Find box... */
  imin = jmin = 9999;
  imax = jmax = -9999;
  for (i = 0; i < wave->nx; i++)
    for (j = 0; j < wave->ny; j++)
      if (gsl_finite(wave->var[i][j])) {
        imin = GSL_MIN(imin, i);
        imax = GSL_MAX(imax, i);
        jmin = GSL_MIN(jmin, j);
        jmax = GSL_MAX(jmax, j);
      }
  nx = imax - imin + 1;
  ny = jmax - jmin + 1;

  /* Copy data... */
  for (i = imin; i <= imax; i++)
    for (j = jmin; j <= jmax; j++) {
      if (gsl_finite(wave->pt[i][j]))
        boxReal[i - imin][j - jmin] = wave->pt[i][j];
      else
        boxReal[i - imin][j - jmin] = 0.0;
      boxImag[i - imin][j - jmin] = 0.0;
    }

  /* FFT of the rows... */
  for (i = 0; i < nx; i++) {
    for (j = 0; j < ny; j++) {
      cutReal[j] = boxReal[i][j];
      cutImag[j] = boxImag[i][j];
    }
    fft_help(cutReal, cutImag, ny);
    for (j = 0; j < ny; j++) {
      boxReal[i][j] = cutReal[j];
      boxImag[i][j] = cutImag[j];
    }
  }

  /* FFT of the columns... */
  for (j = 0; j < ny; j++) {
    for (i = 0; i < nx; i++) {
      cutReal[i] = boxReal[i][j];
      cutImag[i] = boxImag[i][j];
    }
    fft_help(cutReal, cutImag, nx);
    for (i = 0; i < nx; i++) {
      boxReal[i][j] = cutReal[i];
      boxImag[i][j] = cutImag[i];
    }
  }

  /* Get frequencies, amplitude, and phase... */
  for (i = 0; i < nx; i++)
    kx[i] = 2. * M_PI * ((i < nx / 2) ? (double) i : -(double) (nx - i))
      / (nx * fabs(wave->x[imax] - wave->x[imin]) / (nx - 1.0));
  for (j = 0; j < ny; j++)
    ky[j] = 2. * M_PI * ((j < ny / 2) ? (double) j : -(double) (ny - j))
      / (ny * fabs(wave->y[jmax] - wave->y[jmin]) / (ny - 1.0));
  for (i = 0; i < nx; i++)
    for (j = 0; j < ny; j++) {
      A[i][j]
        = (i == 0 && j == 0 ? 1.0 : 2.0) / (nx * ny)
        * sqrt(gsl_pow_2(boxReal[i][j]) + gsl_pow_2(boxImag[i][j]));
      phi[i][j]
        = 180. / M_PI * atan2(boxImag[i][j], boxReal[i][j]);
    }

```

```
  /* Check frequencies... */
  for (i = 0; i < nx; i++)
    for (j = 0; j < ny; j++)
      if (kx[i] == 0 || ky[j] == 0) {
        A[i][j] = GSL_NAN;
        phi[i][j] = GSL_NAN;
      }

  /* Find maximum... */
  *Amax = 0;
  for (i = 0; i < nx; i++)
    for (j = 0; j < ny / 2; j++)
      if (gsl_finite(A[i][j]) && A[i][j] > *Amax) {
        *Amax = A[i][j];
        *phimax = phi[i][j];
        kxmax = kx[i];
        kymax = ky[j];
        imax = i;
        jmax = j;
      }

  /* Get horizontal wavelength... */
  *lhmax = 2 * M_PI / sqrt(gsl_pow_2(kxmax) + gsl_pow_2(kymax));

  /* Get propagation direction in xy-plane... */
  *alphamax = 90. - 180. / M_PI * atan2(kxmax, kymax);

  /* Get propagation direction in lon,lat-plane... */
  *betamax = *alphamax
    +
    180. / M_PI *
    atan2(wave->lat[wave->nx / 2 >
                    0 ? wave->nx / 2 - 1 : wave->nx / 2][wave->ny / 2]
          - wave->lat[wave->nx / 2 <
                      wave->nx - 1 ? wave->nx / 2 +
                      1 : wave->nx / 2][wave->ny / 2],
          wave->lon[wave->nx / 2 >
                    0 ? wave->nx / 2 - 1 : wave->nx / 2][wave->ny / 2]
          - wave->lon[wave->nx / 2 <
                      wave->nx - 1 ? wave->nx / 2 +
                      1 : wave->nx / 2][wave->ny / 2]);

  /* Save FFT data... */
  if (filename != NULL) {

    /* Write info... */
    printf("Write FFT data: %s\n", filename);

    /* Create file... */
    if (!(out = fopen(filename, "w")))
      ERRMSG("Cannot create file!");

    /* Write header... */
    fprintf(out,
            "# $1 = altitude [km]\n"
            "# $2 = wavelength in x-direction [km]\n"
            "# $3 = wavelength in y-direction [km]\n"
            "# $4 = wavenumber in x-direction [1/km]\n"
            "# $5 = wavenumber in y-direction [1/km]\n"
            "# $6 = amplitude [K]\n" "# $7 = phase [rad]\n");

    /* Write data... */
    for (i = nx - 1; i > 0; i--) {
      fprintf(out, "\n");
      for (j = ny / 2; j > 0; j--) {
        i2 = (i == nx / 2 ? 0 : i);
        j2 = (j == ny / 2 ? 0 : j);
        fprintf(out, "%g %g %g %g %g %g %g\n", wave->z,
                (kx[i2] != 0 ? 2 * M_PI / kx[i2] : 0),
                (ky[j2] != 0 ? 2 * M_PI / ky[j2] : 0),
               kx[i2], ky[j2], A[i2][j2], phi[i2][j2]);
      }
    }

    /* Close file... */
    fclose(out);
  }
}
```

Here is the call graph for this function:

5.37.1.12 void gauss (wave_t * *wave*, double *fwhm*)

Apply Gaussian filter to perturbations...

Definition at line 529 of file libairs.c.

```
                 {

  static double d2, help[WX][WY], sigma2, w, wsum;

  int ix, ix2, iy, iy2;

  /* Check parameters... */
  if (fwhm <= 0)
    return;

  /* Compute sigma^2... */
  sigma2 = gsl_pow_2(fwhm / 2.3548);

  /* Loop over data points... */
  for (ix = 0; ix < wave->nx; ix++)
    for (iy = 0; iy < wave->ny; iy++) {

      /* Init... */
      wsum = 0;
      help[ix][iy] = 0;

      /* Average... */
      for (ix2 = 0; ix2 < wave->nx; ix2++)
        for (iy2 = 0; iy2 < wave->ny; iy2++) {
          d2 = gsl_pow_2(wave->x[ix] - wave->x[ix2])
            + gsl_pow_2(wave->y[iy] - wave->y[iy2]);
          if (d2 <= 9 * sigma2) {
            w = exp(-d2 / (2 * sigma2));
            wsum += w;
            help[ix][iy] += w * wave->pt[ix2][iy2];
          }
        }

      /* Normalize... */
      wave->pt[ix][iy] = help[ix][iy] / wsum;
    }
}
```

5.37.1.13 void hamming (wave_t * *wave*, int *nit*)

Apply Hamming filter to perturbations...

Definition at line 571 of file libairs.c.

```
                {

  static double help[WX][WY];

  int iter, ix, iy;

  /* Iterations... */
  for (iter = 0; iter < niter; iter++) {

    /* Filter in x direction... */
    for (ix = 0; ix < wave->nx; ix++)
      for (iy = 0; iy < wave->ny; iy++)
        help[ix][iy]
          = 0.23 * wave->pt[ix > 0 ? ix - 1 : ix][iy]
          + 0.54 * wave->pt[ix][iy]
          + 0.23 * wave->pt[ix < wave->nx - 1 ? ix + 1 : ix][iy];

    /* Filter in y direction... */
    for (ix = 0; ix < wave->nx; ix++)
      for (iy = 0; iy < wave->ny; iy++)
        wave->pt[ix][iy]
          = 0.23 * help[ix][iy > 0 ? iy - 1 : iy]
          + 0.54 * help[ix][iy]
          + 0.23 * help[ix][iy < wave->ny - 1 ? iy + 1 : iy];
  }
}
```

5.37.1.14 void intpol_x (wave_t * wave, int n)

Interpolate to regular grid in x-direction.

Definition at line 602 of file libairs.c.

```
        {

  gsl_interp_accel *acc;
  gsl_spline *spline;

  double dummy, x[WX], xc[WX][3], xc2[WX][3], y[WX];

  int i, ic, ix, iy;

  /* Check parameters... */
  if (n <= 0)
    return;
  if (n > WX)
    ERRMSG("Too many data points!");

  /* Set new x-coordinates... */
  for (i = 0; i < n; i++)
    x[i] = LIN(0.0, wave->x[0], n - 1.0, wave->x[wave->nx - 1], i);

  /* Allocate... */
  acc = gsl_interp_accel_alloc();
  spline = gsl_spline_alloc(gsl_interp_cspline, (size_t) wave->nx);

  /* Loop over scans... */
  for (iy = 0; iy < wave->ny; iy++) {

    /* Interpolate Cartesian coordinates... */
    for (ix = 0; ix < wave->nx; ix++)
      geo2cart(0, wave->lon[ix][iy], wave->lat[ix][iy], xc[ix]);
    for (ic = 0; ic < 3; ic++) {
      for (ix = 0; ix < wave->nx; ix++)
        y[ix] = xc[ix][ic];
      gsl_spline_init(spline, wave->x, y, (size_t) wave->nx);
      for (i = 0; i < n; i++)
        xc2[i][ic] = gsl_spline_eval(spline, x[i], acc);
    }
    for (i = 0; i < n; i++)
      cart2geo(xc2[i], &dummy, &wave->lon[i][iy], &wave->lat[i][iy]);

    /* Interpolate temperature... */
    for (ix = 0; ix < wave->nx; ix++)
      y[ix] = wave->temp[ix][iy];
    gsl_spline_init(spline, wave->x, y, (size_t) wave->nx);
    for (i = 0; i < n; i++)
      wave->temp[i][iy] = gsl_spline_eval(spline, x[i], acc);

    /* Interpolate background... */
    for (ix = 0; ix < wave->nx; ix++)
      y[ix] = wave->bg[ix][iy];
    gsl_spline_init(spline, wave->x, y, (size_t) wave->nx);
    for (i = 0; i < n; i++)
      wave->bg[i][iy] = gsl_spline_eval(spline, x[i], acc);

    /* Interpolate perturbations... */
    for (ix = 0; ix < wave->nx; ix++)
      y[ix] = wave->pt[ix][iy];
    gsl_spline_init(spline, wave->x, y, (size_t) wave->nx);
    for (i = 0; i < n; i++)
      wave->pt[i][iy] = gsl_spline_eval(spline, x[i], acc);

    /* Interpolate variance... */
    for (ix = 0; ix < wave->nx; ix++)
      y[ix] = wave->var[ix][iy];
    gsl_spline_init(spline, wave->x, y, (size_t) wave->nx);
    for (i = 0; i < n; i++)
      wave->var[i][iy] = gsl_spline_eval(spline, x[i], acc);
  }

  /* Free... */
  gsl_spline_free(spline);
  gsl_interp_accel_free(acc);

  /* Set new x-coordinates... */
  for (i = 0; i < n; i++)
    wave->x[i] = x[i];
  wave->nx = n;
}
```

Here is the call graph for this function:

5.37.1.15 void median (wave_t * wave, int dx)

Apply median filter to perturbations...

Definition at line 684 of file libairs.c.

```
            {

  static double data[WX * WY], help[WX][WY];

  int ix, ix2, iy, iy2;

  size_t n;

  /* Check parameters... */
  if (dx <= 0)
    return;

  /* Loop over data points... */
  for (ix = 0; ix < wave->nx; ix++)
    for (iy = 0; iy < wave->ny; iy++) {

      /* Init... */
      n = 0;

      /* Get data... */
      for (ix2 = GSL_MAX(ix - dx, 0); ix2 < GSL_MIN(ix + dx, wave->nx - 1);
           ix2++)
        for (iy2 = GSL_MAX(iy - dx, 0); iy2 < GSL_MIN(iy + dx, wave->ny - 1);
             iy2++) {
          data[n] = wave->pt[ix2][iy2];
          n++;
        }

      /* Normalize... */
      gsl_sort(data, 1, n);
      help[ix][iy] = gsl_stats_median_from_sorted_data(data, 1, n);
    }

  /* Loop over data points... */
  for (ix = 0; ix < wave->nx; ix++)
    for (iy = 0; iy < wave->ny; iy++)
      wave->pt[ix][iy] = help[ix][iy];
}
```

5.37.1.16 void merge_y (wave_t * wave1, wave_t * wave2)

Merge wave structs in y-direction.

Definition at line 727 of file libairs.c.

```
                    {

  double y;

  int ix, iy;

  /* Check data... */
  if (wave1->nx != wave2->nx)
    ERRMSG("Across-track sizes do not match!");
  if (wave1->ny + wave2->ny > WY)
    ERRMSG("Too many data points!");

  /* Get offset in y direction... */
  y =
    wave1->y[wave1->ny - 1] + (wave1->y[wave1->ny - 1] -
                               wave1->y[0]) / (wave1->ny - 1);

  /* Merge data... */
  for (ix = 0; ix < wave2->nx; ix++)
    for (iy = 0; iy < wave2->ny; iy++) {
      wave1->y[wave1->ny + iy] = y + wave2->y[iy];
      wave1->lon[ix][wave1->ny + iy] = wave2->lon[ix][iy];
      wave1->lat[ix][wave1->ny + iy] = wave2->lat[ix][iy];
      wave1->temp[ix][wave1->ny + iy] = wave2->temp[ix][iy];
      wave1->bg[ix][wave1->ny + iy] = wave2->bg[ix][iy];
      wave1->pt[ix][wave1->ny + iy] = wave2->pt[ix][iy];
      wave1->var[ix][wave1->ny + iy] = wave2->var[ix][iy];
    }

  /* Increment counter... */
  wave1->ny += wave2->ny;
}
```

5.37.1.17 void noise (wave_t * *wave*, double * *mu*, double * *sig*)

Estimate noise.

Definition at line 764 of file libairs.c.

```
                 {

  int ix, ix2, iy, iy2, n = 0, okay;

  /* Init... */
  *mu = 0;
  *sig = 0;

  /* Estimate noise (Immerkaer, 1996)... */
  for (ix = 1; ix < wave->nx - 1; ix++)
    for (iy = 1; iy < wave->ny - 1; iy++) {

      /* Check data... */
      okay = 1;
      for (ix2 = ix - 1; ix2 <= ix + 1; ix2++)
        for (iy2 = iy - 1; iy2 <= iy + 1; iy2++)
          if (!gsl_finite(wave->temp[ix2][iy2]))
            okay = 0;
      if (!okay)
        continue;

      /* Get mean noise... */
      n++;
      *mu += wave->temp[ix][iy];
      *sig += gsl_pow_2(+4. / 6. * wave->temp[ix][iy]
                        - 2. / 6. * (wave->temp[ix - 1][iy]
                                     + wave->temp[ix + 1][iy]
                                     + wave->temp[ix][iy - 1]
                                     + wave->temp[ix][iy + 1])
                        + 1. / 6. * (wave->temp[ix - 1][iy - 1]
                                     + wave->temp[ix + 1][iy - 1]
                                     + wave->temp[ix - 1][iy + 1]
                                     + wave->temp[ix + 1][iy + 1]));
    }

  /* Normalize... */
  *mu /= (double) n;
  *sig = sqrt(*sig / (double) n);
}
```

5.37.1.18 void period (wave_t * *wave*, double * *Amax*, double * *phimax*, double * *lhmax*, double * *alphamax*, double * *betamax*, char * *filename*)

Compute periodogram.

Definition at line 809 of file libairs.c.

```
                    {

  FILE *out;

  static double kx[PMAX], ky[PMAX], kx_ny, ky_ny, kxmax, kymax, A[PMAX][PMAX],
    phi[PMAX][PMAX], cx[PMAX][WX], cy[PMAX][WY], sx[PMAX][WX], sy[PMAX][WY],
    a, b, c, lx, ly, lxymax = 1000, dlxy = 10;

  int i, imin, imax, j, jmin, jmax, l, lmax = 0, m, mmax = 0;

  /* Compute wavenumbers and periodogram coefficients... */
  for (lx = -lxymax; lx <= lxymax; lx += dlxy) {
    kx[lmax] = (lx != 0 ? 2 * M_PI / lx : 0);
    for (i = 0; i < wave->nx; i++) {
      cx[lmax][i] = cos(kx[lmax] * wave->x[i]);
      sx[lmax][i] = sin(kx[lmax] * wave->x[i]);
    }
    if ((++lmax) > PMAX)
      ERRMSG("Too many wavenumbers for periodogram!");
  }
  for (ly = 0; ly <= lxymax; ly += dlxy) {
    ky[mmax] = (ly != 0 ? 2 * M_PI / ly : 0);
    for (j = 0; j < wave->ny; j++) {
```

```
      cy[mmax][j] = cos(ky[mmax] * wave->y[j]);
      sy[mmax][j] = sin(ky[mmax] * wave->y[j]);
    }
    if ((++mmax) > PMAX)
      ERRMSG("Too many wavenumbers for periodogram!");
  }

  /* Find area... */
  imin = jmin = 9999;
  imax = jmax = -9999;
  for (i = 0; i < wave->nx; i++)
    for (j = 0; j < wave->ny; j++)
      if (gsl_finite(wave->var[i][j])) {
        imin = GSL_MIN(imin, i);
        imax = GSL_MAX(imax, i);
        jmin = GSL_MIN(jmin, j);
        jmax = GSL_MAX(jmax, j);
      }

  /* Get Nyquist frequencies... */
  kx_ny =
    M_PI / fabs((wave->x[imax] - wave->x[imin]) /
                ((double) imax - (double) imin));
  ky_ny =
    M_PI / fabs((wave->y[jmax] - wave->y[jmin]) /
                ((double) jmax - (double) jmin));

  /* Loop over wavelengths... */
  for (l = 0; l < lmax; l++)
    for (m = 0; m < mmax; m++) {

      /* Check frequencies... */
      if (kx[l] == 0 || fabs(kx[l]) > kx_ny ||
          ky[m] == 0 || fabs(ky[m]) > ky_ny) {
        A[l][m] = GSL_NAN;
        phi[l][m] = GSL_NAN;
        continue;
      }

      /* Compute periodogram... */
      a = b = c = 0;
      for (i = imin; i <= imax; i++)
        for (j = jmin; j <= jmax; j++)
          if (gsl_finite(wave->var[i][j])) {
            a += wave->pt[i][j] * (cx[l][i] * cy[m][j] - sx[l][i] * sy[m][j]);
            b += wave->pt[i][j] * (sx[l][i] * cy[m][j] + cx[l][i] * sy[m][j]);
            c++;
          }
      a *= 2. / c;
      b *= 2. / c;

      /* Get amplitude and phase... */
      A[l][m] = sqrt(gsl_pow_2(a) + gsl_pow_2(b));
      phi[l][m] = atan2(b, a) * 180. / M_PI;
    }

  /* Find maximum... */
  *Amax = 0;
  for (l = 0; l < lmax; l++)
    for (m = 0; m < mmax; m++)
      if (gsl_finite(A[l][m]) && A[l][m] > *Amax) {
        *Amax = A[l][m];
        *phimax = phi[l][m];
        kxmax = kx[l];
        kymax = ky[m];
        imax = i;
        jmax = j;
      }

  /* Get horizontal wavelength... */
  *lhmax = 2 * M_PI / sqrt(gsl_pow_2(kxmax) + gsl_pow_2(kymax));

  /* Get propagation direction in xy-plane... */
  *alphamax = 90. - 180. / M_PI * atan2(kxmax, kymax);

  /* Get propagation direction in lon,lat-plane... */
  *betamax = *alphamax
    +
    180. / M_PI *
    atan2(wave->lat[wave->nx / 2 >
                    0 ? wave->nx / 2 - 1 : wave->nx / 2][wave->ny / 2]
          - wave->lat[wave->nx / 2 <
                      wave->nx - 1 ? wave->nx / 2 +
                      1 : wave->nx / 2][wave->ny / 2],
          wave->lon[wave->nx / 2 >
                    0 ? wave->nx / 2 - 1 : wave->nx / 2][wave->ny / 2]
          - wave->lon[wave->nx / 2 <
```

```
                           wave->nx - 1 ? wave->nx / 2 +
                           1 : wave->nx / 2][wave->ny / 2]);

  /* Save periodogram data... */
  if (filename != NULL) {

    /* Write info... */
    printf("Write periodogram data: %s\n", filename);

    /* Create file... */
    if (!(out = fopen(filename, "w")))
      ERRMSG("Cannot create file!");

    /* Write header... */
    fprintf(out,
            "# $1 = altitude [km]\n"
            "# $2 = wavelength in x-direction [km]\n"
            "# $3 = wavelength in y-direction [km]\n"
            "# $4 = wavenumber in x-direction [1/km]\n"
            "# $5 = wavenumber in y-direction [1/km]\n"
            "# $6 = amplitude [K]\n" "# $7 = phase [rad]\n");

    /* Write data... */
    for (l = 0; l < lmax; l++) {
      fprintf(out, "\n");
      for (m = 0; m < mmax; m++)
        fprintf(out, "%g %g %g %g %g %g %g\n", wave->z,
                (kx[l] != 0 ? 2 * M_PI / kx[l] : 0),
                (ky[m] != 0 ? 2 * M_PI / ky[m] : 0),
                kx[l], ky[m], A[l][m], phi[l][m]);
    }

    /* Close file... */
    fclose(out);
  }
}
```

5.37.1.19 void pert2wave (pert_t * *pert,* wave_t * *wave,* int *track0,* int *track1,* int *xtrack0,* int *xtrack1*)

Convert radiance perturbation data to wave analysis struct.

Definition at line 965 of file libairs.c.

```
                  {

  double x0[3], x1[3];

  int itrack, ixtrack;

  /* Check ranges... */
  track0 = GSL_MIN(GSL_MAX(track0, 0), pert->ntrack - 1);
  track1 = GSL_MIN(GSL_MAX(track1, 0), pert->ntrack - 1);
  xtrack0 = GSL_MIN(GSL_MAX(xtrack0, 0), pert->nxtrack - 1);
  xtrack1 = GSL_MIN(GSL_MAX(xtrack1, 0), pert->nxtrack - 1);

  /* Set size... */
  wave->nx = xtrack1 - xtrack0 + 1;
  if (wave->nx > WX)
    ERRMSG("Too many across-track values!");
  wave->ny = track1 - track0 + 1;
  if (wave->ny > WY)
    ERRMSG("Too many along-track values!");

  /* Loop over footprints... */
  for (itrack = track0; itrack <= track1; itrack++)
    for (ixtrack = xtrack0; ixtrack <= xtrack1; ixtrack++) {

      /* Get distances... */
      if (itrack == track0) {
        wave->x[0] = 0;
        if (ixtrack > xtrack0) {
          geo2cart(0, pert->lon[itrack][ixtrack - 1],
                   pert->lat[itrack][ixtrack - 1], x0);
          geo2cart(0, pert->lon[itrack][ixtrack],
                   pert->lat[itrack][ixtrack], x1);
          wave->x[ixtrack - xtrack0] =
            wave->x[ixtrack - xtrack0 - 1] + DIST(x0, x1);
        }
      }
      if (ixtrack == xtrack0) {
```

```
        wave->y[0] = 0;
        if (itrack > track0) {
          geo2cart(0, pert->lon[itrack - 1][ixtrack],
                   pert->lat[itrack - 1][ixtrack], x0);
          geo2cart(0, pert->lon[itrack][ixtrack],
                   pert->lat[itrack][ixtrack], x1);
          wave->y[itrack - track0] =
            wave->y[itrack - track0 - 1] + DIST(x0, x1);
        }
      }

      /* Save geolocation... */
      wave->time = pert->time[(track0 + track1) / 2][(xtrack0 + xtrack1) / 2];
      wave->z = 0;
      wave->lon[ixtrack - xtrack0][itrack - track0] =
        pert->lon[itrack][ixtrack];
      wave->lat[ixtrack - xtrack0][itrack - track0] =
        pert->lat[itrack][ixtrack];

      /* Save temperature data... */
      wave->temp[ixtrack - xtrack0][itrack - track0]
        = pert->bt[itrack][ixtrack];
      wave->bg[ixtrack - xtrack0][itrack - track0]
        = pert->bt[itrack][ixtrack] - pert->pt[itrack][ixtrack];
      wave->pt[ixtrack - xtrack0][itrack - track0]
        = pert->pt[itrack][ixtrack];
      wave->var[ixtrack - xtrack0][itrack - track0]
        = pert->var[itrack][ixtrack];
    }
}
```

Here is the call graph for this function:

5.37.1.20 void read_l1 (char ∗ *filename,* airs_l1_t ∗ *l1*)

Read AIRS Level-1 data.

Definition at line 1041 of file libairs.c.

```
                    {

  int ncid, varid;

  /* Open netCDF file... */
  printf("Read AIRS Level-1 file: %s\n", filename);
  NC(nc_open(filename, NC_NOWRITE, &ncid));

  /* Read data... */
  NC(nc_inq_varid(ncid, "l1_time", &varid));
  NC(nc_get_var_double(ncid, varid, l1->time[0]));
  NC(nc_inq_varid(ncid, "l1_lon", &varid));
  NC(nc_get_var_double(ncid, varid, l1->lon[0]));
  NC(nc_inq_varid(ncid, "l1_lat", &varid));
  NC(nc_get_var_double(ncid, varid, l1->lat[0]));
  NC(nc_inq_varid(ncid, "l1_sat_z", &varid));
  NC(nc_get_var_double(ncid, varid, l1->sat_z));
  NC(nc_inq_varid(ncid, "l1_sat_lon", &varid));
  NC(nc_get_var_double(ncid, varid, l1->sat_lon));
  NC(nc_inq_varid(ncid, "l1_sat_lat", &varid));
  NC(nc_get_var_double(ncid, varid, l1->sat_lat));
  NC(nc_inq_varid(ncid, "l1_nu", &varid));
  NC(nc_get_var_double(ncid, varid, l1->nu));
  NC(nc_inq_varid(ncid, "l1_rad", &varid));
  NC(nc_get_var_float(ncid, varid, l1->rad[0][0]));

  /* Close file... */
  NC(nc_close(ncid));
}
```

5.37.1.21 void read_l2 (char ∗ *filename,* airs_l2_t ∗ *l2*)

Read AIRS Level-2 data.

Definition at line 1075 of file libairs.c.

```
                {

int ncid, varid;

/* Open netCDF file... */
printf("Read AIRS Level-2 file: %s\n", filename);
NC(nc_open(filename, NC_NOWRITE, &ncid));

/* Read data... */
NC(nc_inq_varid(ncid, "l2_time", &varid));
NC(nc_get_var_double(ncid, varid, l2->time[0]));
NC(nc_inq_varid(ncid, "l2_z", &varid));
NC(nc_get_var_double(ncid, varid, l2->z[0][0]));
NC(nc_inq_varid(ncid, "l2_lon", &varid));
NC(nc_get_var_double(ncid, varid, l2->lon[0]));
NC(nc_inq_varid(ncid, "l2_lat", &varid));
NC(nc_get_var_double(ncid, varid, l2->lat[0]));
NC(nc_inq_varid(ncid, "l2_press", &varid));
NC(nc_get_var_double(ncid, varid, l2->p));
NC(nc_inq_varid(ncid, "l2_temp", &varid));
NC(nc_get_var_double(ncid, varid, l2->t[0][0]));

/* Close file... */
NC(nc_close(ncid));
}
```

5.37.1.22 void read_pert (char * *filename,* char * *pertname,* pert_t * *pert*)

Read radiance perturbation data.

Definition at line 1105 of file libairs.c.

```
              {

static char varname[LEN];

static int dimid[2], ncid, varid;

static size_t itrack, ntrack, nxtrack, start[2] = { 0, 0 }, count[2] = {
1, 1};

/* Write info... */
printf("Read perturbation data: %s\n", filename);

/* Open netCDF file... */
NC(nc_open(filename, NC_NOWRITE, &ncid));

/* Get dimensions... */
NC(nc_inq_dimid(ncid, "NTRACK", &dimid[0]));
NC(nc_inq_dimid(ncid, "NXTRACK", &dimid[1]));
NC(nc_inq_dimlen(ncid, dimid[0], &ntrack));
NC(nc_inq_dimlen(ncid, dimid[1], &nxtrack));
if (nxtrack > PERT_NXTRACK)
  ERRMSG("Too many tracks!");
if (ntrack > PERT_NTRACK)
  ERRMSG("Too many scans!");
pert->ntrack = (int) ntrack;
pert->nxtrack = (int) nxtrack;
count[1] = nxtrack;

/* Read data... */
NC(nc_inq_varid(ncid, "time", &varid));
for (itrack = 0; itrack < ntrack; itrack++) {
  start[0] = itrack;
  NC(nc_get_vara_double(ncid, varid, start, count, pert->time[itrack]));
}

NC(nc_inq_varid(ncid, "lon", &varid));
for (itrack = 0; itrack < ntrack; itrack++) {
  start[0] = itrack;
  NC(nc_get_vara_double(ncid, varid, start, count, pert->lon[itrack]));
}

NC(nc_inq_varid(ncid, "lat", &varid));
for (itrack = 0; itrack < ntrack; itrack++) {
  start[0] = itrack;
  NC(nc_get_vara_double(ncid, varid, start, count, pert->lat[itrack]));
}

NC(nc_inq_varid(ncid, "bt_8mu", &varid));
```

```
  for (itrack = 0; itrack < ntrack; itrack++) {
    start[0] = itrack;
    NC(nc_get_vara_double(ncid, varid, start, count, pert->dc[itrack]));
  }

  sprintf(varname, "bt_%s", pertname);
  NC(nc_inq_varid(ncid, varname, &varid));
  for (itrack = 0; itrack < ntrack; itrack++) {
    start[0] = itrack;
    NC(nc_get_vara_double(ncid, varid, start, count, pert->bt[itrack]));
  }

  sprintf(varname, "bt_%s_pt", pertname);
  NC(nc_inq_varid(ncid, varname, &varid));
  for (itrack = 0; itrack < ntrack; itrack++) {
    start[0] = itrack;
    NC(nc_get_vara_double(ncid, varid, start, count, pert->pt[itrack]));
  }

  sprintf(varname, "bt_%s_var", pertname);
  NC(nc_inq_varid(ncid, varname, &varid));
  for (itrack = 0; itrack < ntrack; itrack++) {
    start[0] = itrack;
    NC(nc_get_vara_double(ncid, varid, start, count, pert->var[itrack]));
  }

  /* Close file... */
  NC(nc_close(ncid));
}
```

5.37.1.23 void read_retr (char * *filename*, ret_t * *ret*)

Read AIRS retrieval data.

Definition at line 1188 of file libairs.c.

```
                {

  static double help[NDS * NPG];

  int dimid, ids = 0, ip, ncid, varid;

  size_t itrack, ixtrack, nds, np, ntrack, nxtrack;

  /* Write info... */
  printf("Read retrieval data: %s\n", filename);

  /* Open netCDF file... */
  NC(nc_open(filename, NC_NOWRITE, &ncid));

  /* Read new retrieval file format... */
  if (nc_inq_dimid(ncid, "L1_NTRACK", &dimid) == NC_NOERR) {

    /* Get dimensions... */
    NC(nc_inq_dimid(ncid, "RET_NP", &dimid));
    NC(nc_inq_dimlen(ncid, dimid, &np));
    ret->np = (int) np;
    if (ret->np > NPG)
      ERRMSG("Too many data points!");

    NC(nc_inq_dimid(ncid, "L1_NTRACK", &dimid));
    NC(nc_inq_dimlen(ncid, dimid, &ntrack));
    NC(nc_inq_dimid(ncid, "L1_NXTRACK", &dimid));
    NC(nc_inq_dimlen(ncid, dimid, &nxtrack));
    ret->nds = (int) (ntrack * nxtrack);
    if (ret->nds > NDS)
      ERRMSG("Too many data sets!");

    /* Read time... */
    NC(nc_inq_varid(ncid, "l1_time", &varid));
    NC(nc_get_var_double(ncid, varid, help));
    ids = 0;
    for (itrack = 0; itrack < ntrack; itrack++)
      for (ixtrack = 0; ixtrack < nxtrack; ixtrack++) {
        for (ip = 0; ip < ret->np; ip++)
          ret->time[ids][ip] = help[ids];
        ids++;
      }

    /* Read altitudes... */
```

```
    NC(nc_inq_varid(ncid, "ret_z", &varid));
    NC(nc_get_var_double(ncid, varid, help));
    ids = 0;
    for (itrack = 0; itrack < ntrack; itrack++)
      for (ixtrack = 0; ixtrack < nxtrack; ixtrack++) {
        for (ip = 0; ip < ret->np; ip++)
          ret->z[ids][ip] = help[ip];
        ids++;
      }

    /* Read longitudes... */
    NC(nc_inq_varid(ncid, "l1_lon", &varid));
    NC(nc_get_var_double(ncid, varid, help));
    ids = 0;
    for (itrack = 0; itrack < ntrack; itrack++)
      for (ixtrack = 0; ixtrack < nxtrack; ixtrack++) {
        for (ip = 0; ip < ret->np; ip++)
          ret->lon[ids][ip] = help[ids];
        ids++;
      }

    /* Read latitudes... */
    NC(nc_inq_varid(ncid, "l1_lat", &varid));
    NC(nc_get_var_double(ncid, varid, help));
    ids = 0;
    for (itrack = 0; itrack < ntrack; itrack++)
      for (ixtrack = 0; ixtrack < nxtrack; ixtrack++) {
        for (ip = 0; ip < ret->np; ip++)
          ret->lat[ids][ip] = help[ids];
        ids++;
      }

    /* Read temperatures... */
    NC(nc_inq_varid(ncid, "ret_temp", &varid));
    NC(nc_get_var_double(ncid, varid, help));
    ids = 0;
    for (itrack = 0; itrack < ntrack; itrack++)
      for (ixtrack = 0; ixtrack < nxtrack; ixtrack++) {
        for (ip = 0; ip < ret->np; ip++)
          ret->t[ids][ip] =
            help[(itrack * nxtrack + ixtrack) * (size_t) np + (size_t) ip];
        ids++;
      }
  }

  /* Read old retrieval file format... */
  if (nc_inq_dimid(ncid, "np", &dimid) == NC_NOERR) {

    /* Get dimensions... */
    NC(nc_inq_dimid(ncid, "np", &dimid));
    NC(nc_inq_dimlen(ncid, dimid, &np));
    ret->np = (int) np;
    if (ret->np > NPG)
      ERRMSG("Too many data points!");

    NC(nc_inq_dimid(ncid, "nds", &dimid));
    NC(nc_inq_dimlen(ncid, dimid, &nds));
    ret->nds = (int) nds;
    if (ret->nds > NDS)
      ERRMSG("Too many data sets!");

    /* Read data... */
    NC(nc_inq_varid(ncid, "time", &varid));
    NC(nc_get_var_double(ncid, varid, help));
    read_retr_help(help, ret->nds, ret->np, ret->time);

    NC(nc_inq_varid(ncid, "z", &varid));
    NC(nc_get_var_double(ncid, varid, help));
    read_retr_help(help, ret->nds, ret->np, ret->z);

    NC(nc_inq_varid(ncid, "lon", &varid));
    NC(nc_get_var_double(ncid, varid, help));
    read_retr_help(help, ret->nds, ret->np, ret->lon);

    NC(nc_inq_varid(ncid, "lat", &varid));
    NC(nc_get_var_double(ncid, varid, help));
    read_retr_help(help, ret->nds, ret->np, ret->lat);

    NC(nc_inq_varid(ncid, "press", &varid));
    NC(nc_get_var_double(ncid, varid, help));
    read_retr_help(help, ret->nds, ret->np, ret->p);

    NC(nc_inq_varid(ncid, "temp", &varid));
    NC(nc_get_var_double(ncid, varid, help));
    read_retr_help(help, ret->nds, ret->np, ret->t);

    NC(nc_inq_varid(ncid, "temp_apr", &varid));
```

```
    NC(nc_get_var_double(ncid, varid, help));
    read_retr_help(help, ret->nds, ret->np, ret->t_apr);

    NC(nc_inq_varid(ncid, "temp_total", &varid));
    NC(nc_get_var_double(ncid, varid, help));
    read_retr_help(help, ret->nds, ret->np, ret->t_tot);

    NC(nc_inq_varid(ncid, "temp_noise", &varid));
    NC(nc_get_var_double(ncid, varid, help));
    read_retr_help(help, ret->nds, ret->np, ret->t_noise);

    NC(nc_inq_varid(ncid, "temp_formod", &varid));
    NC(nc_get_var_double(ncid, varid, help));
    read_retr_help(help, ret->nds, ret->np, ret->t_fm);

    NC(nc_inq_varid(ncid, "temp_cont", &varid));
    NC(nc_get_var_double(ncid, varid, help));
    read_retr_help(help, ret->nds, ret->np, ret->t_cont);

    NC(nc_inq_varid(ncid, "temp_res", &varid));
    NC(nc_get_var_double(ncid, varid, help));
    read_retr_help(help, ret->nds, ret->np, ret->t_res);

    NC(nc_inq_varid(ncid, "chisq", &varid));
    NC(nc_get_var_double(ncid, varid, ret->chisq));
  }

  /* Close file... */
  NC(nc_close(ncid));
}
```

Here is the call graph for this function:

5.37.1.24 void read_retr_help (double ∗ *help,* int *nds,* int *np,* double *mat[NDS][NPG]*)

Convert array.

Definition at line 1354 of file libairs.c.

```
                        {

  int ids, ip, n = 0;

  for (ip = 0; ip < np; ip++)
    for (ids = 0; ids < nds; ids++)
      mat[ids][ip] = help[n++];
}
```

5.37.1.25 void read_wave (char ∗ *filename,* wave_t ∗ *wave*)

Read wave analysis data.

Definition at line 1369 of file libairs.c.

```
                  {

  FILE *in;

  char line[LEN];

  double rtime, rz, rlon, rlat, rx, ry, ryold = -1e10, rtemp, rbg, rpt, rvar;

  /* Init... */
  wave->nx = 0;
  wave->ny = 0;

  /* Write info... */
  printf("Read wave data: %s\n", filename);

  /* Open file... */
  if (!(in = fopen(filename, "r")))
    ERRMSG("Cannot open file!");

```

```
  /* Read data... */
  while (fgets(line, LEN, in))
    if (sscanf(line, "%lg %lg %lg %lg %lg %lg %lg %lg %lg %lg", &rtime,
               &rz, &rlon, &rlat, &rx, &ry, &rtemp, &rbg, &rpt,
               &rvar) == 10) {

      /* Set index... */
      if (ry != ryold) {
        if ((++wave->ny >= WY))
          ERRMSG("Too many y-values!");
        wave->nx = 0;
      } else if ((++wave->nx) >= WX)
        ERRMSG("Too many x-values!");
      ryold = ry;

      /* Save data... */
      wave->time = rtime;
      wave->z = rz;
      wave->lon[wave->nx][wave->ny] = rlon;
      wave->lat[wave->nx][wave->ny] = rlat;
      wave->x[wave->nx] = rx;
      wave->y[wave->ny] = ry;
      wave->temp[wave->nx][wave->ny] = rtemp;
      wave->bg[wave->nx][wave->ny] = rbg;
      wave->pt[wave->nx][wave->ny] = rpt;
      wave->var[wave->nx][wave->ny] = rvar;
    }

  /* Increment counters... */
  wave->nx++;
  wave->ny++;

  /* Close file... */
  fclose(in);
}
```

5.37.1.26 void rad2wave (airs_rad_gran_t * *airs_rad_gran*, double * *nu*, int *nd*, wave_t * *wave*)

Convert AIRS radiance data to wave analysis struct.

Definition at line 1428 of file libairs.c.

```
                  {

  double x0[3], x1[3];

  int ichan[AIRS_RAD_CHANNEL], id, track, xtrack;

  /* Get channel numbers... */
  for (id = 0; id < nd; id++) {
    for (ichan[id] = 0; ichan[id] < AIRS_RAD_CHANNEL; ichan[id]++)
      if (fabs(gran->nominal_freq[ichan[id]] - nu[id]) < 0.1)
        break;
    if (ichan[id] >= AIRS_RAD_CHANNEL)
      ERRMSG("Could not find channel!");
  }

  /* Set size... */
  wave->nx = AIRS_RAD_GEOXTRACK;
  wave->ny = AIRS_RAD_GEOTRACK;
  if (wave->nx > WX || wave->ny > WY)
    ERRMSG("Wave struct too small!");

  /* Set Cartesian coordinates... */
  geo2cart(0, gran->Longitude[0][0], gran->Latitude[0][0], x0);
  for (xtrack = 0; xtrack < AIRS_RAD_GEOXTRACK; xtrack++) {
    geo2cart(0, gran->Longitude[0][xtrack], gran->Latitude[0][xtrack], x1);
    wave->x[xtrack] = DIST(x0, x1);
  }
  for (track = 0; track < AIRS_RAD_GEOTRACK; track++) {
    geo2cart(0, gran->Longitude[track][0], gran->Latitude[track][0], x1);
    wave->y[track] = DIST(x0, x1);
  }

  /* Set geolocation... */
  wave->time =
    gran->Time[AIRS_RAD_GEOTRACK / 2][AIRS_RAD_GEOXTRACK / 2] - 220838400;
  wave->z = 0;
  for (track = 0; track < AIRS_RAD_GEOTRACK; track++)
    for (xtrack = 0; xtrack < AIRS_RAD_GEOXTRACK; xtrack++) {
```

```
      wave->lon[xtrack][track] = gran->Longitude[track][xtrack];
      wave->lat[xtrack][track] = gran->Latitude[track][xtrack];
    }

  /* Set brightness temperature... */
  for (track = 0; track < AIRS_RAD_GEOTRACK; track++)
    for (xtrack = 0; xtrack < AIRS_RAD_GEOXTRACK; xtrack++) {
      wave->temp[xtrack][track] = 0;
      wave->bg[xtrack][track] = 0;
      wave->pt[xtrack][track] = 0;
      wave->var[xtrack][track] = 0;
      for (id = 0; id < nd; id++) {
        if ((gran->state[track][xtrack] != 0)
            || (gran->ExcludedChans[ichan[id]] > 2)
            || (gran->CalChanSummary[ichan[id]] & 8)
            || (gran->CalChanSummary[ichan[id]] & (32 + 64))
            || (gran->CalFlag[track][ichan[id]] & 16))
          wave->temp[xtrack][track] = GSL_NAN;
        else
          wave->temp[xtrack][track]
            += brightness(gran->radiances[track][xtrack][ichan[id]] * 1e-3,
                          gran->nominal_freq[ichan[id]]) / nd;
      }
    }
}
```

Here is the call graph for this function:

5.37.1.27 void ret2wave (ret_t * *ret*, wave_t * *wave*, int *dataset*, int *ip*)

Convert AIRS retrieval results to wave analysis struct.

Definition at line 1498 of file libairs.c.

```
          {

  double x0[3], x1[3];

  int ids, ix, iy;

  /* Initialize... */
  wave->nx = 90;
  if (wave->nx > WX)
    ERRMSG("Too many across-track values!");
  wave->ny = 135;
  if (wave->ny > WY)
    ERRMSG("Too many along-track values!");
  if (ip < 0 || ip >= ret->np)
    ERRMSG("Altitude index out of range!");

  /* Loop over data sets and data points... */
  for (ids = 0; ids < ret->nds; ids++) {

    /* Get horizontal indices... */
    ix = ids % 90;
    iy = ids / 90;

    /* Get distances... */
    if (iy == 0) {
      geo2cart(0.0, ret->lon[0][0], ret->lat[0][0], x0);
      geo2cart(0.0, ret->lon[ids][ip], ret->lat[ids][ip], x1);
      wave->x[ix] = DIST(x0, x1);
    }
    if (ix == 0) {
      geo2cart(0.0, ret->lon[0][0], ret->lat[0][0], x0);
      geo2cart(0.0, ret->lon[ids][ip], ret->lat[ids][ip], x1);
      wave->y[iy] = DIST(x0, x1);
    }

    /* Save geolocation... */
    wave->time = ret->time[0][0];
    if (ix == 0 && iy == 0)
      wave->z = ret->z[ids][ip];
    wave->lon[ix][iy] = ret->lon[ids][ip];
    wave->lat[ix][iy] = ret->lat[ids][ip];

    /* Save temperature... */
    if (dataset == 1)
      wave->temp[ix][iy] = ret->t[ids][ip];
    else if (dataset == 2)
      wave->temp[ix][iy] = ret->t_apr[ids][ip];
  }
}
```

Here is the call graph for this function:

5.37.1.28 double sza (double *sec*, double *lon*, double *lat*)

Calculate solar zenith angle.

Definition at line 1554 of file libairs.c.

```
              {

  double D, dec, e, g, GMST, h, L, LST, q, ra;

  /* Number of days and fraction with respect to 2000-01-01T12:00Z... */
  D = sec / 86400 - 0.5;

  /* Geocentric apparent ecliptic longitude [rad]... */
  g = (357.529 + 0.98560028 * D) * M_PI / 180;
  q = 280.459 + 0.98564736 * D;
  L = (q + 1.915 * sin(g) + 0.020 * sin(2 * g)) * M_PI / 180;

  /* Mean obliquity of the ecliptic [rad]... */
  e = (23.439 - 0.00000036 * D) * M_PI / 180;

  /* Declination [rad]... */
  dec = asin(sin(e) * sin(L));

  /* Right ascension [rad]... */
  ra = atan2(cos(e) * sin(L), cos(L));

  /* Greenwich Mean Sidereal Time [h]... */
  GMST = 18.697374558 + 24.06570982441908 * D;

  /* Local Sidereal Time [h]... */
  LST = GMST + lon / 15;

  /* Hour angle [rad]... */
  h = LST / 12 * M_PI - ra;

  /* Convert latitude... */
  lat *= M_PI / 180;

  /* Return solar zenith angle [deg]... */
  return acos(sin(lat) * sin(dec) +
              cos(lat) * cos(dec) * cos(h)) * 180 / M_PI;
}
```

5.37.1.29 void variance (wave_t * *wave*, double *dh*)

Compute local variance.

Definition at line 1597 of file libairs.c.

```
              {

  double dh2, mu, help;

  int dx, dy, ix, ix2, iy, iy2, n;

  /* Check parameters... */
  if (dh <= 0)
    return;

  /* Compute squared radius... */
  dh2 = gsl_pow_2(dh);

  /* Get sampling distances... */
  dx =
    (int) (dh / fabs(wave->x[wave->nx - 1] - wave->x[0]) * (wave->nx - 1.0) +
           1);
  dy =
    (int) (dh / fabs(wave->y[wave->ny - 1] - wave->y[0]) * (wave->ny - 1.0) +
           1);

  /* Loop over data points... */
  for (ix = 0; ix < wave->nx; ix++)
```

```
    for (iy = 0; iy < wave->ny; iy++) {

      /* Init... */
      mu = help = 0;
      n = 0;

      /* Get data... */
      for (ix2 = GSL_MAX(ix - dx, 0); ix2 <= GSL_MIN(ix + dx, wave->nx - 1);
           ix2++)
        for (iy2 = GSL_MAX(iy - dy, 0); iy2 <= GSL_MIN(iy + dy, wave->ny - 1);
             iy2++)
          if ((gsl_pow_2(wave->x[ix] - wave->x[ix2])
               + gsl_pow_2(wave->y[iy] - wave->y[iy2])) <= dh2)
            if (gsl_finite(wave->pt[ix2][iy2])) {
              mu += wave->pt[ix2][iy2];
              help += gsl_pow_2(wave->pt[ix2][iy2]);
              n++;
            }

      /* Compute local variance... */
      if (n > 1)
        wave->var[ix][iy] = help / n - gsl_pow_2(mu / n);
      else
        wave->var[ix][iy] = GSL_NAN;
    }
}
```

5.37.1.30 void write_l1 (char * *filename*, airs_l1_t * *l1*)

Write AIRS Level-1 data.

Definition at line 1651 of file libairs.c.

```
                  {

  int dimid[10], ncid, time_id, lon_id, lat_id,
    sat_z_id, sat_lon_id, sat_lat_id, nu_id, rad_id;

  /* Open or create netCDF file... */
  printf("Write AIRS Level-1 file: %s\n", filename);
  if (nc_open(filename, NC_WRITE, &ncid) != NC_NOERR) {
    NC(nc_create(filename, NC_CLOBBER, &ncid));
  } else {
    NC(nc_redef(ncid));
  }

  /* Set dimensions... */
  if (nc_inq_dimid(ncid, "L1_NTRACK", &dimid[0]) != NC_NOERR)
    NC(nc_def_dim(ncid, "L1_NTRACK", L1_NTRACK, &dimid[0]));
  if (nc_inq_dimid(ncid, "L1_NXTRACK", &dimid[1]) != NC_NOERR)
    NC(nc_def_dim(ncid, "L1_NXTRACK", L1_NXTRACK, &dimid[1]));
  if (nc_inq_dimid(ncid, "L1_NCHAN", &dimid[2]) != NC_NOERR)
    NC(nc_def_dim(ncid, "L1_NCHAN", L1_NCHAN, &dimid[2]));

  /* Add variables... */
  add_var(ncid, "l1_time", "s", "time (seconds since 2000-01-01T00:00Z)",
          NC_DOUBLE, dimid, &time_id, 2);
  add_var(ncid, "l1_lon", "deg", "longitude", NC_DOUBLE, dimid, &lon_id, 2);
  add_var(ncid, "l1_lat", "deg", "latitude", NC_DOUBLE, dimid, &lat_id, 2);
  add_var(ncid, "l1_sat_z", "km", "satellite altitude",
          NC_DOUBLE, dimid, &sat_z_id, 1);
  add_var(ncid, "l1_sat_lon", "deg", "satellite longitude",
          NC_DOUBLE, dimid, &sat_lon_id, 1);
  add_var(ncid, "l1_sat_lat", "deg", "satellite latitude",
          NC_DOUBLE, dimid, &sat_lat_id, 1);
  add_var(ncid, "l1_nu", "cm^-1", "channel wavenumber",
          NC_DOUBLE, &dimid[2], &nu_id, 1);
  add_var(ncid, "l1_rad", "W/(m^2 sr cm^-1)", "channel radiance",
          NC_FLOAT, dimid, &rad_id, 3);

  /* Leave define mode... */
  NC(nc_enddef(ncid));

  /* Write data... */
  NC(nc_put_var_double(ncid, time_id, l1->time[0]));
  NC(nc_put_var_double(ncid, lon_id, l1->lon[0]));
  NC(nc_put_var_double(ncid, lat_id, l1->lat[0]));
  NC(nc_put_var_double(ncid, sat_z_id, l1->sat_z));
  NC(nc_put_var_double(ncid, sat_lon_id, l1->sat_lon));
  NC(nc_put_var_double(ncid, sat_lat_id, l1->sat_lat));
```

```
  NC(nc_put_var_double(ncid, nu_id, l1->nu));
  NC(nc_put_var_float(ncid, rad_id, l1->rad[0][0]));

  /* Close file... */
  NC(nc_close(ncid));
}
```

Here is the call graph for this function:

5.37.1.31 void write_l2 (char ∗ *filename*, airs_l2_t ∗ *l2*)

Write AIRS Level-2 data.

Definition at line 1709 of file libairs.c.

```
                  {

  int dimid[10], ncid, time_id, z_id, lon_id, lat_id, p_id, t_id;

  /* Create netCDF file... */
  printf("Write AIRS Level-2 file: %s\n", filename);
  if (nc_open(filename, NC_WRITE, &ncid) != NC_NOERR) {
    NC(nc_create(filename, NC_CLOBBER, &ncid));
  } else {
    NC(nc_redef(ncid));
  }

  /* Set dimensions... */
  if (nc_inq_dimid(ncid, "L2_NTRACK", &dimid[0]) != NC_NOERR)
    NC(nc_def_dim(ncid, "L2_NTRACK", L2_NTRACK, &dimid[0]));
  if (nc_inq_dimid(ncid, "L2_NXTRACK", &dimid[1]) != NC_NOERR)
    NC(nc_def_dim(ncid, "L2_NXTRACK", L2_NXTRACK, &dimid[1]));
  if (nc_inq_dimid(ncid, "L2_NLAY", &dimid[2]) != NC_NOERR)
    NC(nc_def_dim(ncid, "L2_NLAY", L2_NLAY, &dimid[2]));

  /* Add variables... */
  add_var(ncid, "l2_time", "s", "time (seconds since 2000-01-01T00:00Z)",
          NC_DOUBLE, dimid, &time_id, 2);
  add_var(ncid, "l2_z", "km", "altitude", NC_DOUBLE, dimid, &z_id, 3);
  add_var(ncid, "l2_lon", "deg", "longitude", NC_DOUBLE, dimid, &lon_id, 2);
  add_var(ncid, "l2_lat", "deg", "latitude", NC_DOUBLE, dimid, &lat_id, 2);
  add_var(ncid, "l2_press", "hPa", "pressure",
          NC_DOUBLE, &dimid[2], &p_id, 1);
  add_var(ncid, "l2_temp", "K", "temperature", NC_DOUBLE, dimid, &t_id, 3);

  /* Leave define mode... */
  NC(nc_enddef(ncid));

  /* Write data... */
  NC(nc_put_var_double(ncid, time_id, l2->time[0]));
  NC(nc_put_var_double(ncid, z_id, l2->z[0][0]));
  NC(nc_put_var_double(ncid, lon_id, l2->lon[0]));
  NC(nc_put_var_double(ncid, lat_id, l2->lat[0]));
  NC(nc_put_var_double(ncid, p_id, l2->p));
  NC(nc_put_var_double(ncid, t_id, l2->t[0][0]));

  /* Close file... */
  NC(nc_close(ncid));
}
```

Here is the call graph for this function:

5.37.1.32 void write_wave (char ∗ *filename*, wave_t ∗ *wave*)

Write wave analysis data.

Definition at line 1758 of file libairs.c.

```
                    {

  FILE *out;

  int i, j;

  /* Write info... */
  printf("Write wave data: %s\n", filename);

  /* Create file... */
  if (!(out = fopen(filename, "w")))
    ERRMSG("Cannot create file!");

  /* Write header... */
  fprintf(out,
          "# $1  = time (seconds since 2000-01-01T00:00Z)\n"
          "# $2  = altitude [km]\n"
          "# $3  = longitude [deg]\n"
          "# $4  = latitude [deg]\n"
          "# $5  = across-track distance [km]\n"
          "# $6  = along-track distance [km]\n"
          "# $7  = temperature [K]\n"
          "# $8  = background [K]\n"
          "# $9  = perturbation [K]\n" "# $10 = variance [K^2]\n");

  /* Write data... */
  for (j = 0; j < wave->ny; j++) {
    fprintf(out, "\n");
    for (i = 0; i < wave->nx; i++)
      fprintf(out, "%.2f %g %g %g %g %g %g %g %g %g\n",
              wave->time, wave->z, wave->lon[i][j], wave->lat[i][j],
              wave->x[i], wave->y[j], wave->temp[i][j], wave->bg[i][j],
              wave->pt[i][j], wave->var[i][j]);
  }

  /* Close file... */
  fclose(out);
}
```

5.38 libairs.h

```
#include <netcdf.h>
#include <gsl/gsl_randist.h>
#include <gsl/gsl_fft_complex.h>
#include <gsl/gsl_multifit.h>
#include <gsl/gsl_poly.h>
#include <gsl/gsl_sort.h>
#include <gsl/gsl_spline.h>
#include <airs_rad_typ.h>
#include <airs_rad_struct.h>
#include <airs_ret_typ.h>
#include <airs_ret_struct.h>
#include "jurassic.h"

/* ------------------------------------------------------------
   Dimensions...
   ------------------------------------------------------------ */

#define NDS 13000

#define NPG 30

#define L1_NCHAN 34

#define L1_NTRACK 135

#define L1_NXTRACK 90

#define L2_NLAY 27

#define L2_NTRACK 45

#define L2_NXTRACK 30

#define PERT_NTRACK 132000

#define PERT_NXTRACK 360

#define WX 300

#define WY 33000

#define PMAX 512
```

```
/* ------------------------------------------------------------
   Macros...
   ------------------------------------------------------------ */
#define NC(cmd) {                                     \
    if((cmd)!=NC_NOERR)                               \
      ERRMSG(nc_strerror(cmd));                       \
  }
/* ------------------------------------------------------------
   Structs...
   ------------------------------------------------------------ */
typedef struct {
  double time[L1_NTRACK][L1_NXTRACK];
  double lon[L1_NTRACK][L1_NXTRACK];
  double lat[L1_NTRACK][L1_NXTRACK];
  double sat_z[L1_NTRACK];
  double sat_lon[L1_NTRACK];
  double sat_lat[L1_NTRACK];
  double nu[L1_NCHAN];
  float rad[L1_NTRACK][L1_NXTRACK][L1_NCHAN];
} airs_l1_t;
typedef struct {
  double time[L2_NTRACK][L2_NXTRACK];
  double z[L2_NTRACK][L2_NXTRACK][L2_NLAY];
  double lon[L2_NTRACK][L2_NXTRACK];
  double lat[L2_NTRACK][L2_NXTRACK];
  double p[L2_NLAY];
  double t[L2_NTRACK][L2_NXTRACK][L2_NLAY];
} airs_l2_t;
typedef struct {
  int ntrack;
  int nxtrack;
  double time[PERT_NTRACK][PERT_NXTRACK];
  double lon[PERT_NTRACK][PERT_NXTRACK];
  double lat[PERT_NTRACK][PERT_NXTRACK];
  double dc[PERT_NTRACK][PERT_NXTRACK];
  double bt[PERT_NTRACK][PERT_NXTRACK];
  double pt[PERT_NTRACK][PERT_NXTRACK];
  double var[PERT_NTRACK][PERT_NXTRACK];
} pert_t;
typedef struct {
  int nds;
  int np;
  double time[NDS][NPG];
  double z[NDS][NPG];
  double lon[NDS][NPG];
  double lat[NDS][NPG];
  double p[NDS][NPG];
```

```
  double t[NDS][NPG];
  double t_apr[NDS][NPG];
  double t_tot[NDS][NPG];
  double t_noise[NDS][NPG];
  double t_fm[NDS][NPG];
  double t_cont[NDS][NPG];
  double t_res[NDS][NPG];
  double chisq[NDS];
} ret_t;
typedef struct {
  int nx;
  int ny;
  double time;
  double z;
  double lon[WX][WY];
  double lat[WX][WY];
  double x[WX];
  double y[WY];
  double temp[WX][WY];
  double bg[WX][WY];
  double pt[WX][WY];
  double var[WX][WY];
} wave_t;
/* ------------------------------------------------------------
   Functions...
   ------------------------------------------------------------ */
void add_var(
  int ncid,
  const char *varname,
  const char *unit,
  const char *longname,
  int type,
  int dimid[],
  int *varid,
  int ndims);
void background_poly(
  wave_t * wave,
  int dim_x,
  int dim_y);
void background_poly_help(
  double *xx,
  double *yy,
  int n,
  int dim);
void background_smooth(
  wave_t * wave,
  int npts_x,
  int npts_y);
void create_background(
  wave_t * wave);
void create_noise(
  wave_t * wave,
  double nedt);
void create_wave(
  wave_t * wave,
  double amp,
```

```
  double lx,
  double ly,
  double phi,
  double fwhm);

void day2doy(
  int year,
  int mon,
  int day,
  int *doy);

void doy2day(
  int year,
  int doy,
  int *mon,
  int *day);

void fft_help(
  double *fcReal,
  double *fcImag,
  int n);

void fft(
  wave_t * wave,
  double *Amax,
  double *phimax,
  double *lhmax,
  double *alphamax,
  double *betamax,
  char *filename);

void gauss(
  wave_t * wave,
  double fwhm);

void hamming(
  wave_t * wave,
  int nit);

void intpol_x(
  wave_t * wave,
  int n);

void median(
  wave_t * wave,
  int dx);

void merge_y(
  wave_t * wave1,
  wave_t * wave2);

void noise(
  wave_t * wave,
  double *mu,
  double *sig);

void period(
  wave_t * wave,
  double *Amax,
  double *phimax,
  double *lhmax,
  double *alphamax,
  double *betamax,
  char *filename);

void pert2wave(
  pert_t * pert,
  wave_t * wave,
  int track0,
  int track1,
  int xtrack0,
  int xtrack1);

void read_l1(
  char *filename,
  airs_l1_t * l1);

void read_l2(
  char *filename,
  airs_l2_t * l2);

void read_pert(
  char *filename,
  char *pertname,
  pert_t * pert);

void read_retr(
```

```
  char *filename,
  ret_t * ret);

void read_retr_help(
  double *help,
  int nds,
  int np,
  double mat[NDS][NPG]);

void read_wave(
  char *filename,
  wave_t * wave);

void rad2wave(
  airs_rad_gran_t * airs_rad_gran,
  double *nu,
  int nd,
  wave_t * wave);

void ret2wave(
  ret_t * ret,
  wave_t * wave,
  int dataset,
  int ip);

double sza(
  double sec,
  double lon,
  double lat);

void variance(
  wave_t * wave,
  double dh);

void write_l1(
  char *filename,
  airs_l1_t * l1);

void write_l2(
  char *filename,
  airs_l2_t * l2);

void write_wave(
  char *filename,
  wave_t * wave);
```

5.39 map_pert.c File Reference

Functions

- double fill_array (double var[PERT_NTRACK][PERT_NXTRACK], int ntrack, int itrack, int ixtrack)
- int main (int argc, char ∗argv[ ])

5.39.1 Function Documentation

5.39.1.1 double fill_array (double *var[PERT_NTRACK][PERT_NXTRACK],* int *ntrack,* int *itrack,* int *ixtrack*)

Definition at line 196 of file map_pert.c.

```
                 {

  double d1 = 0, d2 = 0, v1 = 0, v2 = 0;

  int i;

  /* Find nearest neighbours... */
  for (i = itrack + 1; i < ntrack; i++)
    if (gsl_finite(var[i][ixtrack])) {
      d1 = fabs(i - itrack);
      v1 = var[i][ixtrack];
      break;
    }
  for (i = itrack - 1; i >= 0; i--)
```

```
00214     if (gsl_finite(var[i][ixtrack])) {
00215       d2 = fabs(i - itrack);
00216       v2 = var[i][ixtrack];
00217       break;
00218     }
00219
00220   /* Interpolate... */
00221   if (d1 + d2 > 0)
00222     return (d2 * v1 + d1 * v2) / (d1 + d2);
00223   else
00224     return GSL_NAN;
00225 }
```

5.39.1.2 int main (int *argc*, char ∗ *argv[ ]*)

Definition at line 18 of file map_pert.c.

```
00020                   {
00021
00022   static pert_t *pert, *pert2;
00023   static wave_t wave;
00024
00025   char set[LEN], pertname[LEN];
00026
00027   double orblat, nu, t230 = 230.0, dt230, tbg, nesr, nedt = 0,
00028     var_dh, gauss_fwhm, t0, t1;
00029
00030   int asc, bg_poly_x, bg_poly_y, bg_smooth_x, bg_smooth_y, ham_iter,
00031     itrack, ixtrack, ix, iy, med_dx, orb = 0, orbit, fill;
00032
00033   FILE *out;
00034
00035   /* Check arguments... */
00036   if (argc < 4)
00037     ERRMSG("Give parameters: <ctl> <pert.nc> <map.tab>");
00038
00039   /* Get control parameters... */
00040   scan_ctl(argc, argv, "PERTNAME", -1, "4mu", pertname);
00041   bg_poly_x = (int) scan_ctl(argc, argv, "BG_POLY_X", -1, "0", NULL);
00042   bg_poly_y = (int) scan_ctl(argc, argv, "BG_POLY_Y", -1, "0", NULL);
00043   bg_smooth_x = (int) scan_ctl(argc, argv, "BG_SMOOTH_X", -1, "0", NULL);
00044   bg_smooth_y = (int) scan_ctl(argc, argv, "BG_SMOOTH_Y", -1, "0", NULL);
00045   gauss_fwhm = scan_ctl(argc, argv, "GAUSS_FWHM", -1, "0", NULL);
00046   ham_iter = (int) scan_ctl(argc, argv, "HAM_ITER", -1, "0", NULL);
00047   med_dx = (int) scan_ctl(argc, argv, "MED_DX", -1, "0", NULL);
00048   var_dh = scan_ctl(argc, argv, "VAR_DH", -1, "0", NULL);
00049   scan_ctl(argc, argv, "SET", -1, "full", set);
00050   orbit = (int) scan_ctl(argc, argv, "ORBIT", -1, "-999", NULL);
00051   orblat = scan_ctl(argc, argv, "ORBLAT", -1, "0", NULL);
00052   t0 = scan_ctl(argc, argv, "T0", -1, "-1e100", NULL);
00053   t1 = scan_ctl(argc, argv, "T1", -1, "1e100", NULL);
00054   dt230 = scan_ctl(argc, argv, "DT230", -1, "0.16", NULL);
00055   nu = scan_ctl(argc, argv, "NU", -1, "2345.0", NULL);
00056   fill = (int) scan_ctl(argc, argv, "FILL", -1, "0", NULL);
00057
00058   /* Allocate... */
00059   ALLOC(pert, pert_t, 1);
00060   ALLOC(pert2, pert_t, 1);
00061
00062   /* Read perturbation data... */
00063   read_pert(argv[2], pertname, pert);
00064
00065   /* Recalculate background and perturbations... */
00066   if (bg_poly_x > 0 || bg_poly_y > 0 ||
00067       bg_smooth_x > 0 || bg_smooth_y > 0 ||
00068       gauss_fwhm > 0 || ham_iter > 0 || med_dx > 0 || var_dh > 0) {
00069
00070     /* Convert to wave analysis struct... */
00071     pert2wave(pert, &wave, 0, pert->ntrack - 1, 0, pert->nxtrack - 1);
00072
00073     /* Estimate background... */
00074     background_poly(&wave, bg_poly_x, bg_poly_y);
00075     background_smooth(&wave, bg_smooth_x, bg_smooth_y);
00076
00077     /* Gaussian filter... */
00078     gauss(&wave, gauss_fwhm);
00079
00080     /* Hamming filter... */
00081     hamming(&wave, ham_iter);
00082
00083     /* Median filter... */
```

```
    median(&wave, med_dx);

    /* Compute variance... */
    variance(&wave, var_dh);

    /* Copy data... */
    for (ix = 0; ix < wave.nx; ix++)
      for (iy = 0; iy < wave.ny; iy++) {
        pert->pt[iy][ix] = wave.pt[ix][iy];
        pert->var[iy][ix] = wave.var[ix][iy];
      }
  }

  /* Fill data gaps... */
  if (fill)
    for (itrack = 0; itrack < pert->ntrack; itrack++)
      for (ixtrack = 0; ixtrack < pert->nxtrack; ixtrack++) {
        if (!gsl_finite(pert->dc[itrack][ixtrack]))
          pert->dc[itrack][ixtrack]
            = fill_array(pert->dc, pert->ntrack, itrack, ixtrack);
        if (!gsl_finite(pert->bt[itrack][ixtrack]))
          pert->bt[itrack][ixtrack]
            = fill_array(pert->bt, pert->ntrack, itrack, ixtrack);
        if (!gsl_finite(pert->pt[itrack][ixtrack]))
          pert->pt[itrack][ixtrack]
            = fill_array(pert->pt, pert->ntrack, itrack, ixtrack);
        if (!gsl_finite(pert->var[itrack][ixtrack]))
          pert->var[itrack][ixtrack]
            = fill_array(pert->var, pert->ntrack, itrack, ixtrack);
      }

  /* Interpolate to fine grid... */
  memcpy(pert2, pert, sizeof(pert_t));

  /* Create output file... */
  printf("Write perturbation data: %s\n", argv[3]);
  if (!(out = fopen(argv[3], "w")))
    ERRMSG("Cannot create file!");

  /* Write header... */
  fprintf(out,
          "# $1 = time (seconds since 01-JAN-2000, 00:00 UTC)\n"
          "# $2 = along-track index\n"
          "# $3 = longitude [deg]\n"
          "# $4 = latitude [deg]\n"
          "# $5 = 8mu brightness temperature [K]\n"
          "# $6 = %s brightness temperature [K]\n"
          "# $7 = %s brightness temperature perturbation [K]\n"
          "# $8 = %s brightness temperature variance [K^2]\n",
          pertname, pertname, pertname);

  /* Write data... */
  for (itrack = 0; itrack < pert->ntrack; itrack++) {

    /* Count orbits... */
    if (itrack > 0)
      if (pert->lat[itrack - 1][pert->nxtrack / 2] <= orblat
          && pert->lat[itrack][pert->nxtrack / 2] >= orblat)
        orb++;

    /* Write output... */
    fprintf(out, "\n");

    /* Loop over scan... */
    for (ixtrack = 0; ixtrack < pert->nxtrack; ixtrack++) {

      /* Check data... */
      if (pert->lon[itrack][ixtrack] < -180
          || pert->lon[itrack][ixtrack] > 180
          || pert->lat[itrack][ixtrack] < -90
          || pert->lat[itrack][ixtrack] > 90)
        continue;

      /* Get ascending/descending flag... */
      asc = (pert->lat[itrack > 0 ? itrack : itrack + 1][pert->nxtrack / 2]
             > pert->lat[itrack >
                         0 ? itrack - 1 : itrack][pert->nxtrack / 2]);

      /* Estimate noise... */
      if (dt230 > 0) {
        nesr = planck(t230 + dt230, nu) - planck(t230, nu);
        tbg = pert->bt[itrack][ixtrack] - pert->pt[itrack][ixtrack];
        nedt = brightness(planck(tbg, nu) + nesr, nu) - tbg;
      }

      /* Write data... */
      if (orbit < 0 || orb == orbit)
```

```
        if (set[0] == 'f' || (set[0] == 'a' && asc)
            || (set[0] == 'd' && !asc))
          if (pert->time[itrack][ixtrack] >= t0
              && pert->time[itrack][ixtrack] <= t1)
            fprintf(out, "%.2f %d %g %g %g %g %g %g\n",
                    pert->time[itrack][ixtrack], itrack,
                    pert->lon[itrack][ixtrack], pert->lat[itrack][ixtrack],
                    pert->dc[itrack][ixtrack], pert->bt[itrack][ixtrack],
                    pert->pt[itrack][ixtrack],
                    pert->var[itrack][ixtrack] - gsl_pow_2(nedt));
    }
  }

  /* Close file... */
  fclose(out);

  /* Free... */
  free(pert);
  free(pert2);

  return EXIT_SUCCESS;
}
```

Here is the call graph for this function:

5.40 map_pert.c

```
#include "libairs.h"

/* ------------------------------------------------------------
   Functions...
   ------------------------------------------------------------ */

/* Fill data gaps in perturbation data. */
double fill_array(
  double var[PERT_NTRACK][PERT_NXTRACK],
  int ntrack,
  int itrack,
  int ixtrack);

/* ------------------------------------------------------------
   Main...
   ------------------------------------------------------------ */

int main(
  int argc,
  char *argv[]) {

  static pert_t *pert, *pert2;
  static wave_t wave;

  char set[LEN], pertname[LEN];

  double orblat, nu, t230 = 230.0, dt230, tbg, nesr, nedt = 0,
    var_dh, gauss_fwhm, t0, t1;

  int asc, bg_poly_x, bg_poly_y, bg_smooth_x, bg_smooth_y, ham_iter,
    itrack, ixtrack, ix, iy, med_dx, orb = 0, orbit, fill;

  FILE *out;

  /* Check arguments... */
  if (argc < 4)
    ERRMSG("Give parameters: <ctl> <pert.nc> <map.tab>");

  /* Get control parameters... */
  scan_ctl(argc, argv, "PERTNAME", -1, "4mu", pertname);
  bg_poly_x = (int) scan_ctl(argc, argv, "BG_POLY_X", -1, "0", NULL);
  bg_poly_y = (int) scan_ctl(argc, argv, "BG_POLY_Y", -1, "0", NULL);
  bg_smooth_x = (int) scan_ctl(argc, argv, "BG_SMOOTH_X", -1, "0", NULL);
  bg_smooth_y = (int) scan_ctl(argc, argv, "BG_SMOOTH_Y", -1, "0", NULL);
  gauss_fwhm = scan_ctl(argc, argv, "GAUSS_FWHM", -1, "0", NULL);
  ham_iter = (int) scan_ctl(argc, argv, "HAM_ITER", -1, "0", NULL);
  med_dx = (int) scan_ctl(argc, argv, "MED_DX", -1, "0", NULL);
  var_dh = scan_ctl(argc, argv, "VAR_DH", -1, "0", NULL);
  scan_ctl(argc, argv, "SET", -1, "full", set);
  orbit = (int) scan_ctl(argc, argv, "ORBIT", -1, "-999", NULL);
  orblat = scan_ctl(argc, argv, "ORBLAT", -1, "0", NULL);
  t0 = scan_ctl(argc, argv, "T0", -1, "-1e100", NULL);
  t1 = scan_ctl(argc, argv, "T1", -1, "1e100", NULL);
  dt230 = scan_ctl(argc, argv, "DT230", -1, "0.16", NULL);
```

```
  nu = scan_ctl(argc, argv, "NU", -1, "2345.0", NULL);
  fill = (int) scan_ctl(argc, argv, "FILL", -1, "0", NULL);

  /* Allocate... */
  ALLOC(pert, pert_t, 1);
  ALLOC(pert2, pert_t, 1);

  /* Read perturbation data... */
  read_pert(argv[2], pertname, pert);

  /* Recalculate background and perturbations... */
  if (bg_poly_x > 0 || bg_poly_y > 0 ||
      bg_smooth_x > 0 || bg_smooth_y > 0 ||
      gauss_fwhm > 0 || ham_iter > 0 || med_dx > 0 || var_dh > 0) {

    /* Convert to wave analysis struct... */
    pert2wave(pert, &wave, 0, pert->ntrack - 1, 0, pert->nxtrack - 1);

    /* Estimate background... */
    background_poly(&wave, bg_poly_x, bg_poly_y);
    background_smooth(&wave, bg_smooth_x, bg_smooth_y);

    /* Gaussian filter... */
    gauss(&wave, gauss_fwhm);

    /* Hamming filter... */
    hamming(&wave, ham_iter);

    /* Median filter... */
    median(&wave, med_dx);

    /* Compute variance... */
    variance(&wave, var_dh);

    /* Copy data... */
    for (ix = 0; ix < wave.nx; ix++)
      for (iy = 0; iy < wave.ny; iy++) {
        pert->pt[iy][ix] = wave.pt[ix][iy];
        pert->var[iy][ix] = wave.var[ix][iy];
      }
  }

  /* Fill data gaps... */
  if (fill)
    for (itrack = 0; itrack < pert->ntrack; itrack++)
      for (ixtrack = 0; ixtrack < pert->nxtrack; ixtrack++) {
        if (!gsl_finite(pert->dc[itrack][ixtrack]))
          pert->dc[itrack][ixtrack]
            = fill_array(pert->dc, pert->ntrack, itrack, ixtrack);
        if (!gsl_finite(pert->bt[itrack][ixtrack]))
          pert->bt[itrack][ixtrack]
            = fill_array(pert->bt, pert->ntrack, itrack, ixtrack);
        if (!gsl_finite(pert->pt[itrack][ixtrack]))
          pert->pt[itrack][ixtrack]
            = fill_array(pert->pt, pert->ntrack, itrack, ixtrack);
        if (!gsl_finite(pert->var[itrack][ixtrack]))
          pert->var[itrack][ixtrack]
            = fill_array(pert->var, pert->ntrack, itrack, ixtrack);
      }

  /* Interpolate to fine grid... */
  memcpy(pert2, pert, sizeof(pert_t));

  /* Create output file... */
  printf("Write perturbation data: %s\n", argv[3]);
  if (!(out = fopen(argv[3], "w")))
    ERRMSG("Cannot create file!");

  /* Write header... */
  fprintf(out,
          "# $1 = time (seconds since 01-JAN-2000, 00:00 UTC)\n"
          "# $2 = along-track index\n"
          "# $3 = longitude [deg]\n"
          "# $4 = latitude [deg]\n"
          "# $5 = 8mu brightness temperature [K]\n"
          "# $6 = %s brightness temperature [K]\n"
          "# $7 = %s brightness temperature perturbation [K]\n"
          "# $8 = %s brightness temperature variance [K^2]\n",
          pertname, pertname, pertname);

  /* Write data... */
  for (itrack = 0; itrack < pert->ntrack; itrack++) {

    /* Count orbits... */
    if (itrack > 0)
      if (pert->lat[itrack - 1][pert->nxtrack / 2] <= orblat
          && pert->lat[itrack][pert->nxtrack / 2] >= orblat)
```

```
        orb++;

    /* Write output... */
    fprintf(out, "\n");

    /* Loop over scan... */
    for (ixtrack = 0; ixtrack < pert->nxtrack; ixtrack++) {

      /* Check data... */
      if (pert->lon[itrack][ixtrack] < -180
          || pert->lon[itrack][ixtrack] > 180
          || pert->lat[itrack][ixtrack] < -90
          || pert->lat[itrack][ixtrack] > 90)
        continue;

      /* Get ascending/descending flag... */
      asc = (pert->lat[itrack > 0 ? itrack : itrack + 1][pert->nxtrack / 2]
             > pert->lat[itrack >
                         0 ? itrack - 1 : itrack][pert->nxtrack / 2]);

      /* Estimate noise... */
      if (dt230 > 0) {
        nesr = planck(t230 + dt230, nu) - planck(t230, nu);
        tbg = pert->bt[itrack][ixtrack] - pert->pt[itrack][ixtrack];
        nedt = brightness(planck(tbg, nu) + nesr, nu) - tbg;
      }

      /* Write data... */
      if (orbit < 0 || orb == orbit)
        if (set[0] == 'f' || (set[0] == 'a' && asc)
            || (set[0] == 'd' && !asc))
          if (pert->time[itrack][ixtrack] >= t0
              && pert->time[itrack][ixtrack] <= t1)
            fprintf(out, "%.2f %d %g %g %g %g %g %g\n",
                    pert->time[itrack][ixtrack], itrack,
                    pert->lon[itrack][ixtrack], pert->lat[itrack][ixtrack],
                    pert->dc[itrack][ixtrack], pert->bt[itrack][ixtrack],
                    pert->pt[itrack][ixtrack],
                    pert->var[itrack][ixtrack] - gsl_pow_2(nedt));
    }
  }

  /* Close file... */
  fclose(out);

  /* Free... */
  free(pert);
  free(pert2);

  return EXIT_SUCCESS;
}

/*****************************************************************************/

double fill_array(
  double var[PERT_NTRACK][PERT_NXTRACK],
  int ntrack,
  int itrack,
  int ixtrack) {

  double d1 = 0, d2 = 0, v1 = 0, v2 = 0;

  int i;

  /* Find nearest neighbours... */
  for (i = itrack + 1; i < ntrack; i++)
    if (gsl_finite(var[i][ixtrack])) {
      d1 = fabs(i - itrack);
      v1 = var[i][ixtrack];
      break;
    }
  for (i = itrack - 1; i >= 0; i--)
    if (gsl_finite(var[i][ixtrack])) {
      d2 = fabs(i - itrack);
      v2 = var[i][ixtrack];
      break;
    }

  /* Interpolate... */
  if (d1 + d2 > 0)
    return (d2 * v1 + d1 * v2) / (d1 + d2);
  else
    return GSL_NAN;
}
```

5.41 map_rad.c File Reference

Functions

- int main (int argc, char ∗argv[ ])

5.41.1 Function Documentation

5.41.1.1 int main (int *argc,* char ∗ *argv[ ]*)

Definition at line 3 of file map_rad.c.

```
                  {

  static airs_rad_gran_t airs_rad_gran;
  static wave_t wave, wave2;

  double gauss_fwhm, nu, var_dh;

  int bg_poly_x, bg_poly_y, bg_smooth_x, bg_smooth_y;

  /* Check arguments... */
  if (argc < 6)
    ERRMSG("Give parameters: <ctl> <l1b_file1> <l1b_file2> <nu> <wave.tab>");

  /* Get control parameters... */
  bg_poly_x = (int) scan_ctl(argc, argv, "BG_POLY_X", -1, "5", NULL);
  bg_poly_y = (int) scan_ctl(argc, argv, "BG_POLY_Y", -1, "0", NULL);
  bg_smooth_x = (int) scan_ctl(argc, argv, "BG_SMOOTH_X", -1, "0", NULL);
  bg_smooth_y = (int) scan_ctl(argc, argv, "BG_SMOOTH_Y", -1, "0", NULL);
  gauss_fwhm = scan_ctl(argc, argv, "GAUSS_FWHM", -1, "0", NULL);
  var_dh = scan_ctl(argc, argv, "VAR_DH", -1, "0", NULL);

  /* Get channel.. */
  nu = atof(argv[4]);

  /* Read AIRS data... */
  printf("Read AIRS Level-1B data file: %s\n", argv[2]);
  airs_rad_rdr(argv[2], &airs_rad_gran);

  /* Convert radiance data to wave struct... */
  rad2wave(&airs_rad_gran, &nu, 1, &wave);

  /* Check if second file is available... */
  if (argv[3][0] != '-') {

    /* Read AIRS data... */
    printf("Read AIRS Level-1B data file: %s\n", argv[3]);
    airs_rad_rdr(argv[3], &airs_rad_gran);

    /* Convert radiance data to wave struct... */
    rad2wave(&airs_rad_gran, &nu, 1, &wave2);

    /* Merge with first file... */
    merge_y(&wave, &wave2);
  }

  /* Compute background... */
  background_poly(&wave, bg_poly_x, bg_poly_y);
  background_smooth(&wave, bg_smooth_x, bg_smooth_y);

  /* Gaussian filter... */
  gauss(&wave, gauss_fwhm);

  /* Compute variance... */
  variance(&wave, var_dh);

  /* Write files... */
  write_wave(argv[5], &wave);

  return EXIT_SUCCESS;
}
```

Here is the call graph for this function:

5.42 map_rad.c

```
#include "libairs.h"

int main(
  int argc,
  char *argv[]) {

  static airs_rad_gran_t airs_rad_gran;
  static wave_t wave, wave2;

  double gauss_fwhm, nu, var_dh;

  int bg_poly_x, bg_poly_y, bg_smooth_x, bg_smooth_y;

  /* Check arguments... */
  if (argc < 6)
    ERRMSG("Give parameters: <ctl> <l1b_file1> <l1b_file2> <nu> <wave.tab>");

  /* Get control parameters... */
  bg_poly_x = (int) scan_ctl(argc, argv, "BG_POLY_X", -1, "5", NULL);
  bg_poly_y = (int) scan_ctl(argc, argv, "BG_POLY_Y", -1, "0", NULL);
  bg_smooth_x = (int) scan_ctl(argc, argv, "BG_SMOOTH_X", -1, "0", NULL);
  bg_smooth_y = (int) scan_ctl(argc, argv, "BG_SMOOTH_Y", -1, "0", NULL);
  gauss_fwhm = scan_ctl(argc, argv, "GAUSS_FWHM", -1, "0", NULL);
  var_dh = scan_ctl(argc, argv, "VAR_DH", -1, "0", NULL);

  /* Get channel.. */
  nu = atof(argv[4]);

  /* Read AIRS data... */
  printf("Read AIRS Level-1B data file: %s\n", argv[2]);
  airs_rad_rdr(argv[2], &airs_rad_gran);

  /* Convert radiance data to wave struct... */
  rad2wave(&airs_rad_gran, &nu, 1, &wave);

  /* Check if second file is available... */
  if (argv[3][0] != '-') {

    /* Read AIRS data... */
    printf("Read AIRS Level-1B data file: %s\n", argv[3]);
    airs_rad_rdr(argv[3], &airs_rad_gran);

    /* Convert radiance data to wave struct... */
    rad2wave(&airs_rad_gran, &nu, 1, &wave2);

    /* Merge with first file... */
    merge_y(&wave, &wave2);
  }

  /* Compute background... */
  background_poly(&wave, bg_poly_x, bg_poly_y);
  background_smooth(&wave, bg_smooth_x, bg_smooth_y);

  /* Gaussian filter... */
  gauss(&wave, gauss_fwhm);

  /* Compute variance... */
  variance(&wave, var_dh);

  /* Write files... */
  write_wave(argv[5], &wave);

  return EXIT_SUCCESS;
}
```

5.43 map_ret.c File Reference

Functions

- int main (int argc, char ∗argv[ ])

5.43.1 Function Documentation

5.43.1.1 int main (int *argc,* char ∗ *argv[ ]*)

Definition at line 3 of file map_ret.c.

```
                {

static ret_t ret;
static wave_t wave;

static double tbg[NDS], tabg[NDS], z0;

FILE *out;

char set[LEN];

int asc, bg_poly_x, bg_poly_y, bg_smooth_x, bg_smooth_y, ids, ip, ix, iy;

/* Check arguments... */
if (argc < 4)
  ERRMSG("Give parameters: <ctl> <airs.nc> <map.tab>");

/* Get control parameters... */
scan_ctl(argc, argv, "SET", -1, "full", set);
z0 = scan_ctl(argc, argv, "Z0", -1, "", NULL);
bg_poly_x = (int) scan_ctl(argc, argv, "BG_POLY_X", -1, "5", NULL);
bg_poly_y = (int) scan_ctl(argc, argv, "BG_POLY_Y", -1, "0", NULL);
bg_smooth_x = (int) scan_ctl(argc, argv, "BG_SMOOTH_X", -1, "0", NULL);
bg_smooth_y = (int) scan_ctl(argc, argv, "BG_SMOOTH_Y", -1, "0", NULL);

/* Read AIRS data... */
read_retr(argv[2], &ret);

/* Get altitude index... */
for (ip = 0; ip <= ret.np; ip++) {
  if (ip == ret.np)
    ERRMSG("Altitude level not found!");
  if (fabs(ret.z[0][ip] - z0) < 0.1)
    break;
}

/* Compute background... */
ret2wave(&ret, &wave, 1, ip);
background_poly(&wave, bg_poly_x, bg_poly_y);
background_smooth(&wave, bg_smooth_x, bg_smooth_y);
for (ix = 0; ix < wave.nx; ix++)
  for (iy = 0; iy < wave.ny; iy++)
    tbg[iy * 90 + ix] = wave.bg[ix][iy];
ret2wave(&ret, &wave, 2, ip);
background_poly(&wave, bg_poly_x, bg_poly_y);
background_smooth(&wave, bg_smooth_x, bg_smooth_y);
for (ix = 0; ix < wave.nx; ix++)
  for (iy = 0; iy < wave.ny; iy++)
    tabg[iy * 90 + ix] = wave.bg[ix][iy];

/* Create output file... */
printf("Write AIRS map data: %s\n", argv[3]);
if (!(out = fopen(argv[3], "w")))
  ERRMSG("Cannot create file!");

/* Write header... */
fprintf(out,
        "# $1  = time (seconds since 01-JAN-2000, 00:00 UTC)\n"
        "# $2  = altitude [km]\n"
        "# $3  = longitude [deg]\n"
        "# $4  = latitude [deg]\n"
        "# $5  = pressure [hPa]\n"
        "# $6  = temperature (retrieved) [K]\n"
        "# $7  = temperature (retrieved) perturbation [K]\n"
        "# $8  = temperature (a priori) [K]\n"
        "# $9  = temperature (a priori) perturbation [K]\n");
fprintf(out,
        "# $10 = temperature (total error) [K]\n"
        "# $11 = temperature (noise error) [K]\n"
        "# $12 = temperature (forward model error) [K]\n"
        "# $13 = temperature (measurement content)\n"
        "# $14 = temperature (resolution)\n" "# $15 = normalized chi^2\n");

/* Write data... */
for (ids = 0; ids < ret.nds; ids++) {

  /* Write new line... */
  if (ids % 90 == 0)
    fprintf(out, "\n");

  /* Check data... */
  if (ret.lon[ids][ip] < -180 || ret.lon[ids][ip] > 180
      || ret.lat[ids][ip] < -90 || ret.lat[ids][ip] > 90
      || ret.t[ids][ip] < 100 || ret.t[ids][ip] > 400)
    continue;

  /* Get ascending/descending flag... */
```

```
    asc = (ret.lat[ids > 90 ? ids : ids + 90][0]
           > ret.lat[ids > 90 ? ids - 90 : ids][0]);

    /* Write data... */
    if (set[0] == 'f' || (set[0] == 'a' && asc) || (set[0] == 'd' && !asc))
      fprintf(out, "%.2f %g %g %g %g %g %g %g %g %g %g %g %g %g %g\n",
              ret.time[ids][ip], ret.z[ids][ip],
              ret.lon[ids][ip], ret.lat[ids][ip],
              ret.p[ids][ip], ret.t[ids][ip], ret.t[ids][ip] - tbg[ids],
              ret.t_apr[ids][ip], ret.t_apr[ids][ip] - tabg[ids],
              ret.t_tot[ids][ip], ret.t_noise[ids][ip], ret.t_fm[ids][ip],
              ret.t_cont[ids][ip], ret.t_res[ids][ip], ret.chisq[ids]);
  }

  /* Close file... */
  fclose(out);

  return EXIT_SUCCESS;
}
```

Here is the call graph for this function:

5.44 map_ret.c

```
#include "libairs.h"

int main(
  int argc,
  char *argv[]) {

  static ret_t ret;
  static wave_t wave;

  static double tbg[NDS], tabg[NDS], z0;

  FILE *out;

  char set[LEN];

  int asc, bg_poly_x, bg_poly_y, bg_smooth_x, bg_smooth_y, ids, ip, ix, iy;

  /* Check arguments... */
  if (argc < 4)
    ERRMSG("Give parameters: <ctl> <airs.nc> <map.tab>");

  /* Get control parameters... */
  scan_ctl(argc, argv, "SET", -1, "full", set);
  z0 = scan_ctl(argc, argv, "Z0", -1, "", NULL);
  bg_poly_x = (int) scan_ctl(argc, argv, "BG_POLY_X", -1, "5", NULL);
  bg_poly_y = (int) scan_ctl(argc, argv, "BG_POLY_Y", -1, "0", NULL);
  bg_smooth_x = (int) scan_ctl(argc, argv, "BG_SMOOTH_X", -1, "0", NULL);
  bg_smooth_y = (int) scan_ctl(argc, argv, "BG_SMOOTH_Y", -1, "0", NULL);

  /* Read AIRS data... */
  read_retr(argv[2], &ret);

  /* Get altitude index... */
  for (ip = 0; ip <= ret.np; ip++) {
    if (ip == ret.np)
      ERRMSG("Altitude level not found!");
    if (fabs(ret.z[0][ip] - z0) < 0.1)
      break;
  }

  /* Compute background... */
  ret2wave(&ret, &wave, 1, ip);
  background_poly(&wave, bg_poly_x, bg_poly_y);
  background_smooth(&wave, bg_smooth_x, bg_smooth_y);
  for (ix = 0; ix < wave.nx; ix++)
    for (iy = 0; iy < wave.ny; iy++)
      tbg[iy * 90 + ix] = wave.bg[ix][iy];
  ret2wave(&ret, &wave, 2, ip);
  background_poly(&wave, bg_poly_x, bg_poly_y);
  background_smooth(&wave, bg_smooth_x, bg_smooth_y);
  for (ix = 0; ix < wave.nx; ix++)
    for (iy = 0; iy < wave.ny; iy++)
      tabg[iy * 90 + ix] = wave.bg[ix][iy];

  /* Create output file... */
  printf("Write AIRS map data: %s\n", argv[3]);
  if (!(out = fopen(argv[3], "w")))
```

```
    ERRMSG("Cannot create file!");

  /* Write header... */
  fprintf(out,
          "# $1  = time (seconds since 01-JAN-2000, 00:00 UTC)\n"
          "# $2  = altitude [km]\n"
          "# $3  = longitude [deg]\n"
          "# $4  = latitude [deg]\n"
          "# $5  = pressure [hPa]\n"
          "# $6  = temperature (retrieved) [K]\n"
          "# $7  = temperature (retrieved) perturbation [K]\n"
          "# $8  = temperature (a priori) [K]\n"
          "# $9  = temperature (a priori) perturbation [K]\n");
  fprintf(out,
          "# $10 = temperature (total error) [K]\n"
          "# $11 = temperature (noise error) [K]\n"
          "# $12 = temperature (forward model error) [K]\n"
          "# $13 = temperature (measurement content)\n"
          "# $14 = temperature (resolution)\n" "# $15 = normalized chi^2\n");

  /* Write data... */
  for (ids = 0; ids < ret.nds; ids++) {

    /* Write new line... */
    if (ids % 90 == 0)
      fprintf(out, "\n");

    /* Check data... */
    if (ret.lon[ids][ip] < -180 || ret.lon[ids][ip] > 180
        || ret.lat[ids][ip] < -90 || ret.lat[ids][ip] > 90
        || ret.t[ids][ip] < 100 || ret.t[ids][ip] > 400)
      continue;

    /* Get ascending/descending flag... */
    asc = (ret.lat[ids > 90 ? ids : ids + 90][0]
           > ret.lat[ids > 90 ? ids - 90 : ids][0]);

    /* Write data... */
    if (set[0] == 'f' || (set[0] == 'a' && asc) || (set[0] == 'd' && !asc))
      fprintf(out, "%.2f %g %g %g %g %g %g %g %g %g %g %g %g %g %g\n",
              ret.time[ids][ip], ret.z[ids][ip],
              ret.lon[ids][ip], ret.lat[ids][ip],
              ret.p[ids][ip], ret.t[ids][ip], ret.t[ids][ip] - tbg[ids],
              ret.t_apr[ids][ip], ret.t_apr[ids][ip] - tabg[ids],
              ret.t_tot[ids][ip], ret.t_noise[ids][ip], ret.t_fm[ids][ip],
              ret.t_cont[ids][ip], ret.t_res[ids][ip], ret.chisq[ids]);
  }

  /* Close file... */
  fclose(out);

  return EXIT_SUCCESS;
}
```

5.45 noise_pert.c File Reference

Functions

- int main (int argc, char *argv[ ])

5.45.1 Function Documentation

5.45.1.1 int main (int *argc*, char * *argv[ ]*)

Definition at line 3 of file noise_pert.c.

```
                  {

  static pert_t *pert;
  static wave_t wave;

  FILE *out;

  char pertname[LEN];
```

```
  double maxvar, mu, nedt = -1e99, nedt_old;

  int bsize, itrack;

  /* Check arguments... */
  if (argc < 4)
    ERRMSG("Give parameters: <ctl> <pert.nc> <noise.tab>");

  /* Read control parameters... */
  scan_ctl(argc, argv, "PERTNAME", -1, "4mu", pertname);
  bsize = (int) scan_ctl(argc, argv, "BSIZE", -1, "-999", NULL);
  maxvar = (int) scan_ctl(argc, argv, "MAXVAR", -1, "-999", NULL);

  /* Allocate... */
  ALLOC(pert, pert_t, 1);

  /* Read perturbation data... */
  read_pert(argv[2], pertname, pert);

  /* Set block size... */
  if (bsize < 0)
    bsize = pert->nxtrack;

  /* Create file... */
  printf("Write noise data: %s\n", argv[3]);
  if (!(out = fopen(argv[3], "w")))
    ERRMSG("Cannot create file!");

  /* Write header... */
  fprintf(out,
          "# $1 = longitude [deg]\n"
          "# $2 = latitude [deg]\n"
          "# $3 = mean brightness temperature [K]\n"
          "# $4 = noise estimate [K]\n\n");

  /* Loop over granules... */
  for (itrack = 0; itrack < pert->ntrack; itrack += bsize) {

    /* Convert retrieval data to wave struct... */
    pert2wave(pert, &wave, itrack, itrack + bsize,
              pert->nxtrack / 2 - bsize / 2, pert->nxtrack / 2 + bsize / 2);

    /* Estimate noise... */
    nedt_old = nedt;
    noise(&wave, &mu, &nedt);

    /* Write output... */
    if (maxvar <= 0
        || fabs(200 * (nedt - nedt_old) / (nedt + nedt_old)) < maxvar)
      fprintf(out, "%g %g %g %g\n", wave.lon[wave.nx / 2][wave.ny / 2],
              wave.lat[wave.nx / 2][wave.ny / 2], mu, nedt);
  }

  /* Close file... */
  fclose(out);

  /* Free... */
  free(pert);

  return EXIT_SUCCESS;
}
```

Here is the call graph for this function:

5.46 noise_pert.c

```
#include "libairs.h"

int main(
  int argc,
  char *argv[]) {

  static pert_t *pert;
  static wave_t wave;

  FILE *out;

  char pertname[LEN];

  double maxvar, mu, nedt = -1e99, nedt_old;
```

```
  int bsize, itrack;

  /* Check arguments... */
  if (argc < 4)
    ERRMSG("Give parameters: <ctl> <pert.nc> <noise.tab>");

  /* Read control parameters... */
  scan_ctl(argc, argv, "PERTNAME", -1, "4mu", pertname);
  bsize = (int) scan_ctl(argc, argv, "BSIZE", -1, "-999", NULL);
  maxvar = (int) scan_ctl(argc, argv, "MAXVAR", -1, "-999", NULL);

  /* Allocate... */
  ALLOC(pert, pert_t, 1);

  /* Read perturbation data... */
  read_pert(argv[2], pertname, pert);

  /* Set block size... */
  if (bsize < 0)
    bsize = pert->nxtrack;

  /* Create file... */
  printf("Write noise data: %s\n", argv[3]);
  if (!(out = fopen(argv[3], "w")))
    ERRMSG("Cannot create file!");

  /* Write header... */
  fprintf(out,
          "# $1 = longitude [deg]\n"
          "# $2 = latitude [deg]\n"
          "# $3 = mean brightness temperature [K]\n"
          "# $4 = noise estimate [K]\n\n");

  /* Loop over granules... */
  for (itrack = 0; itrack < pert->ntrack; itrack += bsize) {

    /* Convert retrieval data to wave struct... */
    pert2wave(pert, &wave, itrack, itrack + bsize,
              pert->nxtrack / 2 - bsize / 2, pert->nxtrack / 2 + bsize / 2);

    /* Estimate noise... */
    nedt_old = nedt;
    noise(&wave, &mu, &nedt);

    /* Write output... */
    if (maxvar <= 0
        || fabs(200 * (nedt - nedt_old) / (nedt + nedt_old)) < maxvar)
      fprintf(out, "%g %g %g %g\n", wave.lon[wave.nx / 2][wave.ny / 2],
              wave.lat[wave.nx / 2][wave.ny / 2], mu, nedt);
  }

  /* Close file... */
  fclose(out);

  /* Free... */
  free(pert);

  return EXIT_SUCCESS;
}
```

5.47 noise_ret.c File Reference

Functions

- int main (int argc, char *argv[ ])

5.47.1 Function Documentation

5.47.1.1 int main (int *argc*, char * *argv[ ]*)

Definition at line 3 of file noise_ret.c.

```
                 {

  static ret_t ret;
  static wave_t wave, wave2;

  FILE *out;

  double mu, mu2, nedt, nedt2;

  int ip;

  /* Check arguments... */
  if (argc < 4)
    ERRMSG("Give parameters: <ctl> <airs.nc> <noise.tab>");

  /* Read AIRS data... */
  read_retr(argv[2], &ret);

  /* Create file... */
  printf("Write noise data: %s\n", argv[3]);
  if (!(out = fopen(argv[3], "w")))
    ERRMSG("Cannot create file!");

  /* Write header... */
  fprintf(out,
          "# $1 = altitude [km]\n"
          "# $2 = longitude [deg]\n"
          "# $3 = latitude [deg]\n"
          "# $4 = mean temperature (retrieval) [K]\n"
          "# $5 = noise estimate (retrieval) [K]\n"
          "# $6 = mean temperature (a priori) [K]\n"
          "# $7 = noise estimate (a priori) [K]\n\n");

  /* Loop over altitudes... */
  for (ip = 0; ip < ret.np; ip++) {

    /* Convert retrieval data to wave struct... */
    ret2wave(&ret, &wave, 1, ip);
    ret2wave(&ret, &wave2, 2, ip);

    /* Estimate noise... */
    noise(&wave, &mu, &nedt);
    noise(&wave2, &mu2, &nedt2);

    /* Estimate noise... */
    fprintf(out, "%g %g %g %g %g %g %g\n",
            wave.z,
            wave.lon[wave.nx / 2][wave.ny / 2],
            wave.lat[wave.nx / 2][wave.ny / 2], mu, nedt, mu2, nedt2);
  }

  /* Close file... */
  fclose(out);

  return EXIT_SUCCESS;
}
```

Here is the call graph for this function:

5.48 noise_ret.c

```
#include "libairs.h"

int main(
  int argc,
  char *argv[]) {

  static ret_t ret;
  static wave_t wave, wave2;

  FILE *out;

  double mu, mu2, nedt, nedt2;

  int ip;

  /* Check arguments... */
  if (argc < 4)
    ERRMSG("Give parameters: <ctl> <airs.nc> <noise.tab>");

  /* Read AIRS data... */
```

```
  read_retr(argv[2], &ret);

  /* Create file... */
  printf("Write noise data: %s\n", argv[3]);
  if (!(out = fopen(argv[3], "w")))
    ERRMSG("Cannot create file!");

  /* Write header... */
  fprintf(out,
          "# $1 = altitude [km]\n"
          "# $2 = longitude [deg]\n"
          "# $3 = latitude [deg]\n"
          "# $4 = mean temperature (retrieval) [K]\n"
          "# $5 = noise estimate (retrieval) [K]\n"
          "# $6 = mean temperature (a priori) [K]\n"
          "# $7 = noise estimate (a priori) [K]\n\n");

  /* Loop over altitudes... */
  for (ip = 0; ip < ret.np; ip++) {

    /* Convert retrieval data to wave struct... */
    ret2wave(&ret, &wave, 1, ip);
    ret2wave(&ret, &wave2, 2, ip);

    /* Estimate noise... */
    noise(&wave, &mu, &nedt);
    noise(&wave2, &mu2, &nedt2);

    /* Estimate noise... */
    fprintf(out, "%g %g %g %g %g %g %g\n",
            wave.z,
            wave.lon[wave.nx / 2][wave.ny / 2],
            wave.lat[wave.nx / 2][wave.ny / 2], mu, nedt, mu2, nedt2);
  }

  /* Close file... */
  fclose(out);

  return EXIT_SUCCESS;
}
```

5.49 optimize_si.c File Reference

Functions

- int main (int argc, char ∗argv[ ])

5.49.1 Function Documentation

5.49.1.1 int main (int *argc,* char ∗ *argv[ ]*)

Definition at line 7 of file optimize_si.c.

```
                  {

  static airs_rad_gran_t airs_rad_gran;

  static FILE *out;

  static double bt[AIRS_RAD_CHANNEL], bt2, dbt, lat0, lat1, lon0, lon1,
    mean[AIRS_RAD_CHANNEL][AIRS_RAD_CHANNEL],
    max[AIRS_RAD_CHANNEL][AIRS_RAD_CHANNEL],
    var[AIRS_RAD_CHANNEL][AIRS_RAD_CHANNEL];

  static int chan0, chan1, iarg, iavg, ichan, ichan2,
    n[AIRS_RAD_CHANNEL][AIRS_RAD_CHANNEL], navg, track, xtrack;

  /* Check arguments... */
  if (argc < 10)
    ERRMSG("Give parameters: <opt.tab> <chan0> <chan1>"
           " <lon0> <lon1> <lat0> <lat1> <navg>"
           " <l1b_file1> [<l1b_file2> ...]");

  /* Get parameters... */
```

```
chan0 = GSL_MIN(GSL_MAX(atoi(argv[2]), 0), AIRS_RAD_CHANNEL - 1);
chan1 = GSL_MIN(GSL_MAX(atoi(argv[3]), 0), AIRS_RAD_CHANNEL - 1);
lon0 = atof(argv[4]);
lon1 = atof(argv[5]);
lat0 = atof(argv[6]);
lat1 = atof(argv[7]);
navg = atoi(argv[8]);

/* Loop over HDF files... */
for (iarg = 9; iarg < argc; iarg++) {

  /* Read AIRS data... */
  printf("Read AIRS Level-1B data file: %s\n", argv[iarg]);
  airs_rad_rdr(argv[iarg], &airs_rad_gran);

  /* Loop over footprints... */
  for (track = 0; track < AIRS_RAD_GEOTRACK; track++)
    for (xtrack = 0; xtrack < AIRS_RAD_GEOXTRACK; xtrack++)
      if (airs_rad_gran.Longitude[track][xtrack] >= lon0 &&
          airs_rad_gran.Longitude[track][xtrack] <= lon1 &&
          airs_rad_gran.Latitude[track][xtrack] >= lat0 &&
          airs_rad_gran.Latitude[track][xtrack] <= lat1) {

        /* Get brightness temperature... */
        for (ichan = chan0; ichan <= chan1; ichan++)
          if ((airs_rad_gran.state[track][xtrack] != 0)
              || (airs_rad_gran.ExcludedChans[ichan] > 2)
              || (airs_rad_gran.CalChanSummary[ichan] & 8)
              || (airs_rad_gran.CalChanSummary[ichan] & (32 + 64))
              || (airs_rad_gran.CalFlag[track][ichan] & 16))
            bt[ichan] = GSL_NAN;
          else
            bt[ichan]
              = brightness(airs_rad_gran.radiances[track][xtrack][ichan]
                           * 0.001, airs_rad_gran.nominal_freq[ichan]);

        /* Average channels... */
        for (ichan = chan0; ichan <= chan1; ichan++) {
          bt2 = 0;
          for (iavg = 0; iavg < navg; iavg++)
            bt2 += bt[ichan + iavg];
          bt[ichan] = bt2 / navg;
        }

        /* Get statistics... */
        for (ichan = chan0; ichan <= chan1; ichan++)
          for (ichan2 = chan0; ichan2 <= chan1; ichan2++)
            if (gsl_finite(bt[ichan]) && gsl_finite(bt[ichan2])) {

              /* Get brightness temperature difference... */
              dbt = (bt[ichan2] - bt[ichan]);
              if (fabs(dbt) > 100)
                continue;

              /* Check filter... */
              if (n[ichan][ichan2] <= 0)
                max[ichan][ichan2] = dbt;
              else
                max[ichan][ichan2] = GSL_MAX(max[ichan][ichan2], dbt);
              mean[ichan][ichan2] += dbt;
              var[ichan][ichan2] += gsl_pow_2(dbt);
              n[ichan][ichan2]++;
            }
      }
}

/* Normalize... */
for (ichan = chan0; ichan <= chan1; ichan++)
  for (ichan2 = chan0; ichan2 <= chan1; ichan2++) {
    if (n[ichan][ichan2] > 0) {
      mean[ichan][ichan2] /= n[ichan][ichan2];
      var[ichan][ichan2] = sqrt(var[ichan][ichan2] / n[ichan][ichan2]
                                - gsl_pow_2(mean[ichan][ichan2]));
    } else
      mean[ichan][ichan2] = var[ichan][ichan2] = max[ichan][ichan2] =
        GSL_NAN;
  }

/* Write info... */
printf("Write optimization data: %s\n", argv[1]);

/* Create file... */
if (!(out = fopen(argv[1], "w")))
  ERRMSG("Cannot create file!");

/* Write header... */
fprintf(out,
```

```
          "# $1 = signal channel\n"
          "# $2 = signal wavenumber [cm^-1]\n"
          "# $3 = background channel\n"
          "# $4 = background wavenumber [cm^-1]\n"
          "# $5 = BTD(bg-sig) mean [K]\n"
          "# $6 = BTD(bg-sig) standard deviation [K]\n"
          "# $7 = BTD(bg-sig) maximum [K]\n"
          "# $8 = effective SNR (= max/RMS)\n"
          "# $9 = number of footprints\n");

  /* Write info... */
  for (ichan = chan0; ichan <= chan1; ichan++) {
    fprintf(out, "\n");
    for (ichan2 = chan0; ichan2 <= chan1; ichan2++)
      fprintf(out, "%d %.3f %d %.3f %g %g %g %g %d\n",
              ichan, airs_rad_gran.nominal_freq[ichan],
              ichan2, airs_rad_gran.nominal_freq[ichan2],
              mean[ichan][ichan2], var[ichan][ichan2], max[ichan][ichan2],
              max[ichan][ichan2] / sqrt(gsl_pow_2(var[ichan][ichan2])
                                         + gsl_pow_2(mean[ichan][ichan2])),
              n[ichan][ichan2]);
  }

  /* Close file... */
  fclose(out);

  return EXIT_SUCCESS;
}
```

Here is the call graph for this function:

5.50 optimize_si.c

```
#include "libairs.h"

/* ------------------------------------------------------------
   Main...
   ------------------------------------------------------------ */

int main(
  int argc,
  char *argv[]) {

  static airs_rad_gran_t airs_rad_gran;

  static FILE *out;

  static double bt[AIRS_RAD_CHANNEL], bt2, dbt, lat0, lat1, lon0, lon1,
    mean[AIRS_RAD_CHANNEL][AIRS_RAD_CHANNEL],
    max[AIRS_RAD_CHANNEL][AIRS_RAD_CHANNEL],
    var[AIRS_RAD_CHANNEL][AIRS_RAD_CHANNEL];

  static int chan0, chan1, iarg, iavg, ichan, ichan2,
    n[AIRS_RAD_CHANNEL][AIRS_RAD_CHANNEL], navg, track, xtrack;

  /* Check arguments... */
  if (argc < 10)
    ERRMSG("Give parameters: <opt.tab> <chan0> <chan1>"
           " <lon0> <lon1> <lat0> <lat1> <navg>"
           " <l1b_file1> [<l1b_file2> ...]");

  /* Get parameters... */
  chan0 = GSL_MIN(GSL_MAX(atoi(argv[2]), 0), AIRS_RAD_CHANNEL - 1);
  chan1 = GSL_MIN(GSL_MAX(atoi(argv[3]), 0), AIRS_RAD_CHANNEL - 1);
  lon0 = atof(argv[4]);
  lon1 = atof(argv[5]);
  lat0 = atof(argv[6]);
  lat1 = atof(argv[7]);
  navg = atoi(argv[8]);

  /* Loop over HDF files... */
  for (iarg = 9; iarg < argc; iarg++) {

    /* Read AIRS data... */
    printf("Read AIRS Level-1B data file: %s\n", argv[iarg]);
    airs_rad_rdr(argv[iarg], &airs_rad_gran);

    /* Loop over footprints... */
    for (track = 0; track < AIRS_RAD_GEOTRACK; track++)
      for (xtrack = 0; xtrack < AIRS_RAD_GEOXTRACK; xtrack++)
        if (airs_rad_gran.Longitude[track][xtrack] >= lon0 &&
```

```
          airs_rad_gran.Longitude[track][xtrack] <= lon1 &&
          airs_rad_gran.Latitude[track][xtrack] >= lat0 &&
          airs_rad_gran.Latitude[track][xtrack] <= lat1) {

        /* Get brightness temperature... */
        for (ichan = chan0; ichan <= chan1; ichan++)
          if ((airs_rad_gran.state[track][xtrack] != 0)
              || (airs_rad_gran.ExcludedChans[ichan] > 2)
              || (airs_rad_gran.CalChanSummary[ichan] & 8)
              || (airs_rad_gran.CalChanSummary[ichan] & (32 + 64))
              || (airs_rad_gran.CalFlag[track][ichan] & 16))
            bt[ichan] = GSL_NAN;
          else
            bt[ichan]
              = brightness(airs_rad_gran.radiances[track][xtrack][ichan]
                           * 0.001, airs_rad_gran.nominal_freq[ichan]);

        /* Average channels... */
        for (ichan = chan0; ichan <= chan1; ichan++) {
          bt2 = 0;
          for (iavg = 0; iavg < navg; iavg++)
            bt2 += bt[ichan + iavg];
          bt[ichan] = bt2 / navg;
        }

        /* Get statistics... */
        for (ichan = chan0; ichan <= chan1; ichan++)
          for (ichan2 = chan0; ichan2 <= chan1; ichan2++)
            if (gsl_finite(bt[ichan]) && gsl_finite(bt[ichan2])) {

              /* Get brightness temperature difference... */
              dbt = (bt[ichan2] - bt[ichan]);
              if (fabs(dbt) > 100)
                continue;

              /* Check filter... */
              if (n[ichan][ichan2] <= 0)
                max[ichan][ichan2] = dbt;
              else
                max[ichan][ichan2] = GSL_MAX(max[ichan][ichan2], dbt);
              mean[ichan][ichan2] += dbt;
              var[ichan][ichan2] += gsl_pow_2(dbt);
              n[ichan][ichan2]++;
            }
      }
}

/* Normalize... */
for (ichan = chan0; ichan <= chan1; ichan++)
  for (ichan2 = chan0; ichan2 <= chan1; ichan2++) {
    if (n[ichan][ichan2] > 0) {
      mean[ichan][ichan2] /= n[ichan][ichan2];
      var[ichan][ichan2] = sqrt(var[ichan][ichan2] / n[ichan][ichan2]
                                - gsl_pow_2(mean[ichan][ichan2]));
    } else
      mean[ichan][ichan2] = var[ichan][ichan2] = max[ichan][ichan2] =
        GSL_NAN;
  }

/* Write info... */
printf("Write optimization data: %s\n", argv[1]);

/* Create file... */
if (!(out = fopen(argv[1], "w")))
  ERRMSG("Cannot create file!");

/* Write header... */
fprintf(out,
        "# $1 = signal channel\n"
        "# $2 = signal wavenumber [cm^-1]\n"
        "# $3 = background channel\n"
        "# $4 = background wavenumber [cm^-1]\n"
        "# $5 = BTD(bg-sig) mean [K]\n"
        "# $6 = BTD(bg-sig) standard deviation [K]\n"
        "# $7 = BTD(bg-sig) maximum [K]\n"
        "# $8 = effective SNR (= max/RMS)\n"
        "# $9 = number of footprints\n");

/* Write info... */
for (ichan = chan0; ichan <= chan1; ichan++) {
  fprintf(out, "\n");
  for (ichan2 = chan0; ichan2 <= chan1; ichan2++)
    fprintf(out, "%d %.3f %d %.3f %g %g %g %g %d\n",
            ichan, airs_rad_gran.nominal_freq[ichan],
            ichan2, airs_rad_gran.nominal_freq[ichan2],
            mean[ichan][ichan2], var[ichan][ichan2], max[ichan][ichan2],
            max[ichan][ichan2] / sqrt(gsl_pow_2(var[ichan][ichan2])
```

```
                                          + gsl_pow_2(mean[ichan][ichan2])),
                n[ichan][ichan2]);
  }

  /* Close file... */
  fclose(out);

  return EXIT_SUCCESS;
}
```

5.51 orbit.c File Reference

Functions

- int main (int argc, char ∗argv[ ])

5.51.1 Function Documentation

5.51.1.1 int main (int *argc*, char ∗ *argv[ ]*)

Definition at line 3 of file orbit.c.

```
                  {

  static airs_rad_gran_t airs_rad_gran;

  FILE *out;

  int i, track, xtrack;

  /* Check arguments... */
  if (argc < 3)
    ERRMSG
      ("Give parameters: <orbit.tab> <airs_l1b_file> [ <airs_l1b_file2> ... ]");

  /* Create file... */
  printf("Write orbit data: %s\n", argv[1]);
  if (!(out = fopen(argv[1], "w")))
    ERRMSG("Cannot create file!");

  /* Write header... */
  fprintf(out,
          "# $1 = time (seconds since 01-JAN-2000, 00:00 UTC)\n"
          "# $2 = satellite longitude [deg]\n"
          "# $3 = satellite latitude [deg]\n"
          "# $4 = footprint longitude [deg]\n"
          "# $5 = footprint latitude [deg]\n");

  /* Loop over files... */
  for (i = 2; i < argc; i++) {

    /* Read AIRS data... */
    printf("Read AIRS Level-1B data file: %s\n", argv[i]);
    airs_rad_rdr(argv[i], &airs_rad_gran);

    /* Write data... */
    for (track = 0; track < AIRS_RAD_GEOTRACK; track++) {
      fprintf(out, "\n");
      for (xtrack = 0; xtrack < AIRS_RAD_GEOXTRACK; xtrack++)
        fprintf(out, "%.2f %g %g %g %g\n",
                airs_rad_gran.Time[track][xtrack] - 220838400,
                airs_rad_gran.sat_lon[track],
                airs_rad_gran.sat_lat[track],
                airs_rad_gran.Longitude[track][xtrack],
                airs_rad_gran.Latitude[track][xtrack]);
    }
  }

  /* Close file... */
  fclose(out);

  return EXIT_SUCCESS;
}
```

5.52 orbit.c

```
#include "libairs.h"

int main(
  int argc,
  char *argv[]) {

  static airs_rad_gran_t airs_rad_gran;

  FILE *out;

  int i, track, xtrack;

  /* Check arguments... */
  if (argc < 3)
    ERRMSG
      ("Give parameters: <orbit.tab> <airs_l1b_file> [ <airs_l1b_file2> ... ]");

  /* Create file... */
  printf("Write orbit data: %s\n", argv[1]);
  if (!(out = fopen(argv[1], "w")))
    ERRMSG("Cannot create file!");

  /* Write header... */
  fprintf(out,
          "# $1 = time (seconds since 01-JAN-2000, 00:00 UTC)\n"
          "# $2 = satellite longitude [deg]\n"
          "# $3 = satellite latitude [deg]\n"
          "# $4 = footprint longitude [deg]\n"
          "# $5 = footprint latitude [deg]\n");

  /* Loop over files... */
  for (i = 2; i < argc; i++) {

    /* Read AIRS data... */
    printf("Read AIRS Level-1B data file: %s\n", argv[i]);
    airs_rad_rdr(argv[i], &airs_rad_gran);

    /* Write data... */
    for (track = 0; track < AIRS_RAD_GEOTRACK; track++) {
      fprintf(out, "\n");
      for (xtrack = 0; xtrack < AIRS_RAD_GEOXTRACK; xtrack++)
        fprintf(out, "%.2f %g %g %g %g\n",
                airs_rad_gran.Time[track][xtrack] - 220838400,
                airs_rad_gran.sat_lon[track],
                airs_rad_gran.sat_lat[track],
                airs_rad_gran.Longitude[track][xtrack],
                airs_rad_gran.Latitude[track][xtrack]);
    }
  }

  /* Close file... */
  fclose(out);

  return EXIT_SUCCESS;
}
```

5.53 overpass.c File Reference

Functions

- void write_results (FILE ∗out, pert_t ∗pert, int track0, int xtrack0, int orb, double dmin, double obsz)
- int main (int argc, char ∗argv[ ])

5.53.1 Function Documentation

5.53.1.1 void write_results (FILE ∗ *out*, pert_t ∗ *pert*, int *track0*, int *xtrack0*, int *orb*, double *dmin*, double *obsz*)

Definition at line 118 of file overpass.c.

```
                  {

  double alpha, xf[3], xs[3], xsf[3], remain;

  int asc, i, year, mon, day, hour, min, sec;

  /* Calculate scan angle... */
  geo2cart(0, pert->lon[track0][xtrack0], pert->lat[track0][xtrack0], xf);
  geo2cart(0, pert->lon[track0][pert->nxtrack / 2],
           pert->lat[track0][pert->nxtrack / 2], xsf);
  geo2cart(obsz, pert->lon[track0][pert->nxtrack / 2],
           pert->lat[track0][pert->nxtrack / 2], xs);
  for (i = 0; i < 3; i++) {
    xf[i] -= xs[i];
    xsf[i] -= xs[i];
  }
  alpha = 180. / M_PI * acos(DOTP(xf, xsf) / NORM(xf) / NORM(xsf));
  if (xtrack0 < pert->nxtrack / 2)
    alpha = -alpha;

  /* Get ascending/descending flag... */
  asc = (pert->lat[track0 > 0 ? track0 : track0 + 1][pert->nxtrack / 2]
         > pert->lat[track0 > 0 ? track0 - 1 : track0][pert->nxtrack / 2]);

  /* Write results... */
  jsec2time(pert->time[track0][xtrack0], &year, &mon, &day,
            &hour, &min, &sec, &remain);
  fprintf(out,
          "%.2f %d-%02d-%02dT%02d:%02d:%02dZ %g %g %d %d %d %d %g %g\n",
          pert->time[track0][xtrack0], year, mon, day, hour, min, sec,
          pert->lon[track0][xtrack0], pert->lat[track0][xtrack0],
          track0, xtrack0, orb, asc, alpha, sqrt(dmin));
}
```

Here is the call graph for this function:

5.53.1.2 int main (int *argc,* char ∗ *argv[ ]*)

Definition at line 21 of file overpass.c.

```
                   {

  static pert_t *pert;

  FILE *out;

  char pertname[LEN];

  double dmin = 1e100, lon0, lat0, orblat, rmax, obsz, x0[3], x1[3];

  int orb = 0, track, track0 = 0, xtrack, xtrack0 = 0;

  /* Check arguments... */
  if (argc < 6)
    ERRMSG("Give parameters: <ctl> <pert.nc> <lon0> <lat0> <overpass.tab>");

  /* Get arguments... */
  lon0 = atof(argv[3]);
  lat0 = atof(argv[4]);

  /* Get control parameters... */
  scan_ctl(argc, argv, "PERTNAME", -1, "4mu", pertname);
  orblat = scan_ctl(argc, argv, "ORBLAT", -1, "0", NULL);
  rmax = scan_ctl(argc, argv, "RMAX", -1, "100", NULL);
  obsz = scan_ctl(argc, argv, "OBSZ", -1, "", NULL);

  /* Allocate... */
  ALLOC(pert, pert_t, 1);

  /* Read perturbation data... */
  read_pert(argv[2], pertname, pert);

  /* Get Cartesian coordinates... */
  geo2cart(0, lon0, lat0, x0);

  /* Create file... */
  printf("Write overpass data file: %s\n", argv[5]);
  if (!(out = fopen(argv[5], "w")))
    ERRMSG("Cannot create file!");

```

```
  /* Write header... */
  fprintf(out,
          "# $1  = time (seconds since 2000-01-01T00:00Z)\n"
          "# $2  = time (string)\n"
          "# $3  = longitude [deg]\n"
          "# $4  = latitude [deg]\n"
          "# $5  = along-track index\n"
          "# $6  = across-track index\n"
          "# $7  = orbit number\n"
          "# $8  = ascending (1=yes, 0=no)\n"
          "# $9  = scan angle [deg]\n" "# $10 = distance [km]\n\n");

  /* Find nearest footprint... */
  for (track = 0; track < pert->ntrack; track++) {

    /* Check for new orbit... */
    if (track > 0)
      if (pert->lat[track - 1][pert->nxtrack / 2] <= orblat
          && pert->lat[track][pert->nxtrack / 2] >= orblat) {

        /* Write results... */
        if (sqrt(dmin) <= rmax)
          write_results(out, pert, track0, xtrack0, orb, dmin, obsz);

        /* Set counters... */
        dmin = 1e100;
        orb++;
      }

    /* Check distance of footprints... */
    for (xtrack = 0; xtrack < pert->nxtrack; xtrack++) {
      geo2cart(0, pert->lon[track][xtrack], pert->lat[track][xtrack], x1);
      if (DIST2(x0, x1) < dmin) {
        dmin = DIST2(x0, x1);
        track0 = track;
        xtrack0 = xtrack;
      }
    }
  }

  /* Write results for last orbit... */
  if (sqrt(dmin) <= rmax)
    write_results(out, pert, track0, xtrack0, orb, dmin, obsz);

  /* Close file... */
  fclose(out);

  /* Free... */
  free(pert);

  return EXIT_SUCCESS;
}
```

Here is the call graph for this function:

5.54 overpass.c

```
#include "libairs.h"

/* ------------------------------------------------------------
   Functions...
   ------------------------------------------------------------ */

/* Write results to file. */
void write_results(
  FILE * out,
  pert_t * pert,
  int track0,
  int xtrack0,
  int orb,
  double dmin,
  double obsz);

/* ------------------------------------------------------------
   Main...
   ------------------------------------------------------------ */

int main(
  int argc,
  char *argv[]) {

```

```
  static pert_t *pert;

  FILE *out;

  char pertname[LEN];

  double dmin = 1e100, lon0, lat0, orblat, rmax, obsz, x0[3], x1[3];

  int orb = 0, track, track0 = 0, xtrack, xtrack0 = 0;

  /* Check arguments... */
  if (argc < 6)
    ERRMSG("Give parameters: <ctl> <pert.nc> <lon0> <lat0> <overpass.tab>");

  /* Get arguments... */
  lon0 = atof(argv[3]);
  lat0 = atof(argv[4]);

  /* Get control parameters... */
  scan_ctl(argc, argv, "PERTNAME", -1, "4mu", pertname);
  orblat = scan_ctl(argc, argv, "ORBLAT", -1, "0", NULL);
  rmax = scan_ctl(argc, argv, "RMAX", -1, "100", NULL);
  obsz = scan_ctl(argc, argv, "OBSZ", -1, "", NULL);

  /* Allocate... */
  ALLOC(pert, pert_t, 1);

  /* Read perturbation data... */
  read_pert(argv[2], pertname, pert);

  /* Get Cartesian coordinates... */
  geo2cart(0, lon0, lat0, x0);

  /* Create file... */
  printf("Write overpass data file: %s\n", argv[5]);
  if (!(out = fopen(argv[5], "w")))
    ERRMSG("Cannot create file!");

  /* Write header... */
  fprintf(out,
          "# $1  = time (seconds since 2000-01-01T00:00Z)\n"
          "# $2  = time (string)\n"
          "# $3  = longitude [deg]\n"
          "# $4  = latitude [deg]\n"
          "# $5  = along-track index\n"
          "# $6  = across-track index\n"
          "# $7  = orbit number\n"
          "# $8  = ascending (1=yes, 0=no)\n"
          "# $9  = scan angle [deg]\n" "# $10 = distance [km]\n\n");

  /* Find nearest footprint... */
  for (track = 0; track < pert->ntrack; track++) {

    /* Check for new orbit... */
    if (track > 0)
      if (pert->lat[track - 1][pert->nxtrack / 2] <= orblat
          && pert->lat[track][pert->nxtrack / 2] >= orblat) {

        /* Write results... */
        if (sqrt(dmin) <= rmax)
          write_results(out, pert, track0, xtrack0, orb, dmin, obsz);

        /* Set counters... */
        dmin = 1e100;
        orb++;
      }

    /* Check distance of footprints... */
    for (xtrack = 0; xtrack < pert->nxtrack; xtrack++) {
      geo2cart(0, pert->lon[track][xtrack], pert->lat[track][xtrack], x1);
      if (DIST2(x0, x1) < dmin) {
        dmin = DIST2(x0, x1);
        track0 = track;
        xtrack0 = xtrack;
      }
    }
  }

  /* Write results for last orbit... */
  if (sqrt(dmin) <= rmax)
    write_results(out, pert, track0, xtrack0, orb, dmin, obsz);

  /* Close file... */
  fclose(out);

  /* Free... */
  free(pert);
```

```
  return EXIT_SUCCESS;
}

/*****************************************************************************/

void write_results(
  FILE * out,
  pert_t * pert,
  int track0,
  int xtrack0,
  int orb,
  double dmin,
  double obsz) {

  double alpha, xf[3], xs[3], xsf[3], remain;

  int asc, i, year, mon, day, hour, min, sec;

  /* Calculate scan angle... */
  geo2cart(0, pert->lon[track0][xtrack0], pert->lat[track0][xtrack0], xf);
  geo2cart(0, pert->lon[track0][pert->nxtrack / 2],
           pert->lat[track0][pert->nxtrack / 2], xsf);
  geo2cart(obsz, pert->lon[track0][pert->nxtrack / 2],
           pert->lat[track0][pert->nxtrack / 2], xs);
  for (i = 0; i < 3; i++) {
    xf[i] -= xs[i];
    xsf[i] -= xs[i];
  }
  alpha = 180. / M_PI * acos(DOTP(xf, xsf) / NORM(xf) / NORM(xsf));
  if (xtrack0 < pert->nxtrack / 2)
    alpha = -alpha;

  /* Get ascending/descending flag... */
  asc = (pert->lat[track0 > 0 ? track0 : track0 + 1][pert->nxtrack / 2]
         > pert->lat[track0 > 0 ? track0 - 1 : track0][pert->nxtrack / 2]);

  /* Write results... */
  jsec2time(pert->time[track0][xtrack0], &year, &mon, &day,
            &hour, &min, &sec, &remain);
  fprintf(out,
          "%.2f %d-%02d-%02dT%02d:%02d:%02dZ %g %g %d %d %d %d %g %g\n",
          pert->time[track0][xtrack0], year, mon, day, hour, min, sec,
          pert->lon[track0][xtrack0], pert->lat[track0][xtrack0],
          track0, xtrack0, orb, asc, alpha, sqrt(dmin));
}
```

5.55 pca.c File Reference

Functions

- int main (int argc, char *argv[ ])

5.55.1 Function Documentation

5.55.1.1 int main (int *argc,* char * *argv[ ]*)

Definition at line 3 of file pca.c.

```
                  {

  static airs_rad_gran_t airs_rad_gran;

  static gsl_matrix *a, *v;

  static gsl_vector *s, *w;

  static double lat[AIRS_RAD_GEOTRACK * AIRS_RAD_GEOXTRACK],
    lon[AIRS_RAD_GEOTRACK * AIRS_RAD_GEOXTRACK], mean;

  static size_t channel0, channel1, ichan, itrack, ixtrack, i, j, m, n;

  /* Check arguments... */
  if (argc < 3)
```

```
    ERRMSG("Give parameters: <ctl> <l1b_file1>");

  /* Get arguments... */
  channel0 = (size_t) scan_ctl(argc, argv, "CHANNEL0", -1, "", NULL);
  channel1 = (size_t) scan_ctl(argc, argv, "CHANNEL1", -1, "", NULL);

  /* Read AIRS data... */
  printf("Read AIRS Level-1B data file: %s\n", argv[2]);
  airs_rad_rdr(argv[2], &airs_rad_gran);

  /* Allocate... */
  m = AIRS_RAD_GEOTRACK * AIRS_RAD_GEOXTRACK;
  n = channel1 - channel0 + 1;
  a = gsl_matrix_calloc(m, n);
  v = gsl_matrix_calloc(n, n);
  s = gsl_vector_calloc(n);
  w = gsl_vector_calloc(n);

  /* Build data matrix... */
  for (itrack = 0; itrack < AIRS_RAD_GEOTRACK; itrack++)
    for (ixtrack = 0; ixtrack < AIRS_RAD_GEOXTRACK; ixtrack++) {
      i = itrack * AIRS_RAD_GEOXTRACK + ixtrack;
      lon[i] = airs_rad_gran.Longitude[itrack][ixtrack];
      lat[i] = airs_rad_gran.Latitude[itrack][ixtrack];
      for (ichan = channel0; ichan <= channel1; ichan++)
        if (airs_rad_gran.radiances[itrack][ixtrack][ichan] > 0)
          gsl_matrix_set(a, i, (ichan - channel0),
                         brightness(airs_rad_gran.radiances[itrack][ixtrack]
                                    [ichan] * 0.001,
                                    airs_rad_gran.nominal_freq[ichan]));
    }

  /* Remove column mean... */
  for (j = 0; j < n; j++) {
    mean = 0;
    for (i = 0; i < m; i++)
      mean += gsl_matrix_get(a, i, j) / (double) m;
    printf("mean[%lu] = %g K\n", j, mean);
    for (i = 0; i < m; i++)
      gsl_matrix_set(a, i, j, gsl_matrix_get(a, i, j) - mean);
  }

  /* Calculate SVD... */
  gsl_linalg_SV_decomp(a, v, s, w);

  /*
     https://stats.stackexchange.com/questions/134282/
relationship-between-svd-and-pca-how-to-use-svd-to-perform-pca
   */

  /* Write eigenvalues (variances of PCs)... */
  for (i = 0; i < n; i++)
    printf("lambda_i[%lu] = %g\n", i,
           gsl_pow_2(gsl_vector_get(s, i)) / ((double) n - 1.0));

  /* Calculate principal components (columns of U x S)... */
  for (j = 0; j < n; j++) {
    printf("\n");
    for (i = 0; i < m; i++)
      printf("%lu %lu %g %g %g %g\n", i, j, lon[i], lat[i],
             airs_rad_gran.nominal_freq[channel0 + j], gsl_matrix_get(a, i,
                                                                      j) *
             gsl_vector_get(s, j));
  }

  /* Free... */
  gsl_matrix_free(a);
  gsl_matrix_free(v);
  gsl_vector_free(s);
  gsl_vector_free(w);

  return EXIT_SUCCESS;
}
```

Here is the call graph for this function:

5.56 pca.c

```
#include "libairs.h"

int main(
```

```
int argc,
char *argv[]) {

static airs_rad_gran_t airs_rad_gran;

static gsl_matrix *a, *v;

static gsl_vector *s, *w;

static double lat[AIRS_RAD_GEOTRACK * AIRS_RAD_GEOXTRACK],
  lon[AIRS_RAD_GEOTRACK * AIRS_RAD_GEOXTRACK], mean;

static size_t channel0, channel1, ichan, itrack, ixtrack, i, j, m, n;

/* Check arguments... */
if (argc < 3)
  ERRMSG("Give parameters: <ctl> <l1b_file1>");

/* Get arguments... */
channel0 = (size_t) scan_ctl(argc, argv, "CHANNEL0", -1, "", NULL);
channel1 = (size_t) scan_ctl(argc, argv, "CHANNEL1", -1, "", NULL);

/* Read AIRS data... */
printf("Read AIRS Level-1B data file: %s\n", argv[2]);
airs_rad_rdr(argv[2], &airs_rad_gran);

/* Allocate... */
m = AIRS_RAD_GEOTRACK * AIRS_RAD_GEOXTRACK;
n = channel1 - channel0 + 1;
a = gsl_matrix_calloc(m, n);
v = gsl_matrix_calloc(n, n);
s = gsl_vector_calloc(n);
w = gsl_vector_calloc(n);

/* Build data matrix... */
for (itrack = 0; itrack < AIRS_RAD_GEOTRACK; itrack++)
  for (ixtrack = 0; ixtrack < AIRS_RAD_GEOXTRACK; ixtrack++) {
    i = itrack * AIRS_RAD_GEOXTRACK + ixtrack;
    lon[i] = airs_rad_gran.Longitude[itrack][ixtrack];
    lat[i] = airs_rad_gran.Latitude[itrack][ixtrack];
    for (ichan = channel0; ichan <= channel1; ichan++)
      if (airs_rad_gran.radiances[itrack][ixtrack][ichan] > 0)
        gsl_matrix_set(a, i, (ichan - channel0),
                       brightness(airs_rad_gran.radiances[itrack][ixtrack]
                                  [ichan] * 0.001,
                                  airs_rad_gran.nominal_freq[ichan]));
  }

/* Remove column mean... */
for (j = 0; j < n; j++) {
  mean = 0;
  for (i = 0; i < m; i++)
    mean += gsl_matrix_get(a, i, j) / (double) m;
  printf("mean[%lu] = %g K\n", j, mean);
  for (i = 0; i < m; i++)
    gsl_matrix_set(a, i, j, gsl_matrix_get(a, i, j) - mean);
}

/* Calculate SVD... */
gsl_linalg_SV_decomp(a, v, s, w);

/*
   https://stats.stackexchange.com/questions/134282/
relationship-between-svd-and-pca-how-to-use-svd-to-perform-pca
 */

/* Write eigenvalues (variances of PCs)... */
for (i = 0; i < n; i++)
  printf("lambda_i[%lu] = %g\n", i,
         gsl_pow_2(gsl_vector_get(s, i)) / ((double) n - 1.0));

/* Calculate principal components (columns of U x S)... */
for (j = 0; j < n; j++) {
  printf("\n");
  for (i = 0; i < m; i++)
    printf("%lu %lu %g %g %g %g\n", i, j, lon[i], lat[i],
           airs_rad_gran.nominal_freq[channel0 + j], gsl_matrix_get(a, i,
                                                                    j) *
           gsl_vector_get(s, j));
}

/* Free... */
gsl_matrix_free(a);
gsl_matrix_free(v);
gsl_vector_free(s);
gsl_vector_free(w);

```

```
  return EXIT_SUCCESS;
}
```

5.57 perturbation.c File Reference

Functions

- void addatt (int ncid, int varid, const char ∗unit, const char ∗long_name)
- int main (int argc, char ∗argv[ ])

5.57.1 Function Documentation

5.57.1.1 void addatt (int *ncid,* int *varid,* const char ∗ *unit,* const char ∗ *long_name*)

Definition at line 391 of file perturbation.c.

```
                              {

  /* Set long name... */
  NC(nc_put_att_text(ncid, varid, "long_name", strlen(long_name), long_name));

  /* Set units... */
  NC(nc_put_att_text(ncid, varid, "units", strlen(unit), unit));
}
```

5.57.1.2 int main (int *argc,* char ∗ *argv[ ]*)

Definition at line 31 of file perturbation.c.

```
                  {

  static airs_rad_gran_t airs_rad_gran;

  static pert_t *pert_4mu, *pert_15mu_low, *pert_15mu_high;

  static wave_t wave;

  static double var_dh = 100.;

  static int list_4mu[N4]
    = { 2039, 2040, 2041, 2042, 2043, 2044, 2045, 2046, 2047, 2048,
    2049, 2050, 2051, 2052, 2053, 2054, 2055, 2056, 2057, 2058,
    2059, 2060, 2061, 2062, 2063, 2064, 2071, 2072, 2073, 2074,
    2075, 2076, 2077, 2078, 2079, 2080, 2081, 2082, 2083, 2084,
    2085, 2086
  };

  static int list_15mu_low[N15_LOW]
    = { 4, 10, 16, 22, 29, 35, 41, 55, 83, 88, 94,
    100, 101, 106, 107, 112, 113, 118, 119, 124, 125
  };

  static int list_15mu_high[N15_HIGH]
  = { 74, 75 };

  static int ix, iy, dimid[2], i, n, ncid, track, track0, xtrack,
    time_varid, lon_varid, lat_varid, bt_4mu_varid, bt_4mu_pt_varid,
    bt_4mu_var_varid, bt_8mu_varid, bt_15mu_low_varid, bt_15mu_low_pt_varid,
    bt_15mu_low_var_varid, bt_15mu_high_varid, bt_15mu_high_pt_varid,
    bt_15mu_high_var_varid, iarg;

  static size_t start[2], count[2];

  /* Check arguments... */
  if (argc < 3)
    ERRMSG("Give parameters: <out.nc> <l1b_file1> [<l1b_file2> ...]");

```

```
/* Allocate... */
ALLOC(pert_4mu, pert_t, 1);
ALLOC(pert_15mu_low, pert_t, 1);
ALLOC(pert_15mu_high, pert_t, 1);

/* ------------------------------------------------------------
   Read HDF files...
   ------------------------------------------------------------ */

/* Loop over HDF files... */
for (iarg = 2; iarg < argc; iarg++) {

  /* Read AIRS data... */
  printf("Read AIRS Level-1B data file: %s\n", argv[iarg]);
  airs_rad_rdr(argv[iarg], &airs_rad_gran);

  /* Flag bad observations... */
  for (track = 0; track < AIRS_RAD_GEOTRACK; track++)
    for (xtrack = 0; xtrack < AIRS_RAD_GEOXTRACK; xtrack++)
      for (i = 0; i < AIRS_RAD_CHANNEL; i++)
        if ((airs_rad_gran.state[track][xtrack] != 0)
            || (airs_rad_gran.ExcludedChans[i] > 2)
            || (airs_rad_gran.CalChanSummary[i] & 8)
            || (airs_rad_gran.CalChanSummary[i] & (32 + 64))
            || (airs_rad_gran.CalFlag[track][i] & 16)
            || (airs_rad_gran.Longitude[track][xtrack] < -180)
            || (airs_rad_gran.Longitude[track][xtrack] > 180)
            || (airs_rad_gran.Latitude[track][xtrack] < -90)
            || (airs_rad_gran.Latitude[track][xtrack] > 90))
          airs_rad_gran.radiances[track][xtrack][i] = GSL_NAN;
        else
          airs_rad_gran.radiances[track][xtrack][i] *= 0.001f;

  /* Save geolocation... */
  pert_4mu->ntrack += AIRS_RAD_GEOTRACK;
  if (pert_4mu->ntrack > PERT_NTRACK)
    ERRMSG("Too many granules!");
  pert_4mu->nxtrack = AIRS_RAD_GEOXTRACK;
  if (pert_4mu->nxtrack > PERT_NXTRACK)
    ERRMSG("Too many tracks!");
  for (track = 0; track < AIRS_RAD_GEOTRACK; track++)
    for (xtrack = 0; xtrack < AIRS_RAD_GEOXTRACK; xtrack++) {
      pert_4mu->time[track0 + track][xtrack]
        = airs_rad_gran.Time[track][xtrack] - 220838400.;
      pert_4mu->lon[track0 + track][xtrack]
        = airs_rad_gran.Longitude[track][xtrack];
      pert_4mu->lat[track0 + track][xtrack]
        = airs_rad_gran.Latitude[track][xtrack];
    }

  pert_15mu_low->ntrack += AIRS_RAD_GEOTRACK;
  if (pert_15mu_low->ntrack > PERT_NTRACK)
    ERRMSG("Too many granules!");
  pert_15mu_low->nxtrack = AIRS_RAD_GEOXTRACK;
  if (pert_15mu_low->nxtrack > PERT_NXTRACK)
    ERRMSG("Too many tracks!");
  for (track = 0; track < AIRS_RAD_GEOTRACK; track++)
    for (xtrack = 0; xtrack < AIRS_RAD_GEOXTRACK; xtrack++) {
      pert_15mu_low->time[track0 + track][xtrack]
        = airs_rad_gran.Time[track][xtrack] - 220838400.;
      pert_15mu_low->lon[track0 + track][xtrack]
        = airs_rad_gran.Longitude[track][xtrack];
      pert_15mu_low->lat[track0 + track][xtrack]
        = airs_rad_gran.Latitude[track][xtrack];
    }

  pert_15mu_high->ntrack += AIRS_RAD_GEOTRACK;
  if (pert_15mu_high->ntrack > PERT_NTRACK)
    ERRMSG("Too many granules!");
  pert_15mu_high->nxtrack = AIRS_RAD_GEOXTRACK;
  if (pert_15mu_high->nxtrack > PERT_NXTRACK)
    ERRMSG("Too many tracks!");
  for (track = 0; track < AIRS_RAD_GEOTRACK; track++)
    for (xtrack = 0; xtrack < AIRS_RAD_GEOXTRACK; xtrack++) {
      pert_15mu_high->time[track0 + track][xtrack]
        = airs_rad_gran.Time[track][xtrack] - 220838400.;
      pert_15mu_high->lon[track0 + track][xtrack]
        = airs_rad_gran.Longitude[track][xtrack];
      pert_15mu_high->lat[track0 + track][xtrack]
        = airs_rad_gran.Latitude[track][xtrack];
    }

  /* Get 8.1 micron brightness temperature... */
  for (track = 0; track < AIRS_RAD_GEOTRACK; track++)
    for (xtrack = 0; xtrack < AIRS_RAD_GEOXTRACK; xtrack++)
      pert_4mu->dc[track0 + track][xtrack]
        = brightness(airs_rad_gran.radiances[track][xtrack][1290],
```

```
                        airs_rad_gran.nominal_freq[1290]);

    /* Get 4.3 micron brightness temperature... */
    for (track = 0; track < AIRS_RAD_GEOTRACK; track++)
      for (xtrack = 0; xtrack < AIRS_RAD_GEOXTRACK; xtrack++) {
        n = 0;
        for (i = 0; i < N4; i++)
          if (gsl_finite(airs_rad_gran.radiances[track][xtrack][list_4mu[i]])) {
            pert_4mu->bt[track0 + track][xtrack]
              +=
              brightness(airs_rad_gran.radiances[track][xtrack][list_4mu[i]],
                         airs_rad_gran.nominal_freq[list_4mu[i]]);
            n++;
          }
        if (n > 0.9 * N4)
          pert_4mu->bt[track0 + track][xtrack] /= n;
        else
          pert_4mu->bt[track0 + track][xtrack] = GSL_NAN;
      }

    /* Get 15 micron brightness temperature (low altitudes)... */
    for (track = 0; track < AIRS_RAD_GEOTRACK; track++)
      for (xtrack = 0; xtrack < AIRS_RAD_GEOXTRACK; xtrack++) {
        n = 0;
        for (i = 0; i < N15_LOW; i++)
          if (gsl_finite(airs_rad_gran.radiances
                         [track][xtrack][list_15mu_low[i]])) {
            pert_15mu_low->bt[track0 + track][xtrack]
              += brightness(airs_rad_gran.radiances
                            [track][xtrack][list_15mu_low[i]],
                            airs_rad_gran.nominal_freq[list_15mu_low[i]]);
            n++;
          }
        if (n > 0.9 * N15_LOW)
          pert_15mu_low->bt[track0 + track][xtrack] /= n;
        else
          pert_15mu_low->bt[track0 + track][xtrack] = GSL_NAN;
      }

    /* Get 15 micron brightness temperature (high altitudes)... */
    for (track = 0; track < AIRS_RAD_GEOTRACK; track++)
      for (xtrack = 0; xtrack < AIRS_RAD_GEOXTRACK; xtrack++) {
        n = 0;
        for (i = 0; i < N15_HIGH; i++)
          if (gsl_finite(airs_rad_gran.radiances
                         [track][xtrack][list_15mu_high[i]])) {
            pert_15mu_high->bt[track0 + track][xtrack]
              += brightness(airs_rad_gran.radiances
                            [track][xtrack][list_15mu_high[i]],
                            airs_rad_gran.nominal_freq[list_15mu_high[i]]);
            n++;
          }
        if (n > 0.9 * N15_HIGH)
          pert_15mu_high->bt[track0 + track][xtrack] /= n;
        else
          pert_15mu_high->bt[track0 + track][xtrack] = GSL_NAN;
      }

    /* Increment track counter... */
    track0 += AIRS_RAD_GEOTRACK;
  }

  /* ------------------------------------------------------------
     Calculate perturbations and variances...
     ------------------------------------------------------------ */

  /* Convert to wave analysis struct... */
  pert2wave(pert_4mu, &wave,
            0, pert_4mu->ntrack - 1, 0, pert_4mu->nxtrack - 1);

  /* Estimate background... */
  background_poly(&wave, 5, 0);

  /* Compute variance... */
  variance(&wave, var_dh);

  /* Copy data... */
  for (ix = 0; ix < wave.nx; ix++)
    for (iy = 0; iy < wave.ny; iy++) {
      pert_4mu->pt[iy][ix] = wave.pt[ix][iy];
      pert_4mu->var[iy][ix] = wave.var[ix][iy];
    }

  /* Convert to wave analysis struct... */
  pert2wave(pert_15mu_low, &wave,
            0, pert_15mu_low->ntrack - 1, 0, pert_15mu_low->nxtrack - 1);

```

```
/* Estimate background... */
background_poly(&wave, 5, 0);

/* Compute variance... */
variance(&wave, var_dh);

/* Copy data... */
for (ix = 0; ix < wave.nx; ix++)
  for (iy = 0; iy < wave.ny; iy++) {
    pert_15mu_low->pt[iy][ix] = wave.pt[ix][iy];
    pert_15mu_low->var[iy][ix] = wave.var[ix][iy];
  }

/* Convert to wave analysis struct... */
pert2wave(pert_15mu_high, &wave,
          0, pert_15mu_high->ntrack - 1, 0, pert_15mu_high->nxtrack - 1);

/* Estimate background... */
background_poly(&wave, 5, 0);

/* Compute variance... */
variance(&wave, var_dh);

/* Copy data... */
for (ix = 0; ix < wave.nx; ix++)
  for (iy = 0; iy < wave.ny; iy++) {
    pert_15mu_high->pt[iy][ix] = wave.pt[ix][iy];
    pert_15mu_high->var[iy][ix] = wave.var[ix][iy];
  }

/* ------------------------------------------------------------
   Write to netCDF file...
   ------------------------------------------------------------ */

/* Create netCDF file... */
NC(nc_create(argv[1], NC_CLOBBER, &ncid));

/* Set dimensions... */
NC(nc_def_dim(ncid, "NTRACK", NC_UNLIMITED, &dimid[0]));
NC(nc_def_dim(ncid, "NXTRACK", AIRS_RAD_GEOXTRACK, &dimid[1]));

/* Add variables... */
NC(nc_def_var(ncid, "time", NC_DOUBLE, 2, dimid, &time_varid));
addatt(ncid, time_varid, "s", "time (seconds since 2000-01-01T00:00Z)");
NC(nc_def_var(ncid, "lon", NC_DOUBLE, 2, dimid, &lon_varid));
addatt(ncid, lon_varid, "deg", "footprint longitude");
NC(nc_def_var(ncid, "lat", NC_DOUBLE, 2, dimid, &lat_varid));
addatt(ncid, lat_varid, "deg", "footprint latitude");

NC(nc_def_var(ncid, "bt_8mu", NC_FLOAT, 2, dimid, &bt_8mu_varid));
addatt(ncid, bt_8mu_varid, "K", "brightness temperature at 8.1 micron");

NC(nc_def_var(ncid, "bt_4mu", NC_FLOAT, 2, dimid, &bt_4mu_varid));
addatt(ncid, bt_4mu_varid, "K", "brightness temperature" " at 4.3 micron");
NC(nc_def_var(ncid, "bt_4mu_pt", NC_FLOAT, 2, dimid, &bt_4mu_pt_varid));
addatt(ncid, bt_4mu_pt_varid, "K", "brightness temperature perturbation"
       " at 4.3 micron");
NC(nc_def_var(ncid, "bt_4mu_var", NC_FLOAT, 2, dimid, &bt_4mu_var_varid));
addatt(ncid, bt_4mu_var_varid, "K^2", "brightness temperature variance"
       " at 4.3 micron");

NC(nc_def_var(ncid, "bt_15mu_low", NC_FLOAT, 2, dimid, &bt_15mu_low_varid));
addatt(ncid, bt_15mu_low_varid, "K", "brightness temperature"
       " at 15 micron (low altitudes)");
NC(nc_def_var(ncid, "bt_15mu_low_pt", NC_FLOAT, 2, dimid,
              &bt_15mu_low_pt_varid));
addatt(ncid, bt_15mu_low_pt_varid, "K",
       "brightness temperature perturbation"
       " at 15 micron (low altitudes)");
NC(nc_def_var
   (ncid, "bt_15mu_low_var", NC_FLOAT, 2, dimid, &bt_15mu_low_var_varid));
addatt(ncid, bt_15mu_low_var_varid, "K^2",
       "brightness temperature variance" " at 15 micron (low altitudes)");

NC(nc_def_var(ncid, "bt_15mu_high", NC_FLOAT, 2, dimid,
              &bt_15mu_high_varid));
addatt(ncid, bt_15mu_high_varid, "K", "brightness temperature"
       " at 15 micron (high altitudes)");
NC(nc_def_var(ncid, "bt_15mu_high_pt", NC_FLOAT, 2, dimid,
              &bt_15mu_high_pt_varid));
addatt(ncid, bt_15mu_high_pt_varid, "K",
       "brightness temperature perturbation"
       " at 15 micron (high altitudes)");
NC(nc_def_var
   (ncid, "bt_15mu_high_var", NC_FLOAT, 2, dimid, &bt_15mu_high_var_varid));
addatt(ncid, bt_15mu_high_var_varid, "K^2",
       "brightness temperature variance" " at 15 micron (high altitudes)");
```

```
  /* Leave define mode... */
  NC(nc_enddef(ncid));

  /* Loop over tracks... */
  for (track = 0; track < pert_4mu->ntrack; track++) {

    /* Set array sizes... */
    start[0] = (size_t) track;
    start[1] = 0;
    count[0] = 1;
    count[1] = (size_t) pert_4mu->nxtrack;

    /* Write data... */
    NC(nc_put_vara_double(ncid, time_varid, start, count,
                          pert_4mu->time[track]));
    NC(nc_put_vara_double(ncid, lon_varid, start, count,
                          pert_4mu->lon[track]));
    NC(nc_put_vara_double(ncid, lat_varid, start, count,
                          pert_4mu->lat[track]));

    NC(nc_put_vara_double(ncid, bt_8mu_varid, start, count,
                          pert_4mu->dc[track]));

    NC(nc_put_vara_double(ncid, bt_4mu_varid, start, count,
                          pert_4mu->bt[track]));
    NC(nc_put_vara_double(ncid, bt_4mu_pt_varid, start, count,
                          pert_4mu->pt[track]));
    NC(nc_put_vara_double(ncid, bt_4mu_var_varid, start, count,
                          pert_4mu->var[track]));

    NC(nc_put_vara_double(ncid, bt_15mu_low_varid, start, count,
                          pert_15mu_low->bt[track]));
    NC(nc_put_vara_double(ncid, bt_15mu_low_pt_varid, start, count,
                          pert_15mu_low->pt[track]));
    NC(nc_put_vara_double(ncid, bt_15mu_low_var_varid, start, count,
                          pert_15mu_low->var[track]));

    NC(nc_put_vara_double(ncid, bt_15mu_high_varid, start, count,
                          pert_15mu_high->bt[track]));
    NC(nc_put_vara_double(ncid, bt_15mu_high_pt_varid, start, count,
                          pert_15mu_high->pt[track]));
    NC(nc_put_vara_double(ncid, bt_15mu_high_var_varid, start, count,
                          pert_15mu_high->var[track]));
  }

  /* Close file... */
  NC(nc_close(ncid));

  /* Free... */
  free(pert_4mu);
  free(pert_15mu_low);
  free(pert_15mu_high);

  return EXIT_SUCCESS;
}
```

Here is the call graph for this function:

5.58 perturbation.c

```
#include "libairs.h"

/* ------------------------------------------------------------
   Constants...
   ------------------------------------------------------------ */

/* Number of 4 micron channels: */
#define N4 42

/* Number of 15 micron channels (low altitudes): */
#define N15_LOW 21

/* Number of 15 micron channels (high altitudes): */
#define N15_HIGH 2

/* ------------------------------------------------------------
   Functions...
   ------------------------------------------------------------ */

/* Add variable defintions to netCDF file. */
```

```
void addatt(
  int ncid,
  int varid,
  const char *unit,
  const char *long_name);

/* ------------------------------------------------------------
   Main...
   ------------------------------------------------------------ */

int main(
  int argc,
  char *argv[]) {

  static airs_rad_gran_t airs_rad_gran;

  static pert_t *pert_4mu, *pert_15mu_low, *pert_15mu_high;

  static wave_t wave;

  static double var_dh = 100.;

  static int list_4mu[N4]
    = { 2039, 2040, 2041, 2042, 2043, 2044, 2045, 2046, 2047, 2048,
    2049, 2050, 2051, 2052, 2053, 2054, 2055, 2056, 2057, 2058,
    2059, 2060, 2061, 2062, 2063, 2064, 2071, 2072, 2073, 2074,
    2075, 2076, 2077, 2078, 2079, 2080, 2081, 2082, 2083, 2084,
    2085, 2086
  };

  static int list_15mu_low[N15_LOW]
    = { 4, 10, 16, 22, 29, 35, 41, 55, 83, 88, 94,
    100, 101, 106, 107, 112, 113, 118, 119, 124, 125
  };

  static int list_15mu_high[N15_HIGH]
  = { 74, 75 };

  static int ix, iy, dimid[2], i, n, ncid, track, track0, xtrack,
    time_varid, lon_varid, lat_varid, bt_4mu_varid, bt_4mu_pt_varid,
    bt_4mu_var_varid, bt_8mu_varid, bt_15mu_low_varid, bt_15mu_low_pt_varid,
    bt_15mu_low_var_varid, bt_15mu_high_varid, bt_15mu_high_pt_varid,
    bt_15mu_high_var_varid, iarg;

  static size_t start[2], count[2];

  /* Check arguments... */
  if (argc < 3)
    ERRMSG("Give parameters: <out.nc> <l1b_file1> [<l1b_file2> ...]");

  /* Allocate... */
  ALLOC(pert_4mu, pert_t, 1);
  ALLOC(pert_15mu_low, pert_t, 1);
  ALLOC(pert_15mu_high, pert_t, 1);

  /* ------------------------------------------------------------
     Read HDF files...
     ------------------------------------------------------------ */

  /* Loop over HDF files... */
  for (iarg = 2; iarg < argc; iarg++) {

    /* Read AIRS data... */
    printf("Read AIRS Level-1B data file: %s\n", argv[iarg]);
    airs_rad_rdr(argv[iarg], &airs_rad_gran);

    /* Flag bad observations... */
    for (track = 0; track < AIRS_RAD_GEOTRACK; track++)
      for (xtrack = 0; xtrack < AIRS_RAD_GEOXTRACK; xtrack++)
        for (i = 0; i < AIRS_RAD_CHANNEL; i++)
          if ((airs_rad_gran.state[track][xtrack] != 0)
              || (airs_rad_gran.ExcludedChans[i] > 2)
              || (airs_rad_gran.CalChanSummary[i] & 8)
              || (airs_rad_gran.CalChanSummary[i] & (32 + 64))
              || (airs_rad_gran.CalFlag[track][i] & 16)
              || (airs_rad_gran.Longitude[track][xtrack] < -180)
              || (airs_rad_gran.Longitude[track][xtrack] > 180)
              || (airs_rad_gran.Latitude[track][xtrack] < -90)
              || (airs_rad_gran.Latitude[track][xtrack] > 90))
            airs_rad_gran.radiances[track][xtrack][i] = GSL_NAN;
          else
            airs_rad_gran.radiances[track][xtrack][i] *= 0.001f;

    /* Save geolocation... */
    pert_4mu->ntrack += AIRS_RAD_GEOTRACK;
    if (pert_4mu->ntrack > PERT_NTRACK)
      ERRMSG("Too many granules!");
```

```
  pert_4mu->nxtrack = AIRS_RAD_GEOXTRACK;
  if (pert_4mu->nxtrack > PERT_NXTRACK)
    ERRMSG("Too many tracks!");
  for (track = 0; track < AIRS_RAD_GEOTRACK; track++)
    for (xtrack = 0; xtrack < AIRS_RAD_GEOXTRACK; xtrack++) {
      pert_4mu->time[track0 + track][xtrack]
        = airs_rad_gran.Time[track][xtrack] - 220838400.;
      pert_4mu->lon[track0 + track][xtrack]
        = airs_rad_gran.Longitude[track][xtrack];
      pert_4mu->lat[track0 + track][xtrack]
        = airs_rad_gran.Latitude[track][xtrack];
    }

  pert_15mu_low->ntrack += AIRS_RAD_GEOTRACK;
  if (pert_15mu_low->ntrack > PERT_NTRACK)
    ERRMSG("Too many granules!");
  pert_15mu_low->nxtrack = AIRS_RAD_GEOXTRACK;
  if (pert_15mu_low->nxtrack > PERT_NXTRACK)
    ERRMSG("Too many tracks!");
  for (track = 0; track < AIRS_RAD_GEOTRACK; track++)
    for (xtrack = 0; xtrack < AIRS_RAD_GEOXTRACK; xtrack++) {
      pert_15mu_low->time[track0 + track][xtrack]
        = airs_rad_gran.Time[track][xtrack] - 220838400.;
      pert_15mu_low->lon[track0 + track][xtrack]
        = airs_rad_gran.Longitude[track][xtrack];
      pert_15mu_low->lat[track0 + track][xtrack]
        = airs_rad_gran.Latitude[track][xtrack];
    }

  pert_15mu_high->ntrack += AIRS_RAD_GEOTRACK;
  if (pert_15mu_high->ntrack > PERT_NTRACK)
    ERRMSG("Too many granules!");
  pert_15mu_high->nxtrack = AIRS_RAD_GEOXTRACK;
  if (pert_15mu_high->nxtrack > PERT_NXTRACK)
    ERRMSG("Too many tracks!");
  for (track = 0; track < AIRS_RAD_GEOTRACK; track++)
    for (xtrack = 0; xtrack < AIRS_RAD_GEOXTRACK; xtrack++) {
      pert_15mu_high->time[track0 + track][xtrack]
        = airs_rad_gran.Time[track][xtrack] - 220838400.;
      pert_15mu_high->lon[track0 + track][xtrack]
        = airs_rad_gran.Longitude[track][xtrack];
      pert_15mu_high->lat[track0 + track][xtrack]
        = airs_rad_gran.Latitude[track][xtrack];
    }

  /* Get 8.1 micron brightness temperature... */
  for (track = 0; track < AIRS_RAD_GEOTRACK; track++)
    for (xtrack = 0; xtrack < AIRS_RAD_GEOXTRACK; xtrack++)
      pert_4mu->dc[track0 + track][xtrack]
        = brightness(airs_rad_gran.radiances[track][xtrack][1290],
                     airs_rad_gran.nominal_freq[1290]);

  /* Get 4.3 micron brightness temperature... */
  for (track = 0; track < AIRS_RAD_GEOTRACK; track++)
    for (xtrack = 0; xtrack < AIRS_RAD_GEOXTRACK; xtrack++) {
      n = 0;
      for (i = 0; i < N4; i++)
        if (gsl_finite(airs_rad_gran.radiances[track][xtrack][list_4mu[i]])) {
          pert_4mu->bt[track0 + track][xtrack]
            +=
            brightness(airs_rad_gran.radiances[track][xtrack][list_4mu[i]],
                       airs_rad_gran.nominal_freq[list_4mu[i]]);
          n++;
        }
      if (n > 0.9 * N4)
        pert_4mu->bt[track0 + track][xtrack] /= n;
      else
        pert_4mu->bt[track0 + track][xtrack] = GSL_NAN;
    }

  /* Get 15 micron brightness temperature (low altitudes)... */
  for (track = 0; track < AIRS_RAD_GEOTRACK; track++)
    for (xtrack = 0; xtrack < AIRS_RAD_GEOXTRACK; xtrack++) {
      n = 0;
      for (i = 0; i < N15_LOW; i++)
        if (gsl_finite(airs_rad_gran.radiances
                       [track][xtrack][list_15mu_low[i]])) {
          pert_15mu_low->bt[track0 + track][xtrack]
            += brightness(airs_rad_gran.radiances
                          [track][xtrack][list_15mu_low[i]],
                          airs_rad_gran.nominal_freq[list_15mu_low[i]]);
          n++;
        }
      if (n > 0.9 * N15_LOW)
        pert_15mu_low->bt[track0 + track][xtrack] /= n;
      else
        pert_15mu_low->bt[track0 + track][xtrack] = GSL_NAN;
```

```
      }

    /* Get 15 micron brightness temperature (high altitudes)... */
    for (track = 0; track < AIRS_RAD_GEOTRACK; track++)
      for (xtrack = 0; xtrack < AIRS_RAD_GEOXTRACK; xtrack++) {
        n = 0;
        for (i = 0; i < N15_HIGH; i++)
          if (gsl_finite(airs_rad_gran.radiances
                         [track][xtrack][list_15mu_high[i]])) {
            pert_15mu_high->bt[track0 + track][xtrack]
              += brightness(airs_rad_gran.radiances
                            [track][xtrack][list_15mu_high[i]],
                            airs_rad_gran.nominal_freq[list_15mu_high[i]]);
            n++;
          }
        if (n > 0.9 * N15_HIGH)
          pert_15mu_high->bt[track0 + track][xtrack] /= n;
        else
          pert_15mu_high->bt[track0 + track][xtrack] = GSL_NAN;
      }

    /* Increment track counter... */
    track0 += AIRS_RAD_GEOTRACK;
  }

  /* ------------------------------------------------------------
     Calculate perturbations and variances...
     ------------------------------------------------------------ */

  /* Convert to wave analysis struct... */
  pert2wave(pert_4mu, &wave,
            0, pert_4mu->ntrack - 1, 0, pert_4mu->nxtrack - 1);

  /* Estimate background... */
  background_poly(&wave, 5, 0);

  /* Compute variance... */
  variance(&wave, var_dh);

  /* Copy data... */
  for (ix = 0; ix < wave.nx; ix++)
    for (iy = 0; iy < wave.ny; iy++) {
      pert_4mu->pt[iy][ix] = wave.pt[ix][iy];
      pert_4mu->var[iy][ix] = wave.var[ix][iy];
    }

  /* Convert to wave analysis struct... */
  pert2wave(pert_15mu_low, &wave,
            0, pert_15mu_low->ntrack - 1, 0, pert_15mu_low->nxtrack - 1);

  /* Estimate background... */
  background_poly(&wave, 5, 0);

  /* Compute variance... */
  variance(&wave, var_dh);

  /* Copy data... */
  for (ix = 0; ix < wave.nx; ix++)
    for (iy = 0; iy < wave.ny; iy++) {
      pert_15mu_low->pt[iy][ix] = wave.pt[ix][iy];
      pert_15mu_low->var[iy][ix] = wave.var[ix][iy];
    }

  /* Convert to wave analysis struct... */
  pert2wave(pert_15mu_high, &wave,
            0, pert_15mu_high->ntrack - 1, 0, pert_15mu_high->nxtrack - 1);

  /* Estimate background... */
  background_poly(&wave, 5, 0);

  /* Compute variance... */
  variance(&wave, var_dh);

  /* Copy data... */
  for (ix = 0; ix < wave.nx; ix++)
    for (iy = 0; iy < wave.ny; iy++) {
      pert_15mu_high->pt[iy][ix] = wave.pt[ix][iy];
      pert_15mu_high->var[iy][ix] = wave.var[ix][iy];
    }

  /* ------------------------------------------------------------
     Write to netCDF file...
     ------------------------------------------------------------ */

  /* Create netCDF file... */
  NC(nc_create(argv[1], NC_CLOBBER, &ncid));

```

```
  /* Set dimensions... */
  NC(nc_def_dim(ncid, "NTRACK", NC_UNLIMITED, &dimid[0]));
  NC(nc_def_dim(ncid, "NXTRACK", AIRS_RAD_GEOXTRACK, &dimid[1]));

  /* Add variables... */
  NC(nc_def_var(ncid, "time", NC_DOUBLE, 2, dimid, &time_varid));
  addatt(ncid, time_varid, "s", "time (seconds since 2000-01-01T00:00Z)");
  NC(nc_def_var(ncid, "lon", NC_DOUBLE, 2, dimid, &lon_varid));
  addatt(ncid, lon_varid, "deg", "footprint longitude");
  NC(nc_def_var(ncid, "lat", NC_DOUBLE, 2, dimid, &lat_varid));
  addatt(ncid, lat_varid, "deg", "footprint latitude");

  NC(nc_def_var(ncid, "bt_8mu", NC_FLOAT, 2, dimid, &bt_8mu_varid));
  addatt(ncid, bt_8mu_varid, "K", "brightness temperature at 8.1 micron");

  NC(nc_def_var(ncid, "bt_4mu", NC_FLOAT, 2, dimid, &bt_4mu_varid));
  addatt(ncid, bt_4mu_varid, "K", "brightness temperature" " at 4.3 micron");
  NC(nc_def_var(ncid, "bt_4mu_pt", NC_FLOAT, 2, dimid, &bt_4mu_pt_varid));
  addatt(ncid, bt_4mu_pt_varid, "K", "brightness temperature perturbation"
         " at 4.3 micron");
  NC(nc_def_var(ncid, "bt_4mu_var", NC_FLOAT, 2, dimid, &bt_4mu_var_varid));
  addatt(ncid, bt_4mu_var_varid, "K^2", "brightness temperature variance"
         " at 4.3 micron");

  NC(nc_def_var(ncid, "bt_15mu_low", NC_FLOAT, 2, dimid, &bt_15mu_low_varid));
  addatt(ncid, bt_15mu_low_varid, "K", "brightness temperature"
         " at 15 micron (low altitudes)");
  NC(nc_def_var(ncid, "bt_15mu_low_pt", NC_FLOAT, 2, dimid,
                &bt_15mu_low_pt_varid));
  addatt(ncid, bt_15mu_low_pt_varid, "K",
         "brightness temperature perturbation"
         " at 15 micron (low altitudes)");
  NC(nc_def_var
     (ncid, "bt_15mu_low_var", NC_FLOAT, 2, dimid, &bt_15mu_low_var_varid));
  addatt(ncid, bt_15mu_low_var_varid, "K^2",
         "brightness temperature variance" " at 15 micron (low altitudes)");

  NC(nc_def_var(ncid, "bt_15mu_high", NC_FLOAT, 2, dimid,
                &bt_15mu_high_varid));
  addatt(ncid, bt_15mu_high_varid, "K", "brightness temperature"
         " at 15 micron (high altitudes)");
  NC(nc_def_var(ncid, "bt_15mu_high_pt", NC_FLOAT, 2, dimid,
                &bt_15mu_high_pt_varid));
  addatt(ncid, bt_15mu_high_pt_varid, "K",
         "brightness temperature perturbation"
         " at 15 micron (high altitudes)");
  NC(nc_def_var
     (ncid, "bt_15mu_high_var", NC_FLOAT, 2, dimid, &bt_15mu_high_var_varid));
  addatt(ncid, bt_15mu_high_var_varid, "K^2",
         "brightness temperature variance" " at 15 micron (high altitudes)");

  /* Leave define mode... */
  NC(nc_enddef(ncid));

  /* Loop over tracks... */
  for (track = 0; track < pert_4mu->ntrack; track++) {

    /* Set array sizes... */
    start[0] = (size_t) track;
    start[1] = 0;
    count[0] = 1;
    count[1] = (size_t) pert_4mu->nxtrack;

    /* Write data... */
    NC(nc_put_vara_double(ncid, time_varid, start, count,
                          pert_4mu->time[track]));
    NC(nc_put_vara_double(ncid, lon_varid, start, count,
                          pert_4mu->lon[track]));
    NC(nc_put_vara_double(ncid, lat_varid, start, count,
                          pert_4mu->lat[track]));

    NC(nc_put_vara_double(ncid, bt_8mu_varid, start, count,
                          pert_4mu->dc[track]));

    NC(nc_put_vara_double(ncid, bt_4mu_varid, start, count,
                          pert_4mu->bt[track]));
    NC(nc_put_vara_double(ncid, bt_4mu_pt_varid, start, count,
                          pert_4mu->pt[track]));
    NC(nc_put_vara_double(ncid, bt_4mu_var_varid, start, count,
                          pert_4mu->var[track]));

    NC(nc_put_vara_double(ncid, bt_15mu_low_varid, start, count,
                          pert_15mu_low->bt[track]));
    NC(nc_put_vara_double(ncid, bt_15mu_low_pt_varid, start, count,
                          pert_15mu_low->pt[track]));
    NC(nc_put_vara_double(ncid, bt_15mu_low_var_varid, start, count,
                          pert_15mu_low->var[track]));
```

```
    NC(nc_put_vara_double(ncid, bt_15mu_high_varid, start, count,
                          pert_15mu_high->bt[track]));
    NC(nc_put_vara_double(ncid, bt_15mu_high_pt_varid, start, count,
                          pert_15mu_high->pt[track]));
    NC(nc_put_vara_double(ncid, bt_15mu_high_var_varid, start, count,
                          pert_15mu_high->var[track]));
  }

  /* Close file... */
  NC(nc_close(ncid));

  /* Free... */
  free(pert_4mu);
  free(pert_15mu_low);
  free(pert_15mu_high);

  return EXIT_SUCCESS;
}

/*****************************************************************************/

void addatt(
  int ncid,
  int varid,
  const char *unit,
  const char *long_name) {

  /* Set long name... */
  NC(nc_put_att_text(ncid, varid, "long_name", strlen(long_name), long_name));

  /* Set units... */
  NC(nc_put_att_text(ncid, varid, "units", strlen(unit), unit));
}
```

5.59 rayt.c File Reference

Functions

- double buoyancy (double z0, double p0, double t0, double z1, double p1, double t1)
- double scale_height (double t)
- double temp2theta (double p, double t)
- int main (int argc, char *argv[ ])

5.59.1 Function Documentation

5.59.1.1 double buoyancy (double *z0*, double *p0*, double *t0*, double *z1*, double *p1*, double *t1*)

Definition at line 204 of file rayt.c.

```
              {

  double theta0, theta1;

  /* Get potential temperature... */
  theta0 = temp2theta(p0, t0);
  theta1 = temp2theta(p1, t1);

  /* Get buoyancy frequency... */
  return sqrt(G0 / (0.5 * (theta0 + theta1)) * (theta1 - theta0) /
              ((z1 - z0) * 1e3));
}
```

Here is the call graph for this function:

5.59.1.2 double scale_height (double *t*)

Definition at line 225 of file rayt.c.

```
                {

  return 29.26 * t / 1e3;
}
```

5.59.1.3 double temp2theta (double *p*, double *t*)

Definition at line 233 of file rayt.c.

```
                {

  return t * pow(P0 / p, 0.286);
}
```

5.59.1.4 int main (int *argc*, char ∗ *argv[ ]*)

Definition at line 36 of file rayt.c.

```
                    {

  FILE *in;

  static double f0, k, omin, z[NZ], u[NZ], urel[NZ], v[NZ], bf[NZ], bf2[NZ],
    H[NZ], frel[NZ], osign[NZ], f1[NZ], f2[NZ], delta[NZ], a2[NZ], m[NZ],
    dxdz[NZ], cgz[NZ], dz, path[NZ], tim[NZ], costh, p[NZ], t[NZ], z0, w,
    wsum, dzw = 5 * 1e3, fgb, m0, alpha, lat;

  static int iz, iz2, izcrit, izrefl, nz;

  /* Check arguments... */
  if (argc != 8)
    ERRMSG("Give parameters: <atm.tab> <z_launch> <mode> "
           "<t_gb | lz_launch> <lx> <lat> <direct>");

  /* Get launch level... */
  z0 = atof(argv[2]);
  lat = atof(argv[6]);
  alpha = atof(argv[7]);

  /* Read atmosphere above launch level... */
  if (!(in = fopen(argv[1], "r")))
    ERRMSG("Cannot open atmospheric data file!");
  while (fscanf
         (in, "%lg %lg %lg %lg %lg", &z[nz], &p[nz], &t[nz], &u[nz], &v[nz])
         == 5)
    if (z[nz] >= z0) {
      u[nz] =
        cos(alpha * M_PI / 180.) * u[nz] + sin(alpha * M_PI / 180.) * v[nz];
      if ((++nz) > NZ)
        ERRMSG("Too many altitude levels!");
    }
  fclose(in);

  /* Compute scale height and buoyancy frequency... */
  for (iz = 0; iz < nz; iz++) {
    if (iz < nz - 1)
      bf[iz] = buoyancy(z[iz], p[iz], t[iz], z[iz + 1], p[iz + 1], t[iz + 1]);
    else
      bf[iz] = bf[iz - 1];
    H[iz] = scale_height(t[iz]) * 1e3;
    z[iz] *= 1e3;
  }

  /* Smooth N profile... */
  for (iz = 0; iz < nz; iz++) {
    bf2[iz] = wsum = 0;
    for (iz2 = 0; iz2 < nz; iz2++) {
      if (!gsl_finite(bf[iz2]) ||
          !gsl_finite(bf[GSL_MAX(iz2 - 1, 0)]) ||
```

```
          !gsl_finite(bf[GSL_MIN(iz2 + 1, nz - 1)]))
        continue;
      w =
        (fabs(z[iz] - z[iz2]) < dzw) ? 1.0 - fabs(z[iz] - z[iz2]) / dzw : 0.0;
      bf2[iz] += w * bf[iz2];
      wsum += w;
    }
    bf2[iz] /= wsum;
  }
  for (iz = 0; iz < nz; iz++)
    bf[iz] = bf2[iz];

  /* Get horizontal wavenumber... */
  k = 2 * M_PI / (atof(argv[5]) * 1e3);

  /* Get minimum gravity wave frequency (Coriolis parameter)... */
  omin = 2 * 2 * M_PI / 86400. * sin(lat / 180. * M_PI);

  /* Get initial frequencies... */
  if (argv[3][0] == 't') {

    /* Get ground-based frequency... */
    fgb = 2 * M_PI / (atof(argv[4]) * 60.);

    /* Get intrinsic frequency at launch level... */
    f0 = fgb - k * u[0];

  } else if (argv[3][0] == 'l') {

    /* Get vertical wavenumber... */
    m0 = 2 * M_PI / (atof(argv[4]) * 1e3);

    /* Get intrinsic frequency at launch level... */
    f0 =
      sqrt((bf[0] * bf[0] * k * k +
            omin * omin * (m0 * m0 + 0.25 / (H[0] * H[0])))
           / (m0 * m0 + k * k + 0.25 / (H[0] * H[0])));

    /* Get ground-based frequency... */
    fgb = f0 + k * u[0];

  } else
    ERRMSG("Set <mode> to 't_gb' or 'lz_launch'!");

  /* Loop over layers... */
  for (iz = 0; iz < nz; iz++) {
    urel[iz] = u[iz] - u[0];
    frel[iz] = f0 - k * urel[iz];
    osign[iz] = frel[iz] / fabs(frel[iz]);
    f1[iz] = (bf[iz] * bf[iz] - frel[iz] * frel[iz]) / frel[iz];
    f2[iz] = (frel[iz] * frel[iz] - omin * omin) / frel[iz];
    delta[iz] = k * k * (1 + f1[iz] / f2[iz]);
    a2[iz] = 1. / 4. / (H[iz] * H[iz]);
    m[iz] = (-osign[iz]) * k * sqrt((f1[iz] / f2[iz]) - (a2[iz] / (k * k)));
    dxdz[iz] = (u[iz] * delta[iz] + k * f1[iz]) / (-1 * m[iz] * f2[iz]);
    dz = z[1] - z[0];
    cgz[iz] = f2[iz] * (-1. * m[iz]) / (k * k + m[iz] * m[iz] + a2[iz]);
  }

  /* Integrate via trapezoidal rule... */
  for (iz = 1; iz < nz; iz++) {
    path[iz] = path[iz - 1] + dz * .5 * (dxdz[iz - 1] + dxdz[iz]);
    tim[iz] = tim[iz - 1] + dz * 2. / (cgz[iz - 1] + cgz[iz]);
  }

  /* Find critical level... */
  for (izcrit = 0; izcrit < nz; izcrit++)
    if (f0 / fabs(f0) * frel[izcrit] / fabs(omin) <= 1)
      break;

  /* Find trapping/reflection level... */
  for (izrefl = 0; izrefl < nz; izrefl++) {
    costh = fabs(f0 - k * urel[izrefl])
      / sqrt(bf[izrefl] * bf[izrefl]
             * (1 -
                (1 -
                 (omin / bf[izrefl]) * (omin / bf[izrefl])) / (k * k /
                                                               a2[izrefl] +
                                                               1)));
    if (costh >= 1.0)
      break;
  }

  /* Filter data... */
  for (iz = 0; iz < nz; iz++)
    if (iz >= izcrit || iz >= izrefl)
      path[iz] = tim[iz] = m[iz] = frel[iz] = cgz[iz] = sqrt(-1.0);
```

```

  /* Write output... */
  printf("# $1  = latitude [deg]\n"
         "# $2  = altitude [km]\n"
         "# $3  = pressure [hPa]\n"
         "# $4  = temperature [K]\n"
         "# $5  = potential temperature [K]\n"
         "# $6  = wind speed [m/s]\n"
         "# $7  = buoyancy frequency [1/s]\n"
         "# $8  = scale height [km]\n"
         "# $9  = horizontal distance [km]\n"
         "# $10 = propagation time [min]\n"
         "# $11 = vertical wavelength [km]\n"
         "# $12 = wave period [min]\n"
         "# $13 = vertical group velocity [m/s]\n\n");
  for (iz = 0; iz < nz; iz++)
    printf("%g %g %g %g %g %g %g %g %g %g %g %g %g\n",
           lat, z[iz] / 1e3, p[iz], t[iz], temp2theta(p[iz], t[iz]), u[iz],
           bf[iz], H[iz] / 1e3, path[iz] / 1e3, tim[iz] / 60,
           fabs(2 * M_PI / m[iz] / 1e3), 2. * M_PI / frel[iz] / 60., cgz[iz]);
  printf("\n# z_crit= %g km\n# z_refl= %g km\n",
         z[izcrit - 1] / 1e3, z[izrefl - 1] / 1e3);

  return EXIT_SUCCESS;
}
```

Here is the call graph for this function:

5.60 rayt.c

```
#include "libairs.h"

/* ------------------------------------------------------------
   Dimensions...
   ------------------------------------------------------------ */

/* Maximum number of levels. */
#define NZ 1000

/* ------------------------------------------------------------
   Functions...
   ------------------------------------------------------------ */

/* Compute buoyancy frequency. */
double buoyancy(
  double z0,
  double p0,
  double t0,
  double z1,
  double p1,
  double t1);

/* Compute scale height. */
double scale_height(
  double t);

/* Convert temperature to potential temperature. */
double temp2theta(
  double p,
  double t);

/* ------------------------------------------------------------
   Main...
   ------------------------------------------------------------ */

int main(
  int argc,
  char *argv[]) {

  FILE *in;

  static double f0, k, omin, z[NZ], u[NZ], urel[NZ], v[NZ], bf[NZ], bf2[NZ],
    H[NZ], frel[NZ], osign[NZ], f1[NZ], f2[NZ], delta[NZ], a2[NZ], m[NZ],
    dxdz[NZ], cgz[NZ], dz, path[NZ], tim[NZ], costh, p[NZ], t[NZ], z0, w,
    wsum, dzw = 5 * 1e3, fgb, m0, alpha, lat;

  static int iz, iz2, izcrit, izrefl, nz;

  /* Check arguments... */
  if (argc != 8)
    ERRMSG("Give parameters: <atm.tab> <z_launch> <mode> "
```

```
          "<t_gb | lz_launch> <lx> <lat> <direct>");

  /* Get launch level... */
  z0 = atof(argv[2]);
  lat = atof(argv[6]);
  alpha = atof(argv[7]);

  /* Read atmosphere above launch level... */
  if (!(in = fopen(argv[1], "r")))
    ERRMSG("Cannot open atmospheric data file!");
  while (fscanf
         (in, "%lg %lg %lg %lg %lg", &z[nz], &p[nz], &t[nz], &u[nz], &v[nz])
         == 5)
    if (z[nz] >= z0) {
      u[nz] =
        cos(alpha * M_PI / 180.) * u[nz] + sin(alpha * M_PI / 180.) * v[nz];
      if ((++nz) > NZ)
        ERRMSG("Too many altitude levels!");
    }
  fclose(in);

  /* Compute scale height and buoyancy frequency... */
  for (iz = 0; iz < nz; iz++) {
    if (iz < nz - 1)
      bf[iz] = buoyancy(z[iz], p[iz], t[iz], z[iz + 1], p[iz + 1], t[iz + 1]);
    else
      bf[iz] = bf[iz - 1];
    H[iz] = scale_height(t[iz]) * 1e3;
    z[iz] *= 1e3;
  }

  /* Smooth N profile... */
  for (iz = 0; iz < nz; iz++) {
    bf2[iz] = wsum = 0;
    for (iz2 = 0; iz2 < nz; iz2++) {
      if (!gsl_finite(bf[iz2]) ||
          !gsl_finite(bf[GSL_MAX(iz2 - 1, 0)]) ||
          !gsl_finite(bf[GSL_MIN(iz2 + 1, nz - 1)]))
        continue;
      w =
        (fabs(z[iz] - z[iz2]) < dzw) ? 1.0 - fabs(z[iz] - z[iz2]) / dzw : 0.0;
      bf2[iz] += w * bf[iz2];
      wsum += w;
    }
    bf2[iz] /= wsum;
  }
  for (iz = 0; iz < nz; iz++)
    bf[iz] = bf2[iz];

  /* Get horizontal wavenumber... */
  k = 2 * M_PI / (atof(argv[5]) * 1e3);

  /* Get minimum gravity wave frequency (Coriolis parameter)... */
  omin = 2 * 2 * M_PI / 86400. * sin(lat / 180. * M_PI);

  /* Get initial frequencies... */
  if (argv[3][0] == 't') {

    /* Get ground-based frequency... */
    fgb = 2 * M_PI / (atof(argv[4]) * 60.);

    /* Get intrinsic frequency at launch level... */
    f0 = fgb - k * u[0];

  } else if (argv[3][0] == 'l') {

    /* Get vertical wavenumber... */
    m0 = 2 * M_PI / (atof(argv[4]) * 1e3);

    /* Get intrinsic frequency at launch level... */
    f0 =
      sqrt((bf[0] * bf[0] * k * k +
            omin * omin * (m0 * m0 + 0.25 / (H[0] * H[0])))
           / (m0 * m0 + k * k + 0.25 / (H[0] * H[0])));

    /* Get ground-based frequency... */
    fgb = f0 + k * u[0];

  } else
    ERRMSG("Set <mode> to 't_gb' or 'lz_launch'!");

  /* Loop over layers... */
  for (iz = 0; iz < nz; iz++) {
    urel[iz] = u[iz] - u[0];
    frel[iz] = f0 - k * urel[iz];
    osign[iz] = frel[iz] / fabs(frel[iz]);
    f1[iz] = (bf[iz] * bf[iz] - frel[iz] * frel[iz]) / frel[iz];
```

```
    f2[iz] = (frel[iz] * frel[iz] - omin * omin) / frel[iz];
    delta[iz] = k * k * (1 + f1[iz] / f2[iz]);
    a2[iz] = 1. / 4. / (H[iz] * H[iz]);
    m[iz] = (-osign[iz]) * k * sqrt((f1[iz] / f2[iz]) - (a2[iz] / (k * k)));
    dxdz[iz] = (u[iz] * delta[iz] + k * f1[iz]) / (-1 * m[iz] * f2[iz]);
    dz = z[1] - z[0];
    cgz[iz] = f2[iz] * (-1. * m[iz]) / (k * k + m[iz] * m[iz] + a2[iz]);
  }

  /* Integrate via trapezoidal rule... */
  for (iz = 1; iz < nz; iz++) {
    path[iz] = path[iz - 1] + dz * .5 * (dxdz[iz - 1] + dxdz[iz]);
    tim[iz] = tim[iz - 1] + dz * 2. / (cgz[iz - 1] + cgz[iz]);
  }

  /* Find critical level... */
  for (izcrit = 0; izcrit < nz; izcrit++)
    if (f0 / fabs(f0) * frel[izcrit] / fabs(omin) <= 1)
      break;

  /* Find trapping/reflection level... */
  for (izrefl = 0; izrefl < nz; izrefl++) {
    costh = fabs(f0 - k * urel[izrefl])
      / sqrt(bf[izrefl] * bf[izrefl]
             * (1 -
                (1 -
                 (omin / bf[izrefl]) * (omin / bf[izrefl])) / (k * k /
                                                               a2[izrefl] +
                                                               1)));
    if (costh >= 1.0)
      break;
  }

  /* Filter data... */
  for (iz = 0; iz < nz; iz++)
    if (iz >= izcrit || iz >= izrefl)
      path[iz] = tim[iz] = m[iz] = frel[iz] = cgz[iz] = sqrt(-1.0);

  /* Write output... */
  printf("# $1  = latitude [deg]\n"
         "# $2  = altitude [km]\n"
         "# $3  = pressure [hPa]\n"
         "# $4  = temperature [K]\n"
         "# $5  = potential temperature [K]\n"
         "# $6  = wind speed [m/s]\n"
         "# $7  = buoyancy frequency [1/s]\n"
         "# $8  = scale height [km]\n"
         "# $9  = horizontal distance [km]\n"
         "# $10 = propagation time [min]\n"
         "# $11 = vertical wavelength [km]\n"
         "# $12 = wave period [min]\n"
         "# $13 = vertical group velocity [m/s]\n\n");
  for (iz = 0; iz < nz; iz++)
    printf("%g %g %g %g %g %g %g %g %g %g %g %g %g\n",
           lat, z[iz] / 1e3, p[iz], t[iz], temp2theta(p[iz], t[iz]), u[iz],
           bf[iz], H[iz] / 1e3, path[iz] / 1e3, tim[iz] / 60,
           fabs(2 * M_PI / m[iz] / 1e3), 2. * M_PI / frel[iz] / 60., cgz[iz]);
  printf("\n# z_crit= %g km\n# z_refl= %g km\n",
         z[izcrit - 1] / 1e3, z[izrefl - 1] / 1e3);

  return EXIT_SUCCESS;
}

/*****************************************************************************/

double buoyancy(
  double z0,
  double p0,
  double t0,
  double z1,
  double p1,
  double t1) {

  double theta0, theta1;

  /* Get potential temperature... */
  theta0 = temp2theta(p0, t0);
  theta1 = temp2theta(p1, t1);

  /* Get buoyancy frequency... */
  return sqrt(G0 / (0.5 * (theta0 + theta1)) * (theta1 - theta0) /
              ((z1 - z0) * 1e3));
}

/*****************************************************************************/

double scale_height(
```

```
  double t) {

  return 29.26 * t / 1e3;
}

/*****************************************************************************/

double temp2theta(
  double p,
  double t) {

  return t * pow(P0 / p, 0.286);
}
```

5.61 ret2tab.c File Reference

Functions

- int main (int argc, char ∗argv[ ])

5.61.1 Function Documentation

5.61.1.1 int main (int *argc,* char ∗ *argv[ ]*)

Definition at line 14 of file ret2tab.c.

```
00016                 {
00017 
00018   static airs_ret_gran_t airs_ret_gran;
00019 
00020   FILE *out;
00021 
00022   int lay, track, xtrack;
00023 
00024   /* Check arguments... */
00025   if (argc != 4)
00026     ERRMSG("Give parameters: <airs_l2_file> <layer> <airs.tab>");
00027 
00028   /* Get arguments... */
00029   lay = atoi(argv[2]);
00030 
00031   /* Read AIRS data... */
00032   printf("Read AIRS Level-2 data file: %s\n", argv[1]);
00033   airs_ret_rdr(argv[1], &airs_ret_gran);
00034 
00035   /* Create output file... */
00036   printf("Write ASCII file: %s\n", argv[3]);
00037   if (!(out = fopen(argv[3], "w")))
00038     ERRMSG("Cannot create file!");
00039 
00040   /* Write header... */
00041   fprintf(out,
00042           "# $1  = time (seconds since 01-JAN-2000, 00:00 UTC)\n"
00043           "# $2  = altitude [km]\n"
00044           "# $3  = longitude [deg]\n"
00045           "# $4  = latitude [deg]\n"
00046           "# $5  = pressure [hPa]\n"
00047           "# $6  = temperature [K]\n"
00048           "# $7  = H2O mass mixing ratio\n"
00049           "# $8  = O3 volume mixing ratio\n"
00050           "# $9  = CH4 volume mixing ratio\n"
00051           "# $10 = CO volume mixing ratio\n");
00052 
00053   /* Write data to stdout... */
00054   for (track = 0; track < AIRS_RET_GEOTRACK; track++) {
00055     fprintf(out, "\n");
00056     for (xtrack = 0; xtrack < AIRS_RET_GEOXTRACK; xtrack++)
00057       fprintf(out, "%.2f %g %g %g %g %g %g %g %g %g\n",
00058               airs_ret_gran.Time[track][xtrack] - 220838400,
00059               CHECK(airs_ret_gran.GP_Height[track][xtrack][lay]) / 1000,
00060               CHECK(airs_ret_gran.Longitude[track][xtrack]),
00061               CHECK(airs_ret_gran.Latitude[track][xtrack]),
00062               CHECK(airs_ret_gran.pressStd[lay]),
00063               CHECK(airs_ret_gran.TAirStd[track][xtrack][lay]),
```

```
              CHECK(airs_ret_gran.H2OMMRStd[track][xtrack][lay]),
              CHECK(airs_ret_gran.O3VMRStd[track][xtrack][lay]),
              CHECK(airs_ret_gran.COVMRLevStd[track][xtrack][lay]),
              CHECK(airs_ret_gran.CH4VMRLevStd[track][xtrack][lay]));
  }

  /* Close file... */
  fclose(out);

  return EXIT_SUCCESS;
}
```

5.62 ret2tab.c

```
#include "libairs.h"

/* ------------------------------------------------------------
   Macros...
   ------------------------------------------------------------ */

/* Replace dummy values by nan. */
#define CHECK(x) ((x)!=-9999 ? (x) : GSL_NAN)

/* ------------------------------------------------------------
   Main...
   ------------------------------------------------------------ */

int main(
  int argc,
  char *argv[]) {

  static airs_ret_gran_t airs_ret_gran;

  FILE *out;

  int lay, track, xtrack;

  /* Check arguments... */
  if (argc != 4)
    ERRMSG("Give parameters: <airs_l2_file> <layer> <airs.tab>");

  /* Get arguments... */
  lay = atoi(argv[2]);

  /* Read AIRS data... */
  printf("Read AIRS Level-2 data file: %s\n", argv[1]);
  airs_ret_rdr(argv[1], &airs_ret_gran);

  /* Create output file... */
  printf("Write ASCII file: %s\n", argv[3]);
  if (!(out = fopen(argv[3], "w")))
    ERRMSG("Cannot create file!");

  /* Write header... */
  fprintf(out,
          "# $1  = time (seconds since 01-JAN-2000, 00:00 UTC)\n"
          "# $2  = altitude [km]\n"
          "# $3  = longitude [deg]\n"
          "# $4  = latitude [deg]\n"
          "# $5  = pressure [hPa]\n"
          "# $6  = temperature [K]\n"
          "# $7  = H2O mass mixing ratio\n"
          "# $8  = O3 volume mixing ratio\n"
          "# $9  = CH4 volume mixing ratio\n"
          "# $10 = CO volume mixing ratio\n");

  /* Write data to stdout... */
  for (track = 0; track < AIRS_RET_GEOTRACK; track++) {
    fprintf(out, "\n");
    for (xtrack = 0; xtrack < AIRS_RET_GEOXTRACK; xtrack++)
      fprintf(out, "%.2f %g %g %g %g %g %g %g %g %g\n",
              airs_ret_gran.Time[track][xtrack] - 220838400,
              CHECK(airs_ret_gran.GP_Height[track][xtrack][lay]) / 1000,
              CHECK(airs_ret_gran.Longitude[track][xtrack]),
              CHECK(airs_ret_gran.Latitude[track][xtrack]),
              CHECK(airs_ret_gran.pressStd[lay]),
              CHECK(airs_ret_gran.TAirStd[track][xtrack][lay]),
              CHECK(airs_ret_gran.H2OMMRStd[track][xtrack][lay]),
              CHECK(airs_ret_gran.O3VMRStd[track][xtrack][lay]),
              CHECK(airs_ret_gran.COVMRLevStd[track][xtrack][lay]),
              CHECK(airs_ret_gran.CH4VMRLevStd[track][xtrack][lay]));
  }

```

```
/* Close file... */
fclose(out);

return EXIT_SUCCESS;
}
```

5.63 retrieval.c File Reference

Data Structures

- struct ncd_t
- struct ret_t

    Retrieval results.

Functions

- void add_var (int ncid, const char *varname, const char *unit, const char *longname, int type, int dimid[ ], int *varid, int ndims)

    Add variable to netCDF file.
- void buffer_nc (atm_t *atm, double chisq, ncd_t *ncd, int track, int xtrack, int np0, int np1)
- double cost_function (gsl_vector *dx, gsl_vector *dy, gsl_matrix *s_a_inv, gsl_vector *sig_eps_inv)
- void fill_gaps (double x[L2_NTRACK][L2_NXTRACK][L2_NLAY], double cx, double cy)
- void init_l2 (ncd_t *ncd, int track, int xtrack, ctl_t *ctl, atm_t *atm)
- void matrix_invert (gsl_matrix *a)
- void matrix_product (gsl_matrix *a, gsl_vector *b, int transpose, gsl_matrix *c)
- void optimal_estimation (ret_t *ret, ctl_t *ctl, obs_t *obs_meas, obs_t *obs_i, atm_t *atm_apr, atm_t *atm_i, double *chisq)
- void read_nc (char *filename, ncd_t *ncd)
- void read_ret_ctl (int argc, char *argv[ ], ctl_t *ctl, ret_t *ret)
- void set_cov_apr (ret_t *ret, ctl_t *ctl, atm_t *atm, int *iqa, int *ipa, gsl_matrix *s_a)
- void set_cov_meas (ret_t *ret, ctl_t *ctl, obs_t *obs, gsl_vector *sig_noise, gsl_vector *sig_formod, gsl_↩ vector *sig_eps_inv)
- double sza (double sec, double lon, double lat)

    Calculate solar zenith angle.
- void write_nc (char *filename, ncd_t *ncd)
- int main (int argc, char *argv[ ])

5.63.1 Function Documentation

5.63.1.1 void add_var (int *ncid,* const char * *varname,* const char * *unit,* const char * *longname,* int *type,* int *dimid[ ],* int * *varid,* int *ndims*)

Add variable to netCDF file.

Definition at line 483 of file retrieval.c.

```
               {

  /* Check if variable exists... */
  if (nc_inq_varid(ncid, varname, varid) != NC_NOERR) {

    /* Define variable... */
    NC(nc_def_var(ncid, varname, type, ndims, dimid, varid));

    /* Set long name... */
    NC(nc_put_att_text
       (ncid, *varid, "long_name", strlen(longname), longname));

    /* Set units... */
    NC(nc_put_att_text(ncid, *varid, "units", strlen(unit), unit));
  }
}
```

5.63.1.2 void buffer_nc (atm_t * atm, double chisq, ncd_t * ncd, int track, int xtrack, int np0, int np1)

Definition at line 510 of file retrieval.c.

```
                  {

  int ip;

  /* Set number of data points... */
  ncd->np = np1 - np0 + 1;

  /* Save retrieval data... */
  for (ip = np0; ip <= np1; ip++) {
    ncd->ret_z[ip - np0] = (float) atm->z[ip];
    ncd->ret_p[track * L1_NXTRACK + xtrack] = (float) atm->p[np0];
    ncd->ret_t[(track * L1_NXTRACK + xtrack) * ncd->np + ip - np0] =
      (gsl_finite(chisq) ? (float) atm->t[ip] : GSL_NAN);
  }
}
```

5.63.1.3 double cost_function (gsl_vector * dx, gsl_vector * dy, gsl_matrix * s_a_inv, gsl_vector * sig_eps_inv)

Definition at line 535 of file retrieval.c.

```
                                 {

  gsl_vector *x_aux, *y_aux;

  double chisq_a, chisq_m = 0;

  size_t i, m, n;

  /* Get sizes... */
  m = dy->size;
  n = dx->size;

  /* Allocate... */
  x_aux = gsl_vector_alloc(n);
  y_aux = gsl_vector_alloc(m);

  /* Determine normalized cost function...
     (chi^2 = 1/m * [dy^T * S_eps^{-1} * dy + dx^T * S_a^{-1} * dx]) */
  for (i = 0; i < m; i++)
    chisq_m +=
      gsl_pow_2(gsl_vector_get(dy, i) * gsl_vector_get(sig_eps_inv, i));
  gsl_blas_dgemv(CblasNoTrans, 1.0, s_a_inv, dx, 0.0, x_aux);
  gsl_blas_ddot(dx, x_aux, &chisq_a);

  /* Free... */
  gsl_vector_free(x_aux);
  gsl_vector_free(y_aux);

  /* Return cost function value... */
  return (chisq_m + chisq_a) / (double) m;
}
```

5.63.1.4 void fill_gaps (double x[L2_NTRACK][L2_NXTRACK][L2_NLAY], double cx, double cy)

Definition at line 573 of file retrieval.c.

```
                    {

  double help[L2_NTRACK][L2_NXTRACK], w, wsum;

  int lay, track, track2, xtrack, xtrack2;

  /* Loop over layers... */
  for (lay = 0; lay < L2_NLAY; lay++) {

    /* Loop over grid points... */
    for (track = 0; track < L2_NTRACK; track++)
      for (xtrack = 0; xtrack < L2_NXTRACK; xtrack++) {

```

```
        /* Init... */
        help[track][xtrack] = 0;
        wsum = 0;

        /* Averrage data points... */
        for (track2 = 0; track2 < L2_NTRACK; track2++)
          for (xtrack2 = 0; xtrack2 < L2_NXTRACK; xtrack2++)
            if (gsl_finite(x[track2][xtrack2][lay])
                && x[track2][xtrack2][lay] > 0) {
              w = exp(-gsl_pow_2((xtrack - xtrack2) / cx)
                      - gsl_pow_2((track - track2) / cy));
              help[track][xtrack] += w * x[track2][xtrack2][lay];
              wsum += w;
            }

        /* Normalize... */
        if (wsum > 0)
          help[track][xtrack] /= wsum;
        else
          help[track][xtrack] = GSL_NAN;
      }

    /* Copy grid points... */
    for (track = 0; track < L2_NTRACK; track++)
      for (xtrack = 0; xtrack < L2_NXTRACK; xtrack++)
        x[track][xtrack][lay] = help[track][xtrack];
  }
}
```

5.63.1.5 void init_l2 (ncd_t * *ncd*, int *track*, int *xtrack*, ctl_t * *ctl*, atm_t * *atm*)

Definition at line 620 of file retrieval.c.

```
                 {

  static atm_t atm_airs;

  double k[NW], p, q[NG], t, w, zmax = 0, zmin = 1000;

  int ip, lay;

  /* Reset track- and xtrack-index to match Level-2 data... */
  track /= 3;
  xtrack /= 3;

  /* Store AIRS data in atmospheric data struct... */
  atm_airs.np = 0;
  for (lay = 0; lay < L2_NLAY; lay++)
    if (gsl_finite(ncd->l2_z[track][xtrack][lay])) {
      atm_airs.z[atm_airs.np] = ncd->l2_z[track][xtrack][lay];
      atm_airs.p[atm_airs.np] = ncd->l2_p[lay];
      atm_airs.t[atm_airs.np] = ncd->l2_t[track][xtrack][lay];
      if ((++atm_airs.np) > NP)
        ERRMSG("Too many layers!");
    }

  /* Check number of levels... */
  if (atm_airs.np <= 0)
    return;

  /* Get height range of AIRS data... */
  for (ip = 0; ip < atm_airs.np; ip++) {
    zmax = GSL_MAX(zmax, atm_airs.z[ip]);
    zmin = GSL_MIN(zmin, atm_airs.z[ip]);
  }

  /* Merge AIRS data... */
  for (ip = 0; ip < atm->np; ip++) {

    /* Interpolate AIRS data... */
    intpol_atm(ctl, &atm_airs, atm->z[ip], &p, &t, q, k);

    /* Weighting factor... */
    w = 1;
    if (atm->z[ip] > zmax)
      w = GSL_MAX(1 - (atm->z[ip] - zmax) / 50, 0);
    if (atm->z[ip] < zmin)
      w = GSL_MAX(1 - (zmin - atm->z[ip]) / 50, 0);

    /* Merge... */
    atm->t[ip] = w * t + (1 - w) * atm->t[ip];
    atm->p[ip] = w * p + (1 - w) * atm->p[ip];
  }
}
```

Here is the call graph for this function:

5.63.1.6 void matrix_invert (gsl_matrix * *a*)

Definition at line 679 of file retrieval.c.

```
                    {

  size_t diag = 1, i, j, n;

  /* Get size... */
  n = a->size1;

  /* Check if matrix is diagonal... */
  for (i = 0; i < n && diag; i++)
    for (j = i + 1; j < n; j++)
      if (gsl_matrix_get(a, i, j) != 0) {
        diag = 0;
        break;
      }

  /* Quick inversion of diagonal matrix... */
  if (diag)
    for (i = 0; i < n; i++)
      gsl_matrix_set(a, i, i, 1 / gsl_matrix_get(a, i, i));

  /* Matrix inversion by means of Cholesky decomposition... */
  else {
    gsl_linalg_cholesky_decomp(a);
    gsl_linalg_cholesky_invert(a);
  }
}
```

5.63.1.7 void matrix_product (gsl_matrix * *a*, gsl_vector * *b*, int *transpose*, gsl_matrix * *c*)

Definition at line 709 of file retrieval.c.

```
                    {

  gsl_matrix *aux;

  size_t i, j, m, n;

  /* Set sizes... */
  m = a->size1;
  n = a->size2;

  /* Allocate... */
  aux = gsl_matrix_alloc(m, n);

  /* Compute A^T B A... */
  if (transpose == 1) {

    /* Compute B^1/2 A... */
    for (i = 0; i < m; i++)
      for (j = 0; j < n; j++)
        gsl_matrix_set(aux, i, j,
                       gsl_vector_get(b, i) * gsl_matrix_get(a, i, j));

    /* Compute A^T B A = (B^1/2 A)^T (B^1/2 A)... */
    gsl_blas_dgemm(CblasTrans, CblasNoTrans, 1.0, aux, aux, 0.0, c);
  }

  /* Compute A B A^T... */
  else if (transpose == 2) {

    /* Compute A B^1/2... */
    for (i = 0; i < m; i++)
      for (j = 0; j < n; j++)
        gsl_matrix_set(aux, i, j,
                       gsl_matrix_get(a, i, j) * gsl_vector_get(b, j));

    /* Compute A B A^T = (A B^1/2) (A B^1/2)^T... */
    gsl_blas_dgemm(CblasNoTrans, CblasTrans, 1.0, aux, aux, 0.0, c);
  }

  /* Free... */
  gsl_matrix_free(aux);
}
```

5.63.1.8 void optimal_estimation (ret_t ∗ *ret*, ctl_t ∗ *ctl*, obs_t ∗ *obs_meas*, obs_t ∗ *obs_i*, atm_t ∗ *atm_apr*, atm_t ∗ *atm_i*, double ∗ *chisq*)

Definition at line 758 of file retrieval.c.

```
                  {

  static int ipa[N], iqa[N];

  gsl_matrix *a, *cov, *k_i, *s_a_inv;
  gsl_vector *b, *dx, *dy, *sig_eps_inv, *sig_formod, *sig_noise,
    *x_a, *x_i, *x_step, *y_aux, *y_i, *y_m;

  double chisq_old, disq = 0, lmpar = 0.001;

  int ig, ip, it = 0, it2, iw;

  size_t i, m, n;

  /* ------------------------------------------------------------
     Initialize...
     ------------------------------------------------------------ */

  /* Get sizes... */
  m = obs2y(ctl, obs_meas, NULL, NULL, NULL);
  n = atm2x(ctl, atm_apr, NULL, iqa, ipa);
  if (m <= 0 || n <= 0) {
    *chisq = GSL_NAN;
    return;
  }

  /* Allocate... */
  a = gsl_matrix_alloc(n, n);
  cov = gsl_matrix_alloc(n, n);
  k_i = gsl_matrix_alloc(m, n);
  s_a_inv = gsl_matrix_alloc(n, n);

  b = gsl_vector_alloc(n);
  dx = gsl_vector_alloc(n);
  dy = gsl_vector_alloc(m);
  sig_eps_inv = gsl_vector_alloc(m);
  sig_formod = gsl_vector_alloc(m);
  sig_noise = gsl_vector_alloc(m);
  x_a = gsl_vector_alloc(n);
  x_i = gsl_vector_alloc(n);
  x_step = gsl_vector_alloc(n);
  y_aux = gsl_vector_alloc(m);
  y_i = gsl_vector_alloc(m);
  y_m = gsl_vector_alloc(m);

  /* Set initial state... */
  copy_atm(ctl, atm_i, atm_apr, 0);
  copy_obs(ctl, obs_i, obs_meas, 0);
  formod(ctl, atm_i, obs_i);

  /* Set state vectors and observation vectors... */
  atm2x(ctl, atm_apr, x_a, NULL, NULL);
  atm2x(ctl, atm_i, x_i, NULL, NULL);
  obs2y(ctl, obs_meas, y_m, NULL, NULL);
  obs2y(ctl, obs_i, y_i, NULL, NULL);

  /* Set inverse a priori covariance S_a^-1... */
  set_cov_apr(ret, ctl, atm_apr, iqa, ipa, s_a_inv);
  matrix_invert(s_a_inv);

  /* Get measurement errors... */
  set_cov_meas(ret, ctl, obs_meas, sig_noise, sig_formod, sig_eps_inv);

  /* Determine dx = x_i - x_a and dy = y - F(x_i) ... */
  gsl_vector_memcpy(dx, x_i);
  gsl_vector_sub(dx, x_a);
  gsl_vector_memcpy(dy, y_m);
  gsl_vector_sub(dy, y_i);

  /* Compute cost function... */
  *chisq = cost_function(dx, dy, s_a_inv, sig_eps_inv);

  /* Compute initial kernel... */
  kernel(ctl, atm_i, obs_i, k_i);

  /* ------------------------------------------------------------
     Levenberg-Marquardt minimization...
     ------------------------------------------------------------ */

```

```
  /* Outer loop... */
  for (it = 1; it <= ret->conv_itmax; it++) {

    /* Store current cost function value... */
    chisq_old = *chisq;

    /* Compute kernel matrix K_i... */
    if (it > 1 && it % ret->kernel_recomp == 0)
      kernel(ctl, atm_i, obs_i, k_i);

    /* Compute K_i^T * S_eps^{-1} * K_i ... */
    if (it == 1 || it % ret->kernel_recomp == 0)
      matrix_product(k_i, sig_eps_inv, 1, cov);

    /* Determine b = K_i^T * S_eps^{-1} * dy - S_a^{-1} * dx ... */
    for (i = 0; i < m; i++)
      gsl_vector_set(y_aux, i, gsl_vector_get(dy, i)
                     * gsl_pow_2(gsl_vector_get(sig_eps_inv, i)));
    gsl_blas_dgemv(CblasTrans, 1.0, k_i, y_aux, 0.0, b);
    gsl_blas_dgemv(CblasNoTrans, -1.0, s_a_inv, dx, 1.0, b);

    /* Inner loop... */
    for (it2 = 0; it2 < 20; it2++) {

      /* Compute A = (1 + lmpar) * S_a^{-1} + K_i^T * S_eps^{-1} * K_i ... */
      gsl_matrix_memcpy(a, s_a_inv);
      gsl_matrix_scale(a, 1 + lmpar);
      gsl_matrix_add(a, cov);

      /* Solve A * x_step = b by means of Cholesky decomposition... */
      gsl_linalg_cholesky_decomp(a);
      gsl_linalg_cholesky_solve(a, b, x_step);

      /* Update atmospheric state... */
      gsl_vector_add(x_i, x_step);
      copy_atm(ctl, atm_i, atm_apr, 0);
      copy_obs(ctl, obs_i, obs_meas, 0);
      x2atm(ctl, x_i, atm_i);

      /* Check atmospheric state... */
      for (ip = 0; ip < atm_i->np; ip++) {
        atm_i->p[ip] = GSL_MIN(GSL_MAX(atm_i->p[ip], 5e-7), 5e4);
        atm_i->t[ip] = GSL_MIN(GSL_MAX(atm_i->t[ip], 100), 400);
        for (ig = 0; ig < ctl->ng; ig++)
          atm_i->q[ig][ip] = GSL_MIN(GSL_MAX(atm_i->q[ig][ip], 0), 1);
        for (iw = 0; iw < ctl->nw; iw++)
          atm_i->k[iw][ip] = GSL_MAX(atm_i->k[iw][ip], 0);
      }

      /* Forward calculation... */
      formod(ctl, atm_i, obs_i);
      obs2y(ctl, obs_i, y_i, NULL, NULL);

      /* Determine dx = x_i - x_a and dy = y - F(x_i) ... */
      gsl_vector_memcpy(dx, x_i);
      gsl_vector_sub(dx, x_a);
      gsl_vector_memcpy(dy, y_m);
      gsl_vector_sub(dy, y_i);

      /* Compute cost function... */
      *chisq = cost_function(dx, dy, s_a_inv, sig_eps_inv);

      /* Modify Levenberg-Marquardt parameter... */
      if (*chisq > chisq_old) {
        lmpar *= 10;
        gsl_vector_sub(x_i, x_step);
      } else {
        lmpar /= 10;
        break;
      }
    }

    /* Get normalized step size in state space... */
    gsl_blas_ddot(x_step, b, &disq);
    disq /= (double) n;

    /* Convergence test... */
    if ((it == 1 || it % ret->kernel_recomp == 0) && disq < ret->
conv_dmin)
      break;
  }

  /* ------------------------------------------------------------
     Finalize...
     ------------------------------------------------------------ */

  gsl_matrix_free(a);
```

```
  gsl_matrix_free(cov);
  gsl_matrix_free(k_i);
  gsl_matrix_free(s_a_inv);

  gsl_vector_free(b);
  gsl_vector_free(dx);
  gsl_vector_free(dy);
  gsl_vector_free(sig_eps_inv);
  gsl_vector_free(sig_formod);
  gsl_vector_free(sig_noise);
  gsl_vector_free(x_a);
  gsl_vector_free(x_i);
  gsl_vector_free(x_step);
  gsl_vector_free(y_aux);
  gsl_vector_free(y_i);
  gsl_vector_free(y_m);
}
```

Here is the call graph for this function:

5.63.1.9 void read_nc (char * *filename*, ncd_t * *ncd*)

Definition at line 950 of file retrieval.c.

```
                {

  int varid;

  /* Open netCDF file... */
  printf("Read netCDF file: %s\n", filename);
  NC(nc_open(filename, NC_WRITE, &ncd->ncid));

  /* Read Level-1 data... */
  NC(nc_inq_varid(ncd->ncid, "l1_time", &varid));
  NC(nc_get_var_double(ncd->ncid, varid, ncd->l1_time[0]));
  NC(nc_inq_varid(ncd->ncid, "l1_lon", &varid));
  NC(nc_get_var_double(ncd->ncid, varid, ncd->l1_lon[0]));
  NC(nc_inq_varid(ncd->ncid, "l1_lat", &varid));
  NC(nc_get_var_double(ncd->ncid, varid, ncd->l1_lat[0]));
  NC(nc_inq_varid(ncd->ncid, "l1_sat_z", &varid));
  NC(nc_get_var_double(ncd->ncid, varid, ncd->l1_sat_z));
  NC(nc_inq_varid(ncd->ncid, "l1_sat_lon", &varid));
  NC(nc_get_var_double(ncd->ncid, varid, ncd->l1_sat_lon));
  NC(nc_inq_varid(ncd->ncid, "l1_sat_lat", &varid));
  NC(nc_get_var_double(ncd->ncid, varid, ncd->l1_sat_lat));
  NC(nc_inq_varid(ncd->ncid, "l1_nu", &varid));
  NC(nc_get_var_double(ncd->ncid, varid, ncd->l1_nu));
  NC(nc_inq_varid(ncd->ncid, "l1_rad", &varid));
  NC(nc_get_var_float(ncd->ncid, varid, ncd->l1_rad[0][0]));

  /* Read Level-2 data... */
  NC(nc_inq_varid(ncd->ncid, "l2_z", &varid));
  NC(nc_get_var_double(ncd->ncid, varid, ncd->l2_z[0][0]));
  NC(nc_inq_varid(ncd->ncid, "l2_press", &varid));
  NC(nc_get_var_double(ncd->ncid, varid, ncd->l2_p));
  NC(nc_inq_varid(ncd->ncid, "l2_temp", &varid));
  NC(nc_get_var_double(ncd->ncid, varid, ncd->l2_t[0][0]));
}
```

5.63.1.10 void read_ret_ctl (int *argc*, char * *argv[ ]*, ctl_t * *ctl*, ret_t * *ret*)

Definition at line 989 of file retrieval.c.

```
                {

  int id, ig, iw;

  /* Iteration control... */
  ret->kernel_recomp =
    (int) scan_ctl(argc, argv, "KERNEL_RECOMP", -1, "3", NULL);
  ret->conv_itmax = (int) scan_ctl(argc, argv, "CONV_ITMAX", -1, "30", NULL);
  ret->conv_dmin = scan_ctl(argc, argv, "CONV_DMIN", -1, "0.1", NULL);

  for (id = 0; id < ctl->nd; id++)
    ret->err_formod[id] = scan_ctl(argc, argv, "ERR_FORMOD", id, "0", NULL);
```

```

  for (id = 0; id < ctl->nd; id++)
    ret->err_noise[id] = scan_ctl(argc, argv, "ERR_NOISE", id, "0", NULL);

  ret->err_press = scan_ctl(argc, argv, "ERR_PRESS", -1, "0", NULL);
  ret->err_press_cz = scan_ctl(argc, argv, "ERR_PRESS_CZ", -1, "-999", NULL);
  ret->err_press_ch = scan_ctl(argc, argv, "ERR_PRESS_CH", -1, "-999", NULL);

  ret->err_temp = scan_ctl(argc, argv, "ERR_TEMP", -1, "0", NULL);
  ret->err_temp_cz = scan_ctl(argc, argv, "ERR_TEMP_CZ", -1, "-999", NULL);
  ret->err_temp_ch = scan_ctl(argc, argv, "ERR_TEMP_CH", -1, "-999", NULL);

  for (ig = 0; ig < ctl->ng; ig++) {
    ret->err_q[ig] = scan_ctl(argc, argv, "ERR_Q", ig, "0", NULL);
    ret->err_q_cz[ig] = scan_ctl(argc, argv, "ERR_Q_CZ", ig, "-999", NULL);
    ret->err_q_ch[ig] = scan_ctl(argc, argv, "ERR_Q_CH", ig, "-999", NULL);
  }

  for (iw = 0; iw < ctl->nw; iw++) {
    ret->err_k[iw] = scan_ctl(argc, argv, "ERR_K", iw, "0", NULL);
    ret->err_k_cz[iw] = scan_ctl(argc, argv, "ERR_K_CZ", iw, "-999", NULL);
    ret->err_k_ch[iw] = scan_ctl(argc, argv, "ERR_K_CH", iw, "-999", NULL);
  }
}
```

Here is the call graph for this function:

5.63.1.11 void set_cov_apr (ret_t * *ret*, ctl_t * *ctl*, atm_t * *atm*, int * *iqa*, int * *ipa*, gsl_matrix * *s_a*)

Definition at line 1032 of file retrieval.c.

```
                     {

  gsl_vector *x_a;

  double ch, cz, rho, x0[3], x1[3];

  int ig, iw;

  size_t i, j, n;

  /* Get sizes... */
  n = s_a->size1;

  /* Allocate... */
  x_a = gsl_vector_alloc(n);

  /* Get sigma vector... */
  atm2x(ctl, atm, x_a, NULL, NULL);
  for (i = 0; i < n; i++) {
    if (iqa[i] == IDXP)
      gsl_vector_set(x_a, i, ret->err_press / 100 * gsl_vector_get(x_a, i));
    if (iqa[i] == IDXT)
      gsl_vector_set(x_a, i, ret->err_temp);
    for (ig = 0; ig < ctl->ng; ig++)
      if (iqa[i] == IDXQ(ig))
        gsl_vector_set(x_a, i, ret->err_q[ig] / 100 * gsl_vector_get(x_a, i));
    for (iw = 0; iw < ctl->nw; iw++)
      if (iqa[i] == IDXK(iw))
        gsl_vector_set(x_a, i, ret->err_k[iw]);
  }

  /* Check standard deviations... */
  for (i = 0; i < n; i++)
    if (gsl_pow_2(gsl_vector_get(x_a, i)) <= 0)
      ERRMSG("Check a priori data (zero standard deviation)!");

  /* Initialize diagonal covariance... */
  gsl_matrix_set_zero(s_a);
  for (i = 0; i < n; i++)
    gsl_matrix_set(s_a, i, i, gsl_pow_2(gsl_vector_get(x_a, i)));

  /* Loop over matrix elements... */
  for (i = 0; i < n; i++)
    for (j = 0; j < n; j++)
      if (i != j && iqa[i] == iqa[j]) {

        /* Initialize... */
        cz = ch = 0;

```

```
      /* Set correlation lengths for pressure... */
      if (iqa[i] == IDXP) {
        cz = ret->err_press_cz;
        ch = ret->err_press_ch;
      }

      /* Set correlation lengths for temperature... */
      if (iqa[i] == IDXT) {
        cz = ret->err_temp_cz;
        ch = ret->err_temp_ch;
      }

      /* Set correlation lengths for volume mixing ratios... */
      for (ig = 0; ig < ctl->ng; ig++)
        if (iqa[i] == IDXQ(ig)) {
          cz = ret->err_q_cz[ig];
          ch = ret->err_q_ch[ig];
        }

      /* Set correlation lengths for extinction... */
      for (iw = 0; iw < ctl->nw; iw++)
        if (iqa[i] == IDXK(iw)) {
          cz = ret->err_k_cz[iw];
          ch = ret->err_k_ch[iw];
        }

      /* Compute correlations... */
      if (cz > 0 && ch > 0) {

        /* Get Cartesian coordinates... */
        geo2cart(0, atm->lon[ipa[i]], atm->lat[ipa[i]], x0);
        geo2cart(0, atm->lon[ipa[j]], atm->lat[ipa[j]], x1);

        /* Compute correlations... */
        rho =
          exp(-DIST(x0, x1) / ch -
              fabs(atm->z[ipa[i]] - atm->z[ipa[j]]) / cz);

        /* Set covariance... */
        gsl_matrix_set(s_a, i, j, gsl_vector_get(x_a, i)
                       * gsl_vector_get(x_a, j) * rho);
      }
    }

  /* Free... */
  gsl_vector_free(x_a);
}
```

Here is the call graph for this function:

5.63.1.12 void set_cov_meas (**ret_t** ∗ ***ret,*** **ctl_t** ∗ ***ctl,*** **obs_t** ∗ ***obs,*** gsl_vector ∗ ***sig_noise,*** gsl_vector ∗ ***sig_formod,*** gsl_vector ∗ ***sig_eps_inv***)

Definition at line 1137 of file retrieval.c.

```
                             {

  static obs_t obs_err;

  int id, ir;

  size_t i, m;

  /* Get size... */
  m = sig_eps_inv->size;

  /* Noise error (always considered in retrieval fit)... */
  copy_obs(ctl, &obs_err, obs, 1);
  for (ir = 0; ir < obs_err.nr; ir++)
    for (id = 0; id < ctl->nd; id++)
      obs_err.rad[id][ir]
        = (gsl_finite(obs->rad[id][ir]) ? ret->err_noise[id] : GSL_NAN);
  obs2y(ctl, &obs_err, sig_noise, NULL, NULL);

  /* Forward model error (always considered in retrieval fit)... */
  copy_obs(ctl, &obs_err, obs, 1);
  for (ir = 0; ir < obs_err.nr; ir++)
    for (id = 0; id < ctl->nd; id++)
      obs_err.rad[id][ir]
        = fabs(ret->err_formod[id] / 100 * obs->rad[id][ir]);
```

```
  obs2y(ctl, &obs_err, sig_formod, NULL, NULL);

  /* Total error... */
  for (i = 0; i < m; i++)
    gsl_vector_set(sig_eps_inv, i,
                   1 / sqrt(gsl_pow_2(gsl_vector_get(sig_noise, i))
                            + gsl_pow_2(gsl_vector_get(sig_formod, i))));

  /* Check standard deviations... */
  for (i = 0; i < m; i++)
    if (gsl_vector_get(sig_eps_inv, i) <= 0)
      ERRMSG("Check measurement errors (zero standard deviation)!");
}
```

Here is the call graph for this function:

5.63.1.13 double sza (double *sec,* double *lon,* double *lat*)

Calculate solar zenith angle.

Definition at line 1184 of file retrieval.c.

```
                {

  double D, dec, e, g, GMST, h, L, LST, q, ra;

  /* Number of days and fraction with respect to 2000-01-01T12:00Z... */
  D = sec / 86400 - 0.5;

  /* Geocentric apparent ecliptic longitude [rad]... */
  g = (357.529 + 0.98560028 * D) * M_PI / 180;
  q = 280.459 + 0.98564736 * D;
  L = (q + 1.915 * sin(g) + 0.020 * sin(2 * g)) * M_PI / 180;

  /* Mean obliquity of the ecliptic [rad]... */
  e = (23.439 - 0.00000036 * D) * M_PI / 180;

  /* Declination [rad]... */
  dec = asin(sin(e) * sin(L));

  /* Right ascension [rad]... */
  ra = atan2(cos(e) * sin(L), cos(L));

  /* Greenwich Mean Sidereal Time [h]... */
  GMST = 18.697374558 + 24.06570982441908 * D;

  /* Local Sidereal Time [h]... */
  LST = GMST + lon / 15;

  /* Hour angle [rad]... */
  h = LST / 12 * M_PI - ra;

  /* Convert latitude... */
  lat *= M_PI / 180;

  /* Return solar zenith angle [deg]... */
  return acos(sin(lat) * sin(dec) +
              cos(lat) * cos(dec) * cos(h)) * 180 / M_PI;
}
```

5.63.1.14 void write_nc (char ∗ *filename,* ncd_t ∗ *ncd*)

Definition at line 1227 of file retrieval.c.

```
                 {

  int dimid[10], p_id, t_id, z_id;

  /* Create netCDF file... */
  printf("Write netCDF file: %s\n", filename);

  /* Read existing dimensions... */
  NC(nc_inq_dimid(ncd->ncid, "L1_NTRACK", &dimid[0]));
  NC(nc_inq_dimid(ncd->ncid, "L1_NXTRACK", &dimid[1]));
```

```

  /* Set define mode... */
  NC(nc_redef(ncd->ncid));

  /* Set new dimensions... */
  if (nc_inq_dimid(ncd->ncid, "RET_NP", &dimid[2]) != NC_NOERR)
    NC(nc_def_dim(ncd->ncid, "RET_NP", (size_t) ncd->np, &dimid[2]));

  /* Set new variables... */
  add_var(ncd->ncid, "ret_z", "km", "altitude", NC_FLOAT, &dimid[2], &z_id,
          1);
  add_var(ncd->ncid, "ret_press", "hPa", "pressure", NC_FLOAT, dimid, &p_id,
          2);
  add_var(ncd->ncid, "ret_temp", "K", "temperature", NC_FLOAT, dimid, &t_id,
          3);

  /* Leave define mode... */
  NC(nc_enddef(ncd->ncid));

  /* Write data... */
  NC(nc_put_var_float(ncd->ncid, z_id, ncd->ret_z));
  NC(nc_put_var_float(ncd->ncid, p_id, ncd->ret_p));
  NC(nc_put_var_float(ncd->ncid, t_id, ncd->ret_t));

  /* Close netCDF file... */
  NC(nc_close(ncd->ncid));
}
```

Here is the call graph for this function:

5.63.1.15 int main (int *argc*, char ∗ *argv[ ]*)

Definition at line 263 of file retrieval.c.

```
                  {

  static ctl_t ctl;
  static atm_t atm_apr, atm_clim, atm_i;
  static obs_t obs_i, obs_meas;
  static ncd_t ncd;
  static ret_t ret;

  FILE *in;

  char filename[LEN];

  double chisq, chisq_min, chisq_max, chisq_mean, sx, sy, sza_thresh, z[NP];

  int channel[ND], i, id, ip, iz, m, nz, ntask = -1, rank, size,
    np0, np1, track, track0, track1, xtrack, xtrack0, xtrack1;

  /* ------------------------------------------------------------
     Init...
     ------------------------------------------------------------ */

  /* MPI... */
  MPI_Init(&argc, &argv);
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  MPI_Comm_size(MPI_COMM_WORLD, &size);

  /* Measure CPU time... */
  TIMER("total", 1);

  /* Check arguments... */
  if (argc < 3)
    ERRMSG("Give parameters: <ctl> <filelist>");

  /* Read control parameters... */
  read_ctl(argc, argv, &ctl);
  read_ret_ctl(argc, argv, &ctl, &ret);

  /* Read retrieval grid... */
  nz = (int) scan_ctl(argc, argv, "NZ", -1, "", NULL);
  if (nz > NP)
    ERRMSG("Too many altitudes!");
  for (iz = 0; iz < nz; iz++)
    z[iz] = scan_ctl(argc, argv, "Z", iz, "", NULL);

  /* Read track range... */
  track0 = (int) scan_ctl(argc, argv, "TRACK_MIN", -1, "0", NULL);
```

```
  track1 = (int) scan_ctl(argc, argv, "TRACK_MAX", -1, "134", NULL);

  /* Read xtrack range... */
  xtrack0 = (int) scan_ctl(argc, argv, "XTRACK_MIN", -1, "0", NULL);
  xtrack1 = (int) scan_ctl(argc, argv, "XTRACK_MAX", -1, "89", NULL);

  /* Read height range... */
  np0 = (int) scan_ctl(argc, argv, "NP_MIN", -1, "0", NULL);
  np1 = (int) scan_ctl(argc, argv, "NP_MAX", -1, "100", NULL);
  np1 = GSL_MIN(np1, nz - 1);

  /* Background smoothing... */
  sx = scan_ctl(argc, argv, "SX", -1, "8", NULL);
  sy = scan_ctl(argc, argv, "SY", -1, "2", NULL);

  /* SZA threshold... */
  sza_thresh = scan_ctl(argc, argv, "SZA", -1, "96", NULL);

  /* ------------------------------------------------------------
     Distribute granules...
     ------------------------------------------------------------ */

  /* Open filelist... */
  printf("Read filelist: %s\n", argv[2]);
  if (!(in = fopen(argv[2], "r")))
    ERRMSG("Cannot open filelist!");

  /* Loop over netCDF files... */
  while (fscanf(in, "%s", filename) != EOF) {

    /* Distribute files with MPI... */
    if ((++ntask) % size != rank)
      continue;

    /* Write info... */
    printf("Retrieve file %s on rank %d of %d (with %d threads)...\n",
           filename, rank + 1, size, omp_get_max_threads());

    /* ------------------------------------------------------------
       Initialize retrieval...
       ------------------------------------------------------------ */

    /* Read netCDF file... */
    read_nc(filename, &ncd);

    /* Identify radiance channels... */
    for (id = 0; id < ctl.nd; id++) {
      channel[id] = -999;
      for (i = 0; i < L1_NCHAN; i++)
        if (fabs(ctl.nu[id] - ncd.l1_nu[i]) < 0.1)
          channel[id] = i;
      if (channel[id] < 0)
        ERRMSG("Cannot identify radiance channel!");
    }

    /* Fill data gaps... */
    fill_gaps(ncd.l2_t, sx, sy);
    fill_gaps(ncd.l2_z, sx, sy);

    /* Set climatological data for center of granule... */
    atm_clim.np = nz;
    for (iz = 0; iz < nz; iz++)
      atm_clim.z[iz] = z[iz];
    climatology(&ctl, &atm_clim);

    /* ------------------------------------------------------------
       Retrieval...
       ------------------------------------------------------------ */

    /* Get chi^2 statistics... */
    chisq_min = 1e100;
    chisq_max = -1e100;
    chisq_mean = 0;
    m = 0;

    /* Loop over swaths... */
    for (track = track0; track <= track1; track++) {

      /* Measure CPU time... */
      TIMER("retrieval", 1);

      /* Loop over scan... */
      for (xtrack = xtrack0; xtrack <= xtrack1; xtrack++) {

        /* Store observation data... */
        obs_meas.nr = 1;
        obs_meas.time[0] = ncd.l1_time[track][xtrack];
```

```
        obs_meas.obsz[0] = ncd.l1_sat_z[track];
        obs_meas.obslon[0] = ncd.l1_sat_lon[track];
        obs_meas.obslat[0] = ncd.l1_sat_lat[track];
        obs_meas.vplon[0] = ncd.l1_lon[track][xtrack];
        obs_meas.vplat[0] = ncd.l1_lat[track][xtrack];
        for (id = 0; id < ctl.nd; id++)
          obs_meas.rad[id][0] = ncd.l1_rad[track][xtrack][channel[id]];

        /* Flag out 4 micron channels for daytime measurements... */
        if (sza(obs_meas.time[0], obs_meas.obslon[0], obs_meas.
      obslat[0])
            < sza_thresh)
          for (id = 0; id < ctl.nd; id++)
            if (ctl.nu[id] >= 2000)
              obs_meas.rad[id][0] = GSL_NAN;

        /* Prepare atmospheric data... */
        copy_atm(&ctl, &atm_apr, &atm_clim, 0);
        for (ip = 0; ip < atm_apr.np; ip++) {
          atm_apr.time[ip] = obs_meas.time[0];
          atm_apr.lon[ip] = obs_meas.vplon[0];
          atm_apr.lat[ip] = obs_meas.vplat[0];
        }

        /* Merge Level-2 data... */
        init_l2(&ncd, track, xtrack, &ctl, &atm_apr);

        /* Retrieval... */
        optimal_estimation(&ret, &ctl, &obs_meas, &obs_i,
                           &atm_apr, &atm_i, &chisq);

        /* Get chi^2 statistics... */
        if (gsl_finite(chisq)) {
          chisq_min = GSL_MIN(chisq_min, chisq);
          chisq_max = GSL_MAX(chisq_max, chisq);
          chisq_mean += chisq;
          m++;
        }

        /* Buffer results... */
        buffer_nc(&atm_i, chisq, &ncd, track, xtrack, np0, np1);
      }

      /* Measure CPU time... */
      TIMER("retrieval", 3);
    }

    /* ------------------------------------------------------------
       Finalize...
       ------------------------------------------------------------ */

    /* Write netCDF file... */
    write_nc(filename, &ncd);

    /* Write info... */
    printf("chi^2: min= %g / mean= %g / max= %g / m= %d\n",
           chisq_min, chisq_mean / m, chisq_max, m);
    printf("Retrieval finished on rank %d of %d!\n", rank, size);
  }

  /* Close file list... */
  fclose(in);

  /* Measure CPU time... */
  TIMER("total", 3);

  /* Report memory usage... */
  printf("MEMORY_ATM = %g MByte\n", 4. * sizeof(atm_t) / 1024. / 1024.);
  printf("MEMORY_CTL = %g MByte\n", 1. * sizeof(ctl_t) / 1024. / 1024.);
  printf("MEMORY_NCD = %g MByte\n", 1. * sizeof(ncd_t) / 1024. / 1024.);
  printf("MEMORY_OBS = %g MByte\n", 3. * sizeof(atm_t) / 1024. / 1024.);
  printf("MEMORY_RET = %g MByte\n", 1. * sizeof(ret_t) / 1024. / 1024.);
  printf("MEMORY_TBL = %g MByte\n", 1. * sizeof(tbl_t) / 1024. / 1024.);

  /* Report problem size... */
  printf("SIZE_TASKS = %d\n", size);
  printf("SIZE_THREADS = %d\n", omp_get_max_threads());

  /* MPI... */
  MPI_Finalize();

  return EXIT_SUCCESS;
}
```

Here is the call graph for this function:

5.64 retrieval.c

```
#include <mpi.h>
#include <omp.h>
#include <netcdf.h>
#include "jurassic.h"

/* ------------------------------------------------------------
   Macros...
   ------------------------------------------------------------ */

/* Execute netCDF library command and check result. */
#define NC(cmd) {                                      \
    if((cmd)!=NC_NOERR)                                \
      ERRMSG(nc_strerror(cmd));                        \
  }

/* ------------------------------------------------------------
   Dimensions...
   ------------------------------------------------------------ */

/* Number of AIRS radiance channels (don't change). */
#define L1_NCHAN 34

/* Along-track size of AIRS radiance granule (don't change). */
#define L1_NTRACK 135

/* Across-track size of AIRS radiance granule (don't change). */
#define L1_NXTRACK 90

/* Number of AIRS pressure layers (don't change). */
#define L2_NLAY 27

/* Along-track size of AIRS retrieval granule (don't change). */
#define L2_NTRACK 45

/* Across-track size of AIRS retrieval granule (don't change). */
#define L2_NXTRACK 30

/* ------------------------------------------------------------
   Structs...
   ------------------------------------------------------------ */

/* Buffer for netCDF data. */
typedef struct {

  /* NetCDF file ID. */
  int ncid;

  /* Number of retrieval altitudes. */
  int np;

  /* Time (seconds since 2000-01-01T00:00Z). */
  double l1_time[L1_NTRACK][L1_NXTRACK];

  /* Footprint longitude [deg]. */
  double l1_lon[L1_NTRACK][L1_NXTRACK];

  /* Footprint latitude [deg]. */
  double l1_lat[L1_NTRACK][L1_NXTRACK];

  /* Satellite altitude [km]. */
  double l1_sat_z[L1_NTRACK];

  /* Satellite longitude [deg]. */
  double l1_sat_lon[L1_NTRACK];

  /* Satellite latitude [deg]. */
  double l1_sat_lat[L1_NTRACK];

  /* Channel frequencies [cm^-1]. */
  double l1_nu[L1_NCHAN];

  /* Radiance [W/(m^2 sr cm^-1)]. */
  float l1_rad[L1_NTRACK][L1_NXTRACK][L1_NCHAN];

  /* Altitude [km]. */
  double l2_z[L2_NTRACK][L2_NXTRACK][L2_NLAY];

  /* Pressure [hPa]. */
  double l2_p[L2_NLAY];

  /* Temperature [K]. */
  double l2_t[L2_NTRACK][L2_NXTRACK][L2_NLAY];

  /* Altitude [km]. */
```

```
  float ret_z[NP];

  /* Pressure [hPa]. */
  float ret_p[L1_NTRACK * L1_NXTRACK];

  /* Temperature [K]. */
  float ret_t[L1_NTRACK * L1_NXTRACK * NP];

} ncd_t;

/* Retrieval control parameters. */
typedef struct {

  /* Recomputation of kernel matrix (number of iterations). */
  int kernel_recomp;

  /* Maximum number of iterations. */
  int conv_itmax;

  /* Minimum normalized step size in state space. */
  double conv_dmin;

  /* Forward model error [%]. */
  double err_formod[ND];

  /* Noise error [W/(m^2 sr cm^-1)]. */
  double err_noise[ND];

  /* Pressure error [%]. */
  double err_press;

  /* Vertical correlation length for pressure error [km]. */
  double err_press_cz;

  /* Horizontal correlation length for pressure error [km]. */
  double err_press_ch;

  /* Temperature error [K]. */
  double err_temp;

  /* Vertical correlation length for temperature error [km]. */
  double err_temp_cz;

  /* Horizontal correlation length for temperature error [km]. */
  double err_temp_ch;

  /* Volume mixing ratio error [%]. */
  double err_q[NG];

  /* Vertical correlation length for volume mixing ratio error [km]. */
  double err_q_cz[NG];

  /* Horizontal correlation length for volume mixing ratio error [km]. */
  double err_q_ch[NG];

  /* Extinction error [1/km]. */
  double err_k[NW];

  /* Vertical correlation length for extinction error [km]. */
  double err_k_cz[NW];

  /* Horizontal correlation length for extinction error [km]. */
  double err_k_ch[NW];

} ret_t;

/* ------------------------------------------------------------
   Functions...
   ------------------------------------------------------------ */

/* Create variable in netCDF file. */
void add_var(
  int ncid,
  const char *varname,
  const char *unit,
  const char *longname,
  int type,
  int dimid[],
  int *varid,
  int ndims);

/* Buffer netCDF data. */
void buffer_nc(
  atm_t * atm,
  double chisq,
  ncd_t * ncd,
  int track,
```

```
  int xtrack,
  int np0,
  int np1);

/* Compute cost function. */
double cost_function(
  gsl_vector * dx,
  gsl_vector * dy,
  gsl_matrix * s_a_inv,
  gsl_vector * sig_eps_inv);

/* Fill data gaps in L2 data. */
void fill_gaps(
  double x[L2_NTRACK][L2_NXTRACK][L2_NLAY],
  double cx,
  double cy);

/* Initialize with AIRS Level-2 data. */
void init_l2(
  ncd_t * ncd,
  int track,
  int xtrack,
  ctl_t * ctl,
  atm_t * atm);

/* Invert symmetric matrix. */
void matrix_invert(
  gsl_matrix * a);

/* Compute matrix product A^TBA or ABA^T for diagonal matrix B. */
void matrix_product(
  gsl_matrix * a,
  gsl_vector * b,
  int transpose,
  gsl_matrix * c);

/* Carry out optimal estimation retrieval. */
void optimal_estimation(
  ret_t * ret,
  ctl_t * ctl,
  obs_t * obs_meas,
  obs_t * obs_i,
  atm_t * atm_apr,
  atm_t * atm_i,
  double *chisq);

/* Read netCDF file. */
void read_nc(
  char *filename,
  ncd_t * ncd);

/* Read retrieval control parameters. */
void read_ret_ctl(
  int argc,
  char *argv[],
  ctl_t * ctl,
  ret_t * ret);

/* Set a priori covariance. */
void set_cov_apr(
  ret_t * ret,
  ctl_t * ctl,
  atm_t * atm,
  int *iqa,
  int *ipa,
  gsl_matrix * s_a);

/* Set measurement errors. */
void set_cov_meas(
  ret_t * ret,
  ctl_t * ctl,
  obs_t * obs,
  gsl_vector * sig_noise,
  gsl_vector * sig_formod,
  gsl_vector * sig_eps_inv);

/* Calculate solar zenith angle. */
double sza(
  double sec,
  double lon,
  double lat);

/* Write to netCDF file... */
void write_nc(
  char *filename,
  ncd_t * ncd);

```

```
/* ------------------------------------------------------------
   Main...
   ------------------------------------------------------------ */

int main(
  int argc,
  char *argv[]) {

  static ctl_t ctl;
  static atm_t atm_apr, atm_clim, atm_i;
  static obs_t obs_i, obs_meas;
  static ncd_t ncd;
  static ret_t ret;

  FILE *in;

  char filename[LEN];

  double chisq, chisq_min, chisq_max, chisq_mean, sx, sy, sza_thresh, z[NP];

  int channel[ND], i, id, ip, iz, m, nz, ntask = -1, rank, size,
    np0, np1, track, track0, track1, xtrack, xtrack0, xtrack1;

  /* ------------------------------------------------------------
     Init...
     ------------------------------------------------------------ */

  /* MPI... */
  MPI_Init(&argc, &argv);
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  MPI_Comm_size(MPI_COMM_WORLD, &size);

  /* Measure CPU time... */
  TIMER("total", 1);

  /* Check arguments... */
  if (argc < 3)
    ERRMSG("Give parameters: <ctl> <filelist>");

  /* Read control parameters... */
  read_ctl(argc, argv, &ctl);
  read_ret_ctl(argc, argv, &ctl, &ret);

  /* Read retrieval grid... */
  nz = (int) scan_ctl(argc, argv, "NZ", -1, "", NULL);
  if (nz > NP)
    ERRMSG("Too many altitudes!");
  for (iz = 0; iz < nz; iz++)
    z[iz] = scan_ctl(argc, argv, "Z", iz, "", NULL);

  /* Read track range... */
  track0 = (int) scan_ctl(argc, argv, "TRACK_MIN", -1, "0", NULL);
  track1 = (int) scan_ctl(argc, argv, "TRACK_MAX", -1, "134", NULL);

  /* Read xtrack range... */
  xtrack0 = (int) scan_ctl(argc, argv, "XTRACK_MIN", -1, "0", NULL);
  xtrack1 = (int) scan_ctl(argc, argv, "XTRACK_MAX", -1, "89", NULL);

  /* Read height range... */
  np0 = (int) scan_ctl(argc, argv, "NP_MIN", -1, "0", NULL);
  np1 = (int) scan_ctl(argc, argv, "NP_MAX", -1, "100", NULL);
  np1 = GSL_MIN(np1, nz - 1);

  /* Background smoothing... */
  sx = scan_ctl(argc, argv, "SX", -1, "8", NULL);
  sy = scan_ctl(argc, argv, "SY", -1, "2", NULL);

  /* SZA threshold... */
  sza_thresh = scan_ctl(argc, argv, "SZA", -1, "96", NULL);

  /* ------------------------------------------------------------
     Distribute granules...
     ------------------------------------------------------------ */

  /* Open filelist... */
  printf("Read filelist: %s\n", argv[2]);
  if (!(in = fopen(argv[2], "r")))
    ERRMSG("Cannot open filelist!");

  /* Loop over netCDF files... */
  while (fscanf(in, "%s", filename) != EOF) {

    /* Distribute files with MPI... */
    if ((++ntask) % size != rank)
      continue;

    /* Write info... */
```

```
    printf("Retrieve file %s on rank %d of %d (with %d threads)...\n",
           filename, rank + 1, size, omp_get_max_threads());

    /* ------------------------------------------------------------
       Initialize retrieval...
       ------------------------------------------------------------ */

    /* Read netCDF file... */
    read_nc(filename, &ncd);

    /* Identify radiance channels... */
    for (id = 0; id < ctl.nd; id++) {
      channel[id] = -999;
      for (i = 0; i < L1_NCHAN; i++)
        if (fabs(ctl.nu[id] - ncd.l1_nu[i]) < 0.1)
          channel[id] = i;
      if (channel[id] < 0)
        ERRMSG("Cannot identify radiance channel!");
    }

    /* Fill data gaps... */
    fill_gaps(ncd.l2_t, sx, sy);
    fill_gaps(ncd.l2_z, sx, sy);

    /* Set climatological data for center of granule... */
    atm_clim.np = nz;
    for (iz = 0; iz < nz; iz++)
      atm_clim.z[iz] = z[iz];
    climatology(&ctl, &atm_clim);

    /* ------------------------------------------------------------
       Retrieval...
       ------------------------------------------------------------ */

    /* Get chi^2 statistics... */
    chisq_min = 1e100;
    chisq_max = -1e100;
    chisq_mean = 0;
    m = 0;

    /* Loop over swaths... */
    for (track = track0; track <= track1; track++) {

      /* Measure CPU time... */
      TIMER("retrieval", 1);

      /* Loop over scan... */
      for (xtrack = xtrack0; xtrack <= xtrack1; xtrack++) {

        /* Store observation data... */
        obs_meas.nr = 1;
        obs_meas.time[0] = ncd.l1_time[track][xtrack];
        obs_meas.obsz[0] = ncd.l1_sat_z[track];
        obs_meas.obslon[0] = ncd.l1_sat_lon[track];
        obs_meas.obslat[0] = ncd.l1_sat_lat[track];
        obs_meas.vplon[0] = ncd.l1_lon[track][xtrack];
        obs_meas.vplat[0] = ncd.l1_lat[track][xtrack];
        for (id = 0; id < ctl.nd; id++)
          obs_meas.rad[id][0] = ncd.l1_rad[track][xtrack][channel[id]];

        /* Flag out 4 micron channels for daytime measurements... */
        if (sza(obs_meas.time[0], obs_meas.obslon[0], obs_meas.
  obslat[0])
            < sza_thresh)
          for (id = 0; id < ctl.nd; id++)
            if (ctl.nu[id] >= 2000)
              obs_meas.rad[id][0] = GSL_NAN;

        /* Prepare atmospheric data... */
        copy_atm(&ctl, &atm_apr, &atm_clim, 0);
        for (ip = 0; ip < atm_apr.np; ip++) {
          atm_apr.time[ip] = obs_meas.time[0];
          atm_apr.lon[ip] = obs_meas.vplon[0];
          atm_apr.lat[ip] = obs_meas.vplat[0];
        }

        /* Merge Level-2 data... */
        init_l2(&ncd, track, xtrack, &ctl, &atm_apr);

        /* Retrieval... */
        optimal_estimation(&ret, &ctl, &obs_meas, &obs_i,
                           &atm_apr, &atm_i, &chisq);

        /* Get chi^2 statistics... */
        if (gsl_finite(chisq)) {
          chisq_min = GSL_MIN(chisq_min, chisq);
          chisq_max = GSL_MAX(chisq_max, chisq);
```

```
          chisq_mean += chisq;
          m++;
        }

        /* Buffer results... */
        buffer_nc(&atm_i, chisq, &ncd, track, xtrack, np0, np1);
      }

      /* Measure CPU time... */
      TIMER("retrieval", 3);
    }

    /* ------------------------------------------------------------
       Finalize...
       ------------------------------------------------------------ */

    /* Write netCDF file... */
    write_nc(filename, &ncd);

    /* Write info... */
    printf("chi^2: min= %g / mean= %g / max= %g / m= %d\n",
           chisq_min, chisq_mean / m, chisq_max, m);
    printf("Retrieval finished on rank %d of %d!\n", rank, size);
  }

  /* Close file list... */
  fclose(in);

  /* Measure CPU time... */
  TIMER("total", 3);

  /* Report memory usage... */
  printf("MEMORY_ATM = %g MByte\n", 4. * sizeof(atm_t) / 1024. / 1024.);
  printf("MEMORY_CTL = %g MByte\n", 1. * sizeof(ctl_t) / 1024. / 1024.);
  printf("MEMORY_NCD = %g MByte\n", 1. * sizeof(ncd_t) / 1024. / 1024.);
  printf("MEMORY_OBS = %g MByte\n", 3. * sizeof(atm_t) / 1024. / 1024.);
  printf("MEMORY_RET = %g MByte\n", 1. * sizeof(ret_t) / 1024. / 1024.);
  printf("MEMORY_TBL = %g MByte\n", 1. * sizeof(tbl_t) / 1024. / 1024.);

  /* Report problem size... */
  printf("SIZE_TASKS = %d\n", size);
  printf("SIZE_THREADS = %d\n", omp_get_max_threads());

  /* MPI... */
  MPI_Finalize();

  return EXIT_SUCCESS;
}

/*****************************************************************************/

void add_var(
  int ncid,
  const char *varname,
  const char *unit,
  const char *longname,
  int type,
  int dimid[],
  int *varid,
  int ndims) {

  /* Check if variable exists... */
  if (nc_inq_varid(ncid, varname, varid) != NC_NOERR) {

    /* Define variable... */
    NC(nc_def_var(ncid, varname, type, ndims, dimid, varid));

    /* Set long name... */
    NC(nc_put_att_text
       (ncid, *varid, "long_name", strlen(longname), longname));

    /* Set units... */
    NC(nc_put_att_text(ncid, *varid, "units", strlen(unit), unit));
  }
}

/*****************************************************************************/

void buffer_nc(
  atm_t * atm,
  double chisq,
  ncd_t * ncd,
  int track,
  int xtrack,
  int np0,
  int np1) {

```

```c
  int ip;

  /* Set number of data points... */
  ncd->np = np1 - np0 + 1;

  /* Save retrieval data... */
  for (ip = np0; ip <= np1; ip++) {
    ncd->ret_z[ip - np0] = (float) atm->z[ip];
    ncd->ret_p[track * L1_NXTRACK + xtrack] = (float) atm->p[np0];
    ncd->ret_t[(track * L1_NXTRACK + xtrack) * ncd->np + ip - np0] =
      (gsl_finite(chisq) ? (float) atm->t[ip] : GSL_NAN);
  }
}

/*****************************************************************************/

double cost_function(
  gsl_vector * dx,
  gsl_vector * dy,
  gsl_matrix * s_a_inv,
  gsl_vector * sig_eps_inv) {

  gsl_vector *x_aux, *y_aux;

  double chisq_a, chisq_m = 0;

  size_t i, m, n;

  /* Get sizes... */
  m = dy->size;
  n = dx->size;

  /* Allocate... */
  x_aux = gsl_vector_alloc(n);
  y_aux = gsl_vector_alloc(m);

  /* Determine normalized cost function...
     (chi^2 = 1/m * [dy^T * S_eps^{-1} * dy + dx^T * S_a^{-1} * dx]) */
  for (i = 0; i < m; i++)
    chisq_m +=
      gsl_pow_2(gsl_vector_get(dy, i) * gsl_vector_get(sig_eps_inv, i));
  gsl_blas_dgemv(CblasNoTrans, 1.0, s_a_inv, dx, 0.0, x_aux);
  gsl_blas_ddot(dx, x_aux, &chisq_a);

  /* Free... */
  gsl_vector_free(x_aux);
  gsl_vector_free(y_aux);

  /* Return cost function value... */
  return (chisq_m + chisq_a) / (double) m;
}

/*****************************************************************************/

void fill_gaps(
  double x[L2_NTRACK][L2_NXTRACK][L2_NLAY],
  double cx,
  double cy) {

  double help[L2_NTRACK][L2_NXTRACK], w, wsum;

  int lay, track, track2, xtrack, xtrack2;

  /* Loop over layers... */
  for (lay = 0; lay < L2_NLAY; lay++) {

    /* Loop over grid points... */
    for (track = 0; track < L2_NTRACK; track++)
      for (xtrack = 0; xtrack < L2_NXTRACK; xtrack++) {

        /* Init... */
        help[track][xtrack] = 0;
        wsum = 0;

        /* Averrage data points... */
        for (track2 = 0; track2 < L2_NTRACK; track2++)
          for (xtrack2 = 0; xtrack2 < L2_NXTRACK; xtrack2++)
            if (gsl_finite(x[track2][xtrack2][lay])
                && x[track2][xtrack2][lay] > 0) {
              w = exp(-gsl_pow_2((xtrack - xtrack2) / cx)
                      - gsl_pow_2((track - track2) / cy));
              help[track][xtrack] += w * x[track2][xtrack2][lay];
              wsum += w;
            }

        /* Normalize... */
        if (wsum > 0)
```

```
        help[track][xtrack] /= wsum;
      else
        help[track][xtrack] = GSL_NAN;
    }

    /* Copy grid points... */
    for (track = 0; track < L2_NTRACK; track++)
      for (xtrack = 0; xtrack < L2_NXTRACK; xtrack++)
        x[track][xtrack][lay] = help[track][xtrack];
  }
}

/*****************************************************************************/

void init_l2(
  ncd_t * ncd,
  int track,
  int xtrack,
  ctl_t * ctl,
  atm_t * atm) {

  static atm_t atm_airs;

  double k[NW], p, q[NG], t, w, zmax = 0, zmin = 1000;

  int ip, lay;

  /* Reset track- and xtrack-index to match Level-2 data... */
  track /= 3;
  xtrack /= 3;

  /* Store AIRS data in atmospheric data struct... */
  atm_airs.np = 0;
  for (lay = 0; lay < L2_NLAY; lay++)
    if (gsl_finite(ncd->l2_z[track][xtrack][lay])) {
      atm_airs.z[atm_airs.np] = ncd->l2_z[track][xtrack][lay];
      atm_airs.p[atm_airs.np] = ncd->l2_p[lay];
      atm_airs.t[atm_airs.np] = ncd->l2_t[track][xtrack][lay];
      if ((++atm_airs.np) > NP)
        ERRMSG("Too many layers!");
    }

  /* Check number of levels... */
  if (atm_airs.np <= 0)
    return;

  /* Get height range of AIRS data... */
  for (ip = 0; ip < atm_airs.np; ip++) {
    zmax = GSL_MAX(zmax, atm_airs.z[ip]);
    zmin = GSL_MIN(zmin, atm_airs.z[ip]);
  }

  /* Merge AIRS data... */
  for (ip = 0; ip < atm->np; ip++) {

    /* Interpolate AIRS data... */
    intpol_atm(ctl, &atm_airs, atm->z[ip], &p, &t, q, k);

    /* Weighting factor... */
    w = 1;
    if (atm->z[ip] > zmax)
      w = GSL_MAX(1 - (atm->z[ip] - zmax) / 50, 0);
    if (atm->z[ip] < zmin)
      w = GSL_MAX(1 - (zmin - atm->z[ip]) / 50, 0);

    /* Merge... */
    atm->t[ip] = w * t + (1 - w) * atm->t[ip];
    atm->p[ip] = w * p + (1 - w) * atm->p[ip];
  }
}

/*****************************************************************************/

void matrix_invert(
  gsl_matrix * a) {

  size_t diag = 1, i, j, n;

  /* Get size... */
  n = a->size1;

  /* Check if matrix is diagonal... */
  for (i = 0; i < n && diag; i++)
    for (j = i + 1; j < n; j++)
      if (gsl_matrix_get(a, i, j) != 0) {
        diag = 0;
        break;
```

```
      }

  /* Quick inversion of diagonal matrix... */
  if (diag)
    for (i = 0; i < n; i++)
      gsl_matrix_set(a, i, i, 1 / gsl_matrix_get(a, i, i));

  /* Matrix inversion by means of Cholesky decomposition... */
  else {
    gsl_linalg_cholesky_decomp(a);
    gsl_linalg_cholesky_invert(a);
  }
}

/*****************************************************************************/

void matrix_product(
  gsl_matrix * a,
  gsl_vector * b,
  int transpose,
  gsl_matrix * c) {

  gsl_matrix *aux;

  size_t i, j, m, n;

  /* Set sizes... */
  m = a->size1;
  n = a->size2;

  /* Allocate... */
  aux = gsl_matrix_alloc(m, n);

  /* Compute A^T B A... */
  if (transpose == 1) {

    /* Compute B^1/2 A... */
    for (i = 0; i < m; i++)
      for (j = 0; j < n; j++)
        gsl_matrix_set(aux, i, j,
                       gsl_vector_get(b, i) * gsl_matrix_get(a, i, j));

    /* Compute A^T B A = (B^1/2 A)^T (B^1/2 A)... */
    gsl_blas_dgemm(CblasTrans, CblasNoTrans, 1.0, aux, aux, 0.0, c);
  }

  /* Compute A B A^T... */
  else if (transpose == 2) {

    /* Compute A B^1/2... */
    for (i = 0; i < m; i++)
      for (j = 0; j < n; j++)
        gsl_matrix_set(aux, i, j,
                       gsl_matrix_get(a, i, j) * gsl_vector_get(b, j));

    /* Compute A B A^T = (A B^1/2) (A B^1/2)^T... */
    gsl_blas_dgemm(CblasNoTrans, CblasTrans, 1.0, aux, aux, 0.0, c);
  }

  /* Free... */
  gsl_matrix_free(aux);
}

/*****************************************************************************/

void optimal_estimation(
  ret_t * ret,
  ctl_t * ctl,
  obs_t * obs_meas,
  obs_t * obs_i,
  atm_t * atm_apr,
  atm_t * atm_i,
  double *chisq) {

  static int ipa[N], iqa[N];

  gsl_matrix *a, *cov, *k_i, *s_a_inv;
  gsl_vector *b, *dx, *dy, *sig_eps_inv, *sig_formod, *sig_noise,
    *x_a, *x_i, *x_step, *y_aux, *y_i, *y_m;

  double chisq_old, disq = 0, lmpar = 0.001;

  int ig, ip, it = 0, it2, iw;

  size_t i, m, n;

  /* ------------------------------------------------------------
```

```
     Initialize...
     ------------------------------------------------------------ */

  /* Get sizes... */
  m = obs2y(ctl, obs_meas, NULL, NULL, NULL);
  n = atm2x(ctl, atm_apr, NULL, iqa, ipa);
  if (m <= 0 || n <= 0) {
    *chisq = GSL_NAN;
    return;
  }

  /* Allocate... */
  a = gsl_matrix_alloc(n, n);
  cov = gsl_matrix_alloc(n, n);
  k_i = gsl_matrix_alloc(m, n);
  s_a_inv = gsl_matrix_alloc(n, n);

  b = gsl_vector_alloc(n);
  dx = gsl_vector_alloc(n);
  dy = gsl_vector_alloc(m);
  sig_eps_inv = gsl_vector_alloc(m);
  sig_formod = gsl_vector_alloc(m);
  sig_noise = gsl_vector_alloc(m);
  x_a = gsl_vector_alloc(n);
  x_i = gsl_vector_alloc(n);
  x_step = gsl_vector_alloc(n);
  y_aux = gsl_vector_alloc(m);
  y_i = gsl_vector_alloc(m);
  y_m = gsl_vector_alloc(m);

  /* Set initial state... */
  copy_atm(ctl, atm_i, atm_apr, 0);
  copy_obs(ctl, obs_i, obs_meas, 0);
  formod(ctl, atm_i, obs_i);

  /* Set state vectors and observation vectors... */
  atm2x(ctl, atm_apr, x_a, NULL, NULL);
  atm2x(ctl, atm_i, x_i, NULL, NULL);
  obs2y(ctl, obs_meas, y_m, NULL, NULL);
  obs2y(ctl, obs_i, y_i, NULL, NULL);

  /* Set inverse a priori covariance S_a^-1... */
  set_cov_apr(ret, ctl, atm_apr, iqa, ipa, s_a_inv);
  matrix_invert(s_a_inv);

  /* Get measurement errors... */
  set_cov_meas(ret, ctl, obs_meas, sig_noise, sig_formod, sig_eps_inv);

  /* Determine dx = x_i - x_a and dy = y - F(x_i) ... */
  gsl_vector_memcpy(dx, x_i);
  gsl_vector_sub(dx, x_a);
  gsl_vector_memcpy(dy, y_m);
  gsl_vector_sub(dy, y_i);

  /* Compute cost function... */
  *chisq = cost_function(dx, dy, s_a_inv, sig_eps_inv);

  /* Compute initial kernel... */
  kernel(ctl, atm_i, obs_i, k_i);

  /* ------------------------------------------------------------
     Levenberg-Marquardt minimization...
     ------------------------------------------------------------ */

  /* Outer loop... */
  for (it = 1; it <= ret->conv_itmax; it++) {

    /* Store current cost function value... */
    chisq_old = *chisq;

    /* Compute kernel matrix K_i... */
    if (it > 1 && it % ret->kernel_recomp == 0)
      kernel(ctl, atm_i, obs_i, k_i);

    /* Compute K_i^T * S_eps^{-1} * K_i ... */
    if (it == 1 || it % ret->kernel_recomp == 0)
      matrix_product(k_i, sig_eps_inv, 1, cov);

    /* Determine b = K_i^T * S_eps^{-1} * dy - S_a^{-1} * dx ... */
    for (i = 0; i < m; i++)
      gsl_vector_set(y_aux, i, gsl_vector_get(dy, i)
                     * gsl_pow_2(gsl_vector_get(sig_eps_inv, i)));
    gsl_blas_dgemv(CblasTrans, 1.0, k_i, y_aux, 0.0, b);
    gsl_blas_dgemv(CblasNoTrans, -1.0, s_a_inv, dx, 1.0, b);

    /* Inner loop... */
    for (it2 = 0; it2 < 20; it2++) {
```

```
      /* Compute A = (1 + lmpar) * S_a^{-1} + K_i^T * S_eps^{-1} * K_i ... */
      gsl_matrix_memcpy(a, s_a_inv);
      gsl_matrix_scale(a, 1 + lmpar);
      gsl_matrix_add(a, cov);

      /* Solve A * x_step = b by means of Cholesky decomposition... */
      gsl_linalg_cholesky_decomp(a);
      gsl_linalg_cholesky_solve(a, b, x_step);

      /* Update atmospheric state... */
      gsl_vector_add(x_i, x_step);
      copy_atm(ctl, atm_i, atm_apr, 0);
      copy_obs(ctl, obs_i, obs_meas, 0);
      x2atm(ctl, x_i, atm_i);

      /* Check atmospheric state... */
      for (ip = 0; ip < atm_i->np; ip++) {
        atm_i->p[ip] = GSL_MIN(GSL_MAX(atm_i->p[ip], 5e-7), 5e4);
        atm_i->t[ip] = GSL_MIN(GSL_MAX(atm_i->t[ip], 100), 400);
        for (ig = 0; ig < ctl->ng; ig++)
          atm_i->q[ig][ip] = GSL_MIN(GSL_MAX(atm_i->q[ig][ip], 0), 1);
        for (iw = 0; iw < ctl->nw; iw++)
          atm_i->k[iw][ip] = GSL_MAX(atm_i->k[iw][ip], 0);
      }

      /* Forward calculation... */
      formod(ctl, atm_i, obs_i);
      obs2y(ctl, obs_i, y_i, NULL, NULL);

      /* Determine dx = x_i - x_a and dy = y - F(x_i) ... */
      gsl_vector_memcpy(dx, x_i);
      gsl_vector_sub(dx, x_a);
      gsl_vector_memcpy(dy, y_m);
      gsl_vector_sub(dy, y_i);

      /* Compute cost function... */
      *chisq = cost_function(dx, dy, s_a_inv, sig_eps_inv);

      /* Modify Levenberg-Marquardt parameter... */
      if (*chisq > chisq_old) {
        lmpar *= 10;
        gsl_vector_sub(x_i, x_step);
      } else {
        lmpar /= 10;
        break;
      }
    }

    /* Get normalized step size in state space... */
    gsl_blas_ddot(x_step, b, &disq);
    disq /= (double) n;

    /* Convergence test... */
    if ((it == 1 || it % ret->kernel_recomp == 0) && disq < ret->
conv_dmin)
      break;
  }

  /* ------------------------------------------------------------
     Finalize...
     ------------------------------------------------------------ */

  gsl_matrix_free(a);
  gsl_matrix_free(cov);
  gsl_matrix_free(k_i);
  gsl_matrix_free(s_a_inv);

  gsl_vector_free(b);
  gsl_vector_free(dx);
  gsl_vector_free(dy);
  gsl_vector_free(sig_eps_inv);
  gsl_vector_free(sig_formod);
  gsl_vector_free(sig_noise);
  gsl_vector_free(x_a);
  gsl_vector_free(x_i);
  gsl_vector_free(x_step);
  gsl_vector_free(y_aux);
  gsl_vector_free(y_i);
  gsl_vector_free(y_m);
}

/*****************************************************************************/

void read_nc(
  char *filename,
  ncd_t * ncd) {
```

```

  int varid;

  /* Open netCDF file... */
  printf("Read netCDF file: %s\n", filename);
  NC(nc_open(filename, NC_WRITE, &ncd->ncid));

  /* Read Level-1 data... */
  NC(nc_inq_varid(ncd->ncid, "l1_time", &varid));
  NC(nc_get_var_double(ncd->ncid, varid, ncd->l1_time[0]));
  NC(nc_inq_varid(ncd->ncid, "l1_lon", &varid));
  NC(nc_get_var_double(ncd->ncid, varid, ncd->l1_lon[0]));
  NC(nc_inq_varid(ncd->ncid, "l1_lat", &varid));
  NC(nc_get_var_double(ncd->ncid, varid, ncd->l1_lat[0]));
  NC(nc_inq_varid(ncd->ncid, "l1_sat_z", &varid));
  NC(nc_get_var_double(ncd->ncid, varid, ncd->l1_sat_z));
  NC(nc_inq_varid(ncd->ncid, "l1_sat_lon", &varid));
  NC(nc_get_var_double(ncd->ncid, varid, ncd->l1_sat_lon));
  NC(nc_inq_varid(ncd->ncid, "l1_sat_lat", &varid));
  NC(nc_get_var_double(ncd->ncid, varid, ncd->l1_sat_lat));
  NC(nc_inq_varid(ncd->ncid, "l1_nu", &varid));
  NC(nc_get_var_double(ncd->ncid, varid, ncd->l1_nu));
  NC(nc_inq_varid(ncd->ncid, "l1_rad", &varid));
  NC(nc_get_var_float(ncd->ncid, varid, ncd->l1_rad[0][0]));

  /* Read Level-2 data... */
  NC(nc_inq_varid(ncd->ncid, "l2_z", &varid));
  NC(nc_get_var_double(ncd->ncid, varid, ncd->l2_z[0][0]));
  NC(nc_inq_varid(ncd->ncid, "l2_press", &varid));
  NC(nc_get_var_double(ncd->ncid, varid, ncd->l2_p));
  NC(nc_inq_varid(ncd->ncid, "l2_temp", &varid));
  NC(nc_get_var_double(ncd->ncid, varid, ncd->l2_t[0][0]));
}

/*****************************************************************************/

void read_ret_ctl(
  int argc,
  char *argv[],
  ctl_t * ctl,
  ret_t * ret) {

  int id, ig, iw;

  /* Iteration control... */
  ret->kernel_recomp =
    (int) scan_ctl(argc, argv, "KERNEL_RECOMP", -1, "3", NULL);
  ret->conv_itmax = (int) scan_ctl(argc, argv, "CONV_ITMAX", -1, "30", NULL);
  ret->conv_dmin = scan_ctl(argc, argv, "CONV_DMIN", -1, "0.1", NULL);

  for (id = 0; id < ctl->nd; id++)
    ret->err_formod[id] = scan_ctl(argc, argv, "ERR_FORMOD", id, "0", NULL);

  for (id = 0; id < ctl->nd; id++)
    ret->err_noise[id] = scan_ctl(argc, argv, "ERR_NOISE", id, "0", NULL);

  ret->err_press = scan_ctl(argc, argv, "ERR_PRESS", -1, "0", NULL);
  ret->err_press_cz = scan_ctl(argc, argv, "ERR_PRESS_CZ", -1, "-999", NULL);
  ret->err_press_ch = scan_ctl(argc, argv, "ERR_PRESS_CH", -1, "-999", NULL);

  ret->err_temp = scan_ctl(argc, argv, "ERR_TEMP", -1, "0", NULL);
  ret->err_temp_cz = scan_ctl(argc, argv, "ERR_TEMP_CZ", -1, "-999", NULL);
  ret->err_temp_ch = scan_ctl(argc, argv, "ERR_TEMP_CH", -1, "-999", NULL);

  for (ig = 0; ig < ctl->ng; ig++) {
    ret->err_q[ig] = scan_ctl(argc, argv, "ERR_Q", ig, "0", NULL);
    ret->err_q_cz[ig] = scan_ctl(argc, argv, "ERR_Q_CZ", ig, "-999", NULL);
    ret->err_q_ch[ig] = scan_ctl(argc, argv, "ERR_Q_CH", ig, "-999", NULL);
  }

  for (iw = 0; iw < ctl->nw; iw++) {
    ret->err_k[iw] = scan_ctl(argc, argv, "ERR_K", iw, "0", NULL);
    ret->err_k_cz[iw] = scan_ctl(argc, argv, "ERR_K_CZ", iw, "-999", NULL);
    ret->err_k_ch[iw] = scan_ctl(argc, argv, "ERR_K_CH", iw, "-999", NULL);
  }
}

/*****************************************************************************/

void set_cov_apr(
  ret_t * ret,
  ctl_t * ctl,
  atm_t * atm,
  int *iqa,
  int *ipa,
  gsl_matrix * s_a) {

```

```
  gsl_vector *x_a;

  double ch, cz, rho, x0[3], x1[3];

  int ig, iw;

  size_t i, j, n;

  /* Get sizes... */
  n = s_a->size1;

  /* Allocate... */
  x_a = gsl_vector_alloc(n);

  /* Get sigma vector... */
  atm2x(ctl, atm, x_a, NULL, NULL);
  for (i = 0; i < n; i++) {
    if (iqa[i] == IDXP)
      gsl_vector_set(x_a, i, ret->err_press / 100 * gsl_vector_get(x_a, i));
    if (iqa[i] == IDXT)
      gsl_vector_set(x_a, i, ret->err_temp);
    for (ig = 0; ig < ctl->ng; ig++)
      if (iqa[i] == IDXQ(ig))
        gsl_vector_set(x_a, i, ret->err_q[ig] / 100 * gsl_vector_get(x_a, i));
    for (iw = 0; iw < ctl->nw; iw++)
      if (iqa[i] == IDXK(iw))
        gsl_vector_set(x_a, i, ret->err_k[iw]);
  }

  /* Check standard deviations... */
  for (i = 0; i < n; i++)
    if (gsl_pow_2(gsl_vector_get(x_a, i)) <= 0)
      ERRMSG("Check a priori data (zero standard deviation)!");

  /* Initialize diagonal covariance... */
  gsl_matrix_set_zero(s_a);
  for (i = 0; i < n; i++)
    gsl_matrix_set(s_a, i, i, gsl_pow_2(gsl_vector_get(x_a, i)));

  /* Loop over matrix elements... */
  for (i = 0; i < n; i++)
    for (j = 0; j < n; j++)
      if (i != j && iqa[i] == iqa[j]) {

        /* Initialize... */
        cz = ch = 0;

        /* Set correlation lengths for pressure... */
        if (iqa[i] == IDXP) {
          cz = ret->err_press_cz;
          ch = ret->err_press_ch;
        }

        /* Set correlation lengths for temperature... */
        if (iqa[i] == IDXT) {
          cz = ret->err_temp_cz;
          ch = ret->err_temp_ch;
        }

        /* Set correlation lengths for volume mixing ratios... */
        for (ig = 0; ig < ctl->ng; ig++)
          if (iqa[i] == IDXQ(ig)) {
            cz = ret->err_q_cz[ig];
            ch = ret->err_q_ch[ig];
          }

        /* Set correlation lengths for extinction... */
        for (iw = 0; iw < ctl->nw; iw++)
          if (iqa[i] == IDXK(iw)) {
            cz = ret->err_k_cz[iw];
            ch = ret->err_k_ch[iw];
          }

        /* Compute correlations... */
        if (cz > 0 && ch > 0) {

          /* Get Cartesian coordinates... */
          geo2cart(0, atm->lon[ipa[i]], atm->lat[ipa[i]], x0);
          geo2cart(0, atm->lon[ipa[j]], atm->lat[ipa[j]], x1);

          /* Compute correlations... */
          rho =
            exp(-DIST(x0, x1) / ch -
                fabs(atm->z[ipa[i]] - atm->z[ipa[j]]) / cz);

          /* Set covariance... */
          gsl_matrix_set(s_a, i, j, gsl_vector_get(x_a, i)
```

```
                              * gsl_vector_get(x_a, j) * rho);
        }
      }

  /* Free... */
  gsl_vector_free(x_a);
}

/*****************************************************************************/

void set_cov_meas(
  ret_t * ret,
  ctl_t * ctl,
  obs_t * obs,
  gsl_vector * sig_noise,
  gsl_vector * sig_formod,
  gsl_vector * sig_eps_inv) {

  static obs_t obs_err;

  int id, ir;

  size_t i, m;

  /* Get size... */
  m = sig_eps_inv->size;

  /* Noise error (always considered in retrieval fit)... */
  copy_obs(ctl, &obs_err, obs, 1);
  for (ir = 0; ir < obs_err.nr; ir++)
    for (id = 0; id < ctl->nd; id++)
      obs_err.rad[id][ir]
        = (gsl_finite(obs->rad[id][ir]) ? ret->err_noise[id] : GSL_NAN);
  obs2y(ctl, &obs_err, sig_noise, NULL, NULL);

  /* Forward model error (always considered in retrieval fit)... */
  copy_obs(ctl, &obs_err, obs, 1);
  for (ir = 0; ir < obs_err.nr; ir++)
    for (id = 0; id < ctl->nd; id++)
      obs_err.rad[id][ir]
        = fabs(ret->err_formod[id] / 100 * obs->rad[id][ir]);
  obs2y(ctl, &obs_err, sig_formod, NULL, NULL);

  /* Total error... */
  for (i = 0; i < m; i++)
    gsl_vector_set(sig_eps_inv, i,
                   1 / sqrt(gsl_pow_2(gsl_vector_get(sig_noise, i))
                            + gsl_pow_2(gsl_vector_get(sig_formod, i))));

  /* Check standard deviations... */
  for (i = 0; i < m; i++)
    if (gsl_vector_get(sig_eps_inv, i) <= 0)
      ERRMSG("Check measurement errors (zero standard deviation)!");
}

/*****************************************************************************/

double sza(
  double sec,
  double lon,
  double lat) {

  double D, dec, e, g, GMST, h, L, LST, q, ra;

  /* Number of days and fraction with respect to 2000-01-01T12:00Z... */
  D = sec / 86400 - 0.5;

  /* Geocentric apparent ecliptic longitude [rad]... */
  g = (357.529 + 0.98560028 * D) * M_PI / 180;
  q = 280.459 + 0.98564736 * D;
  L = (q + 1.915 * sin(g) + 0.020 * sin(2 * g)) * M_PI / 180;

  /* Mean obliquity of the ecliptic [rad]... */
  e = (23.439 - 0.00000036 * D) * M_PI / 180;

  /* Declination [rad]... */
  dec = asin(sin(e) * sin(L));

  /* Right ascension [rad]... */
  ra = atan2(cos(e) * sin(L), cos(L));

  /* Greenwich Mean Sidereal Time [h]... */
  GMST = 18.697374558 + 24.06570982441908 * D;

  /* Local Sidereal Time [h]... */
  LST = GMST + lon / 15;

```

```
  /* Hour angle [rad]... */
  h = LST / 12 * M_PI - ra;

  /* Convert latitude... */
  lat *= M_PI / 180;

  /* Return solar zenith angle [deg]... */
  return acos(sin(lat) * sin(dec) +
              cos(lat) * cos(dec) * cos(h)) * 180 / M_PI;
}

/*****************************************************************************/

void write_nc(
  char *filename,
  ncd_t * ncd) {

  int dimid[10], p_id, t_id, z_id;

  /* Create netCDF file... */
  printf("Write netCDF file: %s\n", filename);

  /* Read existing dimensions... */
  NC(nc_inq_dimid(ncd->ncid, "L1_NTRACK", &dimid[0]));
  NC(nc_inq_dimid(ncd->ncid, "L1_NXTRACK", &dimid[1]));

  /* Set define mode... */
  NC(nc_redef(ncd->ncid));

  /* Set new dimensions... */
  if (nc_inq_dimid(ncd->ncid, "RET_NP", &dimid[2]) != NC_NOERR)
    NC(nc_def_dim(ncd->ncid, "RET_NP", (size_t) ncd->np, &dimid[2]));

  /* Set new variables... */
  add_var(ncd->ncid, "ret_z", "km", "altitude", NC_FLOAT, &dimid[2], &z_id,
          1);
  add_var(ncd->ncid, "ret_press", "hPa", "pressure", NC_FLOAT, dimid, &p_id,
          2);
  add_var(ncd->ncid, "ret_temp", "K", "temperature", NC_FLOAT, dimid, &t_id,
          3);

  /* Leave define mode... */
  NC(nc_enddef(ncd->ncid));

  /* Write data... */
  NC(nc_put_var_float(ncd->ncid, z_id, ncd->ret_z));
  NC(nc_put_var_float(ncd->ncid, p_id, ncd->ret_p));
  NC(nc_put_var_float(ncd->ncid, t_id, ncd->ret_t));

  /* Close netCDF file... */
  NC(nc_close(ncd->ncid));
}
```

5.65 sampling.c File Reference

Functions

- int main (int argc, char *argv[ ])

5.65.1 Function Documentation

5.65.1.1 int main (int *argc*, char * *argv[ ]*)

Definition at line 3 of file sampling.c.

```
                  {

  static pert_t *pert;

  double d, dmin, dmax, dmu, x0[3], x1[3], x2[3];

  int i, itrack, ixtrack, n;

  /* Check arguments... */
```

```
  if (argc < 3)
    ERRMSG("Give parameters: <ctl> <pert.nc>");

  /* Allocate... */
  ALLOC(pert, pert_t, 1);

  /* Read perturbation data... */
  read_pert(argv[2], "4mu", pert);

  /* Init... */
  dmin = 1e100;
  dmax = -1e100;
  dmu = 0;
  n = 0;

  /* Get swath width... */
  for (itrack = 0; itrack < pert->ntrack; itrack++) {
    geo2cart(0, pert->lon[itrack][0], pert->lat[itrack][0], x0);
    geo2cart(0, pert->lon[itrack][pert->nxtrack - 1],
             pert->lat[itrack][pert->nxtrack - 1], x1);
    d = 2. * RE * asin(DIST(x0, x1) / (2. * RE));
    dmin = GSL_MIN(dmin, d);
    dmax = GSL_MAX(dmax, d);
    dmu += d;
    n++;
  }

  /* Write output... */
  printf("\nmean_swath_width=    %.1f km\n", dmu / n);
  printf("minimum_swath_width= %.1f km\n", dmin);
  printf("maximum_swath_width= %.1f km\n", dmax);

  /* Init... */
  dmin = 1e100;
  dmax = -1e100;
  dmu = 0;
  n = 0;

  /* Get across-track sampling distances... */
  for (itrack = 0; itrack < pert->ntrack; itrack++) {
    for (ixtrack = 0; ixtrack < pert->nxtrack - 1; ixtrack++) {
      geo2cart(0, pert->lon[itrack][ixtrack], pert->lat[itrack][ixtrack], x0);
      geo2cart(0, pert->lon[itrack][ixtrack + 1],
               pert->lat[itrack][ixtrack + 1], x1);
      d = 2. * RE * asin(DIST(x0, x1) / (2. * RE));
      dmin = GSL_MIN(dmin, d);
      dmax = GSL_MAX(dmax, d);
      dmu += d;
      n++;
    }
  }

  /* Write output... */
  printf("\nmean_across_track_sampling_distance=    %.1f km\n", dmu / n);
  printf("minimum_across_track_sampling_distance= %.1f km\n", dmin);
  printf("maximum_across_track_sampling_distance= %.1f km\n", dmax);

  /* Init... */
  dmin = 1e100;
  dmax = -1e100;
  dmu = 0;
  n = 0;

  /* Get along-track sampling distances... */
  for (itrack = 0; itrack < pert->ntrack - 1; itrack++) {
    for (ixtrack = 0; ixtrack < pert->nxtrack; ixtrack++) {
      geo2cart(0, pert->lon[itrack][ixtrack], pert->lat[itrack][ixtrack], x0);
      geo2cart(0, pert->lon[itrack + 1][ixtrack],
               pert->lat[itrack + 1][ixtrack], x1);
      d = 2. * RE * asin(DIST(x0, x1) / (2. * RE));
      dmin = GSL_MIN(dmin, d);
      dmax = GSL_MAX(dmax, d);
      dmu += d;
      n++;
    }
  }

  /* Write output... */
  printf("\nmean_along_track_sampling_distance=    %.1f km\n", dmu / n);
  printf("minimum_along_track_sampling_distance= %.1f km\n", dmin);
  printf("maximum_along_track_sampling_distance= %.1f km\n", dmax);

  /* Init... */
  dmin = 1e100;
  dmax = -1e100;
  dmu = 0;
  n = 0;
```

```
  /* Get angle between along-track and across-track direction... */
  for (itrack = 0; itrack < pert->ntrack - 1; itrack++) {
    geo2cart(0, pert->lon[itrack][pert->nxtrack / 2],
             pert->lat[itrack][pert->nxtrack / 2], x0);
    geo2cart(0, pert->lon[itrack][pert->nxtrack / 2 + 1],
             pert->lat[itrack][pert->nxtrack / 2 + 1], x1);
    geo2cart(0, pert->lon[itrack + 1][pert->nxtrack / 2],
             pert->lat[itrack + 1][pert->nxtrack / 2], x2);
    for (i = 0; i < 3; i++) {
      x1[i] -= x0[i];
      x2[i] -= x0[i];
    }
    d = acos(DOTP(x1, x2) / (NORM(x1) * NORM(x2))) * 180. / M_PI;
    dmin = GSL_MIN(dmin, d);
    dmax = GSL_MAX(dmax, d);
    dmu += d;
    n++;
  }

  /* Write output... */
  printf("\nmean_across_track_angle=    %.1f deg\n", dmu / n);
  printf("minimum_across_track_angle= %.1f deg\n", dmin);
  printf("maximum_across_track_angle= %.1f deg\n", dmax);

  /* Free... */
  free(pert);

  return EXIT_SUCCESS;
}
```

Here is the call graph for this function:

5.66 sampling.c

```
#include "libairs.h"

int main(
  int argc,
  char *argv[]) {

  static pert_t *pert;

  double d, dmin, dmax, dmu, x0[3], x1[3], x2[3];

  int i, itrack, ixtrack, n;

  /* Check arguments... */
  if (argc < 3)
    ERRMSG("Give parameters: <ctl> <pert.nc>");

  /* Allocate... */
  ALLOC(pert, pert_t, 1);

  /* Read perturbation data... */
  read_pert(argv[2], "4mu", pert);

  /* Init... */
  dmin = 1e100;
  dmax = -1e100;
  dmu = 0;
  n = 0;

  /* Get swath width... */
  for (itrack = 0; itrack < pert->ntrack; itrack++) {
    geo2cart(0, pert->lon[itrack][0], pert->lat[itrack][0], x0);
    geo2cart(0, pert->lon[itrack][pert->nxtrack - 1],
             pert->lat[itrack][pert->nxtrack - 1], x1);
    d = 2. * RE * asin(DIST(x0, x1) / (2. * RE));
    dmin = GSL_MIN(dmin, d);
    dmax = GSL_MAX(dmax, d);
    dmu += d;
    n++;
  }

  /* Write output... */
  printf("\nmean_swath_width=    %.1f km\n", dmu / n);
  printf("minimum_swath_width= %.1f km\n", dmin);
  printf("maximum_swath_width= %.1f km\n", dmax);

  /* Init... */
```

```
  dmin = 1e100;
  dmax = -1e100;
  dmu = 0;
  n = 0;

  /* Get across-track sampling distances... */
  for (itrack = 0; itrack < pert->ntrack; itrack++) {
    for (ixtrack = 0; ixtrack < pert->nxtrack - 1; ixtrack++) {
      geo2cart(0, pert->lon[itrack][ixtrack], pert->lat[itrack][ixtrack], x0);
      geo2cart(0, pert->lon[itrack][ixtrack + 1],
               pert->lat[itrack][ixtrack + 1], x1);
      d = 2. * RE * asin(DIST(x0, x1) / (2. * RE));
      dmin = GSL_MIN(dmin, d);
      dmax = GSL_MAX(dmax, d);
      dmu += d;
      n++;
    }
  }

  /* Write output... */
  printf("\nmean_across_track_sampling_distance=    %.1f km\n", dmu / n);
  printf("minimum_across_track_sampling_distance= %.1f km\n", dmin);
  printf("maximum_across_track_sampling_distance= %.1f km\n", dmax);

  /* Init... */
  dmin = 1e100;
  dmax = -1e100;
  dmu = 0;
  n = 0;

  /* Get along-track sampling distances... */
  for (itrack = 0; itrack < pert->ntrack - 1; itrack++) {
    for (ixtrack = 0; ixtrack < pert->nxtrack; ixtrack++) {
      geo2cart(0, pert->lon[itrack][ixtrack], pert->lat[itrack][ixtrack], x0);
      geo2cart(0, pert->lon[itrack + 1][ixtrack],
               pert->lat[itrack + 1][ixtrack], x1);
      d = 2. * RE * asin(DIST(x0, x1) / (2. * RE));
      dmin = GSL_MIN(dmin, d);
      dmax = GSL_MAX(dmax, d);
      dmu += d;
      n++;
    }
  }

  /* Write output... */
  printf("\nmean_along_track_sampling_distance=    %.1f km\n", dmu / n);
  printf("minimum_along_track_sampling_distance= %.1f km\n", dmin);
  printf("maximum_along_track_sampling_distance= %.1f km\n", dmax);

  /* Init... */
  dmin = 1e100;
  dmax = -1e100;
  dmu = 0;
  n = 0;

  /* Get angle between along-track and across-track direction... */
  for (itrack = 0; itrack < pert->ntrack - 1; itrack++) {
    geo2cart(0, pert->lon[itrack][pert->nxtrack / 2],
             pert->lat[itrack][pert->nxtrack / 2], x0);
    geo2cart(0, pert->lon[itrack][pert->nxtrack / 2 + 1],
             pert->lat[itrack][pert->nxtrack / 2 + 1], x1);
    geo2cart(0, pert->lon[itrack + 1][pert->nxtrack / 2],
             pert->lat[itrack + 1][pert->nxtrack / 2], x2);
    for (i = 0; i < 3; i++) {
      x1[i] -= x0[i];
      x2[i] -= x0[i];
    }
    d = acos(DOTP(x1, x2) / (NORM(x1) * NORM(x2))) * 180. / M_PI;
    dmin = GSL_MIN(dmin, d);
    dmax = GSL_MAX(dmax, d);
    dmu += d;
    n++;
  }

  /* Write output... */
  printf("\nmean_across_track_angle=    %.1f deg\n", dmu / n);
  printf("minimum_across_track_angle= %.1f deg\n", dmin);
  printf("maximum_across_track_angle= %.1f deg\n", dmax);

  /* Free... */
  free(pert);

  return EXIT_SUCCESS;
}
```

5.67 so2.c File Reference

Functions

- double get_noise (double bt, double dt250, double nu)
- void get_so2_column (double si, double dsi, double t, double lat, int set, double ∗scd, double ∗err)
- int main (int argc, char ∗argv[ ])

5.67.1 Function Documentation

5.67.1.1 double get_noise (double *bt,* double *dt250,* double *nu*)

Definition at line 204 of file so2.c.

```
                {
  double nesr;
  nesr = planck(250.0 + dt250, nu) - planck(250.0, nu);
  return brightness(planck(bt, nu) + nesr, nu) - bt;
}
```

Here is the call graph for this function:

5.67.1.2 void get_so2_column (double *si,* double *dsi,* double *t,* double *lat,* int *set,* double ∗ *scd,* double ∗ *err*)

Definition at line 218 of file so2.c.

```
                   {
  static double
    si_low[53] = { -0.377, -0.361, -0.342, -0.318, -0.291, -0.257, -0.217,
    -0.169, -0.112, -0.043, 0.039, 0.138, 0.256, 0.397,
    0.565, 0.766, 1.005, 1.29, 1.629, 2.03, 2.505, 3.065,
    3.725, 4.496, 5.398, 6.44, 7.644, 9.019, 10.574, 12.329,
    14.254, 16.378, 18.638, 21.039, 23.504, 25.989, 28.413,
    30.71, 32.786, 34.622, 36.118, 37.338, 38.216, 38.865,
    39.43, 39.886, 39.741, 39.86, 39.821, 39.832, 39.776,
    39.649, 39.659
  };
  static double
    scd_low[53] = { 0.205917, 0.232053, 0.263417, 0.301053, 0.346217,
    0.400413, 0.465446, 0.543491, 0.637141, 0.749524,
    0.884383, 1.04621, 1.24041, 1.47344, 1.75308,
    2.08865, 2.49133, 2.97455, 3.55441, 4.25026, 5.08524,
    6.08725, 7.28967, 8.73257, 10.464, 12.5418, 15.035,
    18.0271, 21.6174, 25.9259, 31.0959, 37.3, 44.745,
    53.6792, 64.4, 77.2647, 92.7026, 111.228, 133.458,
    160.135, 192.147, 230.562, 276.659, 331.977, 398.357,
    478.011, 1189.33, 1427.18, 2959.33, 3551.19, 5113.68,
    8836.36, 10603.6
  };
  static double
    si_high[60] = { -4.203, -4.199, -4.195, -4.19, -4.184, -4.177, -4.168,
    -4.158, -4.145, -4.13, -4.112, -4.091, -4.065, -4.034,
    -3.996, -3.952, -3.898, -3.834, -3.758, -3.666, -3.557,
    -3.426, -3.27, -3.084, -2.863, -2.599, -2.287, -1.918,
    -1.481, -0.966, -0.363, 0.343, 1.16, 2.107, 3.19, 4.421,
    5.811, 7.35, 9.049, 10.887, 12.852, 14.93, 17.065,
    19.269, 21.482, 23.711, 25.909, 28.064, 30.136, 32.094,
    33.877, 35.466, 36.773, 37.835, 38.59, 39.314, 39.866,
    39.826, 39.737, 39.791
  };
  static double
    scd_high[60] = { 0.205917, 0.232053, 0.263417, 0.301053, 0.346217,
```

```
    0.400413, 0.465446, 0.543491, 0.637141, 0.749524,
    0.884383, 1.04621, 1.24041, 1.47344, 1.75308, 2.08865,
    2.49133, 2.97455, 3.55441, 4.25026, 5.08524, 6.08725,
    7.28967, 8.73257, 10.464, 12.5418, 15.035, 18.0271,
    21.6174, 25.9259, 31.0959, 37.3, 44.745, 53.6792,
    64.4, 77.2647, 92.7026, 111.228, 133.458, 160.135,
    192.147, 230.562, 276.659, 331.977, 398.357, 478.011,
    573.599, 688.305, 825.952, 991.126, 1189.33, 1427.18,
    1712.61, 2055.12, 2466.13, 2959.33, 3551.19, 5113.68,
    7363.64, 10603.6
  };

  double *sia, *scda, scdm, scdp, s1, w_eqn, w_midl, w_psum, w_pwin;

  int i, *n, n_low = 53, n_high = 60;

  /* Set data set... */
  if (set == 1) {
    sia = &si_low[0];
    scda = &scd_low[0];
    n = &n_low;
  } else if (set == 2) {
    sia = &si_high[0];
    scda = &scd_high[0];
    n = &n_high;
  } else
    ERRMSG("Coding error!");

  /* Get weighting factors... */
  if (fabs(lat) <= 45) {
    w_eqn = LIN(0.0, 1.0, 45.0, 0.0, fabs(lat));
    w_midl = 1 - w_eqn;
    w_psum = 0;
    w_pwin = 0;
  } else {
    w_eqn = 0;
    w_midl = LIN(45.0, 1.0, 90.0, 0.0, fabs(lat));
    if (lat > 0) {
      w_psum = 0.5 * (1 - cos(2 * M_PI * t / (86400.0 * 365.25)));
      w_pwin = 1 - w_psum;
    } else {
      w_pwin = 0.5 * (1 - cos(2 * M_PI * t / (86400.0 * 365.25)));
      w_psum = 1 - w_pwin;
    }
    w_psum *= (1 - w_midl);
    w_pwin *= (1 - w_midl);
  }

  /* Get maximum SI... */
  s1 = (w_eqn * 63.75 + w_midl * 39.88 + w_psum * 10.73 + w_pwin * 45.58)
    / (w_eqn + w_midl + w_psum + w_pwin);

  /* Scale SI... */
  si *= sia[*n - 1] / s1;

  /* Estimate column density... */
  if (si <= sia[0]) {
    *scd = 0;
    *err = GSL_NAN;
  } else if (si >= sia[*n - 1]) {
    *scd = GSL_POSINF;
    *err = GSL_POSINF;
  } else {
    i = locate(sia, *n, si);
    *scd = LIN(sia[i], scda[i], sia[i + 1], scda[i + 1], si);

    i = locate(sia, *n, si + dsi + 1.0);
    scdp = LIN(sia[i], scda[i], sia[i + 1], scda[i + 1], si + dsi + 1.0);

    i = locate(sia, *n, si - dsi - 1.0);
    scdm = LIN(sia[i], scda[i], sia[i + 1], scda[i + 1], si - dsi - 1.0);

    *err = GSL_MAX(fabs(scdm - *scd), fabs(scdp - *scd));
  }
}
```

Here is the call graph for this function:

5.67.1.3 int main (int *argc*, char * *argv[ ]*)

Definition at line 27 of file so2.c.

```
                {

FILE *out;

static airs_rad_gran_t airs_rad_gran;

static double ci, ci_err, ci_nedt = 0.0783,
  ai, ai_err, ai_bt1, ai_bt1_nedt = 0.3155, ai_bt2, ai_bt2_nedt = 0.1177,
  si_low, si_low_err, si_low_bt1, si_low_bt1_nedt = 0.1064,
  si_low_bt2, si_low_bt2_nedt = 0.0909,
  si_high, si_high_err, si_high_bt1, si_high_bt1_nedt = 0.1064,
  si_high_bt2, si_high_bt2_nedt = 0.0786,
  scd_low, scd_low_err, scd_high, scd_high_err, scd, scd_err;

static int ichan, track, xtrack, iarg, ai_nu1 = 559, ai_nu2 = 901, ci_nu =
  1290, si_low_nu1 = 1591, si_low_nu2 = 1526, si_high_nu1 =
  1591, si_high_nu2 = 1550;

/* Check arguments... */
if (argc < 3)
  ERRMSG("Give parameters: <out.tab> <l1b_file1> [<l1b_file2> ...]");

/* Create file... */
printf("Write volcanic emission data: %s\n", argv[1]);
if (!(out = fopen(argv[1], "w")))
  ERRMSG("Cannot create file!");

/* Loop over HDF files... */
for (iarg = 2; iarg < argc; iarg++) {

  /* Read AIRS data... */
  printf("Read AIRS Level-1B data file: %s\n", argv[iarg]);
  airs_rad_rdr(argv[iarg], &airs_rad_gran);

  /* Write header... */
  if (iarg == 2) {
    fprintf(out,
            "# $1  = time [s]\n"
            "# $2  = footprint longitude [deg]\n"
            "# $3  = footprint latitude [deg]\n"
            "# $4  = cloud index, BT(%.2f/cm) [K]\n"
            "# $5  = cloud index error [K]\n"
            "# $6  = ash index, BT(%.2f/cm) - BT(%.2f/cm) [K]\n"
            "# $7  = ash index error [K]\n",
            airs_rad_gran.nominal_freq[ci_nu],
            airs_rad_gran.nominal_freq[ai_nu1],
            airs_rad_gran.nominal_freq[ai_nu2]);
    fprintf(out,
            "# $8  = SO2 index (low), BT(%.2f/cm) - BT(%.2f/cm) [K]\n"
            "# $9  = SO2 index (low) error [K]\n"
            "# $10 = SO2 index (high), BT(%.2f/cm) - BT(%.2f/cm) [K]\n"
            "# $11 = SO2 index (high) error [K]\n"
            "# $12 = SO2 column density estimate [DU]\n"
            "# $13 = SO2 column density error [DU]\n",
            airs_rad_gran.nominal_freq[si_low_nu1],
            airs_rad_gran.nominal_freq[si_low_nu2],
            airs_rad_gran.nominal_freq[si_high_nu1],
            airs_rad_gran.nominal_freq[si_high_nu2]);
  }

  /* Flag bad observations... */
  for (track = 0; track < AIRS_RAD_GEOTRACK; track++)
    for (xtrack = 0; xtrack < AIRS_RAD_GEOXTRACK; xtrack++)
      for (ichan = 0; ichan < AIRS_RAD_CHANNEL; ichan++)
        if ((airs_rad_gran.state[track][xtrack] != 0)
            || (airs_rad_gran.ExcludedChans[ichan] > 2)
            || (airs_rad_gran.CalChanSummary[ichan] & 8)
            || (airs_rad_gran.CalChanSummary[ichan] & (32 + 64))
            || (airs_rad_gran.CalFlag[track][ichan] & 16))
          airs_rad_gran.radiances[track][xtrack][ichan] = GSL_NAN;

  /* Loop over scans... */
  for (track = 0; track < AIRS_RAD_GEOTRACK; track++) {

    /* Write output... */
    fprintf(out, "\n");

    /* Loop over footprints... */
    for (xtrack = 0; xtrack < AIRS_RAD_GEOXTRACK; xtrack++) {

      /* cloud index... */
      ci = brightness(airs_rad_gran.radiances[track][xtrack][ci_nu] * 0.001,
                      airs_rad_gran.nominal_freq[ci_nu]);
      ci_err = get_noise(ci, ci_nedt, airs_rad_gran.nominal_freq[ci_nu]);

      /* ash index... */
      ai_bt1 =
```

```
          brightness(airs_rad_gran.radiances[track][xtrack][ai_nu1] * 0.001,
                     airs_rad_gran.nominal_freq[ai_nu1]);
        ai_bt2 =
          brightness(airs_rad_gran.radiances[track][xtrack][ai_nu2] * 0.001,
                     airs_rad_gran.nominal_freq[ai_nu2]);
        ai = ai_bt1 - ai_bt2;
        ai_err = sqrt(gsl_pow_2(get_noise(ai_bt1, ai_bt1_nedt,
                                 airs_rad_gran.nominal_freq[ai_nu1]))
                      + gsl_pow_2(get_noise(ai_bt2, ai_bt2_nedt,
                                  airs_rad_gran.nominal_freq
                                  [ai_nu2])));

        /* SO2 index (low concentrations)... */
        si_low_bt1 =
          brightness(airs_rad_gran.radiances[track][xtrack][si_low_nu1] *
                     0.001, airs_rad_gran.nominal_freq[si_low_nu1]);
        si_low_bt2 =
          brightness(airs_rad_gran.radiances[track][xtrack][si_low_nu2] *
                     0.001, airs_rad_gran.nominal_freq[si_low_nu2]);
        si_low = si_low_bt1 - si_low_bt2;
        si_low_err = sqrt(gsl_pow_2(get_noise(si_low_bt1, si_low_bt1_nedt,
                                    airs_rad_gran.nominal_freq
                                    [si_low_nu1]))
                          +
                          gsl_pow_2(get_noise
                                    (si_low_bt2, si_low_bt2_nedt,
                                     airs_rad_gran.nominal_freq
                                     [si_low_nu2])));

        /* SO2 index (high concentrations)... */
        si_high_bt1 =
          brightness(airs_rad_gran.radiances[track][xtrack][si_high_nu1] *
                     0.001, airs_rad_gran.nominal_freq[si_high_nu1]);
        si_high_bt2 =
          brightness(airs_rad_gran.radiances[track][xtrack][si_high_nu2] *
                     0.001, airs_rad_gran.nominal_freq[si_high_nu2]);
        si_high = si_high_bt1 - si_high_bt2;
        si_high_err = sqrt(gsl_pow_2(get_noise(si_high_bt1, si_high_bt1_nedt,
                                     airs_rad_gran.nominal_freq
                                     [si_high_nu1]))
                           +
                           gsl_pow_2(get_noise
                                     (si_high_bt2, si_high_bt2_nedt,
                                      airs_rad_gran.nominal_freq
                                      [si_high_nu2])));

        /* SO2 column density (low concentrations)... */
        get_so2_column(si_low, si_low_err,
                       airs_rad_gran.Time[track][xtrack] - 220838400,
                       airs_rad_gran.Latitude[track][xtrack],
                       1, &scd_low, &scd_low_err);

        /* SO2 column density (high concentrations)... */
        get_so2_column(si_high, si_high_err,
                       airs_rad_gran.Time[track][xtrack] - 220838400,
                       airs_rad_gran.Latitude[track][xtrack],
                       2, &scd_high, &scd_high_err);

        /* Get optimal estimate... */
        scd =
          (scd_low * gsl_pow_2(scd_high_err) +
           scd_high * gsl_pow_2(scd_low_err))
          / (gsl_pow_2(scd_low_err) + gsl_pow_2(scd_high_err));
        scd_err =
          1 / sqrt(1 / gsl_pow_2(scd_low_err) + 1 / gsl_pow_2(scd_high_err));

        /* Write output... */
        fprintf(out,
                "%.2f %.4f %.4f %.2f %.2f %.2f %.2f "
                "%.2f %.2f %.2f %.2f %.1f %.1f\n",
                airs_rad_gran.Time[track][xtrack] - 220838400,
                airs_rad_gran.Longitude[track][xtrack],
                airs_rad_gran.Latitude[track][xtrack],
                ci, ci_err, GSL_MAX(ai, 0.0), ai_err,
                GSL_MAX(si_low, 0.0), si_low_err,
                GSL_MAX(si_high, 0.0), si_high_err, scd, scd_err);
      }
    }
  }

  /* Close file... */
  fclose(out);

  return EXIT_SUCCESS;
}
```

Here is the call graph for this function:

5.68 so2.c

```
#include "libairs.h"

/* ------------------------------------------------------------
   Functions...
   ------------------------------------------------------------ */

/* Estimate noise. */
double get_noise(
  double bt,
  double dt250,
  double nu);

/* Estimate SO2 column density. */
void get_so2_column(
  double si,
  double dsi,
  double t,
  double lat,
  int set,
  double *scd,
  double *err);

/* ------------------------------------------------------------
   Main...
   ------------------------------------------------------------ */

int main(
  int argc,
  char *argv[]) {

  FILE *out;

  static airs_rad_gran_t airs_rad_gran;

  static double ci, ci_err, ci_nedt = 0.0783,
    ai, ai_err, ai_bt1, ai_bt1_nedt = 0.3155, ai_bt2, ai_bt2_nedt = 0.1177,
    si_low, si_low_err, si_low_bt1, si_low_bt1_nedt = 0.1064,
    si_low_bt2, si_low_bt2_nedt = 0.0909,
    si_high, si_high_err, si_high_bt1, si_high_bt1_nedt = 0.1064,
    si_high_bt2, si_high_bt2_nedt = 0.0786,
    scd_low, scd_low_err, scd_high, scd_high_err, scd, scd_err;

  static int ichan, track, xtrack, iarg, ai_nu1 = 559, ai_nu2 = 901, ci_nu =
    1290, si_low_nu1 = 1591, si_low_nu2 = 1526, si_high_nu1 =
    1591, si_high_nu2 = 1550;

  /* Check arguments... */
  if (argc < 3)
    ERRMSG("Give parameters: <out.tab> <l1b_file1> [<l1b_file2> ...]");

  /* Create file... */
  printf("Write volcanic emission data: %s\n", argv[1]);
  if (!(out = fopen(argv[1], "w")))
    ERRMSG("Cannot create file!");

  /* Loop over HDF files... */
  for (iarg = 2; iarg < argc; iarg++) {

    /* Read AIRS data... */
    printf("Read AIRS Level-1B data file: %s\n", argv[iarg]);
    airs_rad_rdr(argv[iarg], &airs_rad_gran);

    /* Write header... */
    if (iarg == 2) {
      fprintf(out,
              "# $1  = time [s]\n"
              "# $2  = footprint longitude [deg]\n"
              "# $3  = footprint latitude [deg]\n"
              "# $4  = cloud index, BT(%.2f/cm) [K]\n"
              "# $5  = cloud index error [K]\n"
              "# $6  = ash index, BT(%.2f/cm) - BT(%.2f/cm) [K]\n"
              "# $7  = ash index error [K]\n",
              airs_rad_gran.nominal_freq[ci_nu],
              airs_rad_gran.nominal_freq[ai_nu1],
              airs_rad_gran.nominal_freq[ai_nu2]);
      fprintf(out,
              "# $8  = SO2 index (low), BT(%.2f/cm) - BT(%.2f/cm) [K]\n"
              "# $9  = SO2 index (low) error [K]\n"
              "# $10 = SO2 index (high), BT(%.2f/cm) - BT(%.2f/cm) [K]\n"
```

```
            "# $11 = SO2 index (high) error [K]\n"
            "# $12 = SO2 column density estimate [DU]\n"
            "# $13 = SO2 column density error [DU]\n",
            airs_rad_gran.nominal_freq[si_low_nu1],
            airs_rad_gran.nominal_freq[si_low_nu2],
            airs_rad_gran.nominal_freq[si_high_nu1],
            airs_rad_gran.nominal_freq[si_high_nu2]);
  }

  /* Flag bad observations... */
  for (track = 0; track < AIRS_RAD_GEOTRACK; track++)
    for (xtrack = 0; xtrack < AIRS_RAD_GEOXTRACK; xtrack++)
      for (ichan = 0; ichan < AIRS_RAD_CHANNEL; ichan++)
        if ((airs_rad_gran.state[track][xtrack] != 0)
            || (airs_rad_gran.ExcludedChans[ichan] > 2)
            || (airs_rad_gran.CalChanSummary[ichan] & 8)
            || (airs_rad_gran.CalChanSummary[ichan] & (32 + 64))
            || (airs_rad_gran.CalFlag[track][ichan] & 16))
          airs_rad_gran.radiances[track][xtrack][ichan] = GSL_NAN;

  /* Loop over scans... */
  for (track = 0; track < AIRS_RAD_GEOTRACK; track++) {

    /* Write output... */
    fprintf(out, "\n");

    /* Loop over footprints... */
    for (xtrack = 0; xtrack < AIRS_RAD_GEOXTRACK; xtrack++) {

      /* cloud index... */
      ci = brightness(airs_rad_gran.radiances[track][xtrack][ci_nu] * 0.001,
                      airs_rad_gran.nominal_freq[ci_nu]);
      ci_err = get_noise(ci, ci_nedt, airs_rad_gran.nominal_freq[ci_nu]);

      /* ash index... */
      ai_bt1 =
        brightness(airs_rad_gran.radiances[track][xtrack][ai_nu1] * 0.001,
                   airs_rad_gran.nominal_freq[ai_nu1]);
      ai_bt2 =
        brightness(airs_rad_gran.radiances[track][xtrack][ai_nu2] * 0.001,
                   airs_rad_gran.nominal_freq[ai_nu2]);
      ai = ai_bt1 - ai_bt2;
      ai_err = sqrt(gsl_pow_2(get_noise(ai_bt1, ai_bt1_nedt,
                                         airs_rad_gran.nominal_freq[ai_nu1]))
                    + gsl_pow_2(get_noise(ai_bt2, ai_bt2_nedt,
                                           airs_rad_gran.nominal_freq
                                           [ai_nu2])));

      /* SO2 index (low concentrations)... */
      si_low_bt1 =
        brightness(airs_rad_gran.radiances[track][xtrack][si_low_nu1] *
                   0.001, airs_rad_gran.nominal_freq[si_low_nu1]);
      si_low_bt2 =
        brightness(airs_rad_gran.radiances[track][xtrack][si_low_nu2] *
                   0.001, airs_rad_gran.nominal_freq[si_low_nu2]);
      si_low = si_low_bt1 - si_low_bt2;
      si_low_err = sqrt(gsl_pow_2(get_noise(si_low_bt1, si_low_bt1_nedt,
                                             airs_rad_gran.nominal_freq
                                             [si_low_nu1]))
                        +
                        gsl_pow_2(get_noise
                                  (si_low_bt2, si_low_bt2_nedt,
                                   airs_rad_gran.nominal_freq
                                   [si_low_nu2])));

      /* SO2 index (high concentrations)... */
      si_high_bt1 =
        brightness(airs_rad_gran.radiances[track][xtrack][si_high_nu1] *
                   0.001, airs_rad_gran.nominal_freq[si_high_nu1]);
      si_high_bt2 =
        brightness(airs_rad_gran.radiances[track][xtrack][si_high_nu2] *
                   0.001, airs_rad_gran.nominal_freq[si_high_nu2]);
      si_high = si_high_bt1 - si_high_bt2;
      si_high_err = sqrt(gsl_pow_2(get_noise(si_high_bt1, si_high_bt1_nedt,
                                              airs_rad_gran.nominal_freq
                                              [si_high_nu1]))
                         +
                         gsl_pow_2(get_noise
                                   (si_high_bt2, si_high_bt2_nedt,
                                    airs_rad_gran.nominal_freq
                                    [si_high_nu2])));

      /* SO2 column density (low concentrations)... */
      get_so2_column(si_low, si_low_err,
                     airs_rad_gran.Time[track][xtrack] - 220838400,
                     airs_rad_gran.Latitude[track][xtrack],
                     1, &scd_low, &scd_low_err);
```

```
        /* SO2 column density (high concentrations)... */
        get_so2_column(si_high, si_high_err,
                       airs_rad_gran.Time[track][xtrack] - 220838400,
                       airs_rad_gran.Latitude[track][xtrack],
                       2, &scd_high, &scd_high_err);

        /* Get optimal estimate... */
        scd =
          (scd_low * gsl_pow_2(scd_high_err) +
           scd_high * gsl_pow_2(scd_low_err))
          / (gsl_pow_2(scd_low_err) + gsl_pow_2(scd_high_err));
        scd_err =
          1 / sqrt(1 / gsl_pow_2(scd_low_err) + 1 / gsl_pow_2(scd_high_err));

        /* Write output... */
        fprintf(out,
                "%.2f %.4f %.4f %.2f %.2f %.2f %.2f "
                "%.2f %.2f %.2f %.2f %.1f %.1f\n",
                airs_rad_gran.Time[track][xtrack] - 220838400,
                airs_rad_gran.Longitude[track][xtrack],
                airs_rad_gran.Latitude[track][xtrack],
                ci, ci_err, GSL_MAX(ai, 0.0), ai_err,
                GSL_MAX(si_low, 0.0), si_low_err,
                GSL_MAX(si_high, 0.0), si_high_err, scd, scd_err);
      }
    }
  }

  /* Close file... */
  fclose(out);

  return EXIT_SUCCESS;
}

/*****************************************************************************/

double get_noise(
  double bt,
  double dt250,
  double nu) {

  double nesr;

  nesr = planck(250.0 + dt250, nu) - planck(250.0, nu);

  return brightness(planck(bt, nu) + nesr, nu) - bt;
}

/*****************************************************************************/

void get_so2_column(
  double si,
  double dsi,
  double t,
  double lat,
  int set,
  double *scd,
  double *err) {

  static double
    si_low[53] = { -0.377, -0.361, -0.342, -0.318, -0.291, -0.257, -0.217,
    -0.169, -0.112, -0.043, 0.039, 0.138, 0.256, 0.397,
    0.565, 0.766, 1.005, 1.29, 1.629, 2.03, 2.505, 3.065,
    3.725, 4.496, 5.398, 6.44, 7.644, 9.019, 10.574, 12.329,
    14.254, 16.378, 18.638, 21.039, 23.504, 25.989, 28.413,
    30.71, 32.786, 34.622, 36.118, 37.338, 38.216, 38.865,
    39.43, 39.886, 39.741, 39.86, 39.821, 39.832, 39.776,
    39.649, 39.659
  };

  static double
    scd_low[53] = { 0.205917, 0.232053, 0.263417, 0.301053, 0.346217,
    0.400413, 0.465446, 0.543491, 0.637141, 0.749524,
    0.884383, 1.04621, 1.24041, 1.47344, 1.75308,
    2.08865, 2.49133, 2.97455, 3.55441, 4.25026, 5.08524,
    6.08725, 7.28967, 8.73257, 10.464, 12.5418, 15.035,
    18.0271, 21.6174, 25.9259, 31.0959, 37.3, 44.745,
    53.6792, 64.4, 77.2647, 92.7026, 111.228, 133.458,
    160.135, 192.147, 230.562, 276.659, 331.977, 398.357,
    478.011, 1189.33, 1427.18, 2959.33, 3551.19, 5113.68,
    8836.36, 10603.6
  };

  static double
    si_high[60] = { -4.203, -4.199, -4.195, -4.19, -4.184, -4.177, -4.168,
    -4.158, -4.145, -4.13, -4.112, -4.091, -4.065, -4.034,
```

```
    -3.996, -3.952, -3.898, -3.834, -3.758, -3.666, -3.557,
    -3.426, -3.27, -3.084, -2.863, -2.599, -2.287, -1.918,
    -1.481, -0.966, -0.363, 0.343, 1.16, 2.107, 3.19, 4.421,
    5.811, 7.35, 9.049, 10.887, 12.852, 14.93, 17.065,
    19.269, 21.482, 23.711, 25.909, 28.064, 30.136, 32.094,
    33.877, 35.466, 36.773, 37.835, 38.59, 39.314, 39.866,
    39.826, 39.737, 39.791
  };

  static double
    scd_high[60] = { 0.205917, 0.232053, 0.263417, 0.301053, 0.346217,
    0.400413, 0.465446, 0.543491, 0.637141, 0.749524,
    0.884383, 1.04621, 1.24041, 1.47344, 1.75308, 2.08865,
    2.49133, 2.97455, 3.55441, 4.25026, 5.08524, 6.08725,
    7.28967, 8.73257, 10.464, 12.5418, 15.035, 18.0271,
    21.6174, 25.9259, 31.0959, 37.3, 44.745, 53.6792,
    64.4, 77.2647, 92.7026, 111.228, 133.458, 160.135,
    192.147, 230.562, 276.659, 331.977, 398.357, 478.011,
    573.599, 688.305, 825.952, 991.126, 1189.33, 1427.18,
    1712.61, 2055.12, 2466.13, 2959.33, 3551.19, 5113.68,
    7363.64, 10603.6
  };

  double *sia, *scda, scdm, scdp, s1, w_eqn, w_midl, w_psum, w_pwin;

  int i, *n, n_low = 53, n_high = 60;

  /* Set data set... */
  if (set == 1) {
    sia = &si_low[0];
    scda = &scd_low[0];
    n = &n_low;
  } else if (set == 2) {
    sia = &si_high[0];
    scda = &scd_high[0];
    n = &n_high;
  } else
    ERRMSG("Coding error!");

  /* Get weighting factors... */
  if (fabs(lat) <= 45) {
    w_eqn = LIN(0.0, 1.0, 45.0, 0.0, fabs(lat));
    w_midl = 1 - w_eqn;
    w_psum = 0;
    w_pwin = 0;
  } else {
    w_eqn = 0;
    w_midl = LIN(45.0, 1.0, 90.0, 0.0, fabs(lat));
    if (lat > 0) {
      w_psum = 0.5 * (1 - cos(2 * M_PI * t / (86400.0 * 365.25)));
      w_pwin = 1 - w_psum;
    } else {
      w_pwin = 0.5 * (1 - cos(2 * M_PI * t / (86400.0 * 365.25)));
      w_psum = 1 - w_pwin;
    }
    w_psum *= (1 - w_midl);
    w_pwin *= (1 - w_midl);
  }

  /* Get maximum SI... */
  s1 = (w_eqn * 63.75 + w_midl * 39.88 + w_psum * 10.73 + w_pwin * 45.58)
    / (w_eqn + w_midl + w_psum + w_pwin);

  /* Scale SI... */
  si *= sia[*n - 1] / s1;

  /* Estimate column density... */
  if (si <= sia[0]) {
    *scd = 0;
    *err = GSL_NAN;
  } else if (si >= sia[*n - 1]) {
    *scd = GSL_POSINF;
    *err = GSL_POSINF;
  } else {
    i = locate(sia, *n, si);
    *scd = LIN(sia[i], scda[i], sia[i + 1], scda[i + 1], si);

    i = locate(sia, *n, si + dsi + 1.0);
    scdp = LIN(sia[i], scda[i], sia[i + 1], scda[i + 1], si + dsi + 1.0);

    i = locate(sia, *n, si - dsi - 1.0);
    scdm = LIN(sia[i], scda[i], sia[i + 1], scda[i + 1], si - dsi - 1.0);

    *err = GSL_MAX(fabs(scdm - *scd), fabs(scdp - *scd));
  }
}
```

5.69 spec2tab.c File Reference

Functions

- int main (int argc, char ∗argv[ ])

5.69.1 Function Documentation

5.69.1.1 int main (int *argc,* char ∗ *argv[ ]*)

Definition at line 3 of file spec2tab.c.

```
           {

  static airs_rad_gran_t airs_rad_gran;

  FILE *out;

  double dmin = 1e100, x0[3], x1[3];

  int ichan, track = -1, track2, xtrack = -1, xtrack2;

  /* Check arguments... */
  if (argc != 6)
    ERRMSG("Give parameters: <airs_l1b_file> "
           "[index <track> <xtrack> | geo <lon> <lat>] <spec.tab>");

  /* Read AIRS data... */
  printf("Read AIRS Level-1B data file: %s\n", argv[1]);
  airs_rad_rdr(argv[1], &airs_rad_gran);

  /* Get indices... */
  if (argv[2][0] == 'i') {
    track = atoi(argv[3]);
    xtrack = atoi(argv[4]);
  }

  /* Find nearest footprint... */
  else {
    geo2cart(0, atof(argv[3]), atof(argv[4]), x0);
    for (track2 = 0; track2 < AIRS_RAD_GEOTRACK; track2++)
      for (xtrack2 = 0; xtrack2 < AIRS_RAD_GEOXTRACK; xtrack2++) {
        geo2cart(0, airs_rad_gran.Longitude[track2][xtrack2],
                 airs_rad_gran.Latitude[track2][xtrack2], x1);
        if (DIST2(x0, x1) < dmin) {
          dmin = DIST2(x0, x1);
          track = track2;
          xtrack = xtrack2;
        }
      }
    if (dmin > 2500)
      ERRMSG("Geolocation not covered by granule!");
  }

  /* Check indices... */
  if (track < 0 || track >= AIRS_RAD_GEOTRACK)
    ERRMSG("Along-track index out of range!");
  if (xtrack < 0 || xtrack >= AIRS_RAD_GEOXTRACK)
    ERRMSG("Across-track index out of range!");

  /* Flag bad observations... */
  for (ichan = 0; ichan < AIRS_RAD_CHANNEL; ichan++)
    if ((airs_rad_gran.state[track][xtrack] != 0)
        || (airs_rad_gran.ExcludedChans[ichan] > 2)
        || (airs_rad_gran.CalChanSummary[ichan] & 8)
        || (airs_rad_gran.CalChanSummary[ichan] & (32 + 64))
        || (airs_rad_gran.CalFlag[track][ichan] & 16))
      airs_rad_gran.radiances[track][xtrack][ichan]
        = (float) sqrt(-1.0);

  /* Create file... */
  printf("Write spectrum: %s\n", argv[5]);
  if (!(out = fopen(argv[5], "w")))
    ERRMSG("Cannot create file!");

  /* Write header... */
  fprintf(out,
```

```
          "# $1 = time (seconds since 01-JAN-2000, 00:00 UTC)\n"
          "# $2 = satellite longitude [deg]\n"
          "# $3 = satellite latitude [deg]\n"
          "# $4 = footprint longitude [deg]\n"
          "# $5 = footprint latitude [deg]\n"
          "# $6 = wavenumber [cm^-1]\n"
          "# $7 = brightness temperature [K]\n"
          "# $8 = radiance [W/(m^2 sr cm^-1)]\n\n");

  /* Write data... */
  for (ichan = 0; ichan < AIRS_RAD_CHANNEL; ichan++) {
    if (ichan > 0)
      if (fabs(airs_rad_gran.nominal_freq[ichan]
               - airs_rad_gran.nominal_freq[ichan - 1]) > 1.2)
        fprintf(out, "\n");
    fprintf(out, "%.2f %g %g %g %g %g %g %g\n",
            airs_rad_gran.Time[track][xtrack] - 220838400,
            airs_rad_gran.sat_lon[track],
            airs_rad_gran.sat_lat[track],
            airs_rad_gran.Longitude[track][xtrack],
            airs_rad_gran.Latitude[track][xtrack],
            airs_rad_gran.nominal_freq[ichan],
            brightness(airs_rad_gran.radiances[track][xtrack][ichan] * 1e-3,
                       airs_rad_gran.nominal_freq[ichan]),
            airs_rad_gran.radiances[track][xtrack][ichan] * 1e-3);
  }

  /* Close file... */
  fclose(out);

  return EXIT_SUCCESS;
}
```

Here is the call graph for this function:

5.70 spec2tab.c

```
#include "libairs.h"

int main(
  int argc,
  char *argv[]) {

  static airs_rad_gran_t airs_rad_gran;

  FILE *out;

  double dmin = 1e100, x0[3], x1[3];

  int ichan, track = -1, track2, xtrack = -1, xtrack2;

  /* Check arguments... */
  if (argc != 6)
    ERRMSG("Give parameters: <airs_l1b_file> "
           "[index <track> <xtrack> | geo <lon> <lat>] <spec.tab>");

  /* Read AIRS data... */
  printf("Read AIRS Level-1B data file: %s\n", argv[1]);
  airs_rad_rdr(argv[1], &airs_rad_gran);

  /* Get indices... */
  if (argv[2][0] == 'i') {
    track = atoi(argv[3]);
    xtrack = atoi(argv[4]);
  }

  /* Find nearest footprint... */
  else {
    geo2cart(0, atof(argv[3]), atof(argv[4]), x0);
    for (track2 = 0; track2 < AIRS_RAD_GEOTRACK; track2++)
      for (xtrack2 = 0; xtrack2 < AIRS_RAD_GEOXTRACK; xtrack2++) {
        geo2cart(0, airs_rad_gran.Longitude[track2][xtrack2],
                 airs_rad_gran.Latitude[track2][xtrack2], x1);
        if (DIST2(x0, x1) < dmin) {
          dmin = DIST2(x0, x1);
          track = track2;
          xtrack = xtrack2;
        }
      }
    if (dmin > 2500)
      ERRMSG("Geolocation not covered by granule!");
```

```
  }

  /* Check indices... */
  if (track < 0 || track >= AIRS_RAD_GEOTRACK)
    ERRMSG("Along-track index out of range!");
  if (xtrack < 0 || xtrack >= AIRS_RAD_GEOXTRACK)
    ERRMSG("Across-track index out of range!");

  /* Flag bad observations... */
  for (ichan = 0; ichan < AIRS_RAD_CHANNEL; ichan++)
    if ((airs_rad_gran.state[track][xtrack] != 0)
        || (airs_rad_gran.ExcludedChans[ichan] > 2)
        || (airs_rad_gran.CalChanSummary[ichan] & 8)
        || (airs_rad_gran.CalChanSummary[ichan] & (32 + 64))
        || (airs_rad_gran.CalFlag[track][ichan] & 16))
      airs_rad_gran.radiances[track][xtrack][ichan]
        = (float) sqrt(-1.0);

  /* Create file... */
  printf("Write spectrum: %s\n", argv[5]);
  if (!(out = fopen(argv[5], "w")))
    ERRMSG("Cannot create file!");

  /* Write header... */
  fprintf(out,
          "# $1 = time (seconds since 01-JAN-2000, 00:00 UTC)\n"
          "# $2 = satellite longitude [deg]\n"
          "# $3 = satellite latitude [deg]\n"
          "# $4 = footprint longitude [deg]\n"
          "# $5 = footprint latitude [deg]\n"
          "# $6 = wavenumber [cm^-1]\n"
          "# $7 = brightness temperature [K]\n"
          "# $8 = radiance [W/(m^2 sr cm^-1)]\n\n");

  /* Write data... */
  for (ichan = 0; ichan < AIRS_RAD_CHANNEL; ichan++) {
    if (ichan > 0)
      if (fabs(airs_rad_gran.nominal_freq[ichan]
               - airs_rad_gran.nominal_freq[ichan - 1]) > 1.2)
        fprintf(out, "\n");
    fprintf(out, "%.2f %g %g %g %g %g %g %g\n",
            airs_rad_gran.Time[track][xtrack] - 220838400,
            airs_rad_gran.sat_lon[track],
            airs_rad_gran.sat_lat[track],
            airs_rad_gran.Longitude[track][xtrack],
            airs_rad_gran.Latitude[track][xtrack],
            airs_rad_gran.nominal_freq[ichan],
            brightness(airs_rad_gran.radiances[track][xtrack][ichan] * 1e-3,
                       airs_rad_gran.nominal_freq[ichan]),
            airs_rad_gran.radiances[track][xtrack][ichan] * 1e-3);
  }

  /* Close file... */
  fclose(out);

  return EXIT_SUCCESS;
}
```

5.71 statistics.c File Reference

Functions

- int main (int argc, char ∗argv[ ])

5.71.1 Function Documentation

5.71.1.1 int main (int *argc,* char ∗ *argv[ ]*)

Definition at line 14 of file statistics.c.

```
              {

FILE *in;

static double x[NMAX], y[NMAX], w[2 * NMAX];

static size_t n, stride;

/* Write info... */
if (argc != 3)
  ERRMSG("Give parameters: <xy.tab> <stride>");

/* Get stride... */
stride = (size_t) atoi(argv[2]);

/* Read data... */
if (!(in = fopen(argv[1], "r")))
  ERRMSG("Cannot open file!");
while (fscanf(in, "%lg %lg", &x[n], &y[n]) == 2)
  if (gsl_finite(x[n]) && gsl_finite(y[n]))
    if (++n >= NMAX)
      ERRMSG("Too many data points!");
fclose(in);

/* Get statistics... */
printf("                    n= %lu\n\n", n);
printf("        xy_covariance= %g\n",
       gsl_stats_covariance(x, stride, y, stride, n));
printf("       xy_correlation= %g\n",
       gsl_stats_correlation(x, stride, y, stride, n));
printf("          xy_spearman= %g\n",
       gsl_stats_spearman(x, stride, y, stride, n, w));
printf("      x_lag1_autocorr= %g\n",
       gsl_stats_lag1_autocorrelation(x, stride, n));
printf("      y_lag1_autocorr= %g\n\n",
       gsl_stats_lag1_autocorrelation(y, stride, n));

/* Sort data... */
gsl_sort(x, stride, n);
gsl_sort(y, stride, n);

/* Get statistics... */
printf("               x_mean= %g\n", gsl_stats_mean(x, stride, n));
printf("              x_sigma= %g\n", gsl_stats_sd(x, stride, n));
printf("           x_skewness= %g\n", gsl_stats_skew(x, stride, n));
printf("           x_kurtosis= %g\n\n", gsl_stats_kurtosis(x, stride, n));

printf("            x_mininum= %g\n", gsl_stats_min(x, stride, n));
printf("       x_10%%_quantile= %g\n",
       gsl_stats_quantile_from_sorted_data(x, stride, n, 0.1));
printf("       x_25%%_quantile= %g\n",
       gsl_stats_quantile_from_sorted_data(x, stride, n, 0.25));
printf("       x_50%%_quantile= %g\n",
       gsl_stats_quantile_from_sorted_data(x, stride, n, 0.5));
printf("       x_75%%_quantile= %g\n",
       gsl_stats_quantile_from_sorted_data(x, stride, n, 0.75));
printf("       x_90%%_quantile= %g\n",
       gsl_stats_quantile_from_sorted_data(x, stride, n, 0.9));
printf("            x_maximum= %g\n\n", gsl_stats_max(x, stride, n));

printf("        x_absdev_mean= %g\n", gsl_stats_absdev(x, stride, n));
printf("      x_absdev_median= %g\n",
       gsl_stats_absdev_m(x, stride, n,
                          gsl_stats_quantile_from_sorted_data(x, stride, n,
                                                              0.5)));
printf("        x_absdev_zero= %g\n",
       gsl_stats_absdev_m(x, stride, n, 0.0));
printf("x_interquartile_range= %g\n\n",
       gsl_stats_quantile_from_sorted_data(x, stride, n, 0.75)
       - gsl_stats_quantile_from_sorted_data(x, stride, n, 0.25));

printf("               y_mean= %g\n", gsl_stats_mean(y, stride, n));
printf("              y_sigma= %g\n", gsl_stats_sd(y, stride, n));
printf("           y_skewness= %g\n", gsl_stats_skew(y, stride, n));
printf("           y_kurtosis= %g\n\n", gsl_stats_kurtosis(y, stride, n));

printf("            y_mininum= %g\n", gsl_stats_min(y, stride, n));
printf("       y_10%%_quantile= %g\n",
       gsl_stats_quantile_from_sorted_data(y, stride, n, 0.1));
printf("       y_25%%_quantile= %g\n",
       gsl_stats_quantile_from_sorted_data(y, stride, n, 0.25));
printf("       y_50%%_quantile= %g\n",
       gsl_stats_quantile_from_sorted_data(y, stride, n, 0.5));
printf("       y_75%%_quantile= %g\n",
       gsl_stats_quantile_from_sorted_data(y, stride, n, 0.75));
printf("       y_90%%_quantile= %g\n",
       gsl_stats_quantile_from_sorted_data(y, stride, n, 0.9));
```

```
  printf("            y_maximum= %g\n\n", gsl_stats_max(y, stride, n));

  printf("        y_absdev_mean= %g\n", gsl_stats_absdev(y, stride, n));
  printf("      y_absdev_median= %g\n",
         gsl_stats_absdev_m(y, stride, n,
                            gsl_stats_quantile_from_sorted_data(y, stride, n,
                                                                0.5)));
  printf("        y_absdev_zero= %g\n",
         gsl_stats_absdev_m(y, stride, n, 0.0));
  printf("y_interquartile_range= %g\n\n",
         gsl_stats_quantile_from_sorted_data(y, stride, n, 0.75)
         - gsl_stats_quantile_from_sorted_data(y, stride, n, 0.25));


  return (EXIT_SUCCESS);
}
```

5.72 statistics.c

```
#include "libairs.h"

/* ------------------------------------------------------------
   Definitions...
   ------------------------------------------------------------ */

/* Maximum number of data points. */
#define NMAX 1000000

/* ------------------------------------------------------------
   Main...
   ------------------------------------------------------------ */

int main(
  int argc,
  char *argv[]) {

  FILE *in;

  static double x[NMAX], y[NMAX], w[2 * NMAX];

  static size_t n, stride;

  /* Write info... */
  if (argc != 3)
    ERRMSG("Give parameters: <xy.tab> <stride>");

  /* Get stride... */
  stride = (size_t) atoi(argv[2]);

  /* Read data... */
  if (!(in = fopen(argv[1], "r")))
    ERRMSG("Cannot open file!");
  while (fscanf(in, "%lg %lg", &x[n], &y[n]) == 2)
    if (gsl_finite(x[n]) && gsl_finite(y[n]))
      if (++n >= NMAX)
        ERRMSG("Too many data points!");
  fclose(in);

  /* Get statistics... */
  printf("                    n= %lu\n\n", n);
  printf("        xy_covariance= %g\n",
         gsl_stats_covariance(x, stride, y, stride, n));
  printf("       xy_correlation= %g\n",
         gsl_stats_correlation(x, stride, y, stride, n));
  printf("          xy_spearman= %g\n",
         gsl_stats_spearman(x, stride, y, stride, n, w));
  printf("      x_lag1_autocorr= %g\n",
         gsl_stats_lag1_autocorrelation(x, stride, n));
  printf("      y_lag1_autocorr= %g\n\n",
         gsl_stats_lag1_autocorrelation(y, stride, n));

  /* Sort data... */
  gsl_sort(x, stride, n);
  gsl_sort(y, stride, n);

  /* Get statistics... */
  printf("               x_mean= %g\n", gsl_stats_mean(x, stride, n));
  printf("              x_sigma= %g\n", gsl_stats_sd(x, stride, n));
  printf("           x_skewness= %g\n", gsl_stats_skew(x, stride, n));
  printf("           x_kurtosis= %g\n\n", gsl_stats_kurtosis(x, stride, n));

  printf("            x_mininum= %g\n", gsl_stats_min(x, stride, n));
  printf("      x_10%%_quantile= %g\n",
```

```
         gsl_stats_quantile_from_sorted_data(x, stride, n, 0.1));
  printf("       x_25%%_quantile= %g\n",
         gsl_stats_quantile_from_sorted_data(x, stride, n, 0.25));
  printf("       x_50%%_quantile= %g\n",
         gsl_stats_quantile_from_sorted_data(x, stride, n, 0.5));
  printf("       x_75%%_quantile= %g\n",
         gsl_stats_quantile_from_sorted_data(x, stride, n, 0.75));
  printf("       x_90%%_quantile= %g\n",
         gsl_stats_quantile_from_sorted_data(x, stride, n, 0.9));
  printf("            x_maximum= %g\n\n", gsl_stats_max(x, stride, n));

  printf("        x_absdev_mean= %g\n", gsl_stats_absdev(x, stride, n));
  printf("      x_absdev_median= %g\n",
         gsl_stats_absdev_m(x, stride, n,
                            gsl_stats_quantile_from_sorted_data(x, stride, n,
                                                                0.5)));
  printf("        x_absdev_zero= %g\n",
         gsl_stats_absdev_m(x, stride, n, 0.0));
  printf("x_interquartile_range= %g\n\n",
         gsl_stats_quantile_from_sorted_data(x, stride, n, 0.75)
         - gsl_stats_quantile_from_sorted_data(x, stride, n, 0.25));

  printf("               y_mean= %g\n", gsl_stats_mean(y, stride, n));
  printf("              y_sigma= %g\n", gsl_stats_sd(y, stride, n));
  printf("           y_skewness= %g\n", gsl_stats_skew(y, stride, n));
  printf("           y_kurtosis= %g\n\n", gsl_stats_kurtosis(y, stride, n));

  printf("            y_mininum= %g\n", gsl_stats_min(y, stride, n));
  printf("       y_10%%_quantile= %g\n",
         gsl_stats_quantile_from_sorted_data(y, stride, n, 0.1));
  printf("       y_25%%_quantile= %g\n",
         gsl_stats_quantile_from_sorted_data(y, stride, n, 0.25));
  printf("       y_50%%_quantile= %g\n",
         gsl_stats_quantile_from_sorted_data(y, stride, n, 0.5));
  printf("       y_75%%_quantile= %g\n",
         gsl_stats_quantile_from_sorted_data(y, stride, n, 0.75));
  printf("       y_90%%_quantile= %g\n",
         gsl_stats_quantile_from_sorted_data(y, stride, n, 0.9));
  printf("            y_maximum= %g\n\n", gsl_stats_max(y, stride, n));

  printf("        y_absdev_mean= %g\n", gsl_stats_absdev(y, stride, n));
  printf("      y_absdev_median= %g\n",
         gsl_stats_absdev_m(y, stride, n,
                            gsl_stats_quantile_from_sorted_data(y, stride, n,
                                                                0.5)));
  printf("        y_absdev_zero= %g\n",
         gsl_stats_absdev_m(y, stride, n, 0.0));
  printf("y_interquartile_range= %g\n\n",
         gsl_stats_quantile_from_sorted_data(y, stride, n, 0.75)
         - gsl_stats_quantile_from_sorted_data(y, stride, n, 0.25));


  return (EXIT_SUCCESS);
}
```

5.73 sza.c File Reference

Functions

- int main (int argc, char ∗argv[ ])

5.73.1 Function Documentation

5.73.1.1 int main (int *argc,* char ∗ *argv[ ]*)

Definition at line 3 of file sza.c.

```
                {

  double jsec, lon, lat;

  /* Check arguments... */
  if (argc != 4)
    ERRMSG("Give parameters: <jsec> <lon> <lat>");
```

```

  /* Read arguments... */
  jsec = atof(argv[1]);
  lon = atof(argv[2]);
  lat = atof(argv[3]);

  /* Compute solar zenith angle... */
  printf("%g\n", sza(jsec, lon, lat));

  return EXIT_SUCCESS;
}
```

Here is the call graph for this function:

5.74 sza.c

```
#include "libairs.h"

int main(
  int argc,
  char *argv[]) {

  double jsec, lon, lat;

  /* Check arguments... */
  if (argc != 4)
    ERRMSG("Give parameters: <jsec> <lon> <lat>");

  /* Read arguments... */
  jsec = atof(argv[1]);
  lon = atof(argv[2]);
  lat = atof(argv[3]);

  /* Compute solar zenith angle... */
  printf("%g\n", sza(jsec, lon, lat));

  return EXIT_SUCCESS;
}
```

5.75 umfm.c File Reference

Functions

- void background (double temp[NLAT][NLON], double pt[NLAT][NLON], int nlat, int nlon, int bg_poly_x, int bg_smooth_y)
- int main (int argc, char *argv[ ])

5.75.1 Function Documentation

5.75.1.1 void background (double *temp[NLAT][NLON],* double *pt[NLAT][NLON],* int *nlat,* int *nlon,* int *bg_poly_x,* int *bg_smooth_y*)

Definition at line 397 of file umfm.c.

```
                  {

  static double bg[NLAT][NLON];

  gsl_multifit_linear_workspace *work;
  gsl_matrix *cov, *X;
  gsl_vector *c, *x, *y;

  double chisq, bsum, wsum;

  int ilon, ilat, dlat;

```

```
  size_t dim, i, i2, n;

  /* Compute background... */
  for (ilat = 0; ilat < nlat; ilat++) {

    /* Get number of points... */
    n = 0;
    for (ilon = 0; ilon < nlon; ilon++) {
      bg[ilat][ilon] = GSL_NAN;
      if (gsl_finite(temp[ilat][ilon]))
        n++;
    }
    if (n < 10)
      continue;

    /* Allocate... */
    dim = (size_t) bg_poly_x;
    work = gsl_multifit_linear_alloc(n, dim);
    cov = gsl_matrix_alloc(dim, dim);
    X = gsl_matrix_alloc(n, dim);
    c = gsl_vector_alloc(dim);
    x = gsl_vector_alloc(n);
    y = gsl_vector_alloc(n);

    /* Fit polynomial... */
    i = 0;
    for (ilon = 0; ilon < nlon; ilon++)
      if (gsl_finite(temp[ilat][ilon])) {
        gsl_vector_set(x, i, (double) i);
        gsl_vector_set(y, i, temp[ilat][ilon]);
        for (i2 = 0; i2 < dim; i2++)
          gsl_matrix_set(X, i, i2, pow(gsl_vector_get(x, i), (double) i2));
        i++;
      }
    gsl_multifit_linear(X, y, c, cov, &chisq, work);
    i = 0;
    for (ilon = 0; ilon < nlon; ilon++)
      if (gsl_finite(temp[ilat][ilon])) {
        bg[ilat][ilon] =
          gsl_poly_eval(c->data, (int) dim, gsl_vector_get(x, i));
        i++;
      }

    /* Free... */
    gsl_multifit_linear_free(work);
    gsl_matrix_free(cov);
    gsl_matrix_free(X);
    gsl_vector_free(c);
    gsl_vector_free(x);
    gsl_vector_free(y);
  }

  /* Smooth background and calculate perturbations... */
  for (ilon = 0; ilon < nlon; ilon++)
    for (ilat = 0; ilat < nlat; ilat++) {

      /* Smooth background... */
      bsum = wsum = 0;
      for (dlat = -bg_smooth_y; dlat <= bg_smooth_y; dlat++)
        if (ilat + dlat >= 0 && ilat + dlat < nlat) {
          bsum += bg[ilat + dlat][ilon];
          wsum++;
        }

      /* Compute perturbations... */
      pt[ilat][ilon] = temp[ilat][ilon] - bsum / wsum;
    }
}
```

5.75.1.2 int main (int *argc*, char ∗ *argv[ ]*)

Definition at line 29 of file umfm.c.

```
                {

  static ctl_t ctl;
  static atm_t atm;
  static obs_t obs;

  static double z[NZ], p[NZ][NLAT][NLON], t[NZ][NLAT][NLON],
    lon[NLON], lat[NLAT], temp[NLAT][NLON], pt[NLAT][NLON],
```

```
    x0[3], x1[NLAT][NLON][3], wsum, rmax2 = 10. * 10., var_dh;

  static int bg_poly_x, bg_smooth_y, id, ix, iy, oit, oiz,
    ncid, dimid, varid, ilon, ilon2, ilat, ilat2, iz, ncrop, nlon, nlat, nz;

  static size_t start[10], count[10], rs;

  wave_t *wave_airs, *wave_um;

  FILE *out;

  /* ------------------------------------------------------------
     Get control parameters...
     ------------------------------------------------------------ */

  /* Check arguments... */
  if (argc < 10)
    ERRMSG("Give parameters: <ctl> <ump.nc> <umtheta.nc> <it> "
           "<wave_airs.tab> <out_um.tab> <iz> <out_rad.tab> <wave_um.tab>");

  /* Get arguments... */
  oit = atoi(argv[4]);
  oiz = atoi(argv[7]);

  /* Read control parameters... */
  read_ctl(argc, argv, &ctl);

  /* Set control parameters... */
  ctl.write_bbt = 1;

  /* Get control parameters... */
  bg_poly_x = (int) scan_ctl(argc, argv, "BG_POLY_X", -1, "5", NULL);
  bg_smooth_y = (int) scan_ctl(argc, argv, "BG_SMOOTH_Y", -1, "7", NULL);
  ncrop = (int) scan_ctl(argc, argv, "NCROP", -1, "10", NULL);
  var_dh = scan_ctl(argc, argv, "VAR_DH", -1, "0", NULL);

  /* Allocate... */
  ALLOC(wave_airs, wave_t, 1);
  ALLOC(wave_um, wave_t, 1);

  /* ------------------------------------------------------------
     Read UM data...
     ------------------------------------------------------------ */

  /* Read pressure file... */
  printf("Read UM pressure data: %s\n", argv[2]);
  NC(nc_open(argv[2], NC_NOWRITE, &ncid));

  /* Read longitudes... */
  NC(nc_inq_dimid(ncid, "longitude", &dimid));
  NC(nc_inq_dimlen(ncid, dimid, &rs));
  nlon = (int) rs;
  if (nlon >= NLON)
    ERRMSG("Too many longitudes!");
  NC(nc_inq_varid(ncid, "longitude", &varid));
  NC(nc_get_var_double(ncid, varid, lon));

  /* Read latitudes... */
  NC(nc_inq_dimid(ncid, "latitude", &dimid));
  NC(nc_inq_dimlen(ncid, dimid, &rs));
  nlat = (int) rs;
  if (nlat >= NLAT)
    ERRMSG("Too many latitudes!");
  NC(nc_inq_varid(ncid, "latitude", &varid));
  NC(nc_get_var_double(ncid, varid, lat));

  /* Read heights... */
  NC(nc_inq_dimid(ncid, "ht", &dimid));
  NC(nc_inq_dimlen(ncid, dimid, &rs));
  nz = (int) rs;
  if (nz >= NZ)
    ERRMSG("Too many heights!");
  NC(nc_inq_varid(ncid, "ht", &varid));
  NC(nc_get_var_double(ncid, varid, z));

  /* Read pressure... */
  NC(nc_inq_varid(ncid, "p", &varid));
  for (iz = 0; iz < nz; iz++)
    for (ilat = 0; ilat < nlat; ilat++) {
      start[0] = (size_t) oit;
      start[1] = (size_t) iz;
      start[2] = (size_t) ilat;
      start[3] = 0;
      count[0] = 1;
      count[1] = 1;
      count[2] = 1;
      count[3] = (size_t) nlon;
```

```
      NC(nc_get_vara_double(ncid, varid, start, count, p[iz][ilat]));
    }

  /* Close file... */
  NC(nc_close(ncid));

  /* Read theta file... */
  printf("Read UM theta data: %s\n", argv[3]);
  NC(nc_open(argv[3], NC_NOWRITE, &ncid));

  /* Read theta... */
  NC(nc_inq_varid(ncid, "theta", &varid));
  for (iz = 0; iz < nz; iz++)
    for (ilat = 0; ilat < nlat; ilat++) {
      start[0] = (size_t) oit;
      start[1] = (size_t) iz;
      start[2] = (size_t) ilat;
      start[3] = 0;
      count[0] = 1;
      count[1] = 1;
      count[2] = 1;
      count[3] = (size_t) nlon;
      NC(nc_get_vara_double(ncid, varid, start, count, t[iz][ilat]));
    }

  /* Close file... */
  NC(nc_close(ncid));

  /* ------------------------------------------------------------
     Convert UM data...
     ------------------------------------------------------------ */

  /* Modify longitudes... */
  for (ilon = 0; ilon < nlon; ilon++)
    if (lon[ilon] > 180)
      lon[ilon] -= 360;

  /* Scale heights... */
  for (iz = 0; iz < nz; iz++)
    z[iz] /= 1e3;

  /* Scale pressure and theta... */
  for (iz = 0; iz < nz; iz++)
    for (ilat = 0; ilat < nlat; ilat++)
      for (ilon = 0; ilon < nlon; ilon++)
        if (p[iz][ilat][ilon] <= 0 || p[iz][ilat][ilon] >= 1000000 ||
            t[iz][ilat][ilon] <= 0 || t[iz][ilat][ilon] >= 10000) {
          p[iz][ilat][ilon] = GSL_NAN;
          t[iz][ilat][ilon] = GSL_NAN;
        } else {
          p[iz][ilat][ilon] /= 1e2;
          t[iz][ilat][ilon] /= pow(1e3 / p[iz][ilat][ilon], 0.286);
        }

  /* ------------------------------------------------------------
     Write UM data to ASCII...
     ------------------------------------------------------------ */

  /* Check filename... */
  if (argv[6][0] != '-') {

    /* Check height level... */
    if (oiz < 0 || oiz >= nz)
      ERRMSG("Height index out of range!");

    /* Create file... */
    printf("Write UM data: %s\n", argv[6]);
    if (!(out = fopen(argv[6], "w")))
      ERRMSG("Cannot create file!");

    /* Write header... */
    fprintf(out,
            "# $1 = altitude [km]\n"
            "# $2 = longitude [deg]\n"
            "# $3 = latitude [deg]\n"
            "# $4 = pressure [hPa]\n" "# $5 = temperature [K]\n");

    /* Write output... */
    for (ilon = 0; ilon < nlon; ilon++) {
      fprintf(out, "\n");
      for (ilat = 0; ilat < nlat; ilat++)
        fprintf(out, "%g %g %g %g %g\n", z[oiz], lon[ilon], lat[ilat],
                p[oiz][ilat][ilon], t[oiz][ilat][ilon]);
    }

    /* Close file... */
    fclose(out);
```

```
  }

  /* ------------------------------------------------------------
     Run forward model...
     ------------------------------------------------------------ */

  /* Loop over latitudes... */
  for (ilat = 0; ilat < nlat; ilat++) {

    /* Write info... */
    printf("  Compute latitude %d / %d ...\n", ilat + 1, nlat);

    /* Loop over longitudes... */
    for (ilon = 0; ilon < nlon; ilon++) {

      /* Set atmospheric data... */
      atm.np = 0;
      for (iz = 0; iz < nz; iz++)
        if (gsl_finite(p[iz][ilat][ilon]) && gsl_finite(t[iz][ilat][ilon])
            && p[iz][ilat][ilon] > 0 && p[iz][ilat][ilon] < 1200
            && t[iz][ilat][ilon] > 100 && t[iz][ilat][ilon] < 400) {
          atm.z[atm.np] = z[iz];
          if ((++atm.np) >= NP)
            ERRMSG("Too many altitudes!");
        }
      climatology(&ctl, &atm);
      atm.np = 0;
      for (iz = 0; iz < nz; iz++)
        if (gsl_finite(p[iz][ilat][ilon]) && gsl_finite(t[iz][ilat][ilon])
            && p[iz][ilat][ilon] > 0 && p[iz][ilat][ilon] < 1200
            && t[iz][ilat][ilon] > 100 && t[iz][ilat][ilon] < 400) {
          atm.p[atm.np] = p[iz][ilat][ilon];
          atm.t[atm.np] = t[iz][ilat][ilon];
          atm.np++;
        }

      /* Check number of altitudes... */
      if (atm.np < 20) {
        temp[ilat][ilon] = GSL_NAN;
        continue;
      }

      /* Set observation data... */
      obs.nr = 1;
      obs.obsz[0] = 700;

      /* Run forward model... */
      formod(&ctl, &atm, &obs);

      /* Get mean brightness temperature... */
      temp[ilat][ilon] = 0;
      for (id = 0; id < ctl.nd; id++)
        temp[ilat][ilon] += obs.rad[id][0] / ctl.nd;
    }
  }

  /* Crop at boundaries... */
  for (ilat = 0; ilat < nlat; ilat++) {
    for (ilon = 0; ilon < nlon; ilon++)
      if (gsl_finite(temp[ilat][ilon])) {
        for (ilon2 = ilon; ilon2 <= GSL_MIN(ilon + ncrop, nlon - 1); ilon2++)
          temp[ilat][ilon2] = GSL_NAN;
        break;
      }
    for (ilon = nlon - 1; ilon >= 0; ilon--)
      if (gsl_finite(temp[ilat][ilon])) {
        for (ilon2 = ilon; ilon2 >= GSL_MAX(ilon - ncrop, 0); ilon2--)
          temp[ilat][ilon2] = GSL_NAN;
        break;
      }
  }
  for (ilon = 0; ilon < nlon; ilon++) {
    for (ilat = 0; ilat < nlat; ilat++)
      if (gsl_finite(temp[ilat][ilon])) {
        for (ilat2 = ilat; ilat2 <= GSL_MIN(ilat + ncrop, nlat - 1); ilat2++)
          temp[ilat2][ilon] = GSL_NAN;
        break;
      }
    for (ilat = nlat - 1; ilat >= 0; ilat--)
      if (gsl_finite(temp[ilat][ilon])) {
        for (ilat2 = ilat; ilat2 >= GSL_MAX(ilat - ncrop, 0); ilat2--)
          temp[ilat2][ilon] = GSL_NAN;
        break;
      }
  }

  /* Get perturbations... */
```

```
  background(temp, pt, nlat, nlon, bg_poly_x, bg_smooth_y);

  /* ------------------------------------------------------------
     Save forward model output...
     ------------------------------------------------------------ */

  /* Check filename... */
  if (argv[8][0] != '-') {

    /* Create file... */
    printf("Write radiance data: %s\n", argv[8]);
    if (!(out = fopen(argv[8], "w")))
      ERRMSG("Cannot create file!");

    /* Write header... */
    fprintf(out,
            "# $1 = longitude [deg]\n"
            "# $2 = latitude [deg]\n"
            "# $3 = UM brightness temperature [K]\n"
            "# $4 = UM brightness temperature perturbation [K]\n");

    /* Write output... */
    for (ilat = 0; ilat < nlat; ilat++) {
      fprintf(out, "\n");
      for (ilon = 0; ilon < nlon; ilon++)
        fprintf(out, "%g %g %g %g\n", lon[ilon], lat[ilat],
                temp[ilat][ilon], pt[ilat][ilon]);
    }

    /* Close file... */
    fclose(out);
  }

  /* ------------------------------------------------------------
     Read AIRS radiance map and resample model data...
     ------------------------------------------------------------ */

  /* Check filename... */
  if (argv[5][0] != '-') {

    /* Read AIRS wave file... */
    read_wave(argv[5], wave_airs);
    memcpy(wave_um, wave_airs, sizeof(wave_t));

    /* Get Cartesian coordinates for model grid... */
    for (ilat = 0; ilat < nlat; ilat++)
      for (ilon = 0; ilon < nlon; ilon++)
        geo2cart(0, lon[ilon], lat[ilat], x1[ilat][ilon]);

    /* Loop over AIRS geolocations... */
    for (ix = 0; ix < wave_airs->nx; ix++)
      for (iy = 0; iy < wave_airs->ny; iy++) {

        /* Write info... */
        if (iy == 0)
          printf("  Average for xtrack %d / %d ...\n", ix + 1, wave_airs->nx);

        /* Init... */
        wsum = 0;
        wave_um->temp[ix][iy] = 0;
        wave_um->bg[ix][iy] = 0;
        wave_um->pt[ix][iy] = 0;
        wave_um->var[ix][iy] = 0;

        /* Average... */
        geo2cart(0, wave_airs->lon[ix][iy], wave_airs->lat[ix][iy], x0);
        for (ilat = 0; ilat < nlat; ilat++)
          for (ilon = 0; ilon < nlon; ilon++)
            if (DIST2(x0, x1[ilat][ilon]) <= rmax2) {
              wave_um->temp[ix][iy] += temp[ilat][ilon];
              wave_um->bg[ix][iy] += temp[ilat][ilon] - pt[ilat][ilon];
              wave_um->pt[ix][iy] += pt[ilat][ilon];
              wsum++;
            }

        /* Normalize... */
        wave_um->temp[ix][iy] /= wsum;
        wave_um->bg[ix][iy] /= wsum;
        wave_um->pt[ix][iy] /= wsum;
      }

    /* Compute variance... */
    variance(wave_um, var_dh);

    /* Write UM wave struct... */
    write_wave(argv[9], wave_um);
  }
```

```

  /* Free... */
  free(wave_airs);
  free(wave_um);

  return EXIT_SUCCESS;
}
```

Here is the call graph for this function:

5.76 umfm.c

```
#include "libairs.h"

/* ------------------------------------------------------------
   Dimensions...
   ------------------------------------------------------------ */

/* Maximum UM dimensions. */
#define NLON 2310
#define NLAT 740
#define NZ 41

/* ------------------------------------------------------------
   Functions...
   ------------------------------------------------------------ */

/* Estimate background... */
void background(
  double temp[NLAT][NLON],
  double pt[NLAT][NLON],
  int nlat,
  int nlon,
  int bg_poly_x,
  int bg_smooth_y);

/* ------------------------------------------------------------
   Main...
   ------------------------------------------------------------ */

int main(
  int argc,
  char *argv[]) {

  static ctl_t ctl;
  static atm_t atm;
  static obs_t obs;

  static double z[NZ], p[NZ][NLAT][NLON], t[NZ][NLAT][NLON],
    lon[NLON], lat[NLAT], temp[NLAT][NLON], pt[NLAT][NLON],
    x0[3], x1[NLAT][NLON][3], wsum, rmax2 = 10. * 10., var_dh;

  static int bg_poly_x, bg_smooth_y, id, ix, iy, oit, oiz,
    ncid, dimid, varid, ilon, ilon2, ilat, ilat2, iz, ncrop, nlon, nlat, nz;

  static size_t start[10], count[10], rs;

  wave_t *wave_airs, *wave_um;

  FILE *out;

  /* ------------------------------------------------------------
     Get control parameters...
     ------------------------------------------------------------ */

  /* Check arguments... */
  if (argc < 10)
    ERRMSG("Give parameters: <ctl> <ump.nc> <umtheta.nc> <it> "
           "<wave_airs.tab> <out_um.tab> <iz> <out_rad.tab> <wave_um.tab>");

  /* Get arguments... */
  oit = atoi(argv[4]);
  oiz = atoi(argv[7]);

  /* Read control parameters... */
  read_ctl(argc, argv, &ctl);

  /* Set control parameters... */
  ctl.write_bbt = 1;

  /* Get control parameters... */
```

```
  bg_poly_x = (int) scan_ctl(argc, argv, "BG_POLY_X", -1, "5", NULL);
  bg_smooth_y = (int) scan_ctl(argc, argv, "BG_SMOOTH_Y", -1, "7", NULL);
  ncrop = (int) scan_ctl(argc, argv, "NCROP", -1, "10", NULL);
  var_dh = scan_ctl(argc, argv, "VAR_DH", -1, "0", NULL);

  /* Allocate... */
  ALLOC(wave_airs, wave_t, 1);
  ALLOC(wave_um, wave_t, 1);

  /* ------------------------------------------------------------
     Read UM data...
     ------------------------------------------------------------ */

  /* Read pressure file... */
  printf("Read UM pressure data: %s\n", argv[2]);
  NC(nc_open(argv[2], NC_NOWRITE, &ncid));

  /* Read longitudes... */
  NC(nc_inq_dimid(ncid, "longitude", &dimid));
  NC(nc_inq_dimlen(ncid, dimid, &rs));
  nlon = (int) rs;
  if (nlon >= NLON)
    ERRMSG("Too many longitudes!");
  NC(nc_inq_varid(ncid, "longitude", &varid));
  NC(nc_get_var_double(ncid, varid, lon));

  /* Read latitudes... */
  NC(nc_inq_dimid(ncid, "latitude", &dimid));
  NC(nc_inq_dimlen(ncid, dimid, &rs));
  nlat = (int) rs;
  if (nlat >= NLAT)
    ERRMSG("Too many latitudes!");
  NC(nc_inq_varid(ncid, "latitude", &varid));
  NC(nc_get_var_double(ncid, varid, lat));

  /* Read heights... */
  NC(nc_inq_dimid(ncid, "ht", &dimid));
  NC(nc_inq_dimlen(ncid, dimid, &rs));
  nz = (int) rs;
  if (nz >= NZ)
    ERRMSG("Too many heights!");
  NC(nc_inq_varid(ncid, "ht", &varid));
  NC(nc_get_var_double(ncid, varid, z));

  /* Read pressure... */
  NC(nc_inq_varid(ncid, "p", &varid));
  for (iz = 0; iz < nz; iz++)
    for (ilat = 0; ilat < nlat; ilat++) {
      start[0] = (size_t) oit;
      start[1] = (size_t) iz;
      start[2] = (size_t) ilat;
      start[3] = 0;
      count[0] = 1;
      count[1] = 1;
      count[2] = 1;
      count[3] = (size_t) nlon;
      NC(nc_get_vara_double(ncid, varid, start, count, p[iz][ilat]));
    }

  /* Close file... */
  NC(nc_close(ncid));

  /* Read theta file... */
  printf("Read UM theta data: %s\n", argv[3]);
  NC(nc_open(argv[3], NC_NOWRITE, &ncid));

  /* Read theta... */
  NC(nc_inq_varid(ncid, "theta", &varid));
  for (iz = 0; iz < nz; iz++)
    for (ilat = 0; ilat < nlat; ilat++) {
      start[0] = (size_t) oit;
      start[1] = (size_t) iz;
      start[2] = (size_t) ilat;
      start[3] = 0;
      count[0] = 1;
      count[1] = 1;
      count[2] = 1;
      count[3] = (size_t) nlon;
      NC(nc_get_vara_double(ncid, varid, start, count, t[iz][ilat]));
    }

  /* Close file... */
  NC(nc_close(ncid));

  /* ------------------------------------------------------------
     Convert UM data...
     ------------------------------------------------------------ */
```

```
  /* Modify longitudes... */
  for (ilon = 0; ilon < nlon; ilon++)
    if (lon[ilon] > 180)
      lon[ilon] -= 360;

  /* Scale heights... */
  for (iz = 0; iz < nz; iz++)
    z[iz] /= 1e3;

  /* Scale pressure and theta... */
  for (iz = 0; iz < nz; iz++)
    for (ilat = 0; ilat < nlat; ilat++)
      for (ilon = 0; ilon < nlon; ilon++)
        if (p[iz][ilat][ilon] <= 0 || p[iz][ilat][ilon] >= 1000000 ||
            t[iz][ilat][ilon] <= 0 || t[iz][ilat][ilon] >= 10000) {
          p[iz][ilat][ilon] = GSL_NAN;
          t[iz][ilat][ilon] = GSL_NAN;
        } else {
          p[iz][ilat][ilon] /= 1e2;
          t[iz][ilat][ilon] /= pow(1e3 / p[iz][ilat][ilon], 0.286);
        }

  /* ------------------------------------------------------------
     Write UM data to ASCII...
     ------------------------------------------------------------ */

  /* Check filename... */
  if (argv[6][0] != '-') {

    /* Check height level... */
    if (oiz < 0 || oiz >= nz)
      ERRMSG("Height index out of range!");

    /* Create file... */
    printf("Write UM data: %s\n", argv[6]);
    if (!(out = fopen(argv[6], "w")))
      ERRMSG("Cannot create file!");

    /* Write header... */
    fprintf(out,
            "# $1 = altitude [km]\n"
            "# $2 = longitude [deg]\n"
            "# $3 = latitude [deg]\n"
            "# $4 = pressure [hPa]\n" "# $5 = temperature [K]\n");

    /* Write output... */
    for (ilon = 0; ilon < nlon; ilon++) {
      fprintf(out, "\n");
      for (ilat = 0; ilat < nlat; ilat++)
        fprintf(out, "%g %g %g %g %g\n", z[oiz], lon[ilon], lat[ilat],
                p[oiz][ilat][ilon], t[oiz][ilat][ilon]);
    }

    /* Close file... */
    fclose(out);
  }

  /* ------------------------------------------------------------
     Run forward model...
     ------------------------------------------------------------ */

  /* Loop over latitudes... */
  for (ilat = 0; ilat < nlat; ilat++) {

    /* Write info... */
    printf("  Compute latitude %d / %d ...\n", ilat + 1, nlat);

    /* Loop over longitudes... */
    for (ilon = 0; ilon < nlon; ilon++) {

      /* Set atmospheric data... */
      atm.np = 0;
      for (iz = 0; iz < nz; iz++)
        if (gsl_finite(p[iz][ilat][ilon]) && gsl_finite(t[iz][ilat][ilon])
            && p[iz][ilat][ilon] > 0 && p[iz][ilat][ilon] < 1200
            && t[iz][ilat][ilon] > 100 && t[iz][ilat][ilon] < 400) {
          atm.z[atm.np] = z[iz];
          if ((++atm.np) >= NP)
            ERRMSG("Too many altitudes!");
        }
      climatology(&ctl, &atm);
      atm.np = 0;
      for (iz = 0; iz < nz; iz++)
        if (gsl_finite(p[iz][ilat][ilon]) && gsl_finite(t[iz][ilat][ilon])
            && p[iz][ilat][ilon] > 0 && p[iz][ilat][ilon] < 1200
            && t[iz][ilat][ilon] > 100 && t[iz][ilat][ilon] < 400) {
```

```
          atm.p[atm.np] = p[iz][ilat][ilon];
          atm.t[atm.np] = t[iz][ilat][ilon];
          atm.np++;
        }

      /* Check number of altitudes... */
      if (atm.np < 20) {
        temp[ilat][ilon] = GSL_NAN;
        continue;
      }

      /* Set observation data... */
      obs.nr = 1;
      obs.obsz[0] = 700;

      /* Run forward model... */
      formod(&ctl, &atm, &obs);

      /* Get mean brightness temperature... */
      temp[ilat][ilon] = 0;
      for (id = 0; id < ctl.nd; id++)
        temp[ilat][ilon] += obs.rad[id][0] / ctl.nd;
    }
  }

  /* Crop at boundaries... */
  for (ilat = 0; ilat < nlat; ilat++) {
    for (ilon = 0; ilon < nlon; ilon++)
      if (gsl_finite(temp[ilat][ilon])) {
        for (ilon2 = ilon; ilon2 <= GSL_MIN(ilon + ncrop, nlon - 1); ilon2++)
          temp[ilat][ilon2] = GSL_NAN;
        break;
      }
    for (ilon = nlon - 1; ilon >= 0; ilon--)
      if (gsl_finite(temp[ilat][ilon])) {
        for (ilon2 = ilon; ilon2 >= GSL_MAX(ilon - ncrop, 0); ilon2--)
          temp[ilat][ilon2] = GSL_NAN;
        break;
      }
  }
  for (ilon = 0; ilon < nlon; ilon++) {
    for (ilat = 0; ilat < nlat; ilat++)
      if (gsl_finite(temp[ilat][ilon])) {
        for (ilat2 = ilat; ilat2 <= GSL_MIN(ilat + ncrop, nlat - 1); ilat2++)
          temp[ilat2][ilon] = GSL_NAN;
        break;
      }
    for (ilat = nlat - 1; ilat >= 0; ilat--)
      if (gsl_finite(temp[ilat][ilon])) {
        for (ilat2 = ilat; ilat2 >= GSL_MAX(ilat - ncrop, 0); ilat2--)
          temp[ilat2][ilon] = GSL_NAN;
        break;
      }
  }

  /* Get perturbations... */
  background(temp, pt, nlat, nlon, bg_poly_x, bg_smooth_y);

  /* ------------------------------------------------------------
     Save forward model output...
     ------------------------------------------------------------ */

  /* Check filename... */
  if (argv[8][0] != '-') {

    /* Create file... */
    printf("Write radiance data: %s\n", argv[8]);
    if (!(out = fopen(argv[8], "w")))
      ERRMSG("Cannot create file!");

    /* Write header... */
    fprintf(out,
            "# $1 = longitude [deg]\n"
            "# $2 = latitude [deg]\n"
            "# $3 = UM brightness temperature [K]\n"
            "# $4 = UM brightness temperature perturbation [K]\n");

    /* Write output... */
    for (ilat = 0; ilat < nlat; ilat++) {
      fprintf(out, "\n");
      for (ilon = 0; ilon < nlon; ilon++)
        fprintf(out, "%g %g %g %g\n", lon[ilon], lat[ilat],
                temp[ilat][ilon], pt[ilat][ilon]);
    }

    /* Close file... */
    fclose(out);
```

```
  }

  /* ------------------------------------------------------------
     Read AIRS radiance map and resample model data...
     ------------------------------------------------------------ */

  /* Check filename... */
  if (argv[5][0] != '-') {

    /* Read AIRS wave file... */
    read_wave(argv[5], wave_airs);
    memcpy(wave_um, wave_airs, sizeof(wave_t));

    /* Get Cartesian coordinates for model grid... */
    for (ilat = 0; ilat < nlat; ilat++)
      for (ilon = 0; ilon < nlon; ilon++)
        geo2cart(0, lon[ilon], lat[ilat], x1[ilat][ilon]);

    /* Loop over AIRS geolocations... */
    for (ix = 0; ix < wave_airs->nx; ix++)
      for (iy = 0; iy < wave_airs->ny; iy++) {

        /* Write info... */
        if (iy == 0)
          printf("  Average for xtrack %d / %d ...\n", ix + 1, wave_airs->nx);

        /* Init... */
        wsum = 0;
        wave_um->temp[ix][iy] = 0;
        wave_um->bg[ix][iy] = 0;
        wave_um->pt[ix][iy] = 0;
        wave_um->var[ix][iy] = 0;

        /* Average... */
        geo2cart(0, wave_airs->lon[ix][iy], wave_airs->lat[ix][iy], x0);
        for (ilat = 0; ilat < nlat; ilat++)
          for (ilon = 0; ilon < nlon; ilon++)
            if (DIST2(x0, x1[ilat][ilon]) <= rmax2) {
              wave_um->temp[ix][iy] += temp[ilat][ilon];
              wave_um->bg[ix][iy] += temp[ilat][ilon] - pt[ilat][ilon];
              wave_um->pt[ix][iy] += pt[ilat][ilon];
              wsum++;
            }

        /* Normalize... */
        wave_um->temp[ix][iy] /= wsum;
        wave_um->bg[ix][iy] /= wsum;
        wave_um->pt[ix][iy] /= wsum;
      }

    /* Compute variance... */
    variance(wave_um, var_dh);

    /* Write UM wave struct... */
    write_wave(argv[9], wave_um);
  }

  /* Free... */
  free(wave_airs);
  free(wave_um);

  return EXIT_SUCCESS;
}

/*****************************************************************************/

void background(
  double temp[NLAT][NLON],
  double pt[NLAT][NLON],
  int nlat,
  int nlon,
  int bg_poly_x,
  int bg_smooth_y) {

  static double bg[NLAT][NLON];

  gsl_multifit_linear_workspace *work;
  gsl_matrix *cov, *X;
  gsl_vector *c, *x, *y;

  double chisq, bsum, wsum;

  int ilon, ilat, dlat;

  size_t dim, i, i2, n;

  /* Compute background... */
```

```
  for (ilat = 0; ilat < nlat; ilat++) {

    /* Get number of points... */
    n = 0;
    for (ilon = 0; ilon < nlon; ilon++) {
      bg[ilat][ilon] = GSL_NAN;
      if (gsl_finite(temp[ilat][ilon]))
        n++;
    }
    if (n < 10)
      continue;

    /* Allocate... */
    dim = (size_t) bg_poly_x;
    work = gsl_multifit_linear_alloc(n, dim);
    cov = gsl_matrix_alloc(dim, dim);
    X = gsl_matrix_alloc(n, dim);
    c = gsl_vector_alloc(dim);
    x = gsl_vector_alloc(n);
    y = gsl_vector_alloc(n);

    /* Fit polynomial... */
    i = 0;
    for (ilon = 0; ilon < nlon; ilon++)
      if (gsl_finite(temp[ilat][ilon])) {
        gsl_vector_set(x, i, (double) i);
        gsl_vector_set(y, i, temp[ilat][ilon]);
        for (i2 = 0; i2 < dim; i2++)
          gsl_matrix_set(X, i, i2, pow(gsl_vector_get(x, i), (double) i2));
        i++;
      }
    gsl_multifit_linear(X, y, c, cov, &chisq, work);
    i = 0;
    for (ilon = 0; ilon < nlon; ilon++)
      if (gsl_finite(temp[ilat][ilon])) {
        bg[ilat][ilon] =
          gsl_poly_eval(c->data, (int) dim, gsl_vector_get(x, i));
        i++;
      }

    /* Free... */
    gsl_multifit_linear_free(work);
    gsl_matrix_free(cov);
    gsl_matrix_free(X);
    gsl_vector_free(c);
    gsl_vector_free(x);
    gsl_vector_free(y);
  }

  /* Smooth background and calculate perturbations... */
  for (ilon = 0; ilon < nlon; ilon++)
    for (ilat = 0; ilat < nlat; ilat++) {

      /* Smooth background... */
      bsum = wsum = 0;
      for (dlat = -bg_smooth_y; dlat <= bg_smooth_y; dlat++)
        if (ilat + dlat >= 0 && ilat + dlat < nlat) {
          bsum += bg[ilat + dlat][ilon];
          wsum++;
        }

      /* Compute perturbations... */
      pt[ilat][ilon] = temp[ilat][ilon] - bsum / wsum;
    }
}
```

5.77 var1d.c File Reference

Functions

- int main (int argc, char ∗argv[ ])

5.77.1 Function Documentation

5.77.1.1 int main (int *argc,* char ∗ *argv[ ]*)

Definition at line 3 of file var1d.c.

```
              {

gsl_multifit_linear_workspace *work;
gsl_matrix *cov, *X;
gsl_vector *c, *xvec, *yvec, *yfit;

static double chisq, fwhm, lx, dlx, lxmin, lxmax, phi,
  var, var2, vmean, vmean2, width, w, wsum;

static int dim, i, i2, n;

/* Check arguments... */
if (argc != 8)
  ERRMSG("Give parameters: <width> <n> <lxmin> <lxmax> <dlx> <fwhm> <dim>");

/* Get arguments... */
width = atof(argv[1]);
n = atoi(argv[2]);
lxmin = atof(argv[3]);
lxmax = atof(argv[4]);
dlx = atoi(argv[5]);
fwhm = atof(argv[6]);
dim = atoi(argv[7]);

/* Initialize... */
c = gsl_vector_alloc((size_t) dim);
cov = gsl_matrix_alloc((size_t) dim, (size_t) dim);
work = gsl_multifit_linear_alloc((size_t) n, (size_t) dim);
X = gsl_matrix_alloc((size_t) n, (size_t) dim);
xvec = gsl_vector_alloc((size_t) n);
yvec = gsl_vector_alloc((size_t) n);
yfit = gsl_vector_alloc((size_t) n);

/* Loop over wavelengths... */
for (lx = lxmin; lx <= lxmax; lx += dlx) {

  /* Initialize... */
  vmean = 0;
  vmean2 = 0;

  /* Loop over phases... */
  for (phi = 0; phi < 2 * M_PI; phi += M_PI / 180) {

    /* Initialize... */
    var = 0;
    var2 = 0;
    wsum = 0;

    /* Set wave... */
    for (i = 0; i < n; i++) {
      gsl_vector_set(xvec, (size_t) i, width / (n - 1.0) * i - width / 2.);
      gsl_vector_set(yvec, (size_t) i,
                     sin(2 * M_PI / lx * gsl_vector_get(xvec, (size_t) i) +
                         phi));
      if (fwhm > 0) {
        w = gsl_ran_gaussian_pdf(gsl_vector_get(xvec, (size_t) i),
                                 fwhm * lx / 2.3548);
        gsl_vector_set(yvec, (size_t) i,
                       w * gsl_vector_get(yvec, (size_t) i));
        wsum += w;
      }
    }
    if (wsum > 0)
      gsl_vector_scale(yvec, 1 / wsum);

    /* Detrending... */
    for (i = 0; i < n; i++)
      for (i2 = 0; i2 < dim; i2++)
        gsl_matrix_set(X, (size_t) i, (size_t) i2,
                       pow(gsl_vector_get(xvec, (size_t) i), 1. * i2));
    gsl_multifit_linear(X, yvec, c, cov, &chisq, work);
    for (i = 0; i < n; i++)
      gsl_vector_set(yfit, (size_t) i, gsl_vector_get(yvec, (size_t) i)
                     - gsl_poly_eval(c->data, (int) dim,
                                     gsl_vector_get(xvec, (size_t) i)));

    /* Compute variances... */
    for (i = 0; i < n; i++) {
      var += gsl_pow_2(gsl_vector_get(yfit, (size_t) i)) / (double) n;
      var2 += gsl_pow_2(gsl_vector_get(yvec, (size_t) i)) / (double) n;
    }
    vmean += var;
    vmean2 += var2;
  }

  /* Write output... */
  printf("%g %g\n", lx, 100 * vmean / vmean2);
```

```
  }

  return EXIT_SUCCESS;
}
```

5.78 var1d.c

```
#include "libairs.h"

int main(
  int argc,
  char *argv[]) {

  gsl_multifit_linear_workspace *work;
  gsl_matrix *cov, *X;
  gsl_vector *c, *xvec, *yvec, *yfit;

  static double chisq, fwhm, lx, dlx, lxmin, lxmax, phi,
    var, var2, vmean, vmean2, width, w, wsum;

  static int dim, i, i2, n;

  /* Check arguments... */
  if (argc != 8)
    ERRMSG("Give parameters: <width> <n> <lxmin> <lxmax> <dlx> <fwhm> <dim>");

  /* Get arguments... */
  width = atof(argv[1]);
  n = atoi(argv[2]);
  lxmin = atof(argv[3]);
  lxmax = atof(argv[4]);
  dlx = atoi(argv[5]);
  fwhm = atof(argv[6]);
  dim = atoi(argv[7]);

  /* Initialize... */
  c = gsl_vector_alloc((size_t) dim);
  cov = gsl_matrix_alloc((size_t) dim, (size_t) dim);
  work = gsl_multifit_linear_alloc((size_t) n, (size_t) dim);
  X = gsl_matrix_alloc((size_t) n, (size_t) dim);
  xvec = gsl_vector_alloc((size_t) n);
  yvec = gsl_vector_alloc((size_t) n);
  yfit = gsl_vector_alloc((size_t) n);

  /* Loop over wavelengths... */
  for (lx = lxmin; lx <= lxmax; lx += dlx) {

    /* Initialize... */
    vmean = 0;
    vmean2 = 0;

    /* Loop over phases... */
    for (phi = 0; phi < 2 * M_PI; phi += M_PI / 180) {

      /* Initialize... */
      var = 0;
      var2 = 0;
      wsum = 0;

      /* Set wave... */
      for (i = 0; i < n; i++) {
        gsl_vector_set(xvec, (size_t) i, width / (n - 1.0) * i - width / 2.);
        gsl_vector_set(yvec, (size_t) i,
                       sin(2 * M_PI / lx * gsl_vector_get(xvec, (size_t) i) +
                           phi));
        if (fwhm > 0) {
          w = gsl_ran_gaussian_pdf(gsl_vector_get(xvec, (size_t) i),
                                   fwhm * lx / 2.3548);
          gsl_vector_set(yvec, (size_t) i,
                         w * gsl_vector_get(yvec, (size_t) i));
          wsum += w;
        }
      }
      if (wsum > 0)
        gsl_vector_scale(yvec, 1 / wsum);

      /* Detrending... */
      for (i = 0; i < n; i++)
        for (i2 = 0; i2 < dim; i2++)
          gsl_matrix_set(X, (size_t) i, (size_t) i2,
                         pow(gsl_vector_get(xvec, (size_t) i), 1. * i2));
      gsl_multifit_linear(X, yvec, c, cov, &chisq, work);
      for (i = 0; i < n; i++)
```

```
        gsl_vector_set(yfit, (size_t) i, gsl_vector_get(yvec, (size_t) i)
                       - gsl_poly_eval(c->data, (int) dim,
                                       gsl_vector_get(xvec, (size_t) i)));

      /* Compute variances... */
      for (i = 0; i < n; i++) {
        var += gsl_pow_2(gsl_vector_get(yfit, (size_t) i)) / (double) n;
        var2 += gsl_pow_2(gsl_vector_get(yvec, (size_t) i)) / (double) n;
      }
      vmean += var;
      vmean2 += var2;
    }

    /* Write output... */
    printf("%g %g\n", lx, 100 * vmean / vmean2);
  }

  return EXIT_SUCCESS;
}
```

5.79 var3d.c File Reference

Functions

- int main (int argc, char *argv[ ])

5.79.1 Function Documentation

5.79.1.1 int main (int *argc,* char * *argv[ ]*)

Definition at line 3 of file var3d.c.

```
                {

  static ctl_t ctl;
  static atm_t atm;
  static obs_t obs;

  gsl_multifit_linear_workspace *work;
  gsl_matrix *cov, *k, *X;
  gsl_vector *c, *xvec, *yvec;

  static double alpha, alphamax, amp, ampmax, bg[L1_NXTRACK], ca, chisq,
    dalpha, ddx, dx, jac[L1_NXTRACK][NP], ly, lz, mu, phi, rad[L1_NXTRACK],
    radius, sa, t30, var, vmean, vmin, vmax, x, y[L1_NXTRACK];

  static int detrend, dim = 5, i, i2, id, ip, n, nmu, nphi, ndx;

  /* Check arguments... */
  if (argc < 13)
    ERRMSG("Give parameters: <ctl> <atm> <T0_30km> <exp/lin> <radius> "
           "<obsz> <alphamax> <n> <dalpha> <dx> <ddx> <detrend>");
  t30 = atof(argv[3]);
  radius = atof(argv[5]);
  obs.obsz[0] = atof(argv[6]);
  alphamax = atof(argv[7]);
  n = atoi(argv[8]);
  if (n > L1_NXTRACK)
    ERRMSG("Too many tracks!");
  dalpha = atof(argv[9]);
  dx = atof(argv[10]);
  ddx = atof(argv[11]);
  detrend = atoi(argv[12]);

  /* Initialize... */
  c = gsl_vector_alloc((size_t) dim);
  cov = gsl_matrix_alloc((size_t) dim, (size_t) dim);
  work = gsl_multifit_linear_alloc((size_t) n, (size_t) dim);
  X = gsl_matrix_alloc((size_t) n, (size_t) dim);
  xvec = gsl_vector_alloc((size_t) n);
  yvec = gsl_vector_alloc((size_t) n);

  /* Read forward model control parameters... */
  read_ctl(argc, argv, &ctl);
```

```c

/* Read atmospheric data... */
read_atm(NULL, argv[2], &ctl, &atm);

/* ------------------------------------------------------------
   Compute mean radiance and kernel functions...
   ------------------------------------------------------------ */

/* Loop over scans... */
for (i = 0; i < n; i++) {

  /* Set observation geometry... */
  obs.nr = 1;
  alpha =
    -alphamax + 2. * alphamax * i / (n - 1.) + (i % 2 ==
                                                 0 ? 1.0 : -1.0) * dalpha;
  sa = sin(alpha * M_PI / 180.);
  ca = cos(alpha * M_PI / 180.);
  obs.vplat[0] = 180. / M_PI
    * asin(sa / RE * ((RE + obs.obsz[0]) * ca
                      - sqrt(gsl_pow_2(RE) -
                             gsl_pow_2((RE + obs.obsz[0]) * sa))));
  y[i] = obs.vplat[0] / 180 * M_PI * RE;

  /* Run forward model... */
  formod(&ctl, &atm, &obs);
  bg[i] = 0;
  for (id = 0; id < ctl.nd; id++)
    bg[i] += obs.rad[id][0] / ctl.nd;

  /* Compute kernel matrix... */
  ctl.rett_zmin = -10000;
  ctl.rett_zmax = 10000;
  k = gsl_matrix_alloc((size_t) ctl.nd, (size_t) atm.np);
  kernel(&ctl, &atm, &obs, k);
  for (ip = 0; ip < atm.np; ip++) {
    jac[i][ip] = 0;
    for (id = 0; id < ctl.nd; id++)
      jac[i][ip] += gsl_matrix_get(k, (size_t) id, (size_t) ip) / ctl.nd;
  }
  gsl_matrix_free(k);
}

/* ------------------------------------------------------------
   Get variance filter characteristics...
   ------------------------------------------------------------ */

/* Loop over wavelengths... */
for (lz = 10; lz <= 50; lz += 0.5)
  for (ly = 50; ly <= 1500; ly += 10) {

    /* Initialize... */
    vmean = 0;
    vmin = 1e10;
    vmax = -1e10;
    nphi = 0;

    /* Loop over phases... */
    for (phi = 0; phi < 2 * M_PI; phi += M_PI / 24) {

      /* Initialize... */
      nmu = 0;
      mu = var = 0;

      /* Loop over swaths... */
      for (x = -radius; x <= radius;
           x += dx + ((ndx++) % 2 == 0 ? 1.0 : -1.0) * ddx) {

        /* Compute radiances for perturbed profile... */
        for (i = 0; i < n; i++) {
          rad[i] = bg[i];
          for (ip = 0; ip < atm.np; ip++) {
            amp = t30;
            if (argv[4][0] == 'e' || argv[4][0] == 'E') {

              /* Saturation amplitude (Preusse et al., 2008),
                 Tmax = lz / (2*pi) * Tbg / g * N^2... */
              ampmax = lz * 1e3 / (2 * M_PI) * 250 / 9.81 * gsl_pow_2(0.02);

              /* Get wave amplitude... */
              amp *= exp((atm.z[ip] - 30.) / 14.);
              amp = (amp > ampmax) ? ampmax : amp;
            }
            rad[i] += jac[i][ip] * amp
              * sin(2 * M_PI / ly * y[i] + 2 * M_PI / lz * atm.z[ip] + phi);
          }
        }
```

```

          /* Detrending... */
          if (detrend) {
            for (i = 0; i < n; i++) {
              gsl_vector_set(xvec, (size_t) i, y[i]);
              gsl_vector_set(yvec, (size_t) i, rad[i]);
              for (i2 = 0; i2 < dim; i2++)
                gsl_matrix_set(X, (size_t) i, (size_t) i2,
                               pow(gsl_vector_get(xvec, (size_t) i),
                                   1. * i2));
            }
            gsl_multifit_linear(X, yvec, c, cov, &chisq, work);
            for (i = 0; i < n; i++)
              rad[i] -= gsl_poly_eval(c->data, (int) dim,
                                      gsl_vector_get(xvec, (size_t) i));
          }

          /* Compute variance... */
          for (i = 0; i < n; i++)
            if (gsl_pow_2(x) + gsl_pow_2(y[i]) <= gsl_pow_2(radius)) {
              mu += rad[i];
              var += gsl_pow_2(rad[i]);
              nmu++;
            }
        }

        /* Compute variance... */
        mu /= nmu;
        var = var / nmu - mu * mu;
        vmean += var;
        vmax = GSL_MAX(vmax, var);
        vmin = GSL_MIN(vmin, var);
        nphi++;
      }

      /* Write output... */
      printf("obsfilt: %g %g %g %g %g\n", ly, lz, vmean / nphi, vmax, vmin);
    }

  return EXIT_SUCCESS;
}
```

Here is the call graph for this function:

5.80 var3d.c

```
#include "libairs.h"

int main(
  int argc,
  char *argv[]) {

  static ctl_t ctl;
  static atm_t atm;
  static obs_t obs;

  gsl_multifit_linear_workspace *work;
  gsl_matrix *cov, *k, *X;
  gsl_vector *c, *xvec, *yvec;

  static double alpha, alphamax, amp, ampmax, bg[L1_NXTRACK], ca, chisq,
    dalpha, ddx, dx, jac[L1_NXTRACK][NP], ly, lz, mu, phi, rad[L1_NXTRACK],
    radius, sa, t30, var, vmean, vmin, vmax, x, y[L1_NXTRACK];

  static int detrend, dim = 5, i, i2, id, ip, n, nmu, nphi, ndx;

  /* Check arguments... */
  if (argc < 13)
    ERRMSG("Give parameters: <ctl> <atm> <T0_30km> <exp/lin> <radius> "
           "<obsz> <alphamax> <n> <dalpha> <dx> <ddx> <detrend>");
  t30 = atof(argv[3]);
  radius = atof(argv[5]);
  obs.obsz[0] = atof(argv[6]);
  alphamax = atof(argv[7]);
  n = atoi(argv[8]);
  if (n > L1_NXTRACK)
    ERRMSG("Too many tracks!");
  dalpha = atof(argv[9]);
  dx = atof(argv[10]);
  ddx = atof(argv[11]);
  detrend = atoi(argv[12]);
```

```

  /* Initialize... */
  c = gsl_vector_alloc((size_t) dim);
  cov = gsl_matrix_alloc((size_t) dim, (size_t) dim);
  work = gsl_multifit_linear_alloc((size_t) n, (size_t) dim);
  X = gsl_matrix_alloc((size_t) n, (size_t) dim);
  xvec = gsl_vector_alloc((size_t) n);
  yvec = gsl_vector_alloc((size_t) n);

  /* Read forward model control parameters... */
  read_ctl(argc, argv, &ctl);

  /* Read atmospheric data... */
  read_atm(NULL, argv[2], &ctl, &atm);

  /* ------------------------------------------------------------
     Compute mean radiance and kernel functions...
     ------------------------------------------------------------ */

  /* Loop over scans... */
  for (i = 0; i < n; i++) {

    /* Set observation geometry... */
    obs.nr = 1;
    alpha =
      -alphamax + 2. * alphamax * i / (n - 1.) + (i % 2 ==
                                                  0 ? 1.0 : -1.0) * dalpha;
    sa = sin(alpha * M_PI / 180.);
    ca = cos(alpha * M_PI / 180.);
    obs.vplat[0] = 180. / M_PI
      * asin(sa / RE * ((RE + obs.obsz[0]) * ca
                        - sqrt(gsl_pow_2(RE) -
                               gsl_pow_2((RE + obs.obsz[0]) * sa))));
    y[i] = obs.vplat[0] / 180 * M_PI * RE;

    /* Run forward model... */
    formod(&ctl, &atm, &obs);
    bg[i] = 0;
    for (id = 0; id < ctl.nd; id++)
      bg[i] += obs.rad[id][0] / ctl.nd;

    /* Compute kernel matrix... */
    ctl.rett_zmin = -10000;
    ctl.rett_zmax = 10000;
    k = gsl_matrix_alloc((size_t) ctl.nd, (size_t) atm.np);
    kernel(&ctl, &atm, &obs, k);
    for (ip = 0; ip < atm.np; ip++) {
      jac[i][ip] = 0;
      for (id = 0; id < ctl.nd; id++)
        jac[i][ip] += gsl_matrix_get(k, (size_t) id, (size_t) ip) / ctl.nd;
    }
    gsl_matrix_free(k);
  }

  /* ------------------------------------------------------------
     Get variance filter characteristics...
     ------------------------------------------------------------ */

  /* Loop over wavelengths... */
  for (lz = 10; lz <= 50; lz += 0.5)
    for (ly = 50; ly <= 1500; ly += 10) {

      /* Initialize... */
      vmean = 0;
      vmin = 1e10;
      vmax = -1e10;
      nphi = 0;

      /* Loop over phases... */
      for (phi = 0; phi < 2 * M_PI; phi += M_PI / 24) {

        /* Initialize... */
        nmu = 0;
        mu = var = 0;

        /* Loop over swaths... */
        for (x = -radius; x <= radius;
             x += dx + ((ndx++) % 2 == 0 ? 1.0 : -1.0) * ddx) {

          /* Compute radiances for perturbed profile... */
          for (i = 0; i < n; i++) {
            rad[i] = bg[i];
            for (ip = 0; ip < atm.np; ip++) {
              amp = t30;
              if (argv[4][0] == 'e' || argv[4][0] == 'E') {

                /* Saturation amplitude (Preusse et al., 2008),
```

```
                   Tmax = lz / (2*pi) * Tbg / g * N^2... */
                ampmax = lz * 1e3 / (2 * M_PI) * 250 / 9.81 * gsl_pow_2(0.02);

                /* Get wave amplitude... */
                amp *= exp((atm.z[ip] - 30.) / 14.);
                amp = (amp > ampmax) ? ampmax : amp;
              }
              rad[i] += jac[i][ip] * amp
                * sin(2 * M_PI / ly * y[i] + 2 * M_PI / lz * atm.z[ip] + phi);
            }
          }

          /* Detrending... */
          if (detrend) {
            for (i = 0; i < n; i++) {
              gsl_vector_set(xvec, (size_t) i, y[i]);
              gsl_vector_set(yvec, (size_t) i, rad[i]);
              for (i2 = 0; i2 < dim; i2++)
                gsl_matrix_set(X, (size_t) i, (size_t) i2,
                               pow(gsl_vector_get(xvec, (size_t) i),
                                   1. * i2));
            }
            gsl_multifit_linear(X, yvec, c, cov, &chisq, work);
            for (i = 0; i < n; i++)
              rad[i] -= gsl_poly_eval(c->data, (int) dim,
                                      gsl_vector_get(xvec, (size_t) i));
          }

          /* Compute variance... */
          for (i = 0; i < n; i++)
            if (gsl_pow_2(x) + gsl_pow_2(y[i]) <= gsl_pow_2(radius)) {
              mu += rad[i];
              var += gsl_pow_2(rad[i]);
              nmu++;
            }
        }

        /* Compute variance... */
        mu /= nmu;
        var = var / nmu - mu * mu;
        vmean += var;
        vmax = GSL_MAX(vmax, var);
        vmin = GSL_MIN(vmin, var);
        nphi++;
      }

      /* Write output... */
      printf("obsfilt: %g %g %g %g %g\n", ly, lz, vmean / nphi, vmax, vmin);
    }

  return EXIT_SUCCESS;
}
```

5.81 variance.c File Reference

Functions

- int main (int argc, char *argv[ ])

5.81.1 Function Documentation

5.81.1.1 int main (int *argc*, char * *argv[ ]*)

Definition at line 261 of file variance.c.

```
                 {

  static pert_t *pert;

  static wave_t *wave;

  static FILE *in, *out;

  static char pertname[LEN], set[LEN];
```

```
  static double bt[NX][NY], bt_8mu[NX][NY], bt_8mu_min[NX][NY],
    bt_8mu_max[NX][NY], dt[NX][NY], mtime[NX][NY], glat[NY], glon[NX],
    fdc[NX][NY], fwg[NX][NY], fgw[NX][NY], fcw[NX][NY],
    mean[NX][NY], min[NX][NY], max[NX][NY], var[NX][NY],
    t_dc, t_gw, dt_trop = 0, dc_hlat = 25, dc_tlim = 250, dt230,
    nesr, gauss_fwhm, var_dh, nu, lon0, lon1, lat0, lat1,
    thresh_dc, thresh_gw, lt;

  static int asc, ix, iy, nx, ny, iarg, n[NX][NY],
    ndc[NX][NY], ngw[NX][NY], ncw[NX][NY], nwg[NX][NY],
    det_gw, det_cw, det_dc, det_wg, ilat, imon, nmin = 10,
    bg_poly_x, bg_poly_y, bg_smooth_x, bg_smooth_y,
    itrack, itrack2, ixtrack, ixtrack2, iradius = 30;

  /* Check arguments... */
  if (argc < 4)
    ERRMSG("Give parameters: <ctl> <var.tab> <pert1.nc> [<pert2.nc> ...]");

  /* Get control parameters... */
  scan_ctl(argc, argv, "SET", -1, "full", set);
  scan_ctl(argc, argv, "PERTNAME", -1, "4mu", pertname);
  nx = (int) scan_ctl(argc, argv, "NX", -1, "360", NULL);
  lon0 = scan_ctl(argc, argv, "LON0", -1, "-180", NULL);
  lon1 = scan_ctl(argc, argv, "LON1", -1, "180", NULL);
  ny = (int) scan_ctl(argc, argv, "NY", -1, "180", NULL);
  lat0 = scan_ctl(argc, argv, "LAT0", -1, "-90", NULL);
  lat1 = scan_ctl(argc, argv, "LAT1", -1, "90", NULL);
  bg_poly_x = (int) scan_ctl(argc, argv, "BG_POLY_X", -1, "0", NULL);
  bg_poly_y = (int) scan_ctl(argc, argv, "BG_POLY_Y", -1, "0", NULL);
  bg_smooth_x = (int) scan_ctl(argc, argv, "BG_SMOOTH_X", -1, "0", NULL);
  bg_smooth_y = (int) scan_ctl(argc, argv, "BG_SMOOTH_Y", -1, "0", NULL);
  gauss_fwhm = scan_ctl(argc, argv, "GAUSS_FWHM", -1, "0", NULL);
  var_dh = scan_ctl(argc, argv, "VAR_DH", -1, "0", NULL);
  thresh_gw = scan_ctl(argc, argv, "THRESH_GW", -1, "-999", NULL);
  thresh_dc = scan_ctl(argc, argv, "THRESH_DC", -1, "-999", NULL);
  dt230 = scan_ctl(argc, argv, "DT230", -1, "0.16", NULL);
  nu = scan_ctl(argc, argv, "NU", -1, "2345.0", NULL);

  /* Allocate... */
  ALLOC(pert, pert_t, 1);

  /* Check grid dimensions... */
  if (nx < 1 || nx > NX)
    ERRMSG("Set 1 <= NX <= MAX!");
  if (ny < 1 || ny > NY)
    ERRMSG("Set 1 <= NY <= MAX!");

  /* Loop over perturbation files... */
  for (iarg = 3; iarg < argc; iarg++) {

    /* Read perturbation data... */
    if (!(in = fopen(argv[iarg], "r")))
      continue;
    else {
      fclose(in);
      read_pert(argv[iarg], pertname, pert);
    }

    /* Recalculate background and perturbations... */
    if (bg_poly_x > 0 || bg_poly_y > 0 ||
        bg_smooth_x > 0 || bg_smooth_y > 0 || gauss_fwhm > 0 || var_dh > 0) {

      /* Allocate... */
      ALLOC(wave, wave_t, 1);

      /* Convert to wave analysis struct... */
      pert2wave(pert, wave, 0, pert->ntrack - 1, 0, pert->nxtrack - 1);

      /* Estimate background... */
      background_poly(wave, bg_poly_x, bg_poly_y);
      background_smooth(wave, bg_smooth_x, bg_smooth_y);

      /* Gaussian filter... */
      gauss(wave, gauss_fwhm);

      /* Compute variance... */
      variance(wave, var_dh);

      /* Copy data... */
      for (ix = 0; ix < wave->nx; ix++)
        for (iy = 0; iy < wave->ny; iy++) {
          pert->pt[iy][ix] = wave->pt[ix][iy];
          pert->var[iy][ix] = wave->var[ix][iy];
        }

      /* Free... */
```

```
    free(wave);
  }

  /* Detection... */
  for (itrack = 0; itrack < pert->ntrack; itrack++)
    for (ixtrack = 0; ixtrack < pert->nxtrack; ixtrack++) {

      /* Check data... */
      if (pert->time[itrack][ixtrack] < 0
          || pert->lon[itrack][ixtrack] < -180
          || pert->lon[itrack][ixtrack] > 180
          || pert->lat[itrack][ixtrack] < -90
          || pert->lat[itrack][ixtrack] > 90
          || pert->pt[itrack][ixtrack] < -100
          || pert->pt[itrack][ixtrack] > 100
          || !gsl_finite(pert->bt[itrack][ixtrack])
          || !gsl_finite(pert->pt[itrack][ixtrack])
          || !gsl_finite(pert->var[itrack][ixtrack])
          || !gsl_finite(pert->dc[itrack][ixtrack]))
        continue;

      /* Get and check ascending/descending flag... */
      asc = (pert->lat[itrack > 0 ? itrack : itrack + 1][pert->nxtrack / 2]
             > pert->lat[itrack >
                         0 ? itrack - 1 : itrack][pert->nxtrack / 2]);
      if (((set[0] == 'a' || set[0] == 'A') && !asc)
          || ((set[0] == 'd' || set[0] == 'D') && asc))
        continue;

      /* Check am/pm flag... */
      lt = fmod(pert->time[itrack][ixtrack], 86400.) / 3600.;
      if (((set[0] == 'm' || set[0] == 'M') && lt > 12.)
          || ((set[0] == 'n' || set[0] == 'N') && lt < 12.))
        continue;

      /* Get grid indices... */
      ix =
        (int) ((pert->lon[itrack][ixtrack] - lon0) / (lon1 -
                                                      lon0) * (double) nx);
      iy =
        (int) ((pert->lat[itrack][ixtrack] - lat0) / (lat1 -
                                                      lat0) * (double) ny);
      if (ix < 0 || ix >= nx || iy < 0 || iy >= ny)
        continue;

      /* Get month index... */
      imon =
        (int) (fmod(pert->time[0][0] / 60. / 60. / 24. / 365.25, 1.) *
               NMON);
      if (imon < 0 || imon >= NMON)
        continue;

      /* Get gravity wave detection threshold... */
      if (thresh_gw <= 0.0) {
        ilat = locate(t_gw_lat, NLAT_GW, pert->lat[itrack][ixtrack]);
        if (asc)
          t_gw = LIN(t_gw_lat[ilat], t_gw_asc[imon][ilat],
                     t_gw_lat[ilat + 1], t_gw_asc[imon][ilat + 1],
                     pert->lat[itrack][ixtrack]);
        else
          t_gw = LIN(t_gw_lat[ilat], t_gw_dsc[imon][ilat],
                     t_gw_lat[ilat + 1], t_gw_dsc[imon][ilat + 1],
                     pert->lat[itrack][ixtrack]);
      } else
        t_gw = thresh_gw;

      /* Get deep convection detection threshold... */
      if (thresh_dc <= 0.0) {
        ilat = locate(t_trop_lat, NLAT_TROP, pert->lat[itrack][ixtrack]);
        t_dc = LIN(t_trop_lat[ilat], t_trop[imon][ilat],
                   t_trop_lat[ilat + 1], t_trop[imon][ilat + 1],
                   pert->lat[itrack][ixtrack]) + dt_trop;
      } else
        t_dc = thresh_dc;

      /* Detection of gravity waves... */
      det_gw = (pert->var[itrack][ixtrack] >= t_gw);

      /* Detection of convective waves... */
      det_cw = 0;
      if (det_gw)
        for (itrack2 = GSL_MAX(itrack - iradius, 0);
             itrack2 <= GSL_MIN(itrack + iradius, pert->ntrack - 1);
             itrack2++)
          for (ixtrack2 = GSL_MAX(ixtrack - iradius, 0);
               ixtrack2 <= GSL_MIN(ixtrack + iradius, pert->nxtrack - 1);
               ixtrack2++) {
```

```
            if (det_cw)
              break;
            det_cw = (pert->dc[itrack2][ixtrack2] <= t_dc);
          }

      /* Detection of deep convection... */
      det_dc = (pert->dc[itrack][ixtrack] <= t_dc);

      /* Detection of wave generation... */
      det_wg = 0;
      if (det_dc)
        for (itrack2 = GSL_MAX(itrack - iradius, 0);
             itrack2 <= GSL_MIN(itrack + iradius, pert->ntrack - 1);
             itrack2++)
          for (ixtrack2 = GSL_MAX(ixtrack - iradius, 0);
               ixtrack2 <= GSL_MIN(ixtrack + iradius, pert->nxtrack - 1);
               ixtrack2++) {
            if (det_wg)
              break;
            det_wg = (pert->var[itrack2][ixtrack2] >= t_gw);
          }

      /* Count events... */
      n[ix][iy]++;
      if (det_dc)
        ndc[ix][iy]++;
      if (det_wg)
        nwg[ix][iy]++;
      if (det_gw)
        ngw[ix][iy]++;
      if (det_cw)
        ncw[ix][iy]++;

      /* Get statistics of perturbations... */
      mean[ix][iy] += pert->pt[itrack][ixtrack];
      var[ix][iy] += gsl_pow_2(pert->pt[itrack][ixtrack]);
      max[ix][iy] = GSL_MAX(max[ix][iy], pert->pt[itrack][ixtrack]);
      min[ix][iy] = GSL_MIN(min[ix][iy], pert->pt[itrack][ixtrack]);

      /* Get statistics of brightness temperatures... */
      bt[ix][iy] += pert->bt[itrack][ixtrack];
      bt_8mu[ix][iy] += pert->dc[itrack][ixtrack];
      if (n[ix][iy] > 1) {
        bt_8mu_min[ix][iy]
          = GSL_MIN(bt_8mu_min[ix][iy], pert->dc[itrack][ixtrack]);
        bt_8mu_max[ix][iy]
          = GSL_MAX(bt_8mu_max[ix][iy], pert->dc[itrack][ixtrack]);
      } else {
        bt_8mu_min[ix][iy] = pert->dc[itrack][ixtrack];
        bt_8mu_max[ix][iy] = pert->dc[itrack][ixtrack];
      }

      /* Get mean time... */
      mtime[ix][iy] += pert->time[itrack][ixtrack];
    }
  }

  /* Analyze results... */
  for (ix = 0; ix < nx; ix++)
    for (iy = 0; iy < ny; iy++) {

      /* Get geolocation... */
      mtime[ix][iy] /= (double) n[ix][iy];
      glon[ix] = lon0 + (ix + 0.5) / (double) nx *(
  lon1 - lon0);
      glat[iy] = lat0 + (iy + 0.5) / (double) ny *(
  lat1 - lat0);

      /* Normalize brightness temperatures... */
      bt[ix][iy] /= (double) n[ix][iy];
      bt_8mu[ix][iy] /= (double) n[ix][iy];

      /* Get fractions... */
      fdc[ix][iy] = (double) ndc[ix][iy] / (double) n[ix][iy] * 100.;
      fwg[ix][iy] = (double) nwg[ix][iy] / (double) ndc[ix][iy] * 100.;
      fgw[ix][iy] = (double) ngw[ix][iy] / (double) n[ix][iy] * 100.;
      fcw[ix][iy] = (double) ncw[ix][iy] / (double) ngw[ix][iy] * 100.;

      /* Check number of observations... */
      if (n[ix][iy] < nmin) {
        fdc[ix][iy] = GSL_NAN;
        fwg[ix][iy] = GSL_NAN;
        fgw[ix][iy] = GSL_NAN;
        fcw[ix][iy] = GSL_NAN;
        bt_8mu[ix][iy] = GSL_NAN;
        bt_8mu_min[ix][iy] = GSL_NAN;
        bt_8mu_max[ix][iy] = GSL_NAN;
```

```
      }

      /* Check detections of deep convection at high latitudes... */
      if (fabs(glat[iy]) > dc_hlat && bt_8mu[ix][iy] <= dc_tlim) {
        fdc[ix][iy] = GSL_NAN;
        fwg[ix][iy] = GSL_NAN;
        fcw[ix][iy] = GSL_NAN;
      }

      /* Estimate noise... */
      if (dt230 > 0) {
        nesr = planck(230.0 + dt230, nu) - planck(230.0, nu);
        dt[ix][iy] =
          brightness(planck(bt[ix][iy], nu) + nesr, nu) - bt[ix][iy];
      }

      /* Get mean perturbation and variance... */
      mean[ix][iy] /= (double) n[ix][iy];
      var[ix][iy] =
        var[ix][iy] / (double) n[ix][iy] - gsl_pow_2(mean[ix][iy]);
    }

  /* Create file... */
  printf("Write variance statistics: %s\n", argv[2]);
  if (!(out = fopen(argv[2], "w")))
    ERRMSG("Cannot create file!");

  /* Write header... */
  fprintf(out,
          "# $1 = time [s]\n"
          "# $2 = longitude [deg]\n"
          "# $3 = latitude [deg]\n"
          "# $4 = number of footprints\n"
          "# $5 = fraction of convection events [%%]\n"
          "# $6 = fraction of wave generating events [%%]\n"
          "# $7 = fraction of gravity wave events [%%]\n"
          "# $8 = fraction of convective wave events [%%]\n"
          "# $9 = mean perturbation [K]\n"
          "# $10 = minimum perturbation [K]\n");
  fprintf(out,
          "# $11 = maximum perturbation [K]\n"
          "# $12 = variance [K^2]\n"
          "# $13 = mean surface temperature [K]\n"
          "# $14 = minimum surface temperature [K]\n"
          "# $15 = maximum surface temperature [K]\n"
          "# $16 = mean background temperature [K]\n"
          "# $17 = noise estimate [K]\n");

  /* Write results... */
  for (iy = 0; iy < ny; iy++) {
    if (iy == 0 || nx > 1)
      fprintf(out, "\n");
    for (ix = 0; ix < nx; ix++)
      fprintf(out, "%.2f %g %g %d %g %g %g %g %g %g %g %g %g %g %g %g %g\n",
              mtime[ix][iy], glon[ix], glat[iy], n[ix][iy],
              fdc[ix][iy], fwg[ix][iy], fgw[ix][iy], fcw[ix][iy],
              mean[ix][iy], min[ix][iy], max[ix][iy], var[ix][iy],
              bt_8mu[ix][iy], bt_8mu_min[ix][iy], bt_8mu_max[ix][iy],
              bt[ix][iy], dt[ix][iy]);
  }

  /* Close file... */
  fclose(out);

  /* Free... */
  free(pert);

  return EXIT_SUCCESS;
}
```

Here is the call graph for this function:

5.82 variance.c

```
#include "libairs.h"

/* ------------------------------------------------------------
   Dimensions...
   ------------------------------------------------------------ */

/* Number of latitudes for threshold tables. */
```

```
#define NLAT_GW 19
#define NLAT_SURF 6
#define NLAT_TROP 73

/* Number of months for threshold tables. */
#define NMON 12

/* Maximum number of longitudes. */
#define NX 3600

/* Maximum number of latitudes. */
#define NY 1800

/* ------------------------------------------------------------
   Global variables...
   ------------------------------------------------------------ */

/* Latitudes for gravity wave variance thresholds. */
static double t_gw_lat[NLAT_GW]
  = { -90, -80, -70, -60, -50, -40, -30, -20, -10, 0,
  10, 20, 30, 40, 50, 60, 70, 80, 90
};

/* Gravity wave variance thresholds (ascending orbits). */
static double t_gw_asc[NMON][NLAT_GW]
  = { {0.00387, 0.00422, 0.00633, 0.0124, 0.0216, 0.0324,
       0.0553, 0.0791, 0.0501, 0.0136, 0.0134, 0.0151,
       0.0522, 0.321, 0.697, 0.776, 0.696, 0.764, 0.771},
{0.00913, 0.00942, 0.00867, 0.00897, 0.0112, 0.0168,
 0.0314, 0.0484, 0.032, 0.0128, 0.0122, 0.0134,
 0.0382, 0.124, 0.345, 0.404, 0.545, 1.16, 1.18},
{0.0845, 0.0664, 0.0384, 0.0227, 0.0147, 0.0118,
 0.0141, 0.0184, 0.0162, 0.0123, 0.0124, 0.0124,
 0.0159, 0.0509, 0.085, 0.103, 0.188, 0.367, 0.529},
{0.265, 0.297, 0.216, 0.106, 0.0666, 0.0299,
 0.0169, 0.0129, 0.0116, 0.012, 0.0135, 0.0141,
 0.0134, 0.0137, 0.017, 0.0268, 0.0259, 0.0319, 0.0323},
{0.326, 0.44, 0.628, 0.567, 0.434, 0.235,
 0.0601, 0.0214, 0.0132, 0.0113, 0.0144, 0.0185,
 0.0179, 0.0142, 0.0116, 0.00945, 0.00865, 0.00918, 0.00878},
{0.537, 0.73, 1.39, 1.75, 1.35, 0.528,
 0.188, 0.0311, 0.0133, 0.0124, 0.0205, 0.0313,
 0.0297, 0.0216, 0.0166, 0.0131, 0.00983, 0.00606, 0.0049},
{0.382, 1.15, 1.57, 2.13, 1.66, 0.851,
 0.126, 0.0204, 0.0133, 0.0135, 0.0281, 0.0385,
 0.0375, 0.0312, 0.0223, 0.0143, 0.00949, 0.0061, 0.00493},
{0.226, 0.697, 1.68, 1.56, 1.14, 0.496,
 0.0616, 0.0143, 0.0126, 0.013, 0.0216, 0.0252,
 0.0241, 0.0206, 0.0152, 0.0106, 0.00976, 0.0105, 0.00998},
{0.236, 0.489, 0.648, 0.553, 0.524, 0.21,
 0.033, 0.0129, 0.0116, 0.0129, 0.0163, 0.0165,
 0.0153, 0.014, 0.0141, 0.0185, 0.0301, 0.0591, 0.0745},
{0.046, 0.082, 0.112, 0.0806, 0.0516, 0.0469,
 0.0225, 0.0139, 0.0127, 0.0121, 0.0125, 0.0138,
 0.0176, 0.0357, 0.0563, 0.062, 0.133, 0.327, 0.3},
{0.00669, 0.00867, 0.0117, 0.0117, 0.014, 0.015,
 0.0203, 0.0213, 0.0144, 0.0116, 0.0124, 0.0179,
 0.0574, 0.185, 0.346, 0.442, 0.54, 0.669, 0.664},
{0.00355, 0.00381, 0.00658, 0.0125, 0.0217, 0.0304,
 0.0424, 0.0515, 0.0315, 0.0139, 0.0137, 0.0161,
 0.0582, 0.306, 0.999, 1.2, 1.14, 0.621, 0.448}
};

/* Gravity wave variance thresholds (descending orbits). */
static double t_gw_dsc[NMON][NLAT_GW]
  = { {0.00383, 0.00458, 0.00866, 0.019, 0.0348, 0.0598,
       0.144, 0.234, 0.135, 0.0373, 0.0325, 0.0377,
       0.0858, 0.497, 1.4, 1.32, 0.808, 0.771, 0.773},
{0.00999, 0.0123, 0.0141, 0.0148, 0.0177, 0.0286,
 0.0626, 0.102, 0.0717, 0.0302, 0.0261, 0.03,
 0.086, 0.268, 0.631, 0.716, 1.17, 1.24, 1.21},
{0.103, 0.096, 0.0715, 0.0535, 0.0343, 0.0245,
 0.025, 0.0315, 0.0303, 0.0233, 0.023, 0.0257,
 0.0353, 0.118, 0.197, 0.359, 0.541, 0.585, 0.586},
{0.272, 0.293, 0.276, 0.226, 0.146, 0.0689,
 0.0373, 0.0245, 0.0232, 0.0232, 0.0224, 0.0217,
 0.0242, 0.031, 0.0441, 0.0664, 0.0623, 0.053, 0.0361},
{0.331, 0.44, 0.641, 0.868, 0.824, 0.47,
 0.115, 0.0444, 0.0269, 0.0223, 0.0274, 0.0332,
 0.0273, 0.023, 0.0191, 0.0172, 0.0138, 0.0107, 0.00894},
{0.554, 0.716, 1.31, 2.29, 2.43, 1.05,
 0.41, 0.0651, 0.0269, 0.0257, 0.0447, 0.0622,
 0.0497, 0.0357, 0.0258, 0.0182, 0.0117, 0.00697, 0.00502},
{0.427, 0.905, 1.44, 2.78, 2.76, 1.52,
 0.278, 0.041, 0.0279, 0.0296, 0.0629, 0.0818,
 0.0758, 0.0534, 0.0356, 0.0227, 0.012, 0.00692, 0.00513},
{0.245, 0.74, 1.88, 2.32, 1.89, 0.883,
```

```
 0.122, 0.0292, 0.0264, 0.0289, 0.0516, 0.059,
 0.0495, 0.0373, 0.0268, 0.0185, 0.0163, 0.0131, 0.0103},
{0.272, 0.551, 0.812, 0.844, 0.852, 0.486,
 0.0842, 0.0269, 0.0225, 0.0239, 0.0322, 0.0324,
 0.0307, 0.0304, 0.035, 0.0484, 0.0692, 0.0956, 0.0948},
{0.0644, 0.125, 0.177, 0.135, 0.0922, 0.0899,
 0.0524, 0.0249, 0.0214, 0.0218, 0.0251, 0.0293,
 0.0403, 0.0903, 0.168, 0.246, 0.358, 0.378, 0.288},
{0.00676, 0.00923, 0.0148, 0.0195, 0.0261, 0.0286,
 0.0302, 0.0343, 0.0298, 0.024, 0.0252, 0.0403,
 0.131, 0.448, 0.681, 0.923, 0.839, 0.684, 0.629},
{0.00347, 0.00412, 0.00995, 0.0221, 0.0363, 0.0531,
 0.104, 0.168, 0.112, 0.0365, 0.0335, 0.0382,
 0.128, 0.563, 1.62, 1.87, 1.47, 0.652, 0.408}
};

/* Latitudes for zonal mean tropopause temperatures. */
static double t_trop_lat[NLAT_TROP]
  = { 90, 87.5, 85, 82.5, 80, 77.5, 75, 72.5, 70, 67.5, 65, 62.5, 60,
  57.5, 55, 52.5, 50, 47.5, 45, 42.5, 40, 37.5, 35, 32.5, 30, 27.5,
  25, 22.5, 20, 17.5, 15, 12.5, 10, 7.5, 5, 2.5, 0, -2.5, -5, -7.5,
  -10, -12.5, -15, -17.5, -20, -22.5, -25, -27.5, -30, -32.5, -35,
  -37.5, -40, -42.5, -45, -47.5, -50, -52.5, -55, -57.5, -60, -62.5,
  -65, -67.5, -70, -72.5, -75, -77.5, -80, -82.5, -85, -87.5, -90
};

/* Zonal mean tropopause temperatures. */
static double t_trop[NMON][NLAT_TROP]
  = { {211.152, 211.237, 211.434, 211.549, 211.614, 211.776, 211.974,
       212.234, 212.489, 212.808, 213.251, 213.692, 214.193, 214.591,
       214.985, 215.327, 215.658, 215.956, 216.236, 216.446, 216.738,
       216.836, 216.032, 213.607, 209.281, 205, 201.518, 198.969,
       197.123, 195.869, 195.001, 194.409, 193.985, 193.734, 193.617,
       193.573, 193.6, 193.642, 193.707, 193.856, 194.131, 194.558,
       195.121, 195.907, 196.91, 198.192, 199.744, 201.583, 203.672,
       206.012, 208.542, 211.135, 213.681, 216.085, 218.317, 220.329,
       222.071, 223.508, 224.612, 225.357, 225.761, 225.863, 225.657,
       225.287, 224.813, 224.571, 224.385, 224.3, 224.257, 224.173,
       223.786, 222.713, 222.11},
{212.593, 212.621, 212.801, 212.888, 212.912, 213.054, 213.245,
 213.512, 213.726, 213.962, 214.259, 214.508, 214.823, 215.037,
 215.297, 215.545, 215.808, 216.063, 216.323, 216.539, 216.867,
 217.051, 216.532, 214.512, 210.371, 205.658, 201.758, 198.937,
 197.047, 195.817, 194.96, 194.386, 193.993, 193.771, 193.673,
 193.635, 193.658, 193.691, 193.744, 193.872, 194.126, 194.54,
 195.085, 195.847, 196.8, 198.013, 199.489, 201.261, 203.298,
 205.596, 208.082, 210.628, 213.156, 215.563, 217.822, 219.903,
 221.745, 223.311, 224.566, 225.451, 225.947, 226.079, 225.849,
 225.406, 224.889, 224.643, 224.431, 224.246, 224.079, 223.884,
 223.42, 222.402, 221.871},
{215.529, 215.491, 215.539, 215.621, 215.691, 215.808, 215.847,
 215.881, 215.878, 215.907, 216.02, 216.113, 216.297, 216.342,
 216.38, 216.369, 216.342, 216.284, 216.185, 215.989, 215.855,
 215.626, 215.023, 213.432, 209.979, 205.886, 202.212, 199.414,
 197.488, 196.216, 195.327, 194.732, 194.347, 194.158, 194.095,
 194.079, 194.116, 194.154, 194.195, 194.302, 194.534, 194.922,
 195.461, 196.253, 197.288, 198.644, 200.309, 202.293, 204.553,
 207.033, 209.538, 211.911, 214.016, 215.862, 217.572, 219.179,
 220.655, 221.959, 223.052, 223.867, 224.344, 224.451, 224.179,
 223.706, 223.163, 222.876, 222.613, 222.385, 222.154, 221.842,
 221.304, 220.402, 220.06},
{219.921, 219.916, 219.99, 219.989, 219.916, 219.867, 219.73,
 219.522, 219.16, 218.765, 218.448, 218.144, 217.99, 217.756,
 217.553, 217.311, 217.025, 216.684, 216.241, 215.649, 215.05,
 214.302, 213.219, 211.496, 208.729, 205.649, 202.594, 200.066,
 198.144, 196.733, 195.687, 194.991, 194.586, 194.429, 194.418,
 194.443, 194.492, 194.534, 194.59, 194.718, 194.997, 195.481,
 196.165, 197.159, 198.462, 200.142, 202.154, 204.533, 207.208,
 209.848, 212.088, 213.845, 215.222, 216.348, 217.384, 218.383,
 219.313, 220.131, 220.799, 221.271, 221.479, 221.405, 221.012,
 220.4, 219.702, 219.227, 218.827, 218.434, 217.977, 217.477,
 216.783, 215.974, 215.707},
{225.363, 225.255, 225.064, 224.745, 224.351, 224, 223.551,
 222.966, 222.195, 221.435, 220.802, 220.245, 219.871, 219.424,
 218.99, 218.529, 218.013, 217.445, 216.76, 215.859, 214.723,
 213.049, 211.032, 208.767, 206.449, 204.302, 202.113, 200.187,
 198.501, 197.153, 196.117, 195.441, 195.121, 195.073, 195.146,
 195.212, 195.261, 195.288, 195.343, 195.485, 195.772, 196.284,
 197.018, 198.125, 199.624, 201.604, 204.073, 207.036, 210.193,
 212.853, 214.611, 215.635, 216.287, 216.801, 217.284, 217.716,
 218.057, 218.253, 218.282, 218.115, 217.729, 217.15, 216.376,
 215.449, 214.428, 213.574, 212.847, 212.281, 211.718, 211.211,
 210.616, 210.112, 210.056},
{228.431, 228.261, 227.966, 227.457, 226.812, 226.208, 225.518,
 224.71, 223.701, 222.762, 222.045, 221.486, 221.142, 220.761,
 220.361, 219.896, 219.34, 218.646, 217.626, 215.983, 213.624,
 210.817, 208.017, 205.73, 203.8, 202.363, 200.96, 199.778,
```

```
 198.695, 197.845, 197.166, 196.743, 196.6, 196.66, 196.809,
 196.925, 196.985, 196.996, 197.033, 197.135, 197.335, 197.754,
 198.367, 199.335, 200.693, 202.564, 205.001, 208.084, 211.473,
 214.407, 216.208, 217.018, 217.314, 217.394, 217.371, 217.234,
 216.961, 216.517, 215.878, 215.027, 213.952, 212.697, 211.274,
 209.736, 208.172, 206.872, 205.84, 205.093, 204.32, 203.816,
 203.55, 203.49, 203.606},
{229.01, 228.807, 228.45, 227.839, 227.084, 226.377, 225.589,
 224.712, 223.665, 222.724, 222.058, 221.658, 221.519, 221.376,
 221.136, 220.673, 219.926, 218.742, 216.744, 214.028, 210.994,
 208.374, 206.131, 204.563, 203.251, 202.328, 201.313, 200.411,
 199.531, 198.876, 198.356, 198.104, 198.088, 198.21, 198.385,
 198.502, 198.57, 198.601, 198.652, 198.731, 198.869, 199.207,
 199.737, 200.595, 201.802, 203.491, 205.771, 208.765, 212.241,
 215.403, 217.439, 218.251, 218.297, 217.988, 217.533, 216.941,
 216.161, 215.154, 213.887, 212.35, 210.525, 208.481, 206.287,
 204.068, 202.033, 200.405, 199.106, 198.225, 197.435, 197.02,
 197.133, 197.527, 197.808},
{226.525, 226.354, 225.996, 225.433, 224.842, 224.358, 223.818,
 223.202, 222.426, 221.723, 221.266, 220.98, 220.893, 220.707,
 220.392, 219.928, 219.182, 218.015, 216.051, 213.399, 210.617,
 208.318, 206.311, 204.838, 203.515, 202.527, 201.397, 200.423,
 199.494, 198.848, 198.385, 198.212, 198.294, 198.49, 198.707,
 198.853, 198.933, 198.967, 199.01, 199.079, 199.207, 199.537,
 200.081, 200.968, 202.215, 203.946, 206.254, 209.291, 212.876,
 216.262, 218.487, 219.387, 219.436, 219.048, 218.405, 217.527,
 216.372, 214.919, 213.152, 211.096, 208.767, 206.247, 203.609,
 201.029, 198.763, 196.961, 195.578, 194.635, 193.923, 193.54,
 193.632, 193.944, 193.912},
{223.293, 223.158, 222.945, 222.571, 222.126, 221.749, 221.362,
 220.946, 220.404, 219.946, 219.704, 219.599, 219.611, 219.429,
 219.124, 218.702, 218.063, 217.157, 215.827, 213.879, 211.352,
 208.833, 206.504, 204.728, 203.168, 201.992, 200.735, 199.74,
 198.833, 198.213, 197.801, 197.661, 197.765, 197.963, 198.182,
 198.336, 198.42, 198.456, 198.505, 198.609, 198.794, 199.19,
 199.796, 200.758, 202.089, 203.915, 206.262, 209.295, 212.807,
 216.083, 218.329, 219.47, 219.877, 219.846, 219.507, 218.85,
 217.84, 216.448, 214.652, 212.509, 210.083, 207.534, 204.982,
 202.596, 200.463, 198.769, 197.441, 196.546, 195.902, 195.472,
 195.193, 195.066, 195.006},
{219.564, 219.492, 219.415, 219.191, 218.926, 218.801, 218.691,
 218.561, 218.298, 218.06, 217.982, 217.956, 218.038, 217.954,
 217.81, 217.532, 217.08, 216.439, 215.549, 214.31, 212.725,
 210.573, 208.019, 205.585, 203.459, 201.779, 200.162, 198.879,
 197.771, 196.987, 196.459, 196.19, 196.172, 196.274, 196.435,
 196.544, 196.601, 196.644, 196.727, 196.904, 197.184, 197.696,
 198.42, 199.497, 200.934, 202.825, 205.151, 208.005, 211.279,
 214.441, 216.87, 218.493, 219.498, 220.072, 220.353, 220.336,
 219.991, 219.271, 218.142, 216.636, 214.804, 212.776, 210.636,
 208.535, 206.516, 204.825, 203.383, 202.281, 201.365, 200.561,
 199.896, 199.415, 199.382},
{215.926, 215.884, 215.897, 215.814, 215.689, 215.692, 215.707,
 215.767, 215.815, 215.92, 216.138, 216.327, 216.588, 216.668,
 216.664, 216.553, 216.373, 216.112, 215.711, 215.025, 214.106,
 212.596, 210.346, 207.503, 204.604, 202.251, 200.231, 198.607,
 197.228, 196.174, 195.382, 194.87, 194.61, 194.54, 194.579,
 194.615, 194.66, 194.709, 194.82, 195.074, 195.487, 196.103,
 196.904, 198.01, 199.43, 201.246, 203.431, 206.007, 208.905,
 211.81, 214.34, 216.36, 217.918, 219.141, 220.159, 220.965,
 221.514, 221.754, 221.637, 221.135, 220.226, 218.986, 217.475,
 215.879, 214.251, 212.918, 211.84, 211.026, 210.288, 209.553,
 208.791, 208.132, 208.053},
{212.893, 212.911, 213.03, 213.109, 213.224, 213.453, 213.653,
 213.836, 213.98, 214.166, 214.481, 214.787, 215.179, 215.435,
 215.688, 215.908, 216.084, 216.217, 216.262, 216.123, 215.819,
 214.977, 213.173, 210.214, 206.619, 203.437, 200.836, 198.843,
 197.271, 196.078, 195.164, 194.509, 194.057, 193.82, 193.742,
 193.723, 193.762, 193.813, 193.903, 194.121, 194.49, 195.016,
 195.698, 196.627, 197.82, 199.359, 201.204, 203.355, 205.78,
 208.414, 211.057, 213.521, 215.662, 217.504, 219.133, 220.544,
 221.723, 222.631, 223.274, 223.649, 223.737, 223.547, 223.053,
 222.357, 221.52, 220.948, 220.527, 220.247, 220.013, 219.726,
 219.273, 218.506, 218.144}
};

/* ------------------------------------------------------------
   Main...
   ------------------------------------------------------------ */

int main(
  int argc,
  char *argv[]) {

  static pert_t *pert;

  static wave_t *wave;

```

```c
  static FILE *in, *out;

  static char pertname[LEN], set[LEN];

  static double bt[NX][NY], bt_8mu[NX][NY], bt_8mu_min[NX][NY],
    bt_8mu_max[NX][NY], dt[NX][NY], mtime[NX][NY], glat[NY], glon[NX],
    fdc[NX][NY], fwg[NX][NY], fgw[NX][NY], fcw[NX][NY],
    mean[NX][NY], min[NX][NY], max[NX][NY], var[NX][NY],
    t_dc, t_gw, dt_trop = 0, dc_hlat = 25, dc_tlim = 250, dt230,
    nesr, gauss_fwhm, var_dh, nu, lon0, lon1, lat0, lat1,
    thresh_dc, thresh_gw, lt;

  static int asc, ix, iy, nx, ny, iarg, n[NX][NY],
    ndc[NX][NY], ngw[NX][NY], ncw[NX][NY], nwg[NX][NY],
    det_gw, det_cw, det_dc, det_wg, ilat, imon, nmin = 10,
    bg_poly_x, bg_poly_y, bg_smooth_x, bg_smooth_y,
    itrack, itrack2, ixtrack, ixtrack2, iradius = 30;

  /* Check arguments... */
  if (argc < 4)
    ERRMSG("Give parameters: <ctl> <var.tab> <pert1.nc> [<pert2.nc> ...]");

  /* Get control parameters... */
  scan_ctl(argc, argv, "SET", -1, "full", set);
  scan_ctl(argc, argv, "PERTNAME", -1, "4mu", pertname);
  nx = (int) scan_ctl(argc, argv, "NX", -1, "360", NULL);
  lon0 = scan_ctl(argc, argv, "LON0", -1, "-180", NULL);
  lon1 = scan_ctl(argc, argv, "LON1", -1, "180", NULL);
  ny = (int) scan_ctl(argc, argv, "NY", -1, "180", NULL);
  lat0 = scan_ctl(argc, argv, "LAT0", -1, "-90", NULL);
  lat1 = scan_ctl(argc, argv, "LAT1", -1, "90", NULL);
  bg_poly_x = (int) scan_ctl(argc, argv, "BG_POLY_X", -1, "0", NULL);
  bg_poly_y = (int) scan_ctl(argc, argv, "BG_POLY_Y", -1, "0", NULL);
  bg_smooth_x = (int) scan_ctl(argc, argv, "BG_SMOOTH_X", -1, "0", NULL);
  bg_smooth_y = (int) scan_ctl(argc, argv, "BG_SMOOTH_Y", -1, "0", NULL);
  gauss_fwhm = scan_ctl(argc, argv, "GAUSS_FWHM", -1, "0", NULL);
  var_dh = scan_ctl(argc, argv, "VAR_DH", -1, "0", NULL);
  thresh_gw = scan_ctl(argc, argv, "THRESH_GW", -1, "-999", NULL);
  thresh_dc = scan_ctl(argc, argv, "THRESH_DC", -1, "-999", NULL);
  dt230 = scan_ctl(argc, argv, "DT230", -1, "0.16", NULL);
  nu = scan_ctl(argc, argv, "NU", -1, "2345.0", NULL);

  /* Allocate... */
  ALLOC(pert, pert_t, 1);

  /* Check grid dimensions... */
  if (nx < 1 || nx > NX)
    ERRMSG("Set 1 <= NX <= MAX!");
  if (ny < 1 || ny > NY)
    ERRMSG("Set 1 <= NY <= MAX!");

  /* Loop over perturbation files... */
  for (iarg = 3; iarg < argc; iarg++) {

    /* Read perturbation data... */
    if (!(in = fopen(argv[iarg], "r")))
      continue;
    else {
      fclose(in);
      read_pert(argv[iarg], pertname, pert);
    }

    /* Recalculate background and perturbations... */
    if (bg_poly_x > 0 || bg_poly_y > 0 ||
        bg_smooth_x > 0 || bg_smooth_y > 0 || gauss_fwhm > 0 || var_dh > 0) {

      /* Allocate... */
      ALLOC(wave, wave_t, 1);

      /* Convert to wave analysis struct... */
      pert2wave(pert, wave, 0, pert->ntrack - 1, 0, pert->nxtrack - 1);

      /* Estimate background... */
      background_poly(wave, bg_poly_x, bg_poly_y);
      background_smooth(wave, bg_smooth_x, bg_smooth_y);

      /* Gaussian filter... */
      gauss(wave, gauss_fwhm);

      /* Compute variance... */
      variance(wave, var_dh);

      /* Copy data... */
      for (ix = 0; ix < wave->nx; ix++)
        for (iy = 0; iy < wave->ny; iy++) {
          pert->pt[iy][ix] = wave->pt[ix][iy];
          pert->var[iy][ix] = wave->var[ix][iy];
```

```
      }

    /* Free... */
    free(wave);
  }

  /* Detection... */
  for (itrack = 0; itrack < pert->ntrack; itrack++)
    for (ixtrack = 0; ixtrack < pert->nxtrack; ixtrack++) {

      /* Check data... */
      if (pert->time[itrack][ixtrack] < 0
          || pert->lon[itrack][ixtrack] < -180
          || pert->lon[itrack][ixtrack] > 180
          || pert->lat[itrack][ixtrack] < -90
          || pert->lat[itrack][ixtrack] > 90
          || pert->pt[itrack][ixtrack] < -100
          || pert->pt[itrack][ixtrack] > 100
          || !gsl_finite(pert->bt[itrack][ixtrack])
          || !gsl_finite(pert->pt[itrack][ixtrack])
          || !gsl_finite(pert->var[itrack][ixtrack])
          || !gsl_finite(pert->dc[itrack][ixtrack]))
        continue;

      /* Get and check ascending/descending flag... */
      asc = (pert->lat[itrack > 0 ? itrack : itrack + 1][pert->nxtrack / 2]
             > pert->lat[itrack >
                         0 ? itrack - 1 : itrack][pert->nxtrack / 2]);
      if (((set[0] == 'a' || set[0] == 'A') && !asc)
          || ((set[0] == 'd' || set[0] == 'D') && asc))
        continue;

      /* Check am/pm flag... */
      lt = fmod(pert->time[itrack][ixtrack], 86400.) / 3600.;
      if (((set[0] == 'm' || set[0] == 'M') && lt > 12.)
          || ((set[0] == 'n' || set[0] == 'N') && lt < 12.))
        continue;

      /* Get grid indices... */
      ix =
        (int) ((pert->lon[itrack][ixtrack] - lon0) / (lon1 -
                                                      lon0) * (double) nx);
      iy =
        (int) ((pert->lat[itrack][ixtrack] - lat0) / (lat1 -
                                                      lat0) * (double) ny);
      if (ix < 0 || ix >= nx || iy < 0 || iy >= ny)
        continue;

      /* Get month index... */
      imon =
        (int) (fmod(pert->time[0][0] / 60. / 60. / 24. / 365.25, 1.) *
               NMON);
      if (imon < 0 || imon >= NMON)
        continue;

      /* Get gravity wave detection threshold... */
      if (thresh_gw <= 0.0) {
        ilat = locate(t_gw_lat, NLAT_GW, pert->lat[itrack][ixtrack]);
        if (asc)
          t_gw = LIN(t_gw_lat[ilat], t_gw_asc[imon][ilat],
                     t_gw_lat[ilat + 1], t_gw_asc[imon][ilat + 1],
                     pert->lat[itrack][ixtrack]);
        else
          t_gw = LIN(t_gw_lat[ilat], t_gw_dsc[imon][ilat],
                     t_gw_lat[ilat + 1], t_gw_dsc[imon][ilat + 1],
                     pert->lat[itrack][ixtrack]);
      } else
        t_gw = thresh_gw;

      /* Get deep convection detection threshold... */
      if (thresh_dc <= 0.0) {
        ilat = locate(t_trop_lat, NLAT_TROP, pert->lat[itrack][ixtrack]);
        t_dc = LIN(t_trop_lat[ilat], t_trop[imon][ilat],
                   t_trop_lat[ilat + 1], t_trop[imon][ilat + 1],
                   pert->lat[itrack][ixtrack]) + dt_trop;
      } else
        t_dc = thresh_dc;

      /* Detection of gravity waves... */
      det_gw = (pert->var[itrack][ixtrack] >= t_gw);

      /* Detection of convective waves... */
      det_cw = 0;
      if (det_gw)
        for (itrack2 = GSL_MAX(itrack - iradius, 0);
             itrack2 <= GSL_MIN(itrack + iradius, pert->ntrack - 1);
             itrack2++)
```

```
            for (ixtrack2 = GSL_MAX(ixtrack - iradius, 0);
                 ixtrack2 <= GSL_MIN(ixtrack + iradius, pert->nxtrack - 1);
                 ixtrack2++) {
              if (det_cw)
                break;
              det_cw = (pert->dc[itrack2][ixtrack2] <= t_dc);
            }

        /* Detection of deep convection... */
        det_dc = (pert->dc[itrack][ixtrack] <= t_dc);

        /* Detection of wave generation... */
        det_wg = 0;
        if (det_dc)
          for (itrack2 = GSL_MAX(itrack - iradius, 0);
               itrack2 <= GSL_MIN(itrack + iradius, pert->ntrack - 1);
               itrack2++)
            for (ixtrack2 = GSL_MAX(ixtrack - iradius, 0);
                 ixtrack2 <= GSL_MIN(ixtrack + iradius, pert->nxtrack - 1);
                 ixtrack2++) {
              if (det_wg)
                break;
              det_wg = (pert->var[itrack2][ixtrack2] >= t_gw);
            }

        /* Count events... */
        n[ix][iy]++;
        if (det_dc)
          ndc[ix][iy]++;
        if (det_wg)
          nwg[ix][iy]++;
        if (det_gw)
          ngw[ix][iy]++;
        if (det_cw)
          ncw[ix][iy]++;

        /* Get statistics of perturbations... */
        mean[ix][iy] += pert->pt[itrack][ixtrack];
        var[ix][iy] += gsl_pow_2(pert->pt[itrack][ixtrack]);
        max[ix][iy] = GSL_MAX(max[ix][iy], pert->pt[itrack][ixtrack]);
        min[ix][iy] = GSL_MIN(min[ix][iy], pert->pt[itrack][ixtrack]);

        /* Get statistics of brightness temperatures... */
        bt[ix][iy] += pert->bt[itrack][ixtrack];
        bt_8mu[ix][iy] += pert->dc[itrack][ixtrack];
        if (n[ix][iy] > 1) {
          bt_8mu_min[ix][iy]
            = GSL_MIN(bt_8mu_min[ix][iy], pert->dc[itrack][ixtrack]);
          bt_8mu_max[ix][iy]
            = GSL_MAX(bt_8mu_max[ix][iy], pert->dc[itrack][ixtrack]);
        } else {
          bt_8mu_min[ix][iy] = pert->dc[itrack][ixtrack];
          bt_8mu_max[ix][iy] = pert->dc[itrack][ixtrack];
        }

        /* Get mean time... */
        mtime[ix][iy] += pert->time[itrack][ixtrack];
      }
  }

  /* Analyze results... */
  for (ix = 0; ix < nx; ix++)
    for (iy = 0; iy < ny; iy++) {

      /* Get geolocation... */
      mtime[ix][iy] /= (double) n[ix][iy];
      glon[ix] = lon0 + (ix + 0.5) / (double) nx *(
  lon1 - lon0);
      glat[iy] = lat0 + (iy + 0.5) / (double) ny *(
  lat1 - lat0);

      /* Normalize brightness temperatures... */
      bt[ix][iy] /= (double) n[ix][iy];
      bt_8mu[ix][iy] /= (double) n[ix][iy];

      /* Get fractions... */
      fdc[ix][iy] = (double) ndc[ix][iy] / (double) n[ix][iy] * 100.;
      fwg[ix][iy] = (double) nwg[ix][iy] / (double) ndc[ix][iy] * 100.;
      fgw[ix][iy] = (double) ngw[ix][iy] / (double) n[ix][iy] * 100.;
      fcw[ix][iy] = (double) ncw[ix][iy] / (double) ngw[ix][iy] * 100.;

      /* Check number of observations... */
      if (n[ix][iy] < nmin) {
        fdc[ix][iy] = GSL_NAN;
        fwg[ix][iy] = GSL_NAN;
        fgw[ix][iy] = GSL_NAN;
        fcw[ix][iy] = GSL_NAN;
```

```
        bt_8mu[ix][iy] = GSL_NAN;
        bt_8mu_min[ix][iy] = GSL_NAN;
        bt_8mu_max[ix][iy] = GSL_NAN;
      }

      /* Check detections of deep convection at high latitudes... */
      if (fabs(glat[iy]) > dc_hlat && bt_8mu[ix][iy] <= dc_tlim) {
        fdc[ix][iy] = GSL_NAN;
        fwg[ix][iy] = GSL_NAN;
        fcw[ix][iy] = GSL_NAN;
      }

      /* Estimate noise... */
      if (dt230 > 0) {
        nesr = planck(230.0 + dt230, nu) - planck(230.0, nu);
        dt[ix][iy] =
          brightness(planck(bt[ix][iy], nu) + nesr, nu) - bt[ix][iy];
      }

      /* Get mean perturbation and variance... */
      mean[ix][iy] /= (double) n[ix][iy];
      var[ix][iy] =
        var[ix][iy] / (double) n[ix][iy] - gsl_pow_2(mean[ix][iy]);
    }

  /* Create file... */
  printf("Write variance statistics: %s\n", argv[2]);
  if (!(out = fopen(argv[2], "w")))
    ERRMSG("Cannot create file!");

  /* Write header... */
  fprintf(out,
          "# $1 = time [s]\n"
          "# $2 = longitude [deg]\n"
          "# $3 = latitude [deg]\n"
          "# $4 = number of footprints\n"
          "# $5 = fraction of convection events [%%]\n"
          "# $6 = fraction of wave generating events [%%]\n"
          "# $7 = fraction of gravity wave events [%%]\n"
          "# $8 = fraction of convective wave events [%%]\n"
          "# $9 = mean perturbation [K]\n"
          "# $10 = minimum perturbation [K]\n");
  fprintf(out,
          "# $11 = maximum perturbation [K]\n"
          "# $12 = variance [K^2]\n"
          "# $13 = mean surface temperature [K]\n"
          "# $14 = minimum surface temperature [K]\n"
          "# $15 = maximum surface temperature [K]\n"
          "# $16 = mean background temperature [K]\n"
          "# $17 = noise estimate [K]\n");

  /* Write results... */
  for (iy = 0; iy < ny; iy++) {
    if (iy == 0 || nx > 1)
      fprintf(out, "\n");
    for (ix = 0; ix < nx; ix++)
      fprintf(out, "%.2f %g %g %d %g %g %g %g %g %g %g %g %g %g %g %g %g\n",
              mtime[ix][iy], glon[ix], glat[iy], n[ix][iy],
              fdc[ix][iy], fwg[ix][iy], fgw[ix][iy], fcw[ix][iy],
              mean[ix][iy], min[ix][iy], max[ix][iy], var[ix][iy],
              bt_8mu[ix][iy], bt_8mu_min[ix][iy], bt_8mu_max[ix][iy],
              bt[ix][iy], dt[ix][iy]);
  }

  /* Close file... */
  fclose(out);

  /* Free... */
  free(pert);

  return EXIT_SUCCESS;
}
```

5.83 volcano.c File Reference

Functions

- double get_noise (double bt, double dt250, double nu)
- int main (int argc, char *argv[ ])

5.83.1 Function Documentation

5.83.1.1 double get_noise (double *bt,* double *dt250,* double *nu*)

Definition at line 284 of file volcano.c.

```
            {

  double nesr;

  nesr = planck(250.0 + dt250, nu) - planck(250.0, nu);

  return brightness(planck(bt, nu) + nesr, nu) - bt;
}
```

Here is the call graph for this function:

5.83.1.2 int main (int *argc,* char ∗ *argv[ ]*)

Definition at line 17 of file volcano.c.

```
               {

  FILE *out;

  static airs_rad_gran_t airs_rad_gran;

  static double ci, ci_err, ci_nedt = 0.0783,
    ai_low, ai_low_err, ai_low_bt1, ai_low_bt1_nedt =
    0.3698, ai_low_bt2, ai_low_bt2_nedt =
    0.1177, ai_high, ai_high_err, ai_high_bt1, ai_high_bt1_nedt =
    0.0766, ai_high_bt2, ai_high_bt2_nedt =
    0.3706,
    ai_old, ai_old_err, ai_old_bt1, ai_old_bt1_nedt =
    0.3155, ai_old_bt2, ai_old_bt2_nedt =
    0.1177, si_high, si_high_err, si_high_bt1, si_high_bt1_nedt =
    0.1025, si_high_bt2, si_high_bt2_nedt =
    0.1373, si_low, si_low_err, si_low_bt1, si_low_bt1_nedt =
    0.0799, si_low_bt2, si_low_bt2_nedt =
    0.0909, si_old, si_old_err, si_old_bt1, si_old_bt1_nedt =
    0.1064, si_old_bt2, si_old_bt2_nedt =
    0.0909, si_oper, si_oper_err, si_oper_bt1, si_oper_bt1_nedt =
    0.0884, si_oper_bt2, si_oper_bt2_nedt = 0.1159;

  static int ichan, track, xtrack, iarg, ai_low_nu1 = 641, ai_low_nu2 =
    901, ai_high_nu1 = 1295, ai_high_nu2 = 1162, ai_old_nu1 =
    559, ai_old_nu2 = 901, ci_nu = 1290, si_low_nu1 = 1601, si_low_nu2 =
    1526, si_high_nu1 = 1602, si_high_nu2 = 1551, si_old_nu1 =
    1591, si_old_nu2 = 1526, si_oper_nu1 = 1636, si_oper_nu2 = 1507;

  /* Check arguments... */
  if (argc < 3)
    ERRMSG("Give parameters: <out.tab> <l1b_file1> [<l1b_file2> ...]");

  /* Create file... */
  printf("Write volcanic emission data: %s\n", argv[1]);
  if (!(out = fopen(argv[1], "w")))
    ERRMSG("Cannot create file!");

  /* Loop over HDF files... */
  for (iarg = 2; iarg < argc; iarg++) {

    /* Read AIRS data... */
    printf("Read AIRS Level-1B data file: %s\n", argv[iarg]);
    airs_rad_rdr(argv[iarg], &airs_rad_gran);

    /* Write header... */
    if (iarg == 2) {
      fprintf(out,
              "# $1  = time [s]\n"
              "# $2  = footprint longitude [deg]\n"
              "# $3  = footprint latitude [deg]\n"
              "# $4  = satellite altitude [km]\n"
              "# $5  = satellite longitude [deg]\n"
              "# $6  = satellite latitude [deg]\n");
      fprintf(out,
```

```
            "# $7  = cloud index, BT(%.2f/cm) [K]\n"
            "# $8  = cloud index error [K]\n"
            "# $9  = ash index (low wavenumbers),"
            " BT(%.2f/cm) - BT(%.2f/cm) [K]\n"
            "# $10 = ash index (low wavenumbers) error [K]\n"
            "# $11 = ash index (high wavenumbers),"
            " BT(%.2f/cm) - BT(%.2f/cm) [K]\n"
            "# $12 = ash index (high wavenumbers) error [K]\n"
            "# $13 = ash index (Hoffmann et al., 2014),"
            " BT(%.2f/cm) - BT(%.2f/cm) [K]\n"
            "# $14 = ash index (Hoffmann et al., 2014) error [K]\n",
            airs_rad_gran.nominal_freq[ci_nu],
            airs_rad_gran.nominal_freq[ai_low_nu1],
            airs_rad_gran.nominal_freq[ai_low_nu2],
            airs_rad_gran.nominal_freq[ai_high_nu1],
            airs_rad_gran.nominal_freq[ai_high_nu2],
            airs_rad_gran.nominal_freq[ai_old_nu1],
            airs_rad_gran.nominal_freq[ai_old_nu2]);
    fprintf(out,
            "# $15 = SO2 index (low concentrations),"
            " BT(%.2f/cm) - BT(%.2f/cm) [K]\n"
            "# $16 = SO2 index (low concentrations) error [K]\n"
            "# $17 = SO2 index (high concentrations),"
            " BT(%.2f/cm) - BT(%.2f/cm) [K]\n"
            "# $18 = SO2 index (high concentrations) error [K]\n"
            "# $19 = SO2 index (operational),"
            " BT(%.2f/cm) - BT(%.2f/cm) [K]\n"
            "# $20 = SO2 index (operational) error [K]\n"
            "# $21 = SO2 index (Hoffmann et al., 2014),"
            " BT(%.2f/cm) - BT(%.2f/cm) [K]\n"
            "# $22 = SO2 index (Hoffmann et al., 2014) error [K]\n",
            airs_rad_gran.nominal_freq[si_low_nu1],
            airs_rad_gran.nominal_freq[si_low_nu2],
            airs_rad_gran.nominal_freq[si_high_nu1],
            airs_rad_gran.nominal_freq[si_high_nu2],
            airs_rad_gran.nominal_freq[si_oper_nu1],
            airs_rad_gran.nominal_freq[si_oper_nu2],
            airs_rad_gran.nominal_freq[si_old_nu1],
            airs_rad_gran.nominal_freq[si_old_nu2]);
  }

  /* Flag bad observations... */
  for (track = 0; track < AIRS_RAD_GEOTRACK; track++)
    for (xtrack = 0; xtrack < AIRS_RAD_GEOXTRACK; xtrack++)
      for (ichan = 0; ichan < AIRS_RAD_CHANNEL; ichan++)
        if ((airs_rad_gran.state[track][xtrack] != 0)
            || (airs_rad_gran.ExcludedChans[ichan] > 2)
            || (airs_rad_gran.CalChanSummary[ichan] & 8)
            || (airs_rad_gran.CalChanSummary[ichan] & (32 + 64))
            || (airs_rad_gran.CalFlag[track][ichan] & 16))
          airs_rad_gran.radiances[track][xtrack][ichan] = GSL_NAN;

  /* Loop over scans... */
  for (track = 0; track < AIRS_RAD_GEOTRACK; track++) {

    /* Write output... */
    fprintf(out, "\n");

    /* Loop over footprints... */
    for (xtrack = 0; xtrack < AIRS_RAD_GEOXTRACK; xtrack++) {

      /* cloud index... */
      ci = brightness(airs_rad_gran.radiances[track][xtrack][ci_nu] * 0.001,
                      airs_rad_gran.nominal_freq[ci_nu]);
      ci_err = get_noise(ci, ci_nedt, airs_rad_gran.nominal_freq[ci_nu]);

      /* ash index (low wavenumbers)... */
      ai_low_bt1 =
        brightness(airs_rad_gran.radiances[track][xtrack][ai_low_nu1] *
                   0.001, airs_rad_gran.nominal_freq[ai_low_nu1]);
      ai_low_bt2 =
        brightness(airs_rad_gran.radiances[track][xtrack][ai_low_nu2] *
                   0.001, airs_rad_gran.nominal_freq[ai_low_nu2]);
      ai_low = ai_low_bt1 - ai_low_bt2;
      ai_low_err = sqrt(gsl_pow_2(get_noise(ai_low_bt1, ai_low_bt1_nedt,
                                            airs_rad_gran.nominal_freq
                                            [ai_low_nu1]))
                        +
                        gsl_pow_2(get_noise
                                  (ai_low_bt2, ai_low_bt2_nedt,
                                   airs_rad_gran.nominal_freq
                                   [ai_low_nu2])));

      /* ash index (high wavenumbers)... */
      ai_high_bt1 =
        brightness(airs_rad_gran.radiances[track][xtrack][ai_high_nu1] *
                   0.001, airs_rad_gran.nominal_freq[ai_high_nu1]);
```

```
ai_high_bt2 =
  brightness(airs_rad_gran.radiances[track][xtrack][ai_high_nu2] *
             0.001, airs_rad_gran.nominal_freq[ai_high_nu2]);
ai_high = ai_high_bt1 - ai_high_bt2;
ai_high_err = sqrt(gsl_pow_2(get_noise(ai_high_bt1, ai_high_bt1_nedt,
                                       airs_rad_gran.nominal_freq
                                       [ai_high_nu1]))
                   +
                   gsl_pow_2(get_noise
                             (ai_high_bt2, ai_high_bt2_nedt,
                              airs_rad_gran.nominal_freq
                              [ai_high_nu2])));

/* ash index (old)... */
ai_old_bt1 =
  brightness(airs_rad_gran.radiances[track][xtrack][ai_old_nu1] *
             0.001, airs_rad_gran.nominal_freq[ai_old_nu1]);
ai_old_bt2 =
  brightness(airs_rad_gran.radiances[track][xtrack][ai_old_nu2] *
             0.001, airs_rad_gran.nominal_freq[ai_old_nu2]);
ai_old = ai_old_bt1 - ai_old_bt2;
ai_old_err = sqrt(gsl_pow_2(get_noise(ai_old_bt1, ai_old_bt1_nedt,
                                      airs_rad_gran.nominal_freq
                                      [ai_old_nu1]))
                  +
                  gsl_pow_2(get_noise
                            (ai_old_bt2, ai_old_bt2_nedt,
                             airs_rad_gran.nominal_freq
                             [ai_old_nu2])));

/* SO2 index (low concentrations)... */
si_low_bt1 =
  brightness(airs_rad_gran.radiances[track][xtrack][si_low_nu1] *
             0.001, airs_rad_gran.nominal_freq[si_low_nu1]);
si_low_bt2 =
  brightness(airs_rad_gran.radiances[track][xtrack][si_low_nu2] *
             0.001, airs_rad_gran.nominal_freq[si_low_nu2]);
si_low = si_low_bt1 - si_low_bt2;
si_low_err = sqrt(gsl_pow_2(get_noise(si_low_bt1, si_low_bt1_nedt,
                                      airs_rad_gran.nominal_freq
                                      [si_low_nu1]))
                  +
                  gsl_pow_2(get_noise
                            (si_low_bt2, si_low_bt2_nedt,
                             airs_rad_gran.nominal_freq
                             [si_low_nu2])));

/* SO2 index (high concentrations)... */
si_high_bt1 =
  brightness(airs_rad_gran.radiances[track][xtrack][si_high_nu1] *
             0.001, airs_rad_gran.nominal_freq[si_high_nu1]);
si_high_bt2 =
  brightness(airs_rad_gran.radiances[track][xtrack][si_high_nu2] *
             0.001, airs_rad_gran.nominal_freq[si_high_nu2]);
si_high = si_high_bt1 - si_high_bt2;
si_high_err = sqrt(gsl_pow_2(get_noise(si_high_bt1, si_high_bt1_nedt,
                                       airs_rad_gran.nominal_freq
                                       [si_high_nu1]))
                   +
                   gsl_pow_2(get_noise
                             (si_high_bt2, si_high_bt2_nedt,
                              airs_rad_gran.nominal_freq
                              [si_high_nu2])));

/* SO2 index (operational)... */
si_oper_bt1 =
  brightness(airs_rad_gran.radiances[track][xtrack][si_oper_nu1] *
             0.001, airs_rad_gran.nominal_freq[si_oper_nu1]);
si_oper_bt2 =
  brightness(airs_rad_gran.radiances[track][xtrack][si_oper_nu2] *
             0.001, airs_rad_gran.nominal_freq[si_oper_nu2]);
si_oper = si_oper_bt1 - si_oper_bt2;
si_oper_err = sqrt(gsl_pow_2(get_noise(si_oper_bt1, si_oper_bt1_nedt,
                                       airs_rad_gran.nominal_freq
                                       [si_oper_nu1]))
                   +
                   gsl_pow_2(get_noise
                             (si_oper_bt2, si_oper_bt2_nedt,
                              airs_rad_gran.nominal_freq
                              [si_oper_nu2])));

/* SO2 index (old)... */
si_old_bt1 =
  brightness(airs_rad_gran.radiances[track][xtrack][si_old_nu1] *
             0.001, airs_rad_gran.nominal_freq[si_old_nu1]);
si_old_bt2 =
  brightness(airs_rad_gran.radiances[track][xtrack][si_old_nu2] *
```

```
                        0.001, airs_rad_gran.nominal_freq[si_old_nu2]);
          si_old = si_old_bt1 - si_old_bt2;
          si_old_err = sqrt(gsl_pow_2(get_noise(si_old_bt1, si_old_bt1_nedt,
                                                airs_rad_gran.nominal_freq
                                                [si_old_nu1]))
                            +
                            gsl_pow_2(get_noise
                                      (si_old_bt2, si_old_bt2_nedt,
                                       airs_rad_gran.nominal_freq
                                       [si_old_nu2])));

          /* Write output... */
          fprintf(out,
                  "%.2f %.4f %.4f %.3f %.4f %.4f %.2f %.2f %.2f %.2f %.2f %.2f "
                  "%.2f %.2f %.2f %.2f %.2f %.2f %.2f %.2f %.2f %.2f\n",
                  airs_rad_gran.Time[track][xtrack] - 220838400,
                  airs_rad_gran.Longitude[track][xtrack],
                  airs_rad_gran.Latitude[track][xtrack],
                  airs_rad_gran.satheight[track],
                  airs_rad_gran.sat_lon[track],
                  airs_rad_gran.sat_lat[track],
                  ci, ci_err, ai_low, ai_low_err, ai_high, ai_high_err, ai_old,
                  ai_old_err, si_low, si_low_err, si_high, si_high_err, si_oper,
                  si_oper_err, si_old, si_old_err);
        }
      }
    }

    /* Close file... */
    fclose(out);

    return EXIT_SUCCESS;
}
```

Here is the call graph for this function:

5.84 volcano.c

```
#include "libairs.h"

/* ------------------------------------------------------------
   Functions...
   ------------------------------------------------------------ */

/* Estimate noise. */
double get_noise(
  double bt,
  double dt250,
  double nu);

/* ------------------------------------------------------------
   Main...
   ------------------------------------------------------------ */

int main(
  int argc,
  char *argv[]) {

  FILE *out;

  static airs_rad_gran_t airs_rad_gran;

  static double ci, ci_err, ci_nedt = 0.0783,
    ai_low, ai_low_err, ai_low_bt1, ai_low_bt1_nedt =
    0.3698, ai_low_bt2, ai_low_bt2_nedt =
    0.1177, ai_high, ai_high_err, ai_high_bt1, ai_high_bt1_nedt =
    0.0766, ai_high_bt2, ai_high_bt2_nedt =
    0.3706,
    ai_old, ai_old_err, ai_old_bt1, ai_old_bt1_nedt =
    0.3155, ai_old_bt2, ai_old_bt2_nedt =
    0.1177, si_high, si_high_err, si_high_bt1, si_high_bt1_nedt =
    0.1025, si_high_bt2, si_high_bt2_nedt =
    0.1373, si_low, si_low_err, si_low_bt1, si_low_bt1_nedt =
    0.0799, si_low_bt2, si_low_bt2_nedt =
    0.0909, si_old, si_old_err, si_old_bt1, si_old_bt1_nedt =
    0.1064, si_old_bt2, si_old_bt2_nedt =
    0.0909, si_oper, si_oper_err, si_oper_bt1, si_oper_bt1_nedt =
    0.0884, si_oper_bt2, si_oper_bt2_nedt = 0.1159;

  static int ichan, track, xtrack, iarg, ai_low_nu1 = 641, ai_low_nu2 =
    901, ai_high_nu1 = 1295, ai_high_nu2 = 1162, ai_old_nu1 =
```

```
    559, ai_old_nu2 = 901, ci_nu = 1290, si_low_nu1 = 1601, si_low_nu2 =
    1526, si_high_nu1 = 1602, si_high_nu2 = 1551, si_old_nu1 =
    1591, si_old_nu2 = 1526, si_oper_nu1 = 1636, si_oper_nu2 = 1507;

  /* Check arguments... */
  if (argc < 3)
    ERRMSG("Give parameters: <out.tab> <l1b_file1> [<l1b_file2> ...]");

  /* Create file... */
  printf("Write volcanic emission data: %s\n", argv[1]);
  if (!(out = fopen(argv[1], "w")))
    ERRMSG("Cannot create file!");

  /* Loop over HDF files... */
  for (iarg = 2; iarg < argc; iarg++) {

    /* Read AIRS data... */
    printf("Read AIRS Level-1B data file: %s\n", argv[iarg]);
    airs_rad_rdr(argv[iarg], &airs_rad_gran);

    /* Write header... */
    if (iarg == 2) {
      fprintf(out,
              "# $1  = time [s]\n"
              "# $2  = footprint longitude [deg]\n"
              "# $3  = footprint latitude [deg]\n"
              "# $4  = satellite altitude [km]\n"
              "# $5  = satellite longitude [deg]\n"
              "# $6  = satellite latitude [deg]\n");
      fprintf(out,
              "# $7  = cloud index, BT(%.2f/cm) [K]\n"
              "# $8  = cloud index error [K]\n"
              "# $9  = ash index (low wavenumbers),"
              " BT(%.2f/cm) - BT(%.2f/cm) [K]\n"
              "# $10 = ash index (low wavenumbers) error [K]\n"
              "# $11 = ash index (high wavenumbers),"
              " BT(%.2f/cm) - BT(%.2f/cm) [K]\n"
              "# $12 = ash index (high wavenumbers) error [K]\n"
              "# $13 = ash index (Hoffmann et al., 2014),"
              " BT(%.2f/cm) - BT(%.2f/cm) [K]\n"
              "# $14 = ash index (Hoffmann et al., 2014) error [K]\n",
              airs_rad_gran.nominal_freq[ci_nu],
              airs_rad_gran.nominal_freq[ai_low_nu1],
              airs_rad_gran.nominal_freq[ai_low_nu2],
              airs_rad_gran.nominal_freq[ai_high_nu1],
              airs_rad_gran.nominal_freq[ai_high_nu2],
              airs_rad_gran.nominal_freq[ai_old_nu1],
              airs_rad_gran.nominal_freq[ai_old_nu2]);
      fprintf(out,
              "# $15 = SO2 index (low concentrations),"
              " BT(%.2f/cm) - BT(%.2f/cm) [K]\n"
              "# $16 = SO2 index (low concentrations) error [K]\n"
              "# $17 = SO2 index (high concentrations),"
              " BT(%.2f/cm) - BT(%.2f/cm) [K]\n"
              "# $18 = SO2 index (high concentrations) error [K]\n"
              "# $19 = SO2 index (operational),"
              " BT(%.2f/cm) - BT(%.2f/cm) [K]\n"
              "# $20 = SO2 index (operational) error [K]\n"
              "# $21 = SO2 index (Hoffmann et al., 2014),"
              " BT(%.2f/cm) - BT(%.2f/cm) [K]\n"
              "# $22 = SO2 index (Hoffmann et al., 2014) error [K]\n",
              airs_rad_gran.nominal_freq[si_low_nu1],
              airs_rad_gran.nominal_freq[si_low_nu2],
              airs_rad_gran.nominal_freq[si_high_nu1],
              airs_rad_gran.nominal_freq[si_high_nu2],
              airs_rad_gran.nominal_freq[si_oper_nu1],
              airs_rad_gran.nominal_freq[si_oper_nu2],
              airs_rad_gran.nominal_freq[si_old_nu1],
              airs_rad_gran.nominal_freq[si_old_nu2]);
    }

    /* Flag bad observations... */
    for (track = 0; track < AIRS_RAD_GEOTRACK; track++)
      for (xtrack = 0; xtrack < AIRS_RAD_GEOXTRACK; xtrack++)
        for (ichan = 0; ichan < AIRS_RAD_CHANNEL; ichan++)
          if ((airs_rad_gran.state[track][xtrack] != 0)
              || (airs_rad_gran.ExcludedChans[ichan] > 2)
              || (airs_rad_gran.CalChanSummary[ichan] & 8)
              || (airs_rad_gran.CalChanSummary[ichan] & (32 + 64))
              || (airs_rad_gran.CalFlag[track][ichan] & 16))
            airs_rad_gran.radiances[track][xtrack][ichan] = GSL_NAN;

    /* Loop over scans... */
    for (track = 0; track < AIRS_RAD_GEOTRACK; track++) {

      /* Write output... */
      fprintf(out, "\n");
```

```
      /* Loop over footprints... */
      for (xtrack = 0; xtrack < AIRS_RAD_GEOXTRACK; xtrack++) {

        /* cloud index... */
        ci = brightness(airs_rad_gran.radiances[track][xtrack][ci_nu] * 0.001,
                        airs_rad_gran.nominal_freq[ci_nu]);
        ci_err = get_noise(ci, ci_nedt, airs_rad_gran.nominal_freq[ci_nu]);

        /* ash index (low wavenumbers)... */
        ai_low_bt1 =
          brightness(airs_rad_gran.radiances[track][xtrack][ai_low_nu1] *
                     0.001, airs_rad_gran.nominal_freq[ai_low_nu1]);
        ai_low_bt2 =
          brightness(airs_rad_gran.radiances[track][xtrack][ai_low_nu2] *
                     0.001, airs_rad_gran.nominal_freq[ai_low_nu2]);
        ai_low = ai_low_bt1 - ai_low_bt2;
        ai_low_err = sqrt(gsl_pow_2(get_noise(ai_low_bt1, ai_low_bt1_nedt,
                                              airs_rad_gran.nominal_freq
                                              [ai_low_nu1]))
                          +
                          gsl_pow_2(get_noise
                                    (ai_low_bt2, ai_low_bt2_nedt,
                                     airs_rad_gran.nominal_freq
                                     [ai_low_nu2])));

        /* ash index (high wavenumbers)... */
        ai_high_bt1 =
          brightness(airs_rad_gran.radiances[track][xtrack][ai_high_nu1] *
                     0.001, airs_rad_gran.nominal_freq[ai_high_nu1]);
        ai_high_bt2 =
          brightness(airs_rad_gran.radiances[track][xtrack][ai_high_nu2] *
                     0.001, airs_rad_gran.nominal_freq[ai_high_nu2]);
        ai_high = ai_high_bt1 - ai_high_bt2;
        ai_high_err = sqrt(gsl_pow_2(get_noise(ai_high_bt1, ai_high_bt1_nedt,
                                               airs_rad_gran.nominal_freq
                                               [ai_high_nu1]))
                           +
                           gsl_pow_2(get_noise
                                     (ai_high_bt2, ai_high_bt2_nedt,
                                      airs_rad_gran.nominal_freq
                                      [ai_high_nu2])));

        /* ash index (old)... */
        ai_old_bt1 =
          brightness(airs_rad_gran.radiances[track][xtrack][ai_old_nu1] *
                     0.001, airs_rad_gran.nominal_freq[ai_old_nu1]);
        ai_old_bt2 =
          brightness(airs_rad_gran.radiances[track][xtrack][ai_old_nu2] *
                     0.001, airs_rad_gran.nominal_freq[ai_old_nu2]);
        ai_old = ai_old_bt1 - ai_old_bt2;
        ai_old_err = sqrt(gsl_pow_2(get_noise(ai_old_bt1, ai_old_bt1_nedt,
                                              airs_rad_gran.nominal_freq
                                              [ai_old_nu1]))
                          +
                          gsl_pow_2(get_noise
                                    (ai_old_bt2, ai_old_bt2_nedt,
                                     airs_rad_gran.nominal_freq
                                     [ai_old_nu2])));

        /* SO2 index (low concentrations)... */
        si_low_bt1 =
          brightness(airs_rad_gran.radiances[track][xtrack][si_low_nu1] *
                     0.001, airs_rad_gran.nominal_freq[si_low_nu1]);
        si_low_bt2 =
          brightness(airs_rad_gran.radiances[track][xtrack][si_low_nu2] *
                     0.001, airs_rad_gran.nominal_freq[si_low_nu2]);
        si_low = si_low_bt1 - si_low_bt2;
        si_low_err = sqrt(gsl_pow_2(get_noise(si_low_bt1, si_low_bt1_nedt,
                                              airs_rad_gran.nominal_freq
                                              [si_low_nu1]))
                          +
                          gsl_pow_2(get_noise
                                    (si_low_bt2, si_low_bt2_nedt,
                                     airs_rad_gran.nominal_freq
                                     [si_low_nu2])));

        /* SO2 index (high concentrations)... */
        si_high_bt1 =
          brightness(airs_rad_gran.radiances[track][xtrack][si_high_nu1] *
                     0.001, airs_rad_gran.nominal_freq[si_high_nu1]);
        si_high_bt2 =
          brightness(airs_rad_gran.radiances[track][xtrack][si_high_nu2] *
                     0.001, airs_rad_gran.nominal_freq[si_high_nu2]);
        si_high = si_high_bt1 - si_high_bt2;
        si_high_err = sqrt(gsl_pow_2(get_noise(si_high_bt1, si_high_bt1_nedt,
                                               airs_rad_gran.nominal_freq
```

```
                                               [si_high_nu1]))
                           +
                           gsl_pow_2(get_noise
                                     (si_high_bt2, si_high_bt2_nedt,
                                      airs_rad_gran.nominal_freq
                                      [si_high_nu2])));

        /* SO2 index (operational)... */
        si_oper_bt1 =
          brightness(airs_rad_gran.radiances[track][xtrack][si_oper_nu1] *
                     0.001, airs_rad_gran.nominal_freq[si_oper_nu1]);
        si_oper_bt2 =
          brightness(airs_rad_gran.radiances[track][xtrack][si_oper_nu2] *
                     0.001, airs_rad_gran.nominal_freq[si_oper_nu2]);
        si_oper = si_oper_bt1 - si_oper_bt2;
        si_oper_err = sqrt(gsl_pow_2(get_noise(si_oper_bt1, si_oper_bt1_nedt,
                                               airs_rad_gran.nominal_freq
                                               [si_oper_nu1]))
                           +
                           gsl_pow_2(get_noise
                                     (si_oper_bt2, si_oper_bt2_nedt,
                                      airs_rad_gran.nominal_freq
                                      [si_oper_nu2])));

        /* SO2 index (old)... */
        si_old_bt1 =
          brightness(airs_rad_gran.radiances[track][xtrack][si_old_nu1] *
                     0.001, airs_rad_gran.nominal_freq[si_old_nu1]);
        si_old_bt2 =
          brightness(airs_rad_gran.radiances[track][xtrack][si_old_nu2] *
                     0.001, airs_rad_gran.nominal_freq[si_old_nu2]);
        si_old = si_old_bt1 - si_old_bt2;
        si_old_err = sqrt(gsl_pow_2(get_noise(si_old_bt1, si_old_bt1_nedt,
                                              airs_rad_gran.nominal_freq
                                              [si_old_nu1]))
                          +
                          gsl_pow_2(get_noise
                                    (si_old_bt2, si_old_bt2_nedt,
                                     airs_rad_gran.nominal_freq
                                     [si_old_nu2])));

        /* Write output... */
        fprintf(out,
                "%.2f %.4f %.4f %.3f %.4f %.4f %.2f %.2f %.2f %.2f %.2f %.2f "
                "%.2f %.2f %.2f %.2f %.2f %.2f %.2f %.2f %.2f %.2f\n",
                airs_rad_gran.Time[track][xtrack] - 220838400,
                airs_rad_gran.Longitude[track][xtrack],
                airs_rad_gran.Latitude[track][xtrack],
                airs_rad_gran.satheight[track],
                airs_rad_gran.sat_lon[track],
                airs_rad_gran.sat_lat[track],
                ci, ci_err, ai_low, ai_low_err, ai_high, ai_high_err, ai_old,
                ai_old_err, si_low, si_low_err, si_high, si_high_err, si_oper,
                si_oper_err, si_old, si_old_err);
      }
    }
  }

  /* Close file... */
  fclose(out);

  return EXIT_SUCCESS;
}

/*****************************************************************************/

double get_noise(
  double bt,
  double dt250,
  double nu) {

  double nesr;

  nesr = planck(250.0 + dt250, nu) - planck(250.0, nu);

  return brightness(planck(bt, nu) + nesr, nu) - bt;
}
```

5.85 wrffm.c File Reference

Functions

- void background (double temp[NLAT][NLON], double pt[NLAT][NLON], int nlat, int nlon, int dlat, int dlon)
- int main (int argc, char *argv[ ])

5.85.1 Function Documentation

5.85.1.1 void background (double *temp[NLAT][NLON]*, double *pt[NLAT][NLON]*, int *nlat*, int *nlon*, int *dlat*, int *dlon*)

Definition at line 466 of file wrffm.c.

```
                  {

  static double data[NLAT * NLAT];

  int ilon, ilat, ilon2, ilat2, n;

  /* Loop over grid points... */
  for (ilat = 0; ilat < nlat; ilat++)
    for (ilon = 0; ilon < nlon; ilon++) {

      /* Init... */
      n = 0;

      /* Average... */
      for (ilat2 = GSL_MAX(ilat - dlat, 0);
           ilat2 <= GSL_MIN(ilat + dlat, nlat - 1); ilat2++)
        for (ilon2 = GSL_MAX(ilon - dlon, 0);
             ilon2 <= GSL_MIN(ilon + dlon, nlon - 1); ilon2++)
          if (gsl_finite(temp[ilat2][ilon2])) {
            data[n] = temp[ilat2][ilon2];
            n++;
          }

      /* Set perturbation... */
      gsl_sort(data, 1, (size_t) n);
      pt[ilat][ilon] = temp[ilat][ilon]
        - gsl_stats_median_from_sorted_data(data, 1, (size_t) n);
    }
}
```

5.85.1.2 int main (int *argc*, char ∗ *argv[ ]*)

Definition at line 29 of file wrffm.c.

```
                        {

  static ctl_t ctl;
  static atm_t atm;
  static obs_t obs;

  static double z[NZ][NLAT][NLON], z0[NZ][NLAT][NLON],
    p[NZ][NLAT][NLON], p0[NZ][NLAT][NLON], t[NZ][NLAT][NLON],
    lon[NLAT][NLON], lat[NLAT][NLON], temp[NLAT][NLON], pt[NLAT][NLON],
    x0[3], x1[NLAT][NLON][3], w, wsum, rmax2 = 50. * 50., fwhm = 20., var_dh;

  static int id, ix, iy, oit, ncid, dimid, varid, ilon, ilat, iz,
    ncrop, nlon, nlat, nz, nz2, ntime;

  static size_t start[10], count[10], rs;

  wave_t *wave_airs, *wave_wrf;

  FILE *out;

  /* ------------------------------------------------------------
     Get control parameters...
     ------------------------------------------------------------ */

  /* Check arguments... */
  if (argc < 6)
    ERRMSG("Give parameters: <ctl> <wrf.nc> <it> <wrf.tab> <rad.tab> "
           "<wave_airs.tab> <wave_wrf.tab>");

  /* Get arguments... */
  oit = atoi(argv[3]);

  /* Read control parameters... */
  read_ctl(argc, argv, &ctl);

  /* Set control parameters... */
```

```
  ctl.write_bbt = 1;

  /* Get control parameters... */
  ncrop = (int) scan_ctl(argc, argv, "NCROP", -1, "0", NULL);
  var_dh = scan_ctl(argc, argv, "VAR_DH", -1, "0", NULL);

  /* Allocate... */
  ALLOC(wave_airs, wave_t, 1);
  ALLOC(wave_wrf, wave_t, 1);

  /* ------------------------------------------------------------
     Read WRF data...
     ------------------------------------------------------------ */

  /* Open file... */
  printf("Read WRF data: %s\n", argv[2]);
  NC(nc_open(argv[2], NC_NOWRITE, &ncid));

  /* Get dimensions... */
  NC(nc_inq_dimid(ncid, "Time", &dimid));
  NC(nc_inq_dimlen(ncid, dimid, &rs));
  ntime = (int) rs;
  if (oit >= ntime)
    ERRMSG("Timestep out of range!");

  NC(nc_inq_dimid(ncid, "bottom_top", &dimid));
  NC(nc_inq_dimlen(ncid, dimid, &rs));
  nz = (int) rs;
  if (nz > NZ)
    ERRMSG("Too many altitudes!");

  NC(nc_inq_dimid(ncid, "bottom_top_stag", &dimid));
  NC(nc_inq_dimlen(ncid, dimid, &rs));
  nz2 = (int) rs;
  if (nz2 > NZ)
    ERRMSG("Too many altitudes!");

  NC(nc_inq_dimid(ncid, "south_north", &dimid));
  NC(nc_inq_dimlen(ncid, dimid, &rs));
  nlat = (int) rs;
  if (nlat > NLAT)
    ERRMSG("Too many latitudes!");

  NC(nc_inq_dimid(ncid, "west_east", &dimid));
  NC(nc_inq_dimlen(ncid, dimid, &rs));
  nlon = (int) rs;
  if (nlon > NLON)
    ERRMSG("Too many longitudes!");

  /* Read latitudes... */
  NC(nc_inq_varid(ncid, "XLAT", &varid));
  for (ilat = 0; ilat < nlat; ilat++) {
    start[0] = (size_t) oit;
    start[1] = (size_t) ilat;
    start[2] = 0;
    count[0] = 1;
    count[1] = 1;
    count[2] = (size_t) nlon;
    NC(nc_get_vara_double(ncid, varid, start, count, lat[ilat]));
  }

  /* Read longitudes... */
  NC(nc_inq_varid(ncid, "XLONG", &varid));
  for (ilat = 0; ilat < nlat; ilat++) {
    start[0] = (size_t) oit;
    start[1] = (size_t) ilat;
    start[2] = 0;
    count[0] = 1;
    count[1] = 1;
    count[2] = (size_t) nlon;
    NC(nc_get_vara_double(ncid, varid, start, count, lon[ilat]));
  }

  /* Read theta perturbation... */
  NC(nc_inq_varid(ncid, "T", &varid));
  for (iz = 0; iz < nz; iz++)
    for (ilat = 0; ilat < nlat; ilat++) {
      start[0] = (size_t) oit;
      start[1] = (size_t) iz;
      start[2] = (size_t) ilat;
      start[3] = 0;
      count[0] = 1;
      count[1] = 1;
      count[2] = 1;
      count[3] = (size_t) nlon;
      NC(nc_get_vara_double(ncid, varid, start, count, t[iz][ilat]));
    }
```

```
  /* Read geopotential perturbation... */
  NC(nc_inq_varid(ncid, "PH", &varid));
  for (iz = 0; iz < nz2; iz++)
    for (ilat = 0; ilat < nlat; ilat++) {
      start[0] = (size_t) oit;
      start[1] = (size_t) iz;
      start[2] = (size_t) ilat;
      start[3] = 0;
      count[0] = 1;
      count[1] = 1;
      count[2] = 1;
      count[3] = (size_t) nlon;
      NC(nc_get_vara_double(ncid, varid, start, count, z[iz][ilat]));
    }

  /* Read geopotential base... */
  NC(nc_inq_varid(ncid, "PHB", &varid));
  for (iz = 0; iz < nz2; iz++)
    for (ilat = 0; ilat < nlat; ilat++) {
      start[0] = (size_t) oit;
      start[1] = (size_t) iz;
      start[2] = (size_t) ilat;
      start[3] = 0;
      count[0] = 1;
      count[1] = 1;
      count[2] = 1;
      count[3] = (size_t) nlon;
      NC(nc_get_vara_double(ncid, varid, start, count, z0[iz][ilat]));
    }

  /* Read pressure perturbation... */
  NC(nc_inq_varid(ncid, "P", &varid));
  for (iz = 0; iz < nz; iz++)
    for (ilat = 0; ilat < nlat; ilat++) {
      start[0] = (size_t) oit;
      start[1] = (size_t) iz;
      start[2] = (size_t) ilat;
      start[3] = 0;
      count[0] = 1;
      count[1] = 1;
      count[2] = 1;
      count[3] = (size_t) nlon;
      NC(nc_get_vara_double(ncid, varid, start, count, p[iz][ilat]));
    }

  /* Read pressure base... */
  NC(nc_inq_varid(ncid, "PB", &varid));
  for (iz = 0; iz < nz; iz++)
    for (ilat = 0; ilat < nlat; ilat++) {
      start[0] = (size_t) oit;
      start[1] = (size_t) iz;
      start[2] = (size_t) ilat;
      start[3] = 0;
      count[0] = 1;
      count[1] = 1;
      count[2] = 1;
      count[3] = (size_t) nlon;
      NC(nc_get_vara_double(ncid, varid, start, count, p0[iz][ilat]));
    }

  /* Close file... */
  NC(nc_close(ncid));

  /* ------------------------------------------------------------
     Convert WRF data...
     ------------------------------------------------------------ */

  /* Adjust longitudes... */
  for (ilat = 0; ilat < nlat; ilat++)
    for (ilon = 0; ilon < nlon; ilon++)
      if (lon[ilat][ilon] > 180)
        lon[ilat][ilon] -= 360;

  /* Get altitudes... */
  for (iz = 0; iz < nz; iz++)
    for (ilat = 0; ilat < nlat; ilat++)
      for (ilon = 0; ilon < nlon; ilon++)
        z[iz][ilat][ilon]
          = 0.5 * (z[iz + 1][ilat][ilon] + z0[iz + 1][ilat][ilon]
                   + z[iz][ilat][ilon] + z0[iz][ilat][ilon]) / G0 / 1000.;

  /* Get pressure... */
  for (iz = 0; iz < nz; iz++)
    for (ilat = 0; ilat < nlat; ilat++)
      for (ilon = 0; ilon < nlon; ilon++)
        p[iz][ilat][ilon]
```

```
        = (p[iz][ilat][ilon] + p0[iz][ilat][ilon]) / 100.;

  /* Get temperature... */
  for (iz = 0; iz < nz; iz++)
    for (ilat = 0; ilat < nlat; ilat++)
      for (ilon = 0; ilon < nlon; ilon++)
        t[iz][ilat][ilon]
          = (t[iz][ilat][ilon] + 300.) / pow(1000. / p[iz][ilat][ilon],
                                             0.286);

  /* ------------------------------------------------------------
     Write WRF data to ASCII...
     ------------------------------------------------------------ */

  /* Check filename... */
  if (argv[4][0] != '-') {

    /* Create file... */
    printf("Write WRF data: %s\n", argv[4]);
    if (!(out = fopen(argv[4], "w")))
      ERRMSG("Cannot create file!");

    /* Write header... */
    fprintf(out,
            "# $1 = altitude index\n"
            "# $2 = altitude [km]\n"
            "# $3 = longitude [deg]\n"
            "# $4 = latitude [deg]\n"
            "# $5 = pressure [hPa]\n" "# $6 = temperature [K]\n");

    /* Write output... */
    for (iz = 0; iz < nz; iz++)
      for (ilon = 0; ilon < nlon; ilon++) {
        fprintf(out, "\n");
        for (ilat = 0; ilat < nlat; ilat++)
          fprintf(out, "%d %g %g %g %g %g\n", iz, z[iz][ilat][ilon],
                  lon[ilat][ilon], lat[ilat][ilon],
                  p[iz][ilat][ilon], t[iz][ilat][ilon]);
      }

    /* Close file... */
    fclose(out);
  }

  /* ------------------------------------------------------------
     Run forward model...
     ------------------------------------------------------------ */

  /* Loop over latitudes... */
  for (ilat = 0; ilat < nlat; ilat++) {

    /* Write info... */
    printf("  Compute latitude %d / %d ...\n", ilat + 1, nlat);

    /* Loop over longitudes... */
    for (ilon = 0; ilon < nlon; ilon++) {

      /* Set altitude levels... */
      atm.np = 0;
      for (iz = 0; iz < nz; iz++)
        if (gsl_finite(gsl_finite(t[iz][ilat][ilon]))
            && t[iz][ilat][ilon] > 100 && t[iz][ilat][ilon] < 400
            && z[iz][ilat][ilon] > 10) {
          atm.z[atm.np] = z[iz][ilat][ilon];
          if ((++atm.np) >= NP)
            ERRMSG("Too many altitudes!");
        }

      /* Add top level... */
      atm.z[atm.np] = 90.;
      if ((++atm.np) >= NP)
        ERRMSG("Too many altitudes!");

      /* Initialize with climatological data... */
      climatology(&ctl, &atm);

      /* Set temperature and pressure... */
      atm.np = 0;
      for (iz = 0; iz < nz; iz++)
        if (gsl_finite(t[iz][ilat][ilon])
            && t[iz][ilat][ilon] > 100 && t[iz][ilat][ilon] < 400
            && z[iz][ilat][ilon] > 10) {
          atm.p[atm.np] = p[iz][ilat][ilon];
          atm.t[atm.np] = t[iz][ilat][ilon];
          atm.np++;
        }

```

```
      /* Add top level... */
      atm.np++;

      /* Set observation data... */
      obs.nr = 1;
      obs.obsz[0] = 700;

      /* Run forward model... */
      formod(&ctl, &atm, &obs);

      /* Get mean brightness temperature... */
      temp[ilat][ilon] = 0;
      for (id = 0; id < ctl.nd; id++)
        temp[ilat][ilon] += obs.rad[id][0] / ctl.nd;
    }
  }

  /* Crop at boundaries... */
  for (ilat = 0; ilat < ncrop; ilat++)
    for (ilon = 0; ilon < nlon; ilon++)
      temp[ilat][ilon] = GSL_NAN;
  for (ilat = nlat - ncrop; ilat < nlat; ilat++)
    for (ilon = 0; ilon < nlon; ilon++)
      temp[ilat][ilon] = GSL_NAN;
  for (ilon = 0; ilon < ncrop; ilon++)
    for (ilat = 0; ilat < nlat; ilat++)
      temp[ilat][ilon] = GSL_NAN;
  for (ilon = nlon - ncrop; ilon < nlon; ilon++)
    for (ilat = 0; ilat < nlat; ilat++)
      temp[ilat][ilon] = GSL_NAN;

  /* Get perturbations... */
  background(temp, pt, nlat, nlon, 10, 10);

  /* ------------------------------------------------------------
     Save forward model output...
     ------------------------------------------------------------ */

  /* Check filename... */
  if (argv[5][0] != '-') {

    /* Create file... */
    printf("Write radiance data: %s\n", argv[5]);
    if (!(out = fopen(argv[5], "w")))
      ERRMSG("Cannot create file!");

    /* Write header... */
    fprintf(out,
            "# $1 = longitude [deg]\n"
            "# $2 = latitude [deg]\n"
            "# $3 = WRF brightness temperature [K]\n"
            "# $4 = WRF brightness temperature perturbation [K]\n");

    /* Write output... */
    for (ilat = 0; ilat < nlat; ilat++) {
      fprintf(out, "\n");
      for (ilon = 0; ilon < nlon; ilon++)
        fprintf(out, "%g %g %g %g\n", lon[ilat][ilon], lat[ilat][ilon],
                temp[ilat][ilon], pt[ilat][ilon]);
    }

    /* Close file... */
    fclose(out);
  }

  /* ------------------------------------------------------------
     Read AIRS radiance map and resample model data...
     ------------------------------------------------------------ */

  /* Check filename... */
  if (argv[6][0] != '-') {

    /* Read AIRS wave file... */
    read_wave(argv[6], wave_airs);
    memcpy(wave_wrf, wave_airs, sizeof(wave_t));

    /* Get Cartesian coordinates for model grid... */
    for (ilat = 0; ilat < nlat; ilat++)
      for (ilon = 0; ilon < nlon; ilon++)
        geo2cart(0, lon[ilat][ilon], lat[ilat][ilon], x1[ilat][ilon]);

    /* Loop over AIRS geolocations... */
    for (ix = 0; ix < wave_airs->nx; ix++)
      for (iy = 0; iy < wave_airs->ny; iy++) {

        /* Write info... */
        if (iy == 0)
```

```
          printf("  Average for xtrack %d / %d ...\n", ix + 1, wave_airs->nx);

        /* Init... */
        wsum = 0;
        wave_wrf->temp[ix][iy] = 0;
        wave_wrf->bg[ix][iy] = 0;
        wave_wrf->pt[ix][iy] = 0;
        wave_wrf->var[ix][iy] = 0;

        /* Average... */
        geo2cart(0, wave_airs->lon[ix][iy], wave_airs->lat[ix][iy], x0);
        for (ilat = 0; ilat < nlat; ilat++)
          for (ilon = 0; ilon < nlon; ilon++)
            if (DIST2(x0, x1[ilat][ilon]) <= rmax2) {
              w =
                exp(-DIST2(x0, x1[ilat][ilon]) /
                    (2. * gsl_pow_2(fwhm / 2.3548)));
              wave_wrf->temp[ix][iy] += w * temp[ilat][ilon];
              wave_wrf->bg[ix][iy] += w * (temp[ilat][ilon] - pt[ilat][ilon]);
              wave_wrf->pt[ix][iy] += w * pt[ilat][ilon];
              wsum += w;
            }

        /* Normalize... */
        if (wsum > 0) {
          wave_wrf->temp[ix][iy] /= wsum;
          wave_wrf->bg[ix][iy] /= wsum;
          wave_wrf->pt[ix][iy] /= wsum;
        } else {
          wave_wrf->temp[ix][iy] = GSL_NAN;
          wave_wrf->bg[ix][iy] = GSL_NAN;
          wave_wrf->pt[ix][iy] = GSL_NAN;
        }
      }

    /* Compute variance... */
    variance(wave_wrf, var_dh);

    /* Write WRF wave struct... */
    write_wave(argv[7], wave_wrf);
  }

  /* Free... */
  free(wave_airs);
  free(wave_wrf);

  return EXIT_SUCCESS;
}
```

Here is the call graph for this function:

5.86 wrffm.c

```
#include "libairs.h"

/* ------------------------------------------------------------
   Dimensions...
   ------------------------------------------------------------ */

/* Maximum WRF dimensions. */
#define NLON 450
#define NLAT 450
#define NZ 150

/* ------------------------------------------------------------
   Functions...
   ------------------------------------------------------------ */

/* Estimate background... */
void background(
  double temp[NLAT][NLON],
  double pt[NLAT][NLON],
  int nlat,
  int nlon,
  int dlat,
  int dlon);

/* ------------------------------------------------------------
   Main...
   ------------------------------------------------------------ */

```

```
int main(
  int argc,
  char *argv[]) {

  static ctl_t ctl;
  static atm_t atm;
  static obs_t obs;

  static double z[NZ][NLAT][NLON], z0[NZ][NLAT][NLON],
    p[NZ][NLAT][NLON], p0[NZ][NLAT][NLON], t[NZ][NLAT][NLON],
    lon[NLAT][NLON], lat[NLAT][NLON], temp[NLAT][NLON], pt[NLAT][NLON],
    x0[3], x1[NLAT][NLON][3], w, wsum, rmax2 = 50. * 50., fwhm = 20., var_dh;

  static int id, ix, iy, oit, ncid, dimid, varid, ilon, ilat, iz,
    ncrop, nlon, nlat, nz, nz2, ntime;

  static size_t start[10], count[10], rs;

  wave_t *wave_airs, *wave_wrf;

  FILE *out;

  /* ------------------------------------------------------------
     Get control parameters...
     ------------------------------------------------------------ */

  /* Check arguments... */
  if (argc < 6)
    ERRMSG("Give parameters: <ctl> <wrf.nc> <it> <wrf.tab> <rad.tab> "
           "<wave_airs.tab> <wave_wrf.tab>");

  /* Get arguments... */
  oit = atoi(argv[3]);

  /* Read control parameters... */
  read_ctl(argc, argv, &ctl);

  /* Set control parameters... */
  ctl.write_bbt = 1;

  /* Get control parameters... */
  ncrop = (int) scan_ctl(argc, argv, "NCROP", -1, "0", NULL);
  var_dh = scan_ctl(argc, argv, "VAR_DH", -1, "0", NULL);

  /* Allocate... */
  ALLOC(wave_airs, wave_t, 1);
  ALLOC(wave_wrf, wave_t, 1);

  /* ------------------------------------------------------------
     Read WRF data...
     ------------------------------------------------------------ */

  /* Open file... */
  printf("Read WRF data: %s\n", argv[2]);
  NC(nc_open(argv[2], NC_NOWRITE, &ncid));

  /* Get dimensions... */
  NC(nc_inq_dimid(ncid, "Time", &dimid));
  NC(nc_inq_dimlen(ncid, dimid, &rs));
  ntime = (int) rs;
  if (oit >= ntime)
    ERRMSG("Timestep out of range!");

  NC(nc_inq_dimid(ncid, "bottom_top", &dimid));
  NC(nc_inq_dimlen(ncid, dimid, &rs));
  nz = (int) rs;
  if (nz > NZ)
    ERRMSG("Too many altitudes!");

  NC(nc_inq_dimid(ncid, "bottom_top_stag", &dimid));
  NC(nc_inq_dimlen(ncid, dimid, &rs));
  nz2 = (int) rs;
  if (nz2 > NZ)
    ERRMSG("Too many altitudes!");

  NC(nc_inq_dimid(ncid, "south_north", &dimid));
  NC(nc_inq_dimlen(ncid, dimid, &rs));
  nlat = (int) rs;
  if (nlat > NLAT)
    ERRMSG("Too many latitudes!");

  NC(nc_inq_dimid(ncid, "west_east", &dimid));
  NC(nc_inq_dimlen(ncid, dimid, &rs));
  nlon = (int) rs;
  if (nlon > NLON)
    ERRMSG("Too many longitudes!");

```

```
  /* Read latitudes... */
  NC(nc_inq_varid(ncid, "XLAT", &varid));
  for (ilat = 0; ilat < nlat; ilat++) {
    start[0] = (size_t) oit;
    start[1] = (size_t) ilat;
    start[2] = 0;
    count[0] = 1;
    count[1] = 1;
    count[2] = (size_t) nlon;
    NC(nc_get_vara_double(ncid, varid, start, count, lat[ilat]));
  }

  /* Read longitudes... */
  NC(nc_inq_varid(ncid, "XLONG", &varid));
  for (ilat = 0; ilat < nlat; ilat++) {
    start[0] = (size_t) oit;
    start[1] = (size_t) ilat;
    start[2] = 0;
    count[0] = 1;
    count[1] = 1;
    count[2] = (size_t) nlon;
    NC(nc_get_vara_double(ncid, varid, start, count, lon[ilat]));
  }

  /* Read theta perturbation... */
  NC(nc_inq_varid(ncid, "T", &varid));
  for (iz = 0; iz < nz; iz++)
    for (ilat = 0; ilat < nlat; ilat++) {
      start[0] = (size_t) oit;
      start[1] = (size_t) iz;
      start[2] = (size_t) ilat;
      start[3] = 0;
      count[0] = 1;
      count[1] = 1;
      count[2] = 1;
      count[3] = (size_t) nlon;
      NC(nc_get_vara_double(ncid, varid, start, count, t[iz][ilat]));
    }

  /* Read geopotential perturbation... */
  NC(nc_inq_varid(ncid, "PH", &varid));
  for (iz = 0; iz < nz2; iz++)
    for (ilat = 0; ilat < nlat; ilat++) {
      start[0] = (size_t) oit;
      start[1] = (size_t) iz;
      start[2] = (size_t) ilat;
      start[3] = 0;
      count[0] = 1;
      count[1] = 1;
      count[2] = 1;
      count[3] = (size_t) nlon;
      NC(nc_get_vara_double(ncid, varid, start, count, z[iz][ilat]));
    }

  /* Read geopotential base... */
  NC(nc_inq_varid(ncid, "PHB", &varid));
  for (iz = 0; iz < nz2; iz++)
    for (ilat = 0; ilat < nlat; ilat++) {
      start[0] = (size_t) oit;
      start[1] = (size_t) iz;
      start[2] = (size_t) ilat;
      start[3] = 0;
      count[0] = 1;
      count[1] = 1;
      count[2] = 1;
      count[3] = (size_t) nlon;
      NC(nc_get_vara_double(ncid, varid, start, count, z0[iz][ilat]));
    }

  /* Read pressure perturbation... */
  NC(nc_inq_varid(ncid, "P", &varid));
  for (iz = 0; iz < nz; iz++)
    for (ilat = 0; ilat < nlat; ilat++) {
      start[0] = (size_t) oit;
      start[1] = (size_t) iz;
      start[2] = (size_t) ilat;
      start[3] = 0;
      count[0] = 1;
      count[1] = 1;
      count[2] = 1;
      count[3] = (size_t) nlon;
      NC(nc_get_vara_double(ncid, varid, start, count, p[iz][ilat]));
    }

  /* Read pressure base... */
  NC(nc_inq_varid(ncid, "PB", &varid));
  for (iz = 0; iz < nz; iz++)
```

```
    for (ilat = 0; ilat < nlat; ilat++) {
      start[0] = (size_t) oit;
      start[1] = (size_t) iz;
      start[2] = (size_t) ilat;
      start[3] = 0;
      count[0] = 1;
      count[1] = 1;
      count[2] = 1;
      count[3] = (size_t) nlon;
      NC(nc_get_vara_double(ncid, varid, start, count, p0[iz][ilat]));
    }

  /* Close file... */
  NC(nc_close(ncid));

  /* ------------------------------------------------------------
     Convert WRF data...
     ------------------------------------------------------------ */

  /* Adjust longitudes... */
  for (ilat = 0; ilat < nlat; ilat++)
    for (ilon = 0; ilon < nlon; ilon++)
      if (lon[ilat][ilon] > 180)
        lon[ilat][ilon] -= 360;

  /* Get altitudes... */
  for (iz = 0; iz < nz; iz++)
    for (ilat = 0; ilat < nlat; ilat++)
      for (ilon = 0; ilon < nlon; ilon++)
        z[iz][ilat][ilon]
          = 0.5 * (z[iz + 1][ilat][ilon] + z0[iz + 1][ilat][ilon]
                   + z[iz][ilat][ilon] + z0[iz][ilat][ilon]) / G0 / 1000.;

  /* Get pressure... */
  for (iz = 0; iz < nz; iz++)
    for (ilat = 0; ilat < nlat; ilat++)
      for (ilon = 0; ilon < nlon; ilon++)
        p[iz][ilat][ilon]
          = (p[iz][ilat][ilon] + p0[iz][ilat][ilon]) / 100.;

  /* Get temperature... */
  for (iz = 0; iz < nz; iz++)
    for (ilat = 0; ilat < nlat; ilat++)
      for (ilon = 0; ilon < nlon; ilon++)
        t[iz][ilat][ilon]
          = (t[iz][ilat][ilon] + 300.) / pow(1000. / p[iz][ilat][ilon],
                                             0.286);

  /* ------------------------------------------------------------
     Write WRF data to ASCII...
     ------------------------------------------------------------ */

  /* Check filename... */
  if (argv[4][0] != '-') {

    /* Create file... */
    printf("Write WRF data: %s\n", argv[4]);
    if (!(out = fopen(argv[4], "w")))
      ERRMSG("Cannot create file!");

    /* Write header... */
    fprintf(out,
            "# $1 = altitude index\n"
            "# $2 = altitude [km]\n"
            "# $3 = longitude [deg]\n"
            "# $4 = latitude [deg]\n"
            "# $5 = pressure [hPa]\n" "# $6 = temperature [K]\n");

    /* Write output... */
    for (iz = 0; iz < nz; iz++)
      for (ilon = 0; ilon < nlon; ilon++) {
        fprintf(out, "\n");
        for (ilat = 0; ilat < nlat; ilat++)
          fprintf(out, "%d %g %g %g %g %g\n", iz, z[iz][ilat][ilon],
                  lon[ilat][ilon], lat[ilat][ilon],
                  p[iz][ilat][ilon], t[iz][ilat][ilon]);
      }

    /* Close file... */
    fclose(out);
  }

  /* ------------------------------------------------------------
     Run forward model...
     ------------------------------------------------------------ */

  /* Loop over latitudes... */
```

```
  for (ilat = 0; ilat < nlat; ilat++) {

    /* Write info... */
    printf("  Compute latitude %d / %d ...\n", ilat + 1, nlat);

    /* Loop over longitudes... */
    for (ilon = 0; ilon < nlon; ilon++) {

      /* Set altitude levels... */
      atm.np = 0;
      for (iz = 0; iz < nz; iz++)
        if (gsl_finite(gsl_finite(t[iz][ilat][ilon]))
            && t[iz][ilat][ilon] > 100 && t[iz][ilat][ilon] < 400
            && z[iz][ilat][ilon] > 10) {
          atm.z[atm.np] = z[iz][ilat][ilon];
          if ((++atm.np) >= NP)
            ERRMSG("Too many altitudes!");
        }

      /* Add top level... */
      atm.z[atm.np] = 90.;
      if ((++atm.np) >= NP)
        ERRMSG("Too many altitudes!");

      /* Initialize with climatological data... */
      climatology(&ctl, &atm);

      /* Set temperature and pressure... */
      atm.np = 0;
      for (iz = 0; iz < nz; iz++)
        if (gsl_finite(t[iz][ilat][ilon])
            && t[iz][ilat][ilon] > 100 && t[iz][ilat][ilon] < 400
            && z[iz][ilat][ilon] > 10) {
          atm.p[atm.np] = p[iz][ilat][ilon];
          atm.t[atm.np] = t[iz][ilat][ilon];
          atm.np++;
        }

      /* Add top level... */
      atm.np++;

      /* Set observation data... */
      obs.nr = 1;
      obs.obsz[0] = 700;

      /* Run forward model... */
      formod(&ctl, &atm, &obs);

      /* Get mean brightness temperature... */
      temp[ilat][ilon] = 0;
      for (id = 0; id < ctl.nd; id++)
        temp[ilat][ilon] += obs.rad[id][0] / ctl.nd;
    }
  }

  /* Crop at boundaries... */
  for (ilat = 0; ilat < ncrop; ilat++)
    for (ilon = 0; ilon < nlon; ilon++)
      temp[ilat][ilon] = GSL_NAN;
  for (ilat = nlat - ncrop; ilat < nlat; ilat++)
    for (ilon = 0; ilon < nlon; ilon++)
      temp[ilat][ilon] = GSL_NAN;
  for (ilon = 0; ilon < ncrop; ilon++)
    for (ilat = 0; ilat < nlat; ilat++)
      temp[ilat][ilon] = GSL_NAN;
  for (ilon = nlon - ncrop; ilon < nlon; ilon++)
    for (ilat = 0; ilat < nlat; ilat++)
      temp[ilat][ilon] = GSL_NAN;

  /* Get perturbations... */
  background(temp, pt, nlat, nlon, 10, 10);

  /* ------------------------------------------------------------
     Save forward model output...
     ------------------------------------------------------------ */

  /* Check filename... */
  if (argv[5][0] != '-') {

    /* Create file... */
    printf("Write radiance data: %s\n", argv[5]);
    if (!(out = fopen(argv[5], "w")))
      ERRMSG("Cannot create file!");

    /* Write header... */
    fprintf(out,
            "# $1 = longitude [deg]\n"
```

```
          "# $2 = latitude [deg]\n"
          "# $3 = WRF brightness temperature [K]\n"
          "# $4 = WRF brightness temperature perturbation [K]\n");

    /* Write output... */
    for (ilat = 0; ilat < nlat; ilat++) {
      fprintf(out, "\n");
      for (ilon = 0; ilon < nlon; ilon++)
        fprintf(out, "%g %g %g %g\n", lon[ilat][ilon], lat[ilat][ilon],
                temp[ilat][ilon], pt[ilat][ilon]);
    }

    /* Close file... */
    fclose(out);
  }

  /* ------------------------------------------------------------
     Read AIRS radiance map and resample model data...
     ------------------------------------------------------------ */

  /* Check filename... */
  if (argv[6][0] != '-') {

    /* Read AIRS wave file... */
    read_wave(argv[6], wave_airs);
    memcpy(wave_wrf, wave_airs, sizeof(wave_t));

    /* Get Cartesian coordinates for model grid... */
    for (ilat = 0; ilat < nlat; ilat++)
      for (ilon = 0; ilon < nlon; ilon++)
        geo2cart(0, lon[ilat][ilon], lat[ilat][ilon], x1[ilat][ilon]);

    /* Loop over AIRS geolocations... */
    for (ix = 0; ix < wave_airs->nx; ix++)
      for (iy = 0; iy < wave_airs->ny; iy++) {

        /* Write info... */
        if (iy == 0)
          printf("  Average for xtrack %d / %d ...\n", ix + 1, wave_airs->nx);

        /* Init... */
        wsum = 0;
        wave_wrf->temp[ix][iy] = 0;
        wave_wrf->bg[ix][iy] = 0;
        wave_wrf->pt[ix][iy] = 0;
        wave_wrf->var[ix][iy] = 0;

        /* Average... */
        geo2cart(0, wave_airs->lon[ix][iy], wave_airs->lat[ix][iy], x0);
        for (ilat = 0; ilat < nlat; ilat++)
          for (ilon = 0; ilon < nlon; ilon++)
            if (DIST2(x0, x1[ilat][ilon]) <= rmax2) {
              w =
                exp(-DIST2(x0, x1[ilat][ilon]) /
                    (2. * gsl_pow_2(fwhm / 2.3548)));
              wave_wrf->temp[ix][iy] += w * temp[ilat][ilon];
              wave_wrf->bg[ix][iy] += w * (temp[ilat][ilon] - pt[ilat][ilon]);
              wave_wrf->pt[ix][iy] += w * pt[ilat][ilon];
              wsum += w;
            }

        /* Normalize... */
        if (wsum > 0) {
          wave_wrf->temp[ix][iy] /= wsum;
          wave_wrf->bg[ix][iy] /= wsum;
          wave_wrf->pt[ix][iy] /= wsum;
        } else {
          wave_wrf->temp[ix][iy] = GSL_NAN;
          wave_wrf->bg[ix][iy] = GSL_NAN;
          wave_wrf->pt[ix][iy] = GSL_NAN;
        }
      }

    /* Compute variance... */
    variance(wave_wrf, var_dh);

    /* Write WRF wave struct... */
    write_wave(argv[7], wave_wrf);
  }

  /* Free... */
  free(wave_airs);
  free(wave_wrf);

  return EXIT_SUCCESS;
}
```

```
/*****************************************************************************/

void background(
  double temp[NLAT][NLON],
  double pt[NLAT][NLON],
  int nlat,
  int nlon,
  int dlat,
  int dlon) {

  static double data[NLAT * NLAT];

  int ilon, ilat, ilon2, ilat2, n;

  /* Loop over grid points... */
  for (ilat = 0; ilat < nlat; ilat++)
    for (ilon = 0; ilon < nlon; ilon++) {

      /* Init... */
      n = 0;

      /* Average... */
      for (ilat2 = GSL_MAX(ilat - dlat, 0);
           ilat2 <= GSL_MIN(ilat + dlat, nlat - 1); ilat2++)
        for (ilon2 = GSL_MAX(ilon - dlon, 0);
             ilon2 <= GSL_MIN(ilon + dlon, nlon - 1); ilon2++)
          if (gsl_finite(temp[ilat2][ilon2])) {
            data[n] = temp[ilat2][ilon2];
            n++;
          }

      /* Set perturbation... */
      gsl_sort(data, 1, (size_t) n);
      pt[ilat][ilon] = temp[ilat][ilon]
        - gsl_stats_median_from_sorted_data(data, 1, (size_t) n);
    }
}
```

5.87 zm_ret.c File Reference

Functions

- int main (int argc, char *argv[ ])

5.87.1 Function Documentation

5.87.1.1 int main (int *argc*, char * *argv[ ]*)

Definition at line 14 of file zm_ret.c.

```
                  {

  static ret_t ret;
  static wave_t wave;

  static double apr_tm[NPG][NLAT], apr_var[NPG][NLAT], apr_noise[NPG][NLAT],
    ret_tm[NPG][NLAT], ret_var[NPG][NLAT], ret_noise[NPG][NLAT],
    ret_time[NPG][NLAT], mu, sig_apr, sig_ret, tbg[NDS], tabg[NDS];

  static int bg_poly_x, bg_poly_y, bg_smooth_x, bg_smooth_y,
    i, ids, ilat, ip, ix, iy, nlat, n[NPG][NLAT], ncid;

  FILE *out;

  /* Check arguments... */
  if (argc < 4)
    ERRMSG("Give parameters: <ctl> <zm.tab> <airs1.nc> [<airs2.nc> ...]");

  /* Get control parameters... */
  bg_poly_x = (int) scan_ctl(argc, argv, "BG_POLY_X", -1, "5", NULL);
  bg_poly_y = (int) scan_ctl(argc, argv, "BG_POLY_Y", -1, "0", NULL);
  bg_smooth_x = (int) scan_ctl(argc, argv, "BG_SMOOTH_X", -1, "0", NULL);
  bg_smooth_y = (int) scan_ctl(argc, argv, "BG_SMOOTH_Y", -1, "0", NULL);
  nlat = (int) scan_ctl(argc, argv, "NLAT", -1, "36", NULL);
```

```
  if (nlat > NLAT)
    ERRMSG("Too many latitudes!");

  /* Loop over files... */
  for (i = 3; i < argc; i++) {

    /* Read AIRS data... */
    if (nc_open(argv[i], NC_WRITE, &ncid) != NC_NOERR)
      continue;
    else
      nc_close(ncid);
    read_retr(argv[i], &ret);

    /* Loop over altitudes... */
    for (ip = 0; ip < ret.np; ip++) {

      /* Compute background... */
      ret2wave(&ret, &wave, 1, ip);
      background_poly(&wave, bg_poly_x, bg_poly_y);
      background_smooth(&wave, bg_smooth_x, bg_smooth_y);
      for (ix = 0; ix < wave.nx; ix++)
        for (iy = 0; iy < wave.ny; iy++)
          tbg[iy * 90 + ix] = wave.bg[ix][iy];
      noise(&wave, &mu, &sig_ret);
      ret2wave(&ret, &wave, 2, ip);
      background_poly(&wave, bg_poly_x, bg_poly_y);
      background_smooth(&wave, bg_smooth_x, bg_smooth_y);
      for (ix = 0; ix < wave.nx; ix++)
        for (iy = 0; iy < wave.ny; iy++)
          tabg[iy * 90 + ix] = wave.bg[ix][iy];
      noise(&wave, &mu, &sig_apr);

      /* Loop over data sets... */
      for (ids = 0; ids < ret.nds; ids++) {

        /* Check data... */
        if (ret.lon[ids][ip] < -180 || ret.lon[ids][ip] > 180
            || ret.lat[ids][ip] < -90 || ret.lat[ids][ip] > 90
            || ret.t[ids][ip] < 110 || ret.t[ids][ip] > 390
            || !gsl_finite(ret.t[ids][ip]))
          continue;

        /* Get latitude index... */
        ilat = (int) ((ret.lat[ids][ip] + 90.) / 180. * (double) nlat);
        if (ilat < 0 || ilat >= nlat)
          continue;

        /* Get zonal mean... */
        if (gsl_finite(ret.t[ids][ip]) && gsl_finite(tbg[ids])) {
          ret_time[ip][ilat] += ret.time[ids][ip];
          ret_tm[ip][ilat] += ret.t[ids][ip];
          ret_var[ip][ilat] += gsl_pow_2(ret.t[ids][ip] - tbg[ids]);
          ret_noise[ip][ilat] += gsl_pow_2(sig_ret);
          apr_tm[ip][ilat] += ret.t_apr[ids][ip];
          apr_var[ip][ilat] += gsl_pow_2(ret.t_apr[ids][ip] - tabg[ids]);
          apr_noise[ip][ilat] += gsl_pow_2(sig_apr);
          n[ip][ilat]++;
        }
      }
    }
  }

  /* Create output file... */
  printf("Write AIRS zonal mean data: %s\n", argv[2]);
  if (!(out = fopen(argv[2], "w")))
    ERRMSG("Cannot create file!");

  /* Write header... */
  fprintf(out,
          "# $1  = time (seconds since 01-JAN-2000, 00:00 UTC)\n"
          "# $2  = altitude [km]\n"
          "# $3  = latitude [deg]\n"
          "# $4  = mean temperature (retrieved) [K]\n"
          "# $5  = temperature variance (retrieved) [K^2]\n"
          "# $6  = noise estimate (retrieved) [K^2]\n"
          "# $7  = mean temperature (a priori) [K]\n"
          "# $8  = temperature variance (a priori) [K^2]\n"
          "# $9  = noise estimate (a priori) [K^2]\n"
          "# $10 = number of data points\n");

  /* Loop over latitudes... */
  for (ilat = 0; ilat < nlat; ilat++) {

    /* Write empty line... */
    fprintf(out, "\n");

    /* Loop over altitudes... */
```

```
    for (ip = 0; ip < ret.np; ip++) {

      /* Write data... */
      fprintf(out, "%.2f %g %g %g %g %g %g %g %g %d\n",
              ret_time[ip][ilat] / n[ip][ilat],
              ret.z[0][ip], (ilat + 0.5) / nlat * 180. - 90.,
              ret_tm[ip][ilat] / n[ip][ilat],
              sqrt(ret_var[ip][ilat] / n[ip][ilat]),
              sqrt(ret_noise[ip][ilat] / n[ip][ilat]),
              apr_tm[ip][ilat] / n[ip][ilat],
              sqrt(apr_var[ip][ilat] / n[ip][ilat]),
              sqrt(apr_noise[ip][ilat] / n[ip][ilat]), n[ip][ilat]);
    }
  }

  /* Close file... */
  fclose(out);

  return EXIT_SUCCESS;
}
```

Here is the call graph for this function:

5.88 zm_ret.c

```
#include "libairs.h"

/* ------------------------------------------------------------
   Dimensions...
   ------------------------------------------------------------ */

/* Maximum number of latitudes. */
#define NLAT 180

/* ------------------------------------------------------------
   Main...
   ------------------------------------------------------------ */

int main(
  int argc,
  char *argv[]) {

  static ret_t ret;
  static wave_t wave;

  static double apr_tm[NPG][NLAT], apr_var[NPG][NLAT], apr_noise[NPG][NLAT],
    ret_tm[NPG][NLAT], ret_var[NPG][NLAT], ret_noise[NPG][NLAT],
    ret_time[NPG][NLAT], mu, sig_apr, sig_ret, tbg[NDS], tabg[NDS];

  static int bg_poly_x, bg_poly_y, bg_smooth_x, bg_smooth_y,
    i, ids, ilat, ip, ix, iy, nlat, n[NPG][NLAT], ncid;

  FILE *out;

  /* Check arguments... */
  if (argc < 4)
    ERRMSG("Give parameters: <ctl> <zm.tab> <airs1.nc> [<airs2.nc> ...]");

  /* Get control parameters... */
  bg_poly_x = (int) scan_ctl(argc, argv, "BG_POLY_X", -1, "5", NULL);
  bg_poly_y = (int) scan_ctl(argc, argv, "BG_POLY_Y", -1, "0", NULL);
  bg_smooth_x = (int) scan_ctl(argc, argv, "BG_SMOOTH_X", -1, "0", NULL);
  bg_smooth_y = (int) scan_ctl(argc, argv, "BG_SMOOTH_Y", -1, "0", NULL);
  nlat = (int) scan_ctl(argc, argv, "NLAT", -1, "36", NULL);
  if (nlat > NLAT)
    ERRMSG("Too many latitudes!");

  /* Loop over files... */
  for (i = 3; i < argc; i++) {

    /* Read AIRS data... */
    if (nc_open(argv[i], NC_WRITE, &ncid) != NC_NOERR)
      continue;
    else
      nc_close(ncid);
    read_retr(argv[i], &ret);

    /* Loop over altitudes... */
    for (ip = 0; ip < ret.np; ip++) {

      /* Compute background... */
```

```
      ret2wave(&ret, &wave, 1, ip);
      background_poly(&wave, bg_poly_x, bg_poly_y);
      background_smooth(&wave, bg_smooth_x, bg_smooth_y);
      for (ix = 0; ix < wave.nx; ix++)
        for (iy = 0; iy < wave.ny; iy++)
          tbg[iy * 90 + ix] = wave.bg[ix][iy];
      noise(&wave, &mu, &sig_ret);
      ret2wave(&ret, &wave, 2, ip);
      background_poly(&wave, bg_poly_x, bg_poly_y);
      background_smooth(&wave, bg_smooth_x, bg_smooth_y);
      for (ix = 0; ix < wave.nx; ix++)
        for (iy = 0; iy < wave.ny; iy++)
          tabg[iy * 90 + ix] = wave.bg[ix][iy];
      noise(&wave, &mu, &sig_apr);

      /* Loop over data sets... */
      for (ids = 0; ids < ret.nds; ids++) {

        /* Check data... */
        if (ret.lon[ids][ip] < -180 || ret.lon[ids][ip] > 180
            || ret.lat[ids][ip] < -90 || ret.lat[ids][ip] > 90
            || ret.t[ids][ip] < 110 || ret.t[ids][ip] > 390
            || !gsl_finite(ret.t[ids][ip]))
          continue;

        /* Get latitude index... */
        ilat = (int) ((ret.lat[ids][ip] + 90.) / 180. * (double) nlat);
        if (ilat < 0 || ilat >= nlat)
          continue;

        /* Get zonal mean... */
        if (gsl_finite(ret.t[ids][ip]) && gsl_finite(tbg[ids])) {
          ret_time[ip][ilat] += ret.time[ids][ip];
          ret_tm[ip][ilat] += ret.t[ids][ip];
          ret_var[ip][ilat] += gsl_pow_2(ret.t[ids][ip] - tbg[ids]);
          ret_noise[ip][ilat] += gsl_pow_2(sig_ret);
          apr_tm[ip][ilat] += ret.t_apr[ids][ip];
          apr_var[ip][ilat] += gsl_pow_2(ret.t_apr[ids][ip] - tabg[ids]);
          apr_noise[ip][ilat] += gsl_pow_2(sig_apr);
          n[ip][ilat]++;
        }
      }
    }
  }

  /* Create output file... */
  printf("Write AIRS zonal mean data: %s\n", argv[2]);
  if (!(out = fopen(argv[2], "w")))
    ERRMSG("Cannot create file!");

  /* Write header... */
  fprintf(out,
          "# $1  = time (seconds since 01-JAN-2000, 00:00 UTC)\n"
          "# $2  = altitude [km]\n"
          "# $3  = latitude [deg]\n"
          "# $4  = mean temperature (retrieved) [K]\n"
          "# $5  = temperature variance (retrieved) [K^2]\n"
          "# $6  = noise estimate (retrieved) [K^2]\n"
          "# $7  = mean temperature (a priori) [K]\n"
          "# $8  = temperature variance (a priori) [K^2]\n"
          "# $9  = noise estimate (a priori) [K^2]\n"
          "# $10 = number of data points\n");

  /* Loop over latitudes... */
  for (ilat = 0; ilat < nlat; ilat++) {

    /* Write empty line... */
    fprintf(out, "\n");

    /* Loop over altitudes... */
    for (ip = 0; ip < ret.np; ip++) {

      /* Write data... */
      fprintf(out, "%.2f %g %g %g %g %g %g %g %g %d\n",
              ret_time[ip][ilat] / n[ip][ilat],
              ret.z[0][ip], (ilat + 0.5) / nlat * 180. - 90.,
              ret_tm[ip][ilat] / n[ip][ilat],
              sqrt(ret_var[ip][ilat] / n[ip][ilat]),
              sqrt(ret_noise[ip][ilat] / n[ip][ilat]),
              apr_tm[ip][ilat] / n[ip][ilat],
              sqrt(apr_var[ip][ilat] / n[ip][ilat]),
              sqrt(apr_noise[ip][ilat] / n[ip][ilat]), n[ip][ilat]);
    }
  }

  /* Close file... */
  fclose(out);
```

```

  return EXIT_SUCCESS;
}
```

Index