

## AIRS Code Collection

Generated by Doxygen 1.8.11

## Contents

<b>1</b>	<b>Main Page</b>	<b>1</b>
<b>2</b>	<b>Data Structure Index</b>	<b>1</b>
2.1	Data Structures . . . . .	1
<b>3</b>	<b>File Index</b>	<b>2</b>
3.1	File List . . . . .	2
<b>4</b>	<b>Data Structure Documentation</b>	<b>3</b>
4.1	airs_l1_t Struct Reference . . . . .	3
4.1.1	Detailed Description . . . . .	4
4.1.2	Field Documentation . . . . .	4
4.2	airs_l2_t Struct Reference . . . . .	5
4.2.1	Detailed Description . . . . .	5
4.2.2	Field Documentation . . . . .	6
4.3	atm_t Struct Reference . . . . .	6
4.3.1	Detailed Description . . . . .	7
4.3.2	Field Documentation . . . . .	7
4.4	ctl_t Struct Reference . . . . .	8
4.4.1	Detailed Description . . . . .	10
4.4.2	Field Documentation . . . . .	10
4.5	los_t Struct Reference . . . . .	13
4.5.1	Detailed Description . . . . .	13
4.5.2	Field Documentation . . . . .	14
4.6	ncd_t Struct Reference . . . . .	15
4.6.1	Detailed Description . . . . .	16
4.6.2	Field Documentation . . . . .	16
4.7	obs_t Struct Reference . . . . .	18
4.7.1	Detailed Description . . . . .	19
4.7.2	Field Documentation . . . . .	19
4.8	pert_t Struct Reference . . . . .	20
4.8.1	Detailed Description . . . . .	21
4.8.2	Field Documentation . . . . .	21
4.9	ret_t Struct Reference . . . . .	22
4.9.1	Detailed Description . . . . .	24
4.9.2	Field Documentation . . . . .	24
4.10	tbl_t Struct Reference . . . . .	28
4.10.1	Detailed Description . . . . .	28
4.10.2	Field Documentation . . . . .	28
4.11	wave_t Struct Reference . . . . .	30
4.11.1	Detailed Description . . . . .	30
4.11.2	Field Documentation . . . . .	30

<b>5 File Documentation</b>	<b>32</b>
5.1 day2doy.c File Reference	32
5.1.1 Function Documentation	32
5.2 day2doy.c	33
5.3 diff_apr.c File Reference	33
5.3.1 Function Documentation	33
5.4 diff_apr.c	35
5.5 diff_ret.c File Reference	38
5.5.1 Function Documentation	38
5.6 diff_ret.c	39
5.7 distance.c File Reference	40
5.7.1 Function Documentation	40
5.8 distance.c	41
5.9 doy2day.c File Reference	41
5.9.1 Function Documentation	42
5.10 doy2day.c	42
5.11 events.c File Reference	42
5.11.1 Function Documentation	43
5.12 events.c	45
5.13 extract.c File Reference	47
5.13.1 Function Documentation	47
5.13.2 Variable Documentation	49
5.14 extract.c	50
5.15 hurricane.c File Reference	51
5.15.1 Function Documentation	52
5.16 hurricane.c	57
5.17 island.c File Reference	62
5.17.1 Function Documentation	62
5.18 island.c	67
5.19 issifm.c File Reference	71

5.19.1 Function Documentation . . . . .	72
5.20 issifm.c . . . . .	81
5.21 jurassic.c File Reference . . . . .	90
5.21.1 Detailed Description . . . . .	92
5.21.2 Function Documentation . . . . .	92
5.22 jurassic.c . . . . .	159
5.23 jurassic.h File Reference . . . . .	216
5.23.1 Detailed Description . . . . .	219
5.23.2 Function Documentation . . . . .	219
5.24 jurassic.h . . . . .	286
5.25 libairs.c File Reference . . . . .	293
5.25.1 Function Documentation . . . . .	295
5.26 libairs.c . . . . .	321
5.27 libairs.h File Reference . . . . .	342
5.27.1 Function Documentation . . . . .	344
5.28 libairs.h . . . . .	370
5.29 map_pert.c File Reference . . . . .	374
5.29.1 Function Documentation . . . . .	374
5.30 map_pert.c . . . . .	378
5.31 map_rad.c File Reference . . . . .	380
5.31.1 Function Documentation . . . . .	380
5.32 map_rad.c . . . . .	382
5.33 map_ret.c File Reference . . . . .	383
5.33.1 Function Documentation . . . . .	383
5.34 map_ret.c . . . . .	385
5.35 noise_pert.c File Reference . . . . .	386
5.35.1 Function Documentation . . . . .	386
5.36 noise_pert.c . . . . .	388
5.37 noise_ret.c File Reference . . . . .	389
5.37.1 Function Documentation . . . . .	389

5.38	<a href="#">noise_ret.c</a>	390
5.39	<a href="#">optimize_btd.c</a> File Reference	391
5.39.1	<a href="#">Function Documentation</a>	391
5.40	<a href="#">optimize_btd.c</a>	393
5.41	<a href="#">orbit.c</a> File Reference	395
5.41.1	<a href="#">Function Documentation</a>	395
5.42	<a href="#">orbit.c</a>	396
5.43	<a href="#">overpass.c</a> File Reference	396
5.43.1	<a href="#">Function Documentation</a>	397
5.44	<a href="#">overpass.c</a>	399
5.45	<a href="#">perturbation.c</a> File Reference	401
5.45.1	<a href="#">Function Documentation</a>	401
5.46	<a href="#">perturbation.c</a>	406
5.47	<a href="#">rayt.c</a> File Reference	410
5.47.1	<a href="#">Function Documentation</a>	410
5.48	<a href="#">rayt.c</a>	413
5.49	<a href="#">ret2tab.c</a> File Reference	416
5.49.1	<a href="#">Function Documentation</a>	416
5.50	<a href="#">ret2tab.c</a>	417
5.51	<a href="#">retrieval.c</a> File Reference	418
5.51.1	<a href="#">Function Documentation</a>	419
5.52	<a href="#">retrieval.c</a>	434
5.53	<a href="#">sampling.c</a> File Reference	448
5.53.1	<a href="#">Function Documentation</a>	448
5.54	<a href="#">sampling.c</a>	450
5.55	<a href="#">spec2tab.c</a> File Reference	452
5.55.1	<a href="#">Function Documentation</a>	452
5.56	<a href="#">spec2tab.c</a>	453
5.57	<a href="#">spec_ana.c</a> File Reference	455
5.57.1	<a href="#">Function Documentation</a>	455

5.58 spec_ana.c . . . . .	457
5.59 sza.c File Reference . . . . .	458
5.59.1 Function Documentation . . . . .	458
5.60 sza.c . . . . .	458
5.61 var1d.c File Reference . . . . .	459
5.61.1 Function Documentation . . . . .	459
5.62 var1d.c . . . . .	460
5.63 variance.c File Reference . . . . .	461
5.63.1 Function Documentation . . . . .	461
5.64 variance.c . . . . .	467
5.65 volcano.c File Reference . . . . .	475
5.65.1 Function Documentation . . . . .	475
5.66 volcano.c . . . . .	479
5.67 zm_ret.c File Reference . . . . .	482
5.67.1 Function Documentation . . . . .	483
5.68 zm_ret.c . . . . .	485
<b>Index</b>	<b>489</b>

## 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:

<b>airs_l1_t</b> <b>AIRS Level-1 data</b>	<b>3</b>
<b>airs_l2_t</b> <b>AIRS Level-2 data</b>	<b>5</b>

<a href="#">atm_t</a>	Atmospheric data	6
<a href="#">ctl_t</a>	Forward model control parameters	8
<a href="#">los_t</a>	Line-of-sight data	13
<a href="#">ncd_t</a>	Buffer for netCDF data	15
<a href="#">obs_t</a>	Observation geometry and radiance data	18
<a href="#">pert_t</a>	Perturbation data	20
<a href="#">ret_t</a>	Retrieval results	22
<a href="#">tbl_t</a>	Emissivity look-up tables	28
<a href="#">wave_t</a>	Wave analysis data	30

## 3 File Index

### 3.1 File List

Here is a list of all files with brief descriptions:

day2doy.c	32
diff_apr.c	33
diff_ret.c	38
distance.c	40
doy2day.c	41
events.c	42
extract.c	47
hurricane.c	51
island.c	62
issifm.c	71
<b>jurassic.c</b>	
<b>JURASSIC library definitions</b>	<b>90</b>
<b>jurassic.h</b>	
<b>JURASSIC library declarations</b>	<b>216</b>

<a href="#">libairs.c</a>	293
<a href="#">libairs.h</a>	342
<a href="#">map_pert.c</a>	374
<a href="#">map_rad.c</a>	380
<a href="#">map_ret.c</a>	383
<a href="#">noise_pert.c</a>	386
<a href="#">noise_ret.c</a>	389
<a href="#">optimize_btd.c</a>	391
<a href="#">orbit.c</a>	395
<a href="#">overpass.c</a>	396
<a href="#">perturbation.c</a>	401
<a href="#">rayt.c</a>	410
<a href="#">ret2tab.c</a>	416
<a href="#">retrieval.c</a>	418
<a href="#">sampling.c</a>	448
<a href="#">spec2tab.c</a>	452
<a href="#">spec_ana.c</a>	455
<a href="#">sza.c</a>	458
<a href="#">var1d.c</a>	459
<a href="#">variance.c</a>	461
<a href="#">volcano.c</a>	475
<a href="#">zm_ret.c</a>	482

## 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<sup>-1</sup>].*
- float [rad](#) [L1\_NTRACK][L1\_NXTRACK][L1\_NCHAN]
- Radiance [W/(m<sup>2</sup> sr cm<sup>-1</sup>)].*

#### 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 [ $\text{cm}^{-1}$ ].

Definition at line 93 of file [libairs.h](#).

## 4.1.2.8 float airs\_l1\_t::rad[L1\_NTRACK][L1\_NXTRACK][L1\_NCHAN]

Radiance [ $\text{W}/(\text{m}^2 \text{ sr cm}^{-1})$ ].

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 [222](#) of file [jurassic.h](#).

#### 4.3.2 Field Documentation

##### 4.3.2.1 int atm\_t::np

Number of data points.

Definition at line [225](#) of file [jurassic.h](#).

##### 4.3.2.2 double atm\_t::time[NP]

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

Definition at line [228](#) of file [jurassic.h](#).

##### 4.3.2.3 double atm\_t::z[NP]

Altitude [km].

Definition at line [231](#) of file [jurassic.h](#).

#### 4.3.2.4 double atm\_t::lon[NP]

Longitude [deg].

Definition at line 234 of file [jurassic.h](#).

#### 4.3.2.5 double atm\_t::lat[NP]

Latitude [deg].

Definition at line 237 of file [jurassic.h](#).

#### 4.3.2.6 double atm\_t::p[NP]

Pressure [hPa].

Definition at line 240 of file [jurassic.h](#).

#### 4.3.2.7 double atm\_t::t[NP]

Temperature [K].

Definition at line 243 of file [jurassic.h](#).

#### 4.3.2.8 double atm\_t::q[NG][NP]

Volume mixing ratio.

Definition at line 246 of file [jurassic.h](#).

#### 4.3.2.9 double atm\_t::k[NW][NP]

Extinction [1/km].

Definition at line 249 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 [ $\text{cm}^{-1}$ ].*
- 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 254 of file [jurassic.h](#).

#### 4.4.2 Field Documentation

##### 4.4.2.1 `int ctl_t::ng`

Number of emitters.

Definition at line 257 of file [jurassic.h](#).

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

Name of each emitter.

Definition at line 260 of file [jurassic.h](#).

##### 4.4.2.3 `int ctl_t::nd`

Number of radiance channels.

Definition at line 263 of file [jurassic.h](#).

##### 4.4.2.4 `int ctl_t::nw`

Number of spectral windows.

Definition at line 266 of file [jurassic.h](#).

##### 4.4.2.5 `double ctl_t::nu[ND]`

Centroid wavenumber of each channel [ $\text{cm}^{-1}$ ].

Definition at line 269 of file [jurassic.h](#).

##### 4.4.2.6 `int ctl_t::window[ND]`

Window index of each channel.

Definition at line 272 of file [jurassic.h](#).

##### 4.4.2.7 `char ctl_t::tblbase[LEN]`

Basename for table files and filter function files.

Definition at line 275 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 278 of file [jurassic.h](#).

#### 4.4.2.9 `int ctl_t::ctm_co2`

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

Definition at line 281 of file [jurassic.h](#).

#### 4.4.2.10 `int ctl_t::ctm_h2o`

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

Definition at line 284 of file [jurassic.h](#).

#### 4.4.2.11 `int ctl_t::ctm_n2`

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

Definition at line 287 of file [jurassic.h](#).

#### 4.4.2.12 `int ctl_t::ctm_o2`

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

Definition at line 290 of file [jurassic.h](#).

#### 4.4.2.13 `int ctl_t::refrac`

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

Definition at line 293 of file [jurassic.h](#).

#### 4.4.2.14 `double ctl_t::rayds`

Maximum step length for raytracing [km].

Definition at line 296 of file [jurassic.h](#).

#### 4.4.2.15 `double ctl_t::raydz`

Vertical step length for raytracing [km].

Definition at line 299 of file [jurassic.h](#).

#### 4.4.2.16 `char ctl_t::fov[LEN]`

Field-of-view data file.

Definition at line 302 of file [jurassic.h](#).



#### 4.4.2.17 `double ctl_t::retp_zmin`

Minimum altitude for pressure retrieval [km].

Definition at line 305 of file [jurassic.h](#).

#### 4.4.2.18 `double ctl_t::retp_zmax`

Maximum altitude for pressure retrieval [km].

Definition at line 308 of file [jurassic.h](#).

#### 4.4.2.19 `double ctl_t::rett_zmin`

Minimum altitude for temperature retrieval [km].

Definition at line 311 of file [jurassic.h](#).

#### 4.4.2.20 `double ctl_t::rett_zmax`

Maximum altitude for temperature retrieval [km].

Definition at line 314 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 317 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 320 of file [jurassic.h](#).

#### 4.4.2.23 `double ctl_t::retk_zmin[NW]`

Minimum altitude for extinction retrieval [km].

Definition at line 323 of file [jurassic.h](#).

#### 4.4.2.24 `double ctl_t::retk_zmax[NW]`

Maximum altitude for extinction retrieval [km].

Definition at line 326 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 329 of file [jurassic.h](#).

## 4.4.2.26 int ctl\_t::write\_matrix

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

Definition at line 332 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<sup>2</sup>].*

## 4.5.1 Detailed Description

Line-of-sight data.

Definition at line 337 of file [jurassic.h](#).

## 4.5.2 Field Documentation

### 4.5.2.1 `int los_t::np`

Number of LOS points.

Definition at line 340 of file [jurassic.h](#).

### 4.5.2.2 `double los_t::z[NLOS]`

Altitude [km].

Definition at line 343 of file [jurassic.h](#).

### 4.5.2.3 `double los_t::lon[NLOS]`

Longitude [deg].

Definition at line 346 of file [jurassic.h](#).

### 4.5.2.4 `double los_t::lat[NLOS]`

Latitude [deg].

Definition at line 349 of file [jurassic.h](#).

### 4.5.2.5 `double los_t::p[NLOS]`

Pressure [hPa].

Definition at line 352 of file [jurassic.h](#).

### 4.5.2.6 `double los_t::t[NLOS]`

Temperature [K].

Definition at line 355 of file [jurassic.h](#).

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

Volume mixing ratio.

Definition at line 358 of file [jurassic.h](#).

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

Extinction [1/km].

Definition at line 361 of file [jurassic.h](#).

## 4.5.2.9 double los\_t::tsurf

Surface temperature [K].

Definition at line 364 of file [jurassic.h](#).

## 4.5.2.10 double los\_t::ds[NLOS]

Segment length [km].

Definition at line 367 of file [jurassic.h](#).

## 4.5.2.11 double los\_t::u[NG][NLOS]

Column density [molecules/cm<sup>2</sup>].

Definition at line 370 of file [jurassic.h](#).

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

- [jurassic.h](#)

## 4.6 ncd\_t Struct Reference

Buffer for netCDF data.

## Data Fields

- int [ncid](#)  
*NetCDF file ID.*
- int [np](#)  
*Number of retrieval altitudes.*
- double [l1\\_time](#) [L1\_NTRACK][L1\_NXTRACK]  
*Time (seconds since 2000-01-01T00:00Z).*
- double [l1\\_lon](#) [L1\_NTRACK][L1\_NXTRACK]  
*Footprint longitude [deg].*
- double [l1\\_lat](#) [L1\_NTRACK][L1\_NXTRACK]  
*Footprint latitude [deg].*
- double [l1\\_sat\\_z](#) [L1\_NTRACK]  
*Satellite altitude [km].*
- double [l1\\_sat\\_lon](#) [L1\_NTRACK]  
*Satellite longitude [deg].*
- double [l1\\_sat\\_lat](#) [L1\_NTRACK]  
*Satellite latitude [deg].*
- double [l1\\_nu](#) [L1\_NCHAN]  
*Channel frequencies [cm<sup>-1</sup>].*
- float [l1\\_rad](#) [L1\_NTRACK][L1\_NXTRACK][L1\_NCHAN]  
*Radiance [W/(m<sup>2</sup> sr cm<sup>-1</sup>)].*
- double [l2\\_z](#) [L2\_NTRACK][L2\_NXTRACK][L2\_NLAY]  
*Altitude [km].*

- double `l2_p` [L2\_NLAY]  
*Pressure [hPa].*
- double `l2_t` [L2\_NTRACK][L2\_NXTRACK][L2\_NLAY]  
*Temperature [K].*
- float `ret_z` [NP]  
*Altitude [km].*
- float `ret_p` [L1\_NTRACK \*L1\_NXTRACK]  
*Pressure [hPa].*
- float `ret_t` [L1\_NTRACK \*L1\_NXTRACK \*NP]  
*Temperature [K].*

#### 4.6.1 Detailed Description

Buffer for netCDF data.

Definition at line 42 of file [diff\\_apr.c](#).

#### 4.6.2 Field Documentation

##### 4.6.2.1 int ncd\_t::ncid

NetCDF file ID.

Definition at line 45 of file [diff\\_apr.c](#).

##### 4.6.2.2 int ncd\_t::np

Number of retrieval altitudes.

Definition at line 48 of file [diff\\_apr.c](#).

##### 4.6.2.3 double ncd\_t::l1\_time

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

Definition at line 51 of file [diff\\_apr.c](#).

##### 4.6.2.4 double ncd\_t::l1\_lon

Footprint longitude [deg].

Definition at line 54 of file [diff\\_apr.c](#).

##### 4.6.2.5 double ncd\_t::l1\_lat

Footprint latitude [deg].

Definition at line 57 of file [diff\\_apr.c](#).

#### 4.6.2.6 double ncd\_t::l1\_sat\_z

Satellite altitude [km].

Definition at line 60 of file [diff\\_apr.c](#).

#### 4.6.2.7 double ncd\_t::l1\_sat\_lon

Satellite longitude [deg].

Definition at line 63 of file [diff\\_apr.c](#).

#### 4.6.2.8 double ncd\_t::l1\_sat\_lat

Satellite latitude [deg].

Definition at line 66 of file [diff\\_apr.c](#).

#### 4.6.2.9 double ncd\_t::l1\_nu

Channel frequencies [ $\text{cm}^{-1}$ ].

Definition at line 69 of file [diff\\_apr.c](#).

#### 4.6.2.10 float ncd\_t::l1\_rad

Radiance [ $\text{W}/(\text{m}^2 \text{ sr cm}^{-1})$ ].

Definition at line 72 of file [diff\\_apr.c](#).

#### 4.6.2.11 double ncd\_t::l2\_z

Altitude [km].

Definition at line 75 of file [diff\\_apr.c](#).

#### 4.6.2.12 double ncd\_t::l2\_p

Pressure [hPa].

Definition at line 78 of file [diff\\_apr.c](#).

#### 4.6.2.13 double ncd\_t::l2\_t

Temperature [K].

Definition at line 81 of file [diff\\_apr.c](#).

#### 4.6.2.14 float ncd\_t::ret\_z

Altitude [km].

Definition at line 84 of file [diff\\_apr.c](#).

#### 4.6.2.15 float ncd\_t::ret\_p

Pressure [hPa].

Definition at line 87 of file [diff\\_apr.c](#).

#### 4.6.2.16 float ncd\_t::ret\_t

Temperature [K].

Definition at line 90 of file [diff\\_apr.c](#).

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

- [diff\\_apr.c](#)
- [retrieval.c](#)

### 4.7 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 [ $W/(m^2 sr cm^{-1})$ ].*

#### 4.7.1 Detailed Description

Observation geometry and radiance data.

Definition at line 375 of file [jurassic.h](#).

#### 4.7.2 Field Documentation

##### 4.7.2.1 `int obs_t::nr`

Number of ray paths.

Definition at line 378 of file [jurassic.h](#).

##### 4.7.2.2 `double obs_t::time[NR]`

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

Definition at line 381 of file [jurassic.h](#).

##### 4.7.2.3 `double obs_t::obsz[NR]`

Observer altitude [km].

Definition at line 384 of file [jurassic.h](#).

##### 4.7.2.4 `double obs_t::obslon[NR]`

Observer longitude [deg].

Definition at line 387 of file [jurassic.h](#).

##### 4.7.2.5 `double obs_t::obslat[NR]`

Observer latitude [deg].

Definition at line 390 of file [jurassic.h](#).

##### 4.7.2.6 `double obs_t::vpz[NR]`

View point altitude [km].

Definition at line 393 of file [jurassic.h](#).

##### 4.7.2.7 `double obs_t::vplon[NR]`

View point longitude [deg].

Definition at line 396 of file [jurassic.h](#).



#### 4.7.2.8 double obs\_t::vplat[NR]

View point latitude [deg].

Definition at line 399 of file [jurassic.h](#).

#### 4.7.2.9 double obs\_t::tpz[NR]

Tangent point altitude [km].

Definition at line 402 of file [jurassic.h](#).

#### 4.7.2.10 double obs\_t::tpon[NR]

Tangent point longitude [deg].

Definition at line 405 of file [jurassic.h](#).

#### 4.7.2.11 double obs\_t::tplat[NR]

Tangent point latitude [deg].

Definition at line 408 of file [jurassic.h](#).

#### 4.7.2.12 double obs\_t::tau[ND][NR]

Transmittance of ray path.

Definition at line 411 of file [jurassic.h](#).

#### 4.7.2.13 double obs\_t::rad[ND][NR]

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

Definition at line 414 of file [jurassic.h](#).

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

- [jurassic.h](#)

## 4.8 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].*

## 4.8.1 Detailed Description

Perturbation data.

Definition at line 124 of file `libairs.h`.

## 4.8.2 Field Documentation

4.8.2.1 `int pert_t::ntrack`

Number of along-track values.

Definition at line 127 of file `libairs.h`.

4.8.2.2 `int pert_t::nxtrack`

Number of across-track values.

Definition at line 130 of file `libairs.h`.

4.8.2.3 `double pert_t::time[PERT_NTRACK][PERT_NXTRACK]`

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

Definition at line 133 of file `libairs.h`.

#### 4.8.2.4 double pert\_t::lon[PERT\_NTRACK][PERT\_NXTRACK]

Longitude [deg].

Definition at line 136 of file [libairs.h](#).

#### 4.8.2.5 double pert\_t::lat[PERT\_NTRACK][PERT\_NXTRACK]

Latitude [deg].

Definition at line 139 of file [libairs.h](#).

#### 4.8.2.6 double pert\_t::dc[PERT\_NTRACK][PERT\_NXTRACK]

Brightness temperature (8 micron) [K].

Definition at line 142 of file [libairs.h](#).

#### 4.8.2.7 double pert\_t::bt[PERT\_NTRACK][PERT\_NXTRACK]

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

Definition at line 145 of file [libairs.h](#).

#### 4.8.2.8 double pert\_t::pt[PERT\_NTRACK][PERT\_NXTRACK]

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

Definition at line 148 of file [libairs.h](#).

#### 4.8.2.9 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](#)

## 4.9 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](#)  
*Recomputation of kernel matrix (number of iterations).*
- int [conv\\_itmax](#)  
*Maximum number of iterations.*
- double [conv\\_dmin](#)  
*Minimum normalized step size in state space.*
- double [err\\_formod](#) [ND]  
*Forward model error [%].*
- double [err\\_noise](#) [ND]  
*Noise error [ $W/(m^2 \text{ sr cm}^{-1})$ ].*
- double [err\\_press](#)  
*Pressure error [%].*
- double [err\\_press\\_cz](#)  
*Vertical correlation length for pressure error [km].*
- double [err\\_press\\_ch](#)  
*Horizontal correlation length for pressure error [km].*
- double [err\\_temp](#)  
*Temperature error [K].*
- double [err\\_temp\\_cz](#)

- Vertical correlation length for temperature error [km].*
- double `err_temp_ch`
  - Horizontal correlation length for temperature error [km].*
- double `err_q` [NG]
  - Volume mixing ratio error [%].*
- double `err_q_cz` [NG]
  - Vertical correlation length for volume mixing ratio error [km].*
- double `err_q_ch` [NG]
  - Horizontal correlation length for volume mixing ratio error [km].*
- double `err_k` [NW]
  - Extinction error [1/km].*
- double `err_k_cz` [NW]
  - Vertical correlation length for extinction error [km].*
- double `err_k_ch` [NW]
  - Horizontal correlation length for extinction error [km].*

#### 4.9.1 Detailed Description

Retrieval results.

Retrieval control parameters.

Definition at line 156 of file `libairs.h`.

#### 4.9.2 Field Documentation

##### 4.9.2.1 `int ret_t::nds`

Number of data sets.

Definition at line 159 of file `libairs.h`.

##### 4.9.2.2 `int ret_t::np`

Number of data points.

Definition at line 162 of file `libairs.h`.

##### 4.9.2.3 `double ret_t::time[NDS][NPG]`

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

Definition at line 165 of file `libairs.h`.

##### 4.9.2.4 `double ret_t::z[NDS][NPG]`

Altitude [km].

Definition at line 168 of file `libairs.h`.

#### 4.9.2.5 double ret\_t::lon[NDS][NPG]

Longitude [deg].

Definition at line 171 of file [libairs.h](#).

#### 4.9.2.6 double ret\_t::lat[NDS][NPG]

Latitude [deg].

Definition at line 174 of file [libairs.h](#).

#### 4.9.2.7 double ret\_t::p[NDS][NPG]

Pressure [hPa].

Definition at line 177 of file [libairs.h](#).

#### 4.9.2.8 double ret\_t::t[NDS][NPG]

Temperature [K].

Definition at line 180 of file [libairs.h](#).

#### 4.9.2.9 double ret\_t::t\_apr[NDS][NPG]

Temperature (a priori data) [K].

Definition at line 183 of file [libairs.h](#).

#### 4.9.2.10 double ret\_t::t\_tot[NDS][NPG]

Temperature (total error) [K].

Definition at line 186 of file [libairs.h](#).

#### 4.9.2.11 double ret\_t::t\_noise[NDS][NPG]

Temperature (noise error) [K].

Definition at line 189 of file [libairs.h](#).

#### 4.9.2.12 double ret\_t::t\_fm[NDS][NPG]

Temperature (forward model error) [K].

Definition at line 192 of file [libairs.h](#).

#### 4.9.2.13 double ret\_t::t\_cont[NDS][NPG]

Temperature (measurement content).

Definition at line 195 of file [libairs.h](#).

#### 4.9.2.14 `double ret_t::t_res[NDS][NPG]`

Temperature (resolution).

Definition at line 198 of file [libairs.h](#).

#### 4.9.2.15 `double ret_t::chisq[NDS]`

$\chi^2$ .

Definition at line 201 of file [libairs.h](#).

#### 4.9.2.16 `int ret_t::kernel_recomp`

Recomputation of kernel matrix (number of iterations).

Definition at line 99 of file [retrieval.c](#).

#### 4.9.2.17 `int ret_t::conv_itmax`

Maximum number of iterations.

Definition at line 102 of file [retrieval.c](#).

#### 4.9.2.18 `double ret_t::conv_dmin`

Minimum normalized step size in state space.

Definition at line 105 of file [retrieval.c](#).

#### 4.9.2.19 `double ret_t::err_formod[ND]`

Forward model error [%].

Definition at line 108 of file [retrieval.c](#).

#### 4.9.2.20 `double ret_t::err_noise[ND]`

Noise error [ $\text{W}/(\text{m}^2 \text{ sr cm}^{-1})$ ].

Definition at line 111 of file [retrieval.c](#).

#### 4.9.2.21 `double ret_t::err_press`

Pressure error [%].

Definition at line 114 of file [retrieval.c](#).

#### 4.9.2.22 `double ret_t::err_press_cz`

Vertical correlation length for pressure error [km].

Definition at line 117 of file [retrieval.c](#).

**4.9.2.23 double ret\_t::err\_press\_ch**

Horizontal correlation length for pressure error [km].

Definition at line 120 of file [retrieval.c](#).

**4.9.2.24 double ret\_t::err\_temp**

Temperature error [K].

Definition at line 123 of file [retrieval.c](#).

**4.9.2.25 double ret\_t::err\_temp\_cz**

Vertical correlation length for temperature error [km].

Definition at line 126 of file [retrieval.c](#).

**4.9.2.26 double ret\_t::err\_temp\_ch**

Horizontal correlation length for temperature error [km].

Definition at line 129 of file [retrieval.c](#).

**4.9.2.27 double ret\_t::err\_q[NG]**

Volume mixing ratio error [%].

Definition at line 132 of file [retrieval.c](#).

**4.9.2.28 double ret\_t::err\_q\_cz[NG]**

Vertical correlation length for volume mixing ratio error [km].

Definition at line 135 of file [retrieval.c](#).

**4.9.2.29 double ret\_t::err\_q\_ch[NG]**

Horizontal correlation length for volume mixing ratio error [km].

Definition at line 138 of file [retrieval.c](#).

**4.9.2.30 double ret\_t::err\_k[NW]**

Extinction error [1/km].

Definition at line 141 of file [retrieval.c](#).

**4.9.2.31 double ret\_t::err\_k\_cz[NW]**

Vertical correlation length for extinction error [km].

Definition at line 144 of file [retrieval.c](#).



#### 4.9.2.32 double ret\_t::err\_k\_ch[NW]

Horizontal correlation length for extinction error [km].

Definition at line 147 of file [retrieval.c](#).

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

- [libairs.h](#)
- [retrieval.c](#)

### 4.10 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<sup>2</sup>].*
- float [eps](#) [NG][ND][TBLNP][TBLNT][TBLNU]  
*Emissivity.*
- double [st](#) [TBLNS]  
*Source function temperature [K].*
- double [sr](#) [ND][TBLNS]  
*Source function radiance [W/(m<sup>2</sup> sr cm<sup>-1</sup>)].*

#### 4.10.1 Detailed Description

Emissivity look-up tables.

Definition at line 419 of file [jurassic.h](#).

#### 4.10.2 Field Documentation

##### 4.10.2.1 int tbl\_t::np[NG][ND]

Number of pressure levels.

Definition at line 422 of file [jurassic.h](#).

#### 4.10.2.2 int tbl\_t::nt[NG][ND][TBLNP]

Number of temperatures.

Definition at line 425 of file [jurassic.h](#).

#### 4.10.2.3 int tbl\_t::nu[NG][ND][TBLNP][TBLNT]

Number of column densities.

Definition at line 428 of file [jurassic.h](#).

#### 4.10.2.4 double tbl\_t::p[NG][ND][TBLNP]

Pressure [hPa].

Definition at line 431 of file [jurassic.h](#).

#### 4.10.2.5 double tbl\_t::t[NG][ND][TBLNP][TBLNT]

Temperature [K].

Definition at line 434 of file [jurassic.h](#).

#### 4.10.2.6 float tbl\_t::u[NG][ND][TBLNP][TBLNT][TBLNU]

Column density [molecules/cm<sup>2</sup>].

Definition at line 437 of file [jurassic.h](#).

#### 4.10.2.7 float tbl\_t::eps[NG][ND][TBLNP][TBLNT][TBLNU]

Emissivity.

Definition at line 440 of file [jurassic.h](#).

#### 4.10.2.8 double tbl\_t::st[TBLNS]

Source function temperature [K].

Definition at line 443 of file [jurassic.h](#).

#### 4.10.2.9 double tbl\_t::sr[ND][TBLNS]

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

Definition at line 446 of file [jurassic.h](#).

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

- [jurassic.h](#)

## 4.11 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].*

### 4.11.1 Detailed Description

Wave analysis data.

Definition at line [206](#) of file [libairs.h](#).

### 4.11.2 Field Documentation

#### 4.11.2.1 int wave\_t::nx

Number of across-track values.

Definition at line [209](#) of file [libairs.h](#).

4.11.2.2 `int wave_t::ny`

Number of along-track values.

Definition at line 212 of file [libairs.h](#).

4.11.2.3 `double wave_t::time`

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

Definition at line 215 of file [libairs.h](#).

4.11.2.4 `double wave_t::z`

Altitude [km].

Definition at line 218 of file [libairs.h](#).

4.11.2.5 `double wave_t::lon[WX][WY]`

Longitude [deg].

Definition at line 221 of file [libairs.h](#).

4.11.2.6 `double wave_t::lat[WX][WY]`

Latitude [deg].

Definition at line 224 of file [libairs.h](#).

4.11.2.7 `double wave_t::x[WX]`

Across-track distance [km].

Definition at line 227 of file [libairs.h](#).

4.11.2.8 `double wave_t::y[WY]`

Along-track distance [km].

Definition at line 230 of file [libairs.h](#).

4.11.2.9 `double wave_t::temp[WX][WY]`

Temperature [K].

Definition at line 233 of file [libairs.h](#).

4.11.2.10 `double wave_t::bg[WX][WY]`

Background [K].

Definition at line 236 of file [libairs.h](#).

#### 4.11.2.11 double wave\_t::pt[WX][WY]

Perturbation [K].

Definition at line 239 of file [libairs.h](#).

#### 4.11.2.12 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 day2doy.c File Reference

#### Functions

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

#### 5.1.1 Function Documentation

##### 5.1.1.1 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:



## 5.2 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 }

```

## 5.3 diff\_apr.c File Reference

### Data Structures

- struct [ncd\\_t](#)  
*Buffer for netCDF data.*

### Functions

- void [read\\_nc](#) (char \*filename, [ncd\\_t](#) \*ncd)
- int [main](#) (int argc, char \*argv[])

### 5.3.1 Function Documentation

#### 5.3.1.1 void read\_nc ( char \* filename, ncd\_t \* ncd )

Definition at line 205 of file [diff\\_apr.c](#).

```

00207     {
00208
00209     int varid;
00210
00211     /* Open netCDF file... */
00212     printf("Read netCDF file: %s\n", filename);
00213     NC(nc_open(filename, NC_WRITE, &ncd->ncid));
00214
00215     /* Read Level-1 data... */
00216     NC(nc_inq_varid(ncd->ncid, "l1_time", &varid));
00217     NC(nc_get_var_double(ncd->ncid, varid, ncd->l1_time[0]));
00218     NC(nc_inq_varid(ncd->ncid, "l1_lon", &varid));
00219     NC(nc_get_var_double(ncd->ncid, varid, ncd->l1_lon[0]));
00220     NC(nc_inq_varid(ncd->ncid, "l1_lat", &varid));
00221     NC(nc_get_var_double(ncd->ncid, varid, ncd->l1_lat[0]));
00222     NC(nc_inq_varid(ncd->ncid, "l1_sat_z", &varid));
00223     NC(nc_get_var_double(ncd->ncid, varid, ncd->l1_sat_z));
00224     NC(nc_inq_varid(ncd->ncid, "l1_sat_lon", &varid));
00225     NC(nc_get_var_double(ncd->ncid, varid, ncd->l1_sat_lon));
00226     NC(nc_inq_varid(ncd->ncid, "l1_sat_lat", &varid));
00227     NC(nc_get_var_double(ncd->ncid, varid, ncd->l1_sat_lat));
00228     NC(nc_inq_varid(ncd->ncid, "l1_nu", &varid));
00229     NC(nc_get_var_double(ncd->ncid, varid, ncd->l1_nu));

```

```

00230 NC(nc_inq_varid(ncd->ncid, "l1_rad", &varid));
00231 NC(nc_get_var_float(ncd->ncid, varid, ncd->l1_rad[0][0]));
00232
00233 /* Read Level-2 data... */
00234 NC(nc_inq_varid(ncd->ncid, "l2_z", &varid));
00235 NC(nc_get_var_double(ncd->ncid, varid, ncd->l2_z[0][0]));
00236 NC(nc_inq_varid(ncd->ncid, "l2_press", &varid));
00237 NC(nc_get_var_double(ncd->ncid, varid, ncd->l2_p));
00238 NC(nc_inq_varid(ncd->ncid, "l2_temp", &varid));
00239 NC(nc_get_var_double(ncd->ncid, varid, ncd->l2_t[0][0]));
00240 }

```

### 5.3.1.2 int main ( int argc, char \* argv[] )

Definition at line 107 of file [diff\\_apr.c](#).

```

00109 {
00110
00111     static ctl_t ctl;
00112
00113     static ncd_t ncd, ncd2;
00114
00115     static FILE *out;
00116
00117     static double mean[L2_NLAY], sigma[L2_NLAY], min[L2_NLAY], max[L2_NLAY],
00118         tt[L2_NLAY], lon[L2_NLAY], lat[L2_NLAY], temp[L2_NLAY], press[L2_NLAY],
00119         z[L2_NLAY], tip;
00120
00121     static int idx, ip, itrack, ixtrack;
00122
00123     /* Check arguments... */
00124     if (argc < 5)
00125         ERRMSG("Give parameters: <ctl> <airs.nc> <airs2.nc> <diff.tab>");
00126
00127     /* Read control parameters... */
00128     read_ctl(argc, argv, &ctl);
00129
00130     /* Read netCDF files... */
00131     read_nc(argv[2], &ncd);
00132     read_nc(argv[3], &ncd2);
00133
00134     /* Compute differences... */
00135     for (itrack = 0; itrack < L2_NTRACK; itrack++)
00136         for (ixtrack = 0; ixtrack < L2_NXTRACK; ixtrack++) {
00137             for (ip = 0; ip < L2_NLAY; ip++) {
00138                 if (ncd.l1_time[3 * itrack + 1][3 * ixtrack + 1] !=
00139                     ncd2.l1_time[3 * itrack + 1][3 * ixtrack + 1] ||
00140                     ncd.l1_lon[3 * itrack + 1][3 * ixtrack + 1] !=
00141                     ncd2.l1_lon[3 * itrack + 1][3 * ixtrack + 1] ||
00142                     ncd.l1_lat[3 * itrack + 1][3 * ixtrack + 1] !=
00143                     ncd2.l1_lat[3 * itrack + 1][3 * ixtrack + 1])
00144                     ERRMSG("Data files do not match!");
00145                 tt[ip] += ncd.l1_time[3 * itrack + 1][3 * ixtrack + 1];
00146                 lon[ip] += ncd.l1_lon[3 * itrack + 1][3 * ixtrack + 1];
00147                 lat[ip] += ncd.l1_lat[3 * itrack + 1][3 * ixtrack + 1];
00148                 z[ip] += ncd.l2_z[itrack][ixtrack][ip];
00149                 press[ip] += ncd.l2_p[ip];
00150                 temp[ip] += ncd.l2_t[itrack][ixtrack][ip];
00151                 idx =
00152                     locate_irr(ncd2.l2_z[itrack][ixtrack], L2_NLAY,
00153                             ncd.l2_z[itrack][ixtrack][ip]);
00154                 tip =
00155                     LIN(ncd2.l2_z[itrack][ixtrack][idx],
00156                        ncd2.l2_t[itrack][ixtrack][idx],
00157                        ncd2.l2_z[itrack][ixtrack][idx + 1],
00158                        ncd2.l2_t[itrack][ixtrack][idx + 1],
00159                        ncd.l2_z[itrack][ixtrack][ip]);
00160                 mean[ip] += tip - ncd.l2_t[itrack][ixtrack][ip];
00161                 sigma[ip] += gsl_pow_2(tip - ncd.l2_t[itrack][ixtrack][ip]);
00162                 min[ip] = GSL_MIN(min[ip], tip - ncd.l2_t[itrack][ixtrack][ip]);
00163                 max[ip] = GSL_MAX(max[ip], tip - ncd.l2_t[itrack][ixtrack][ip]);
00164             }
00165         }
00166
00167     /* Create output file... */
00168     printf("Write a priori differences data: %s\n", argv[4]);
00169     if (!(out = fopen(argv[4], "w")))
00170         ERRMSG("Cannot create file!");
00171
00172     /* Write header... */
00173     fprintf(out,

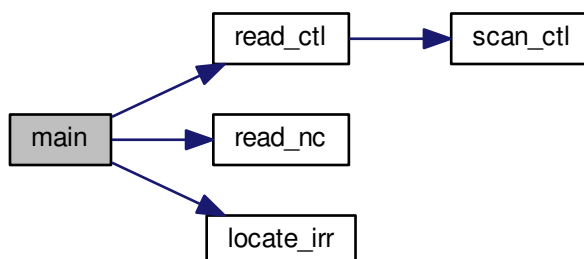
```

```

00174     "# $1 = time (seconds since 01-JAN-2000, 00:00 UTC)\n"
00175     "# $2 = altitude [km]\n"
00176     "# $3 = longitude [deg]\n"
00177     "# $4 = latitude [deg]\n"
00178     "# $5 = pressure (set 1) [hPa]\n"
00179     "# $6 = temperature (set 1) [K]\n"
00180     "# $7 = temperature difference (mean, set 2 - set 1) [K]\n"
00181     "# $8 = temperature difference (sigma, set 2 - set 1) [K]\n"
00182     "# $9 = temperature difference (minimum, set 2 - set 1) [K]\n"
00183     "# $10 = temperature difference (maximum, set 2 - set 1) [K]\n\n");
00184
00185 /* Write output... */
00186 for (ip = 0; ip < L2_NLAY; ip++)
00187     fprintf(out, "%.2f %g %g %g %g %g %g %g %g\n",
00188         tt[ip] / (L2_NTRACK * L2_NXTRACK),
00189         z[ip] / (L2_NTRACK * L2_NXTRACK),
00190         lon[ip] / (L2_NTRACK * L2_NXTRACK),
00191         lat[ip] / (L2_NTRACK * L2_NXTRACK),
00192         press[ip] / (L2_NTRACK * L2_NXTRACK),
00193         temp[ip] / (L2_NTRACK * L2_NXTRACK),
00194         mean[ip] / (L2_NTRACK * L2_NXTRACK),
00195         sqrt(sigma[ip] / (L2_NTRACK * L2_NXTRACK)) -
00196         gsl_pow_2(mean[ip] / (L2_NTRACK * L2_NXTRACK))), min[ip],
00197         max[ip]);
00198
00199 /* Close file... */
00200 fclose(out);
00201 }

```

Here is the call graph for this function:



## 5.4 diff\_apr.c

```

00001 #include <omp.h>
00002 #include <netcdf.h>
00003 #include "jurassic.h"
00004
00005 /* -----
00006     Macros...
00007     ----- */
00008
00009 /* Execute netCDF library command and check result. */
00010 #define NC(cmd) {
00011     if ((cmd) != NC_NOERR)
00012         ERRMSG(nc_strerror(cmd));
00013 }
00014
00015 /* -----
00016     Dimensions...
00017     ----- */
00018
00019 /* Number of AIRS radiance channels (don't change). */
00020 #define L1_NCHAN 34
00021
00022 /* Along-track size of AIRS radiance granule (don't change). */
00023 #define L1_NTRACK 135

```



```

00024
00025 /* Across-track size of AIRS radiance granule (don't change). */
00026 #define L1_NXTRACK 90
00027
00028 /* Number of AIRS pressure layers (don't change). */
00029 #define L2_NLAY 27
00030
00031 /* Along-track size of AIRS retrieval granule (don't change). */
00032 #define L2_NTRACK 45
00033
00034 /* Across-track size of AIRS retrieval granule (don't change). */
00035 #define L2_NXTRACK 30
00036
00037 /* -----
00038     Structs...
00039     ----- */
00040
00041 /* Buffer for netCDF data. */
00042 typedef struct {
00043
00044     /* NetCDF file ID. */
00045     int ncid;
00046
00047     /* Number of retrieval altitudes. */
00048     int np;
00049
00050     /* Time (seconds since 2000-01-01T00:00Z). */
00051     double l1_time[L1_NTRACK][L1_NXTRACK];
00052
00053     /* Footprint longitude [deg]. */
00054     double l1_lon[L1_NTRACK][L1_NXTRACK];
00055
00056     /* Footprint latitude [deg]. */
00057     double l1_lat[L1_NTRACK][L1_NXTRACK];
00058
00059     /* Satellite altitude [km]. */
00060     double l1_sat_z[L1_NTRACK];
00061
00062     /* Satellite longitude [deg]. */
00063     double l1_sat_lon[L1_NTRACK];
00064
00065     /* Satellite latitude [deg]. */
00066     double l1_sat_lat[L1_NTRACK];
00067
00068     /* Channel frequencies [cm^-1]. */
00069     double l1_nu[L1_NCHAN];
00070
00071     /* Radiance [W/(m^2 sr cm^-1)]. */
00072     float l1_rad[L1_NTRACK][L1_NXTRACK][L1_NCHAN];
00073
00074     /* Altitude [km]. */
00075     double l2_z[L2_NTRACK][L2_NXTRACK][L2_NLAY];
00076
00077     /* Pressure [hPa]. */
00078     double l2_p[L2_NLAY];
00079
00080     /* Temperature [K]. */
00081     double l2_t[L2_NTRACK][L2_NXTRACK][L2_NLAY];
00082
00083     /* Altitude [km]. */
00084     float ret_z[NP];
00085
00086     /* Pressure [hPa]. */
00087     float ret_p[L1_NTRACK * L1_NXTRACK];
00088
00089     /* Temperature [K]. */
00090     float ret_t[L1_NTRACK * L1_NXTRACK * NP];
00091
00092 } ncd_t;
00093
00094 /* -----
00095     Functions...
00096     ----- */
00097
00098 /* Read netCDF file. */
00099 void read_nc(
00100     char *filename,
00101     ncd_t * ncd);
00102
00103 /* -----
00104     Main...
00105     ----- */
00106
00107 int main(
00108     int argc,
00109     char *argv[]) {
00110

```

```

00111 static ctl_t ctl;
00112
00113 static ncd_t ncd, ncd2;
00114
00115 static FILE *out;
00116
00117 static double mean[L2_NLAY], sigma[L2_NLAY], min[L2_NLAY], max[L2_NLAY],
00118             tt[L2_NLAY], lon[L2_NLAY], lat[L2_NLAY], temp[L2_NLAY], press[L2_NLAY],
00119             z[L2_NLAY], tip;
00120
00121 static int idx, ip, itrack, ixtrack;
00122
00123 /* Check arguments... */
00124 if (argc < 5)
00125     ERRMSG("Give parameters: <ctl> <airs.nc> <airs2.nc> <diff.tab>");
00126
00127 /* Read control parameters... */
00128 read_ctl(argc, argv, &ctl);
00129
00130 /* Read netCDF files... */
00131 read_nc(argv[2], &ncd);
00132 read_nc(argv[3], &ncd2);
00133
00134 /* Compute differences... */
00135 for (itrack = 0; itrack < L2_NTRACK; itrack++)
00136     for (ixtrack = 0; ixtrack < L2_NXTRACK; ixtrack++) {
00137         for (ip = 0; ip < L2_NLAY; ip++) {
00138             if (ncd.l1_time[3 * itrack + 1][3 * ixtrack + 1] !=
00139                 ncd2.l1_time[3 * itrack + 1][3 * ixtrack + 1] ||
00140                 ncd.l1_lon[3 * itrack + 1][3 * ixtrack + 1] !=
00141                 ncd2.l1_lon[3 * itrack + 1][3 * ixtrack + 1] ||
00142                 ncd.l1_lat[3 * itrack + 1][3 * ixtrack + 1] !=
00143                 ncd2.l1_lat[3 * itrack + 1][3 * ixtrack + 1])
00144                 ERRMSG("Data files do not match!");
00145             tt[ip] += ncd.l1_time[3 * itrack + 1][3 * ixtrack + 1];
00146             lon[ip] += ncd.l1_lon[3 * itrack + 1][3 * ixtrack + 1];
00147             lat[ip] += ncd.l1_lat[3 * itrack + 1][3 * ixtrack + 1];
00148             z[ip] += ncd.l2_z[itrack][ixtrack][ip];
00149             press[ip] += ncd.l2_p[ip];
00150             temp[ip] += ncd.l2_t[itrack][ixtrack][ip];
00151             idx =
00152                 locate_irr(ncd2.l2_z[itrack][ixtrack], L2_NLAY,
00153                           ncd.l2_z[itrack][ixtrack][ip]);
00154             tip =
00155                 LIN(ncd2.l2_z[itrack][ixtrack][idx],
00156                   ncd2.l2_t[itrack][ixtrack][idx],
00157                   ncd2.l2_z[itrack][ixtrack][idx + 1],
00158                   ncd2.l2_t[itrack][ixtrack][idx + 1],
00159                   ncd.l2_z[itrack][ixtrack][ip]);
00160             mean[ip] += tip - ncd.l2_t[itrack][ixtrack][ip];
00161             sigma[ip] += gsl_pow_2(tip - ncd.l2_t[itrack][ixtrack][ip]);
00162             min[ip] = GSL_MIN(min[ip], tip - ncd.l2_t[itrack][ixtrack][ip]);
00163             max[ip] = GSL_MAX(max[ip], tip - ncd.l2_t[itrack][ixtrack][ip]);
00164         }
00165     }
00166
00167 /* Create output file... */
00168 printf("Write a priori differences data: %s\n", argv[4]);
00169 if (!(out = fopen(argv[4], "w")))
00170     ERRMSG("Cannot create file!");
00171
00172 /* Write header... */
00173 fprintf(out,
00174         "# $1 = time (seconds since 01-JAN-2000, 00:00 UTC)\n"
00175         "# $2 = altitude [km]\n"
00176         "# $3 = longitude [deg]\n"
00177         "# $4 = latitude [deg]\n"
00178         "# $5 = pressure (set 1) [hPa]\n"
00179         "# $6 = temperature (set 1) [K]\n"
00180         "# $7 = temperature difference (mean, set 2 - set 1) [K]\n"
00181         "# $8 = temperature difference (sigma, set 2 - set 1) [K]\n"
00182         "# $9 = temperature difference (minimum, set 2 - set 1) [K]\n"
00183         "# $10 = temperature difference (maximum, set 2 - set 1) [K]\n\n");
00184
00185 /* Write output... */
00186 for (ip = 0; ip < L2_NLAY; ip++)
00187     fprintf(out, "%.2f %g %g %g %g %g %g %g %g %g\n",
00188             tt[ip] / (L2_NTRACK * L2_NXTRACK),
00189             z[ip] / (L2_NTRACK * L2_NXTRACK),
00190             lon[ip] / (L2_NTRACK * L2_NXTRACK),
00191             lat[ip] / (L2_NTRACK * L2_NXTRACK),
00192             press[ip] / (L2_NTRACK * L2_NXTRACK),
00193             temp[ip] / (L2_NTRACK * L2_NXTRACK),
00194             mean[ip] / (L2_NTRACK * L2_NXTRACK),
00195             sqrt(sigma[ip] / (L2_NTRACK * L2_NXTRACK)) -
00196             gsl_pow_2(mean[ip] / (L2_NTRACK * L2_NXTRACK))), min[ip],
00197             max[ip]);

```

```

00198
00199  /* Close file... */
00200  fclose(out);
00201 }
00202
00203 /*****
00204
00205 void read_nc(
00206     char *filename,
00207     ncd_t * ncd) {
00208
00209     int varid;
00210
00211     /* Open netCDF file... */
00212     printf("Read netCDF file: %s\n", filename);
00213     NC(nc_open(filename, NC_WRITE, &ncd->ncid));
00214
00215     /* Read Level-1 data... */
00216     NC(nc_inq_varid(ncd->ncid, "l1_time", &varid));
00217     NC(nc_get_var_double(ncd->ncid, varid, ncd->l1_time[0]));
00218     NC(nc_inq_varid(ncd->ncid, "l1_lon", &varid));
00219     NC(nc_get_var_double(ncd->ncid, varid, ncd->l1_lon[0]));
00220     NC(nc_inq_varid(ncd->ncid, "l1_lat", &varid));
00221     NC(nc_get_var_double(ncd->ncid, varid, ncd->l1_lat[0]));
00222     NC(nc_inq_varid(ncd->ncid, "l1_sat_z", &varid));
00223     NC(nc_get_var_double(ncd->ncid, varid, ncd->l1_sat_z));
00224     NC(nc_inq_varid(ncd->ncid, "l1_sat_lon", &varid));
00225     NC(nc_get_var_double(ncd->ncid, varid, ncd->l1_sat_lon));
00226     NC(nc_inq_varid(ncd->ncid, "l1_sat_lat", &varid));
00227     NC(nc_get_var_double(ncd->ncid, varid, ncd->l1_sat_lat));
00228     NC(nc_inq_varid(ncd->ncid, "l1_nu", &varid));
00229     NC(nc_get_var_double(ncd->ncid, varid, ncd->l1_nu));
00230     NC(nc_inq_varid(ncd->ncid, "l1_rad", &varid));
00231     NC(nc_get_var_float(ncd->ncid, varid, ncd->l1_rad[0][0]));
00232
00233     /* Read Level-2 data... */
00234     NC(nc_inq_varid(ncd->ncid, "l2_z", &varid));
00235     NC(nc_get_var_double(ncd->ncid, varid, ncd->l2_z[0][0]));
00236     NC(nc_inq_varid(ncd->ncid, "l2_press", &varid));
00237     NC(nc_get_var_double(ncd->ncid, varid, ncd->l2_p));
00238     NC(nc_inq_varid(ncd->ncid, "l2_temp", &varid));
00239     NC(nc_get_var_double(ncd->ncid, varid, ncd->l2_t[0][0]));
00240 }

```

## 5.5 diff\_ret.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 3 of file `diff_ret.c`.

```

00005     {
00006
00007     static ret_t ret, ret2;
00008
00009     static FILE *out;
00010
00011     static double mean[NPG], sigma[NPG], min[NPG], max[NPG],
00012         tt[NPG], lon[NPG], lat[NPG], temp[NPG], press[NPG];
00013
00014     static int ids, ip;
00015
00016     /* Check arguments... */
00017     if (argc < 5)
00018         ERRMSG("Give parameters: <ctl> <airs.nc> <airs2.nc> <diff.tab>");
00019
00020     /* Read AIRS data... */
00021     read_retr(argv[2], &ret);
00022     read_retr(argv[3], &ret2);

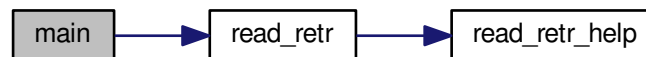
```

```

00023
00024 /* Compute differences... */
00025 for (ids = 0; ids < ret.nds; ids++)
00026     for (ip = 0; ip < ret.np; ip++) {
00027         if (ret.time[ids][ip] != ret2.time[ids][ip] ||
00028             ret.lon[ids][ip] != ret2.lon[ids][ip] ||
00029             ret.lat[ids][ip] != ret2.lat[ids][ip])
00030             ERRMSG("Data files do not match!");
00031         tt[ip] += ret.time[ids][ip];
00032         lon[ip] += ret.lon[ids][ip];
00033         lat[ip] += ret.lat[ids][ip];
00034         press[ip] += ret.p[ids][ip];
00035         temp[ip] += ret.t[ids][ip];
00036         mean[ip] += ret2.t[ids][ip] - ret.t[ids][ip];
00037         sigma[ip] += gsl_pow_2(ret2.t[ids][ip] - ret.t[ids][ip]);
00038         min[ip] = GSL_MIN(min[ip], ret2.t[ids][ip] - ret.t[ids][ip]);
00039         max[ip] = GSL_MAX(max[ip], ret2.t[ids][ip] - ret.t[ids][ip]);
00040     }
00041
00042 /* Create output file... */
00043 printf("Write retrieval differences data: %s\n", argv[4]);
00044 if (!out = fopen(argv[4], "w"))
00045     ERRMSG("Cannot create file!");
00046
00047 /* Write header... */
00048 fprintf(out,
00049     "# $1 = time (seconds since 01-JAN-2000, 00:00 UTC)\n"
00050     "# $2 = altitude [km]\n"
00051     "# $3 = longitude [deg]\n"
00052     "# $4 = latitude [deg]\n"
00053     "# $5 = pressure (set 1) [hPa]\n"
00054     "# $6 = temperature (set 1) [K]\n"
00055     "# $7 = temperature difference (mean, set 2 - set 1) [K]\n"
00056     "# $8 = temperature difference (sigma, set 2 - set 1) [K]\n"
00057     "# $9 = temperature difference (minimum, set 2 - set 1) [K]\n"
00058     "# $10 = temperature difference (maximum, set 2 - set 1) [K]\n\n");
00059
00060 /* Write output... */
00061 for (ip = 0; ip < ret.np; ip++)
00062     fprintf(out, "%.2f %g %g %g %g %g %g %g %g %g\n",
00063         tt[ip] / ret.nds, ret.z[0][ip], lon[ip] / ret.nds,
00064         lat[ip] / ret.nds, press[ip] / ret.nds, temp[ip] / ret.nds,
00065         mean[ip] / ret.nds,
00066         sqrt(sigma[ip] / ret.nds - gsl_pow_2(mean[ip] / ret.nds)),
00067         min[ip], max[ip]);
00068
00069 /* Close file... */
00070 fclose(out);
00071
00072 return EXIT_SUCCESS;
00073 }

```

Here is the call graph for this function:



## 5.6 diff\_ret.c

```

00001 #include "libairs.h"
00002
00003 int main(
00004     int argc,
00005     char *argv[]) {
00006
00007     static ret_t ret, ret2;
00008
00009     static FILE *out;

```

```

00010
00011 static double mean[NPG], sigma[NPG], min[NPG], max[NPG],
00012         tt[NPG], lon[NPG], lat[NPG], temp[NPG], press[NPG];
00013
00014 static int ids, ip;
00015
00016 /* Check arguments... */
00017 if (argc < 5)
00018     ERRMSG("Give parameters: <ctl> <airs.nc> <airs2.nc> <diff.tab>");
00019
00020 /* Read AIRS data... */
00021 read_retr(argv[2], &ret);
00022 read_retr(argv[3], &ret2);
00023
00024 /* Compute differences... */
00025 for (ids = 0; ids < ret.nds; ids++)
00026     for (ip = 0; ip < ret.np; ip++) {
00027         if (ret.time[ids][ip] != ret2.time[ids][ip] ||
00028             ret.lon[ids][ip] != ret2.lon[ids][ip] ||
00029             ret.lat[ids][ip] != ret2.lat[ids][ip])
00030             ERRMSG("Data files do not match!");
00031         tt[ip] += ret.time[ids][ip];
00032         lon[ip] += ret.lon[ids][ip];
00033         lat[ip] += ret.lat[ids][ip];
00034         press[ip] += ret.p[ids][ip];
00035         temp[ip] += ret.t[ids][ip];
00036         mean[ip] += ret2.t[ids][ip] - ret.t[ids][ip];
00037         sigma[ip] += gsl_pow_2(ret2.t[ids][ip] - ret.t[ids][ip]);
00038         min[ip] = GSL_MIN(min[ip], ret2.t[ids][ip] - ret.t[ids][ip]);
00039         max[ip] = GSL_MAX(max[ip], ret2.t[ids][ip] - ret.t[ids][ip]);
00040     }
00041
00042 /* Create output file... */
00043 printf("Write retrieval differences data: %s\n", argv[4]);
00044 if (!(out = fopen(argv[4], "w")))
00045     ERRMSG("Cannot create file!");
00046
00047 /* Write header... */
00048 fprintf(out,
00049         "# $1 = time (seconds since 01-JAN-2000, 00:00 UTC)\n"
00050         "# $2 = altitude [km]\n"
00051         "# $3 = longitude [deg]\n"
00052         "# $4 = latitude [deg]\n"
00053         "# $5 = pressure (set 1) [hPa]\n"
00054         "# $6 = temperature (set 1) [K]\n"
00055         "# $7 = temperature difference (mean, set 2 - set 1) [K]\n"
00056         "# $8 = temperature difference (sigma, set 2 - set 1) [K]\n"
00057         "# $9 = temperature difference (minimum, set 2 - set 1) [K]\n"
00058         "# $10 = temperature difference (maximum, set 2 - set 1) [K]\n\n");
00059
00060 /* Write output... */
00061 for (ip = 0; ip < ret.np; ip++)
00062     fprintf(out, "%.2f %g %g %g %g %g %g %g %g %g\n",
00063             tt[ip] / ret.nds, ret.z[0][ip], lon[ip] / ret.nds,
00064             lat[ip] / ret.nds, press[ip] / ret.nds, temp[ip] / ret.nds,
00065             mean[ip] / ret.nds,
00066             sqrt(sigma[ip] / ret.nds - gsl_pow_2(mean[ip] / ret.nds)),
00067             min[ip], max[ip]);
00068
00069 /* Close file... */
00070 fclose(out);
00071
00072 return EXIT_SUCCESS;
00073 }

```

## 5.7 distance.c File Reference

### Functions

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

#### 5.7.1 Function Documentation

##### 5.7.1.1 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:



## 5.8 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 }

```

## 5.9 doy2day.c File Reference

### Functions

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

## 5.9.1 Function Documentation

### 5.9.1.1 `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:



## 5.10 `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 }
```

## 5.11 `events.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 [events.c](#).

```

00005         {
00006
00007     static pert_t *pert;
00008
00009     static wave_t *wave;
00010
00011     static FILE *in, *out;
00012
00013     static char pertname[LEN];
00014
00015     static double gauss_fwhm, var_dh, varmin, varmax, nu, t230 = 230.0,
00016         dt230, tbg, nesr, nedt = 0;
00017
00018     static int iarg, ix, iy, bg_poly_x, bg_poly_y, bg_smooth_x, bg_smooth_y,
00019         itrack, itrack2, itrackmax, ixtrack, ixtrack2, ixtrackmax, dtrack = 15,
00020         dxtrack = 15;
00021
00022     /* Check arguments... */
00023     if (argc < 4)
00024         ERRMSG("Give parameters: <ctl> <events.tab> <pert1.nc> [<pert2.nc> ...]");
00025
00026     /* Get control parameters... */
00027     scan_ctl(argc, argv, "PERTNAME", -1, "4mu", pertname);
00028     bg_poly_x = (int) scan_ctl(argc, argv, "BG_POLY_X", -1, "0", NULL);
00029     bg_poly_y = (int) scan_ctl(argc, argv, "BG_POLY_Y", -1, "0", NULL);
00030     bg_smooth_x = (int) scan_ctl(argc, argv, "BG_SMOOTH_X", -1, "0", NULL);
00031     bg_smooth_y = (int) scan_ctl(argc, argv, "BG_SMOOTH_Y", -1, "0", NULL);
00032     gauss_fwhm = scan_ctl(argc, argv, "GAUSS_FWHM", -1, "0", NULL);
00033     var_dh = scan_ctl(argc, argv, "VAR_DH", -1, "0", NULL);
00034     varmin = scan_ctl(argc, argv, "VARMIN", -1, "", NULL);
00035     dt230 = scan_ctl(argc, argv, "DT230", -1, "0.16", NULL);
00036     nu = scan_ctl(argc, argv, "NU", -1, "2345.0", NULL);
00037
00038     /* Alloc... */
00039     ALLOC(pert, pert_t, 1);
00040
00041     /* Create file... */
00042     printf("Write event data: %s\n", argv[2]);
00043     if (!(out = fopen(argv[2], "w")))
00044         ERRMSG("Cannot create file!");
00045
00046     /* Write header... */
00047     fprintf(out,
00048         "# $1 = time [s]\n"
00049         "# $2 = longitude [deg]\n"
00050         "# $3 = latitude [deg]\n" "# $4 = maximum variance [K^2]\n\n");
00051
00052     /* Loop over perturbation files... */
00053     for (iarg = 3; iarg < argc; iarg++) {
00054
00055         /* Read perturbation data... */
00056         if (!(in = fopen(argv[iarg], "r")))
00057             continue;
00058         else {
00059             fclose(in);
00060             read_pert(argv[iarg], pertname, pert);
00061         }
00062
00063         /* Recalculate background and perturbations... */
00064         if (bg_poly_x > 0 || bg_poly_y > 0 ||
00065             bg_smooth_x > 0 || bg_smooth_y > 0 || gauss_fwhm > 0 || var_dh > 0) {
00066
00067             /* Allocate... */
00068             ALLOC(wave, wave_t, 1);
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             /* Compute variance... */

```

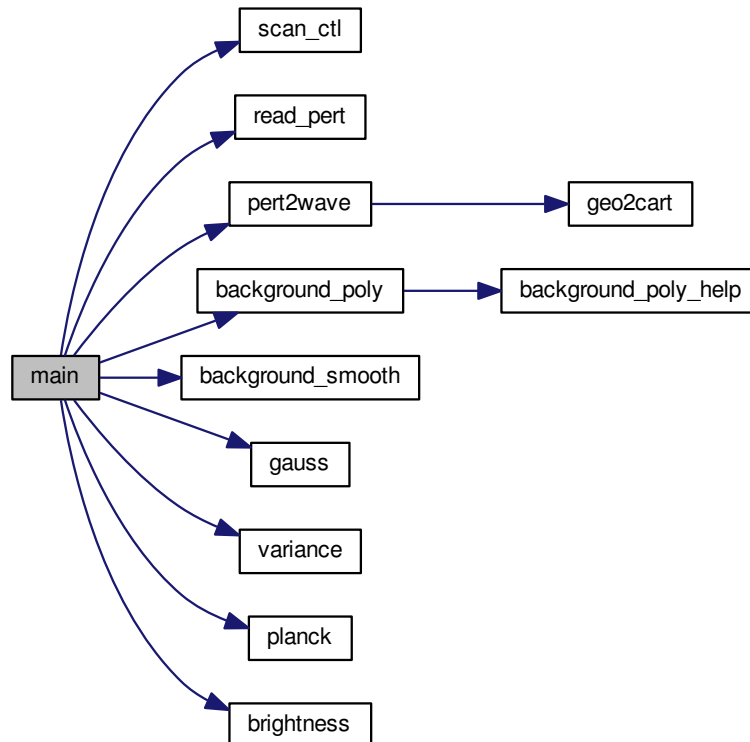


```

00081     variance(wave, var_dh);
00082
00083     /* Copy data... */
00084     for (ix = 0; ix < wave->nx; ix++)
00085         for (iy = 0; iy < wave->ny; iy++) {
00086             pert->pt[ix][iy] = wave->pt[ix][iy];
00087             pert->var[ix][iy] = wave->var[ix][iy];
00088         }
00089
00090     /* Free... */
00091     free(wave);
00092 }
00093
00094 /* Apply noise correction... */
00095 if (dt230 > 0)
00096     for (itrack = 0; itrack < pert->ntrack; itrack++)
00097         for (ixtrack = 0; ixtrack < pert->ntrack; ixtrack++) {
00098             nesr = planck(t230 + dt230, nu) - planck(t230, nu);
00099             tbg = pert->bt[ixtrack][ixtrack] - pert->pt[ixtrack][ixtrack];
00100             nedt = brightness(planck(tbg, nu) + nesr, nu) - tbg;
00101             pert->var[ixtrack][ixtrack] -= gsl_pow_2(nedt);
00102         }
00103
00104 /* Find local maxima... */
00105 for (itrack = 0; itrack < pert->ntrack; itrack += 2 * dtrack)
00106     for (ixtrack = dxtrack / 2; ixtrack < pert->ntrack;
00107          ixtrack += 2 * dxtrack) {
00108
00109         /* Init... */
00110         varmax = 0;
00111         itrackmax = -999;
00112         ixtrackmax = -999;
00113
00114         /* Loop over box... */
00115         for (itrack2 = itrack;
00116              itrack2 < GSL_MIN(itrack + dtrack, pert->ntrack); itrack2++)
00117             for (ixtrack2 = ixtrack;
00118                  ixtrack2 < GSL_MIN(ixtrack + dxtrack, pert->ntrack);
00119                  ixtrack2++)
00120                 if (pert->var[ixtrack2][ixtrack2] >= varmax) {
00121                     varmax = pert->var[ixtrack2][ixtrack2];
00122                     itrackmax = itrack2;
00123                     ixtrackmax = ixtrack2;
00124                 }
00125
00126         /* Report event... */
00127         if (itrackmax >= 0 && ixtrackmax >= 0 && varmax >= varmin)
00128             fprintf(out, "%.2f %g %g %g\n",
00129                     pert->time[ixtrackmax][ixtrackmax],
00130                     pert->lon[ixtrackmax][ixtrackmax],
00131                     pert->lat[ixtrackmax][ixtrackmax],
00132                     pert->var[ixtrackmax][ixtrackmax]);
00133     }
00134 }
00135
00136 /* Close file... */
00137 fclose(out);
00138
00139 /* Free... */
00140 free(pert);
00141
00142 return EXIT_SUCCESS;
00143 }

```

Here is the call graph for this function:



## 5.12 events.c

```

00001 #include "libairs.h"
00002
00003 int main(
00004     int argc,
00005     char *argv[]) {
00006
00007     static pert_t *pert;
00008
00009     static wave_t *wave;
00010
00011     static FILE *in, *out;
00012
00013     static char pertname[LEN];
00014
00015     static double gauss_fwhm, var_dh, varmin, varmax, nu, t230 = 230.0,
00016         dt230, tbg, nesr, nedt = 0;
00017
00018     static int iarg, ix, iy, bg_poly_x, bg_poly_y, bg_smooth_x, bg_smooth_y,
00019         itrack, itrack2, itrackmax, ixtrack, ixtrack2, ixtrackmax, dtrack = 15,
00020         dxtrack = 15;
00021
00022     /* Check arguments... */
00023     if (argc < 4)
00024         ERRMSG("Give parameters: <ctl> <events.tab> <pert1.nc> [<pert2.nc> ...]");
00025
00026     /* Get control parameters... */
00027     scan_ctl(argc, argv, "PERTNAME", -1, "4mu", pertname);
00028     bg_poly_x = (int) scan_ctl(argc, argv, "BG_POLY_X", -1, "0", NULL);
00029     bg_poly_y = (int) scan_ctl(argc, argv, "BG_POLY_Y", -1, "0", NULL);
00030     bg_smooth_x = (int) scan_ctl(argc, argv, "BG_SMOOTH_X", -1, "0", NULL);
00031     bg_smooth_y = (int) scan_ctl(argc, argv, "BG_SMOOTH_Y", -1, "0", NULL);
00032     gauss_fwhm = scan_ctl(argc, argv, "GAUSS_FWHM", -1, "0", NULL);
  
```

```

00033 var_dh = scan_ctl(argc, argv, "VAR_DH", -1, "0", NULL);
00034 varmin = scan_ctl(argc, argv, "VARMIN", -1, "", NULL);
00035 dt230 = scan_ctl(argc, argv, "DT230", -1, "0.16", NULL);
00036 nu = scan_ctl(argc, argv, "NU", -1, "2345.0", NULL);
00037
00038 /* Alloc... */
00039 ALLOC(pert, pert_t, 1);
00040
00041 /* Create file... */
00042 printf("Write event data: %s\n", argv[2]);
00043 if (!(out = fopen(argv[2], "w")))
00044     ERRMSG("Cannot create file!");
00045
00046 /* Write header... */
00047 fprintf(out,
00048         "# $1 = time [s]\n"
00049         "# $2 = longitude [deg]\n"
00050         "# $3 = latitude [deg]\n" "# $4 = maximum variance [K^2]\n\n");
00051
00052 /* Loop over perturbation files... */
00053 for (iarg = 3; iarg < argc; iarg++) {
00054
00055     /* Read perturbation data... */
00056     if (!(in = fopen(argv[iarg], "r")))
00057         continue;
00058     else {
00059         fclose(in);
00060         read_pert(argv[iarg], pertname, pert);
00061     }
00062
00063     /* Recalculate background and perturbations... */
00064     if (bg_poly_x > 0 || bg_poly_y > 0 ||
00065         bg_smooth_x > 0 || bg_smooth_y > 0 || gauss_fwhm > 0 || var_dh > 0) {
00066
00067         /* Allocate... */
00068         ALLOC(wave, wave_t, 1);
00069
00070         /* Convert to wave analysis struct... */
00071         pert2wave(pert, wave, 0, pert->ntrack - 1, 0, pert->ntrack - 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         /* Compute variance... */
00081         variance(wave, var_dh);
00082
00083         /* Copy data... */
00084         for (ix = 0; ix < wave->nx; ix++)
00085             for (iy = 0; iy < wave->ny; iy++) {
00086                 pert->pt[ix][iy] = wave->pt[ix][iy];
00087                 pert->var[ix][iy] = wave->var[ix][iy];
00088             }
00089
00090         /* Free... */
00091         free(wave);
00092     }
00093
00094     /* Apply noise correction... */
00095     if (dt230 > 0)
00096         for (itrack = 0; itrack < pert->ntrack; itrack++)
00097             for (ixtrack = 0; ixtrack < pert->ntrack; ixtrack++) {
00098                 nesr = planck(t230 + dt230, nu) - planck(t230, nu);
00099                 tbgr = pert->bt[ixtrack][ixtrack] - pert->pt[ixtrack][ixtrack];
00100                 nedt = brightness(planck(tbgr, nu) + nesr, nu) - tbgr;
00101                 pert->var[ixtrack][ixtrack] -= gsl_pow_2(nedt);
00102             }
00103
00104     /* Find local maxima... */
00105     for (itrack = 0; itrack < pert->ntrack; itrack += 2 * dtrack)
00106         for (ixtrack = dxtrack / 2; ixtrack < pert->ntrack;
00107             ixtrack += 2 * dxtrack) {
00108
00109             /* Init... */
00110             varmax = 0;
00111             itrackmax = -999;
00112             ixtrackmax = -999;
00113
00114             /* Loop over box... */
00115             for (itrack2 = itrack;
00116                 itrack2 < GSL_MIN(itrack + dtrack, pert->ntrack); itrack2++)
00117                 for (ixtrack2 = ixtrack;
00118                     ixtrack2 < GSL_MIN(ixtrack + dxtrack, pert->ntrack);
00119                     ixtrack2++)

```

```

00120         if (pert->var[itrack2][ixtrack2] >= varmax) {
00121             varmax = pert->var[itrack2][ixtrack2];
00122             itrackmax = itrack2;
00123             ixtrackmax = ixtrack2;
00124         }
00125
00126         /* Report event... */
00127         if (itrackmax >= 0 && ixtrackmax >= 0 && varmax >= varmin)
00128             fprintf(out, "%.2f %g %g %g\n",
00129                 pert->time[itrackmax][ixtrackmax],
00130                 pert->lon[itrackmax][ixtrackmax],
00131                 pert->lat[itrackmax][ixtrackmax],
00132                 pert->var[itrackmax][ixtrackmax]);
00133     }
00134 }
00135
00136 /* Close file... */
00137 fclose(out);
00138
00139 /* Free... */
00140 free(pert);
00141
00142 return EXIT_SUCCESS;
00143 }

```

## 5.13 extract.c File Reference

### Functions

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

### Variables

- int [airs\\_chan](#) [L1\_NCHAN]

#### 5.13.1 Function Documentation

##### 5.13.1.1 double [gph2z](#) ( double *gph* )

Definition at line [140](#) of file [extract.c](#).

```

00141     {
00142
00143         double a = 3.086e-3;
00144
00145         return G0 / a - sqrt(gsl_pow_2(G0 / a) - 2 * G0 * gph / a);
00146     }

```

##### 5.13.1.2 int [main](#) ( int *argc*, char \* *argv*[] )

Definition at line [26](#) of file [extract.c](#).

```

00028         {
00029
00030     static airs_rad_gran_t airs_rad_gran;
00031     static airs_ret_gran_t airs_ret_gran;
00032
00033     static airs_ll_t ll;
00034     static airs_l2_t l2;
00035
00036     int ichan, lay, track, xtrack;
00037
00038     /* Check arguments... */
00039     if (argc != 4)
00040         ERRMSG("Give parameters: <airs_ll_file> <airs_l2_file> <out.nc>");
00041
00042     /* Check Level-1 filename... */
00043     if (argv[1][0] != '-') {
00044
00045         /* Read data... */
00046         printf("Read AIRS Level-1 file: %s\n", argv[1]);
00047         airs_rad_rdr(argv[1], &airs_rad_gran);
00048
00049         /* Flag bad data... */
00050         for (track = 0; track < AIRS_RAD_GEOTRACK; track++)
00051             for (xtrack = 0; xtrack < AIRS_RAD_GEOXTRACK; xtrack++)
00052                 for (ichan = 0; ichan < Ll_NCHAN; ichan++)
00053                     if ((airs_rad_gran.state[track][xtrack] != 0)
00054                         || (airs_rad_gran.ExcludedChans[airs_chan[ichan]] > 2)
00055                         || (airs_rad_gran.CalChanSummary[airs_chan[ichan]] & 8)
00056                         || (airs_rad_gran.CalChanSummary[airs_chan[ichan]] & (32 + 64))
00057                         || (airs_rad_gran.CalFlag[track][airs_chan[ichan]] & 16))
00058                         airs_rad_gran.radiances[track][xtrack][airs_chan[ichan]]
00059                             = GSL_NAN;
00060
00061         /* Copy data to struct... */
00062         for (track = 0; track < AIRS_RAD_GEOTRACK; track++)
00063             for (xtrack = 0; xtrack < AIRS_RAD_GEOXTRACK; xtrack++) {
00064                 ll.time[track][xtrack]
00065                     = airs_rad_gran.Time[track][xtrack] - 220838400.;
00066                 ll.lon[track][xtrack]
00067                     = airs_rad_gran.Longitude[track][xtrack];
00068                 ll.lat[track][xtrack]
00069                     = airs_rad_gran.Latitude[track][xtrack];
00070                 ll.sat_z[track]
00071                     = airs_rad_gran.satheight[track];
00072                 ll.sat_lon[track]
00073                     = airs_rad_gran.sat_lon[track];
00074                 ll.sat_lat[track]
00075                     = airs_rad_gran.sat_lat[track];
00076                 for (ichan = 0; ichan < Ll_NCHAN; ichan++) {
00077                     ll.nu[ichan]
00078                         = airs_rad_gran.nominal_freq[airs_chan[ichan]];
00079                     ll.rad[track][xtrack][ichan]
00080                         = airs_rad_gran.radiances[track][xtrack][airs_chan[ichan]] *
00081                             0.001f;
00082                 }
00083             }
00084
00085         /* Write netCDF file... */
00086         write_ll(argv[3], &ll);
00087     }
00088
00089     /* Check Level-2 filename... */
00090     if (argv[2][0] != '-') {
00091
00092         /* Read data... */
00093         printf("Read AIRS Level-2 file: %s\n", argv[2]);
00094         airs_ret_rdr(argv[2], &airs_ret_gran);
00095
00096         /* Flag bad data... */
00097         for (track = 0; track < AIRS_RET_GEOTRACK; track++)
00098             for (xtrack = 0; xtrack < AIRS_RET_GEOXTRACK; xtrack++)
00099                 for (lay = 1; lay < AIRS_RET_STDPRESSURELAY; lay++)
00100                     if (airs_ret_gran.GP_Height[track][xtrack][lay] <= -9000.
00101                         || airs_ret_gran.TAirStd[track][xtrack][lay] <= -9000.) {
00102                         airs_ret_gran.GP_Height[track][xtrack][lay] = GSL_NAN;
00103                         airs_ret_gran.TAirStd[track][xtrack][lay] = GSL_NAN;
00104                     }
00105
00106         /* Save data in struct... */
00107         for (track = 0; track < AIRS_RET_GEOTRACK; track++)
00108             for (xtrack = 0; xtrack < AIRS_RET_GEOXTRACK; xtrack++)
00109                 for (lay = 1; lay < AIRS_RET_STDPRESSURELAY; lay++) {
00110                     ll2.time[track][xtrack]
00111                         = airs_ret_gran.Time[track][xtrack] - 220838400.;
00112                     ll2.z[track][xtrack][lay - 1]
00113                         = airs_ret_gran.GP_Height[track][xtrack][lay] / 1000.;
00114                     ll2.lon[track][xtrack]

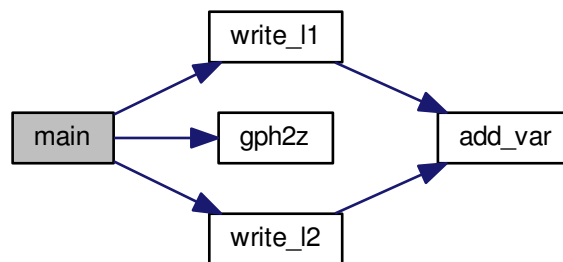
```

```

00115         = airs_ret_gran.Longitude[track][xtrack];
00116     l2.lat[track][xtrack]
00117     = airs_ret_gran.Latitude[track][xtrack];
00118     l2.p[lay - 1]
00119     = airs_ret_gran.pressStd[lay];
00120     l2.t[track][xtrack][lay - 1]
00121     = airs_ret_gran.TAirStd[track][xtrack][lay];
00122 }
00123
00124 /* Convert geopotential heights to geometric heights... */
00125 for (track = 0; track < L2_NTRACK; track++)
00126     for (xtrack = 0; xtrack < L2_NXTRACK; xtrack++)
00127         for (lay = 0; lay < L2_NLAY; lay++)
00128             l2.z[track][xtrack][lay]
00129             = gph2z(l2.z[track][xtrack][lay]);
00130
00131 /* Write netCDF file... */
00132 write_l2(argv[3], &l2);
00133 }
00134
00135 return EXIT_SUCCESS;
00136 }

```

Here is the call graph for this function:



## 5.13.2 Variable Documentation

### 5.13.2.1 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](#).

## 5.14 extract.c

```

00001 #include "libairs.h"
00002
00003 /* -----
00004     Global variables...
00005     ----- */
00006
00007 /* List of AIRS channels (don't change). */
00008 int airs_chan[L1_NCHAN] = { 54, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82,
00009     2035, 2036, 2040, 2041, 2052, 2053, 2054, 2055,
00010     2067, 2075, 2076, 2077, 2078, 2079, 2080, 2081,
00011     2082, 2086, 2088, 2089, 2091, 2092, 2093
00012 };
00013
00014 /* -----
00015     Functions...
00016     ----- */
00017
00018 /* Convert geopotential height to geometric altitude. */
00019 double gph2z(
00020     double gph);
00021
00022 /* -----
00023     Main...
00024     ----- */
00025
00026 int main(
00027     int argc,
00028     char *argv[]) {
00029
00030     static airs_rad_gran_t airs_rad_gran;
00031     static airs_ret_gran_t airs_ret_gran;
00032
00033     static airs_l1_t l1;
00034     static airs_l2_t l2;
00035
00036     int ichan, lay, track, xtrack;
00037
00038     /* Check arguments... */
00039     if (argc != 4)
00040         ERRMSG("Give parameters: <airs_l1_file> <airs_l2_file> <out.nc>");
00041
00042     /* Check Level-1 filename... */
00043     if (argv[1][0] != '-') {
00044
00045         /* Read data... */
00046         printf("Read AIRS Level-1 file: %s\n", argv[1]);
00047         airs_rad_rdr(argv[1], &airs_rad_gran);
00048
00049         /* Flag bad data... */
00050         for (track = 0; track < AIRS_RAD_GEOTRACK; track++)
00051             for (xtrack = 0; xtrack < AIRS_RAD_GEOXTRACK; xtrack++)
00052                 for (ichan = 0; ichan < L1_NCHAN; ichan++)
00053                     if ((airs_rad_gran.state[track][xtrack] != 0)
00054                         || (airs_rad_gran.ExcludedChans[airs_chan[ichan]] > 2)
00055                         || (airs_rad_gran.CalChanSummary[airs_chan[ichan]] & 8)
00056                         || (airs_rad_gran.CalChanSummary[airs_chan[ichan]] & (32 + 64))
00057                         || (airs_rad_gran.CalFlag[track][airs_chan[ichan]] & 16))
00058                         airs_rad_gran.radiances[track][xtrack][airs_chan[ichan]]
00059                             = GSL_NAN;
00060
00061         /* Copy data to struct... */
00062         for (track = 0; track < AIRS_RAD_GEOTRACK; track++)
00063             for (xtrack = 0; xtrack < AIRS_RAD_GEOXTRACK; xtrack++) {
00064                 l1.time[track][xtrack]
00065                     = airs_rad_gran.Time[track][xtrack] - 220838400.;
00066                 l1.lon[track][xtrack]
00067                     = airs_rad_gran.Longitude[track][xtrack];
00068                 l1.lat[track][xtrack]
00069                     = airs_rad_gran.Latitude[track][xtrack];
00070                 l1.sat_z[track]
00071                     = airs_rad_gran.satheight[track];
00072                 l1.sat_lon[track]
00073                     = airs_rad_gran.sat_lon[track];
00074                 l1.sat_lat[track]
00075                     = airs_rad_gran.sat_lat[track];
00076                 for (ichan = 0; ichan < L1_NCHAN; ichan++) {
00077                     l1.nu[ichan]
00078                         = airs_rad_gran.nominal_freq[airs_chan[ichan]];
00079                     l1.rad[track][xtrack][ichan]
00080                         = airs_rad_gran.radiances[track][xtrack][airs_chan[ichan]] *
00081                             0.001f;
00082                 }
00083             }
00084

```

```

00085     /* Write netCDF file... */
00086     write_l1(argv[3], &l1);
00087 }
00088
00089 /* Check Level-2 filename... */
00090 if (argv[2][0] != '-') {
00091
00092     /* Read data... */
00093     printf("Read AIRS Level-2 file: %s\n", argv[2]);
00094     airs_ret_rdr(argv[2], &airs_ret_gran);
00095
00096     /* Flag bad data... */
00097     for (track = 0; track < AIRS_RET_GEOTRACK; track++)
00098         for (xtrack = 0; xtrack < AIRS_RET_GEOXTRACK; xtrack++)
00099             for (lay = 1; lay < AIRS_RET_STDPRESSURELAY; lay++)
00100                 if (airs_ret_gran.GP_Height[track][xtrack][lay] <= -9000.
00101                     || airs_ret_gran.TAirStd[track][xtrack][lay] <= -9000.) {
00102                     airs_ret_gran.GP_Height[track][xtrack][lay] = GSL_NAN;
00103                     airs_ret_gran.TAirStd[track][xtrack][lay] = GSL_NAN;
00104                 }
00105
00106     /* Save data in struct... */
00107     for (track = 0; track < AIRS_RET_GEOTRACK; track++)
00108         for (xtrack = 0; xtrack < AIRS_RET_GEOXTRACK; xtrack++)
00109             for (lay = 1; lay < AIRS_RET_STDPRESSURELAY; lay++) {
00110                 l2.time[track][xtrack]
00111                     = airs_ret_gran.Time[track][xtrack] - 220838400.;
00112                 l2.z[track][xtrack][lay - 1]
00113                     = airs_ret_gran.GP_Height[track][xtrack][lay] / 1000.;
00114                 l2.lon[track][xtrack]
00115                     = airs_ret_gran.Longitude[track][xtrack];
00116                 l2.lat[track][xtrack]
00117                     = airs_ret_gran.Latitude[track][xtrack];
00118                 l2.p[lay - 1]
00119                     = airs_ret_gran.pressStd[lay];
00120                 l2.t[track][xtrack][lay - 1]
00121                     = airs_ret_gran.TAirStd[track][xtrack][lay];
00122             }
00123
00124     /* Convert geopotential heights to geometric heights... */
00125     for (track = 0; track < L2_NTRACK; track++)
00126         for (xtrack = 0; xtrack < L2_NXTRACK; xtrack++)
00127             for (lay = 0; lay < L2_NLAY; lay++)
00128                 l2.z[track][xtrack][lay]
00129                     = gph2z(l2.z[track][xtrack][lay]);
00130
00131     /* Write netCDF file... */
00132     write_l2(argv[3], &l2);
00133 }
00134
00135 return EXIT_SUCCESS;
00136 }
00137
00138 /*****
00139
00140 double gph2z(
00141     double gph) {
00142
00143     double a = 3.086e-3;
00144
00145     return G0 / a - sqrt(gsl_pow_2(G0 / a) - 2 * G0 * gph / a);
00146 }

```

## 5.15 hurricane.c File Reference

### Functions

- int [get\\_storm\\_pos](#) (int nob, 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 nob, double x[NSTORM][NTIME])
- int [main](#) (int argc, char \*argv[])



### 5.15.1 Function Documentation

**5.15.1.1** `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 )`

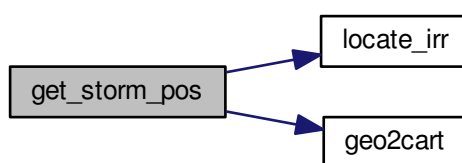
Definition at line 341 of file [hurricane.c](#).

```

00355         {
00356
00357     double w, x0[3], x1[3];
00358
00359     int i;
00360
00361     /* Check time range... */
00362     if (t < time_wmo[0] || t > time_wmo[nobs - 1])
00363         return 0;
00364
00365     /* Interpolate position... */
00366     i = locate_irr(time_wmo, nobs, t);
00367     w = (t - time_wmo[i]) / (time_wmo[i + 1] - time_wmo[i]);
00368     geo2cart(0, lon_wmo[i], lat_wmo[i], x0);
00369     geo2cart(0, lon_wmo[i + 1], lat_wmo[i + 1], x1);
00370     x[0] = (1 - w) * x0[0] + w * x1[0];
00371     x[1] = (1 - w) * x0[1] + w * x1[1];
00372     x[2] = (1 - w) * x0[2] + w * x1[2];
00373
00374     /* Interpolate wind and pressure... */
00375     *pres = (1 - w) * pres_wmo[i] + w * pres_wmo[i + 1];
00376     *wind = (1 - w) * wind_wmo[i] + w * wind_wmo[i + 1];
00377
00378     /* Get pressure and wind change... */
00379     *dpres = (pres_wmo[i + 1 + st] - pres_wmo[GSL_MAX(i - dt + st, 0)])
00380         / (time_wmo[i + 1 + st] - time_wmo[GSL_MAX(i - dt + st, 0)]) * 3600.;
00381     *dwind = (wind_wmo[i + 1 + st] - wind_wmo[GSL_MAX(i - dt + st, 0)])
00382         / (time_wmo[i + 1 + st] - time_wmo[GSL_MAX(i - dt + st, 0)]) * 3600.;
00383
00384     return 1;
00385 }

```

Here is the call graph for this function:



**5.15.1.2** `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](#).

```

00394         {
00395
00396     int varid;
00397
00398     size_t count[2], istorm, start[2];
00399
00400     /* Read pressure... */

```

```

00401     NC(nc_inq_varid(ncid, varname, &varid));
00402     for (istorm = 0; istorm < nstorm; istorm++) {
00403         start[0] = istorm;
00404         start[1] = 0;
00405         count[0] = 1;
00406         count[1] = (size_t) nobs[istorm];
00407         NC(nc_get_vara_double(ncid, varid, start, count, x[istorm]));
00408     }
00409 }

```

### 5.15.1.3 int main ( int argc, char \* argv[] )

Definition at line 46 of file [hurricane.c](#).

```

00048     {
00049
00050     static pert_t *pert;
00051
00052     static FILE *in, *out;
00053
00054     static char filter[LEN], pertname[LEN], set[LEN];
00055
00056     static double bt4_mean, bt4_var, bt8_min, dpres, dpresbest, dt230, dwind,
00057         dwindbest, lat_wmo[NSTORM][NTIME], latbest, lon_wmo[NSTORM][NTIME],
00058         lonbest, lonsat, lonstorm, nedt, nesr, nu, pmin, pres_wmo[NSTORM][NTIME],
00059         pres, presbest, r2, r2best = 1e100, rmax, wind_wmo[NSTORM][NTIME], wind,
00060         windbest, wmax, time_max_pres[NSTORM], time_max_wind[NSTORM],
00061         time_wmo[NSTORM][NTIME], timebest, xf[PERT_NTRACK][PERT_NXTRACK][3],
00062         xs[3], z;
00063
00064     static int asc, dimid, dt, iarg, iobs, itrack, itrack2, ixtrack2, n,
00065         ncid, nobs[NSTORM], st, varid;
00066
00067     static size_t istorm, nstorm, ntime;
00068
00069     /* Check arguments... */
00070     if (argc < 5)
00071         ERRMSG("Give parameters: <ctl> <hurr.tab> <ibtracs.nc>"
00072             " <pert1.nc> [<pert2.nc> ...]");
00073
00074     /* Get control parameters... */
00075     scan_ctl(argc, argv, "SET", -1, "full", set);
00076     scan_ctl(argc, argv, "PERTNAME", -1, "4mu", pertname);
00077     scan_ctl(argc, argv, "FILTER", -1, "both", filter);
00078     dt230 = scan_ctl(argc, argv, "DT230", -1, "0.16", NULL);
00079     nu = scan_ctl(argc, argv, "NU", -1, "2345.0", NULL);
00080     rmax = scan_ctl(argc, argv, "RMAX", -1, "500", NULL);
00081     dt = (int) scan_ctl(argc, argv, "DT", -1, "0", NULL);
00082     st = (int) scan_ctl(argc, argv, "ST", -1, "0", NULL);
00083
00084     /* Allocate... */
00085     ALLOC(pert, pert_t, 1);
00086
00087     /* -----
00088        Read hurricane tracks...
00089        ----- */
00090
00091     /* Write info... */
00092     printf("Read hurricane tracks: %s\n", argv[3]);
00093
00094     /* Open netCDF file... */
00095     NC(nc_open(argv[3], NC_NOWRITE, &ncid));
00096
00097     /* Get dimensions... */
00098     NC(nc_inq_dimid(ncid, "storm", &dimid));
00099     NC(nc_inq_dimlen(ncid, dimid, &nstorm));
00100     NC(nc_inq_dimid(ncid, "time", &dimid));
00101     NC(nc_inq_dimlen(ncid, dimid, &ntime));
00102     if (nstorm > NSTORM)
00103         ERRMSG("Too many storms!");
00104     if (ntime > NTIME)
00105         ERRMSG("Too many time steps!");
00106
00107     /* Read number of observations per storm... */
00108     NC(nc_inq_varid(ncid, "numObs", &varid));
00109     NC(nc_get_var_int(ncid, varid, nobs));
00110
00111     /* Read data... */
00112     read_var(ncid, "lat_wmo", nstorm, nobs, lat_wmo);
00113     read_var(ncid, "lon_wmo", nstorm, nobs, lon_wmo);
00114     read_var(ncid, "time_wmo", nstorm, nobs, time_wmo);

```

```

00115 read_var(ncid, "wind_wmo", nstorm, nob, wind_wmo);
00116 read_var(ncid, "pres_wmo", nstorm, nob, pres_wmo);
00117
00118 /* Convert units.. */
00119 for (istorm = 0; istorm < nstorm; istorm++)
00120     for (iobs = 0; iobs < nob[istorm]; iobs++) {
00121         time_wmo[istorm][iobs] *= 86400.;
00122         time_wmo[istorm][iobs] -= 4453401600.00;
00123         lon_wmo[istorm][iobs] *= 0.01;
00124         lat_wmo[istorm][iobs] *= 0.01;
00125         wind_wmo[istorm][iobs] *= 0.0514444;
00126         pres_wmo[istorm][iobs] *= 0.1;
00127     }
00128
00129 /* Check data... */
00130 for (istorm = 0; istorm < nstorm; istorm++)
00131     for (iobs = 0; iobs < nob[istorm]; iobs++) {
00132         if (pres_wmo[istorm][iobs] <= 800 || pres_wmo[istorm][iobs] >= 1200)
00133             pres_wmo[istorm][iobs] = GSL_NAN;
00134         if (wind_wmo[istorm][iobs] <= 0.1)
00135             wind_wmo[istorm][iobs] = GSL_NAN;
00136     }
00137
00138 /* Find time of maximum intensity (lowest pressure)... */
00139 for (istorm = 0; istorm < nstorm; istorm++) {
00140     pmin = 1e100;
00141     time_max_pres[istorm] = GSL_NAN;
00142     for (iobs = 0; iobs < nob[istorm]; iobs++)
00143         if (gsl_finite(pres_wmo[istorm][iobs]) && pres_wmo[istorm][iobs] < pmin) {
00144             pmin = pres_wmo[istorm][iobs];
00145             time_max_pres[istorm] = time_wmo[istorm][iobs];
00146         }
00147 }
00148
00149 /* Find time of maximum intensity (maximum wind)... */
00150 for (istorm = 0; istorm < nstorm; istorm++) {
00151     wmax = -1e100;
00152     time_max_wind[istorm] = GSL_NAN;
00153     for (iobs = 0; iobs < nob[istorm]; iobs++)
00154         if (gsl_finite(wind_wmo[istorm][iobs]) && wind_wmo[istorm][iobs] > wmax) {
00155             wmax = wind_wmo[istorm][iobs];
00156             time_max_wind[istorm] = time_wmo[istorm][iobs];
00157         }
00158 }
00159
00160 /* Close netCDF file... */
00161 nc_close(ncid);
00162
00163 /* -----
00164 Analyze AIRS data...
00165 ----- */
00166
00167 /* Create file... */
00168 printf("Write hurricane data: %s\n", argv[2]);
00169 if (!(out = fopen(argv[2], "w")))
00170     ERRMSG("Cannot create file!");
00171
00172 /* Write header... */
00173 fprintf(out,
00174     "# $1 = storm number\n"
00175     "# $2 = storm time since first report [hr]\n"
00176     "# $3 = storm time since wind maximum [hr]\n"
00177     "# $4 = storm time since pressure minimum [hr]\n"
00178     "# $5 = match time [s]\n"
00179     "# $6 = match longitude [deg]\n"
00180     "# $7 = match latitude [deg]\n"
00181     "# $8 = match distance [km]\n"
00182     "# $9 = wind speed [m/s]\n"
00183     "# $10 = wind speed change [m/s/hr]\n");
00184 fprintf(out,
00185     "# $11 = pressure [hPa]\n"
00186     "# $12 = pressure change [hPa/hr]\n"
00187     "# $13 = 8.1 micron BT minimum [K]\n"
00188     "# $14 = 4.3 micron BT variance [K^2]\n"
00189     "# $15 = 4.3 micron BT variance (noise-corrected) [K^2]\n"
00190     "# $16 = number of footprints\n");
00191
00192 /* Loop over perturbation files... */
00193 for (iarg = 4; iarg < argc; iarg++) {
00194
00195     /* Read perturbation data... */
00196     if (!(in = fopen(argv[iarg], "r")))
00197         continue;
00198     else {
00199         fclose(in);
00200         read_pert(argv[iarg], pertname, pert);
00201     }

```

```

00202
00203 /* Get Cartesian coordinates... */
00204 for (itrack2 = 0; itrack2 < pert->ntrack; itrack2++)
00205     for (ixtrack2 = 0; ixtrack2 < pert->nxtrack; ixtrack2++)
00206         geo2cart(0, pert->lon[itrack2][ixtrack2],
00207                 pert->lat[itrack2][ixtrack2], xf[itrack2][ixtrack2]);
00208
00209 /* Loop over storms... */
00210 for (istorm = 0; istorm < nstorm; istorm++) {
00211
00212     /* Loop along AIRS center track... */
00213     for (itrack = 0; itrack < pert->ntrack; itrack++) {
00214
00215         /* Get storm position... */
00216         if (get_storm_pos(nobs[istorm], time_wmo[istorm], lon_wmo[istorm],
00217                         lat_wmo[istorm], wind_wmo[istorm], pres_wmo[istorm],
00218                         pert->time[itrack][pert->nxtrack / 2], dt, st, xs,
00219                         &wind, &dwind, &pres, &dpres)) {
00220
00221             /* Get distance... */
00222             r2 = DIST2(xs, xf[itrack][pert->nxtrack / 2]);
00223
00224             /* Find best match... */
00225             if (r2 < r2best) {
00226
00227                 /* Save position... */
00228                 r2best = r2;
00229                 timebest = pert->time[itrack][pert->nxtrack / 2];
00230                 cart2geo(xs, &z, &lonbest, &latbest);
00231
00232                 /* Save wind... */
00233                 windbest = wind;
00234                 dwindbest = dwind;
00235                 presbest = pres;
00236                 dpresbest = dpres;
00237
00238                 /* Get BT data... */
00239                 n = 0;
00240                 bt8_min = 1e100;
00241                 bt4_mean = 0;
00242                 bt4_var = 0;
00243                 for (itrack2 = GSL_MAX(itrack - ((int) (rmax / 17) + 1), 0);
00244                     itrack2 <= GSL_MIN(itrack + ((int) (rmax / 17) + 1),
00245                                         pert->ntrack - 1); itrack2++)
00246                     for (ixtrack2 = 0; ixtrack2 < pert->nxtrack; ixtrack2++) {
00247
00248                         /* Check data... */
00249                         if (pert->time[itrack2][ixtrack2] < 0
00250                             || pert->lon[itrack2][ixtrack2] < -180
00251                             || pert->lon[itrack2][ixtrack2] > 180
00252                             || pert->lat[itrack2][ixtrack2] < -90
00253                             || pert->lat[itrack2][ixtrack2] > 90
00254                             || pert->pt[itrack2][ixtrack2] < -100
00255                             || pert->pt[itrack2][ixtrack2] > 100
00256                             || !gsl_finite(pert->bt[itrack2][ixtrack2])
00257                             || !gsl_finite(pert->pt[itrack2][ixtrack2])
00258                             || !gsl_finite(pert->var[itrack2][ixtrack2])
00259                             || !gsl_finite(pert->dc[itrack2][ixtrack2]))
00260                             continue;
00261
00262                         /* Check east/west filter... */
00263                         lonsat = pert->lon[itrack2][ixtrack2];
00264                         while (lonsat < 20)
00265                             lonsat += 360;
00266                         lonstorm = lonbest;
00267                         while (lonstorm < 20)
00268                             lonstorm += 360;
00269                         if ((filter[0] == 'e' || filter[0] == 'E')
00270                             && lonsat < lonstorm)
00271                             continue;
00272                         if ((filter[0] == 'w' || filter[0] == 'W')
00273                             && lonsat > lonstorm)
00274                             continue;
00275
00276                         /* Get distance... */
00277                         if (DIST2(xs, xf[itrack2][ixtrack2]) < rmax * rmax) {
00278                             bt8_min = GSL_MIN(bt8_min, pert->dc[itrack2][ixtrack2]);
00279                             bt4_mean += pert->bt[itrack2][ixtrack2];
00280                             bt4_var += gsl_pow_2(pert->pt[itrack2][ixtrack2]);
00281                             n++;
00282                         }
00283                     }
00284             }
00285         }
00286     }
00287
00288     /* Output over poles... */
00289     if (fabs(pert->lat[itrack][pert->nxtrack / 2]) > 80.) {

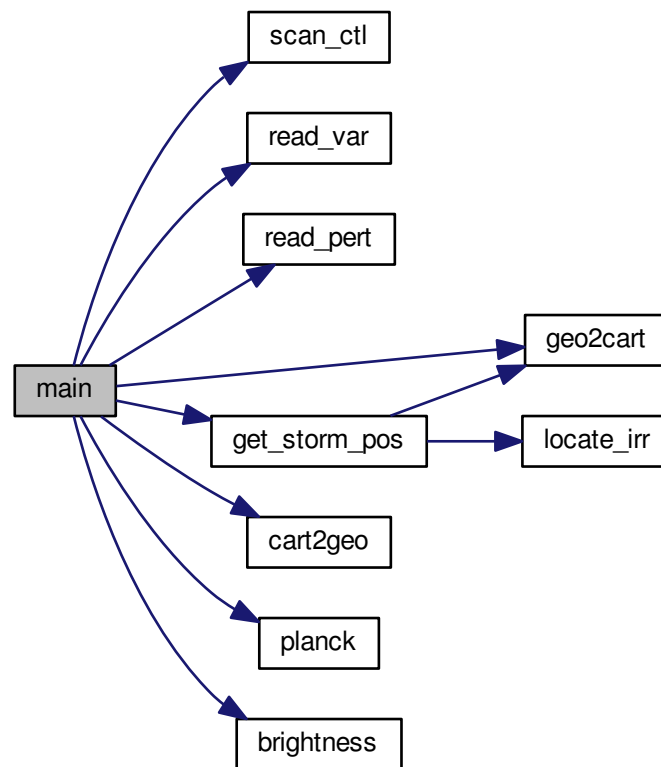
```

```

00289
00290     /* Get and check ascending/descending flag... */
00291     asc =
00292     (pert->lat[itrack > 0 ? itrack : itrack + 1][pert->nxtrack / 2]
00293      > pert->lat[itrack >
00294       0 ? itrack - 1 : itrack][pert->nxtrack / 2]);
00295     if ((set[0] == 'f' || set[0] == 'F')
00296         || ((set[0] == 'a' || set[0] == 'A') && asc)
00297         || ((set[0] == 'd' || set[0] == 'D') && !asc)) {
00298
00299         /* Check for match... */
00300         if (r2best < 890. * 890.) {
00301
00302             /* Estimate noise... */
00303             if (dt230 > 0) {
00304                 nesr = planck(230.0 + dt230, nu) - planck(230.0, nu);
00305                 nedt =
00306                     brightness(planck(bt4_mean / n, nu) + nesr,
00307                                nu) - bt4_mean / n;
00308             }
00309
00310             /* Write output... */
00311             if (n > 0)
00312                 fprintf(out,
00313                         "%lu %g %g %g %.2f %g %g %g %g %g %g %g %g %d\n",
00314                         istorm, (timebest - time_wmo[istorm][0]) / 3600.,
00315                         (timebest - time_max_wind[istorm]) / 3600.,
00316                         (timebest - time_max_pres[istorm]) / 3600.,
00317                         timebest, lonbest, latbest, sqrt(r2best), windbest,
00318                         dwindbest, presbest, dpresbest, bt8_min, bt4_var / n,
00319                         bt4_var / n - gsl_pow_2(nedt), n);
00320             }
00321         }
00322
00323         /* Reset... */
00324         r2best = 1e100;
00325     }
00326 }
00327 }
00328 }
00329
00330 /* Close file... */
00331 fclose(out);
00332
00333 /* Free... */
00334 free(pert);
00335
00336 return EXIT_SUCCESS;
00337 }

```

Here is the call graph for this function:



## 5.16 hurricane.c

```

00001 #include "libairs.h"
00002
00003 /* -----
00004     Dimensions...
00005     ----- */
00006
00007 /* Maximum number of storms. */
00008 #define NSTORM 9000
00009
00010 /* Maximum number of observation times. */
00011 #define NTIME 140
00012
00013 /* -----
00014     Functions...
00015     ----- */
00016
00017 /* Get storm position at given time... */
00018 int get_storm_pos(
00019     int nobs,
00020     double time_wmo[NTIME],
00021     double lon_wmo[NTIME],
00022     double lat_wmo[NTIME],
00023     double wind_wmo[NTIME],
00024     double pres_wmo[NTIME],
00025     double t,
00026     int dt,
00027     int st,
00028     double x[3],
00029     double *wind,
00030     double *dwind,

```

```

00031 double *pres,
00032 double *dpres);
00033
00034 /* Read variable from netCDF file... */
00035 void read_var(
00036     int ncid,
00037     const char varname[],
00038     size_t nstorm,
00039     int nobs[NSTORM],
00040     double x[NSTORM][NTIME]);
00041
00042 /* -----
00043     Main...
00044     ----- */
00045
00046 int main(
00047     int argc,
00048     char *argv[]) {
00049
00050     static pert_t *pert;
00051
00052     static FILE *in, *out;
00053
00054     static char filter[LEN], pertname[LEN], set[LEN];
00055
00056     static double bt4_mean, bt4_var, bt8_min, dpres, dpresbest, dt230, dwind,
00057         dwinbest, lat_wmo[NSTORM][NTIME], latbest, lon_wmo[NSTORM][NTIME],
00058         lonbest, lonsat, lonstorm, nedt, nesr, nu, pmin, pres_wmo[NSTORM][NTIME],
00059         pres, presbest, r2, r2best = 1e100, rmax, wind_wmo[NSTORM][NTIME], wind,
00060         winbest, wmax, time_max_pres[NSTORM], time_max_wind[NSTORM],
00061         time_wmo[NSTORM][NTIME], timebest, xf[PERT_NTRACK][PERT_NXTRACK][3],
00062         xs[3], z;
00063
00064     static int asc, dimid, dt, iarg, iobs, itrack, itrack2, ixtrack2, n,
00065         ncid, nobs[NSTORM], st, varid;
00066
00067     static size_t istorm, nstorm, ntime;
00068
00069     /* Check arguments... */
00070     if (argc < 5)
00071         ERRMSG("Give parameters: <ctl> <hurr.tab> <ibtracs.nc>"
00072             " <pert1.nc> [<pert2.nc> ...]");
00073
00074     /* Get control parameters... */
00075     scan_ctl(argc, argv, "SET", -1, "full", set);
00076     scan_ctl(argc, argv, "PERTNAME", -1, "4mu", pertname);
00077     scan_ctl(argc, argv, "FILTER", -1, "both", filter);
00078     dt230 = scan_ctl(argc, argv, "DT230", -1, "0.16", NULL);
00079     nu = scan_ctl(argc, argv, "NU", -1, "2345.0", NULL);
00080     rmax = scan_ctl(argc, argv, "RMAX", -1, "500", NULL);
00081     dt = (int) scan_ctl(argc, argv, "DT", -1, "0", NULL);
00082     st = (int) scan_ctl(argc, argv, "ST", -1, "0", NULL);
00083
00084     /* Allocate... */
00085     ALLOC(pert, pert_t, 1);
00086
00087     /* -----
00088         Read hurricane tracks...
00089         ----- */
00090
00091     /* Write info... */
00092     printf("Read hurricane tracks: %s\n", argv[3]);
00093
00094     /* Open netCDF file... */
00095     NC(nc_open(argv[3], NC_NOWRITE, &ncid));
00096
00097     /* Get dimensions... */
00098     NC(nc_inq_dimid(ncid, "storm", &dimid));
00099     NC(nc_inq_dimlen(ncid, dimid, &nstorm));
00100     NC(nc_inq_dimid(ncid, "time", &dimid));
00101     NC(nc_inq_dimlen(ncid, dimid, &ntime));
00102     if (nstorm > NSTORM)
00103         ERRMSG("Too many storms!");
00104     if (ntime > NTIME)
00105         ERRMSG("Too many time steps!");
00106
00107     /* Read number of observations per storm... */
00108     NC(nc_inq_varid(ncid, "numObs", &varid));
00109     NC(nc_get_var_int(ncid, varid, nobs));
00110
00111     /* Read data... */
00112     read_var(ncid, "lat_wmo", nstorm, nobs, lat_wmo);
00113     read_var(ncid, "lon_wmo", nstorm, nobs, lon_wmo);
00114     read_var(ncid, "time_wmo", nstorm, nobs, time_wmo);
00115     read_var(ncid, "wind_wmo", nstorm, nobs, wind_wmo);
00116     read_var(ncid, "pres_wmo", nstorm, nobs, pres_wmo);
00117

```

```

00118  /* Convert units.. */
00119  for (istorm = 0; istorm < nstorm; istorm++)
00120      for (iobs = 0; iobs < nob[istorm]; iobs++) {
00121          time_wmo[istorm][iobs] *= 86400.;
00122          time_wmo[istorm][iobs] -= 4453401600.00;
00123          lon_wmo[istorm][iobs] *= 0.01;
00124          lat_wmo[istorm][iobs] *= 0.01;
00125          wind_wmo[istorm][iobs] *= 0.0514444;
00126          pres_wmo[istorm][iobs] *= 0.1;
00127      }
00128
00129  /* Check data... */
00130  for (istorm = 0; istorm < nstorm; istorm++)
00131      for (iobs = 0; iobs < nob[istorm]; iobs++) {
00132          if (pres_wmo[istorm][iobs] <= 800 || pres_wmo[istorm][iobs] >= 1200)
00133              pres_wmo[istorm][iobs] = GSL_NAN;
00134          if (wind_wmo[istorm][iobs] <= 0.1)
00135              wind_wmo[istorm][iobs] = GSL_NAN;
00136      }
00137
00138  /* Find time of maximum intensity (lowest pressure)... */
00139  for (istorm = 0; istorm < nstorm; istorm++) {
00140      pmin = 1e100;
00141      time_max_pres[istorm] = GSL_NAN;
00142      for (iobs = 0; iobs < nob[istorm]; iobs++)
00143          if (gsl_finite(pres_wmo[istorm][iobs]) && pres_wmo[istorm][iobs] < pmin) {
00144              pmin = pres_wmo[istorm][iobs];
00145              time_max_pres[istorm] = time_wmo[istorm][iobs];
00146          }
00147      }
00148
00149  /* Find time of maximum intensity (maximum wind)... */
00150  for (istorm = 0; istorm < nstorm; istorm++) {
00151      wmax = -1e100;
00152      time_max_wind[istorm] = GSL_NAN;
00153      for (iobs = 0; iobs < nob[istorm]; iobs++)
00154          if (gsl_finite(wind_wmo[istorm][iobs]) && wind_wmo[istorm][iobs] > wmax) {
00155              wmax = wind_wmo[istorm][iobs];
00156              time_max_wind[istorm] = time_wmo[istorm][iobs];
00157          }
00158      }
00159
00160  /* Close netCDF file... */
00161  NC(nc_close(ncid));
00162
00163  /* -----
00164  Analyze AIRS data...
00165  ----- */
00166
00167  /* Create file... */
00168  printf("Write hurricane data: %s\n", argv[2]);
00169  if (!(out = fopen(argv[2], "w")))
00170      ERRMSG("Cannot create file!");
00171
00172  /* Write header... */
00173  fprintf(out,
00174      "# $1 = storm number\n"
00175      "# $2 = storm time since first report [hr]\n"
00176      "# $3 = storm time since wind maximum [hr]\n"
00177      "# $4 = storm time since pressure minimum [hr]\n"
00178      "# $5 = match time [s]\n"
00179      "# $6 = match longitude [deg]\n"
00180      "# $7 = match latitude [deg]\n"
00181      "# $8 = match distance [km]\n"
00182      "# $9 = wind speed [m/s]\n"
00183      "# $10 = wind speed change [m/s/hr]\n");
00184  fprintf(out,
00185      "# $11 = pressure [hPa]\n"
00186      "# $12 = pressure change [hPa/hr]\n"
00187      "# $13 = 8.1 micron BT minimum [K]\n"
00188      "# $14 = 4.3 micron BT variance [K^2]\n"
00189      "# $15 = 4.3 micron BT variance (noise-corrected) [K^2]\n"
00190      "# $16 = number of footprints\n");
00191
00192  /* Loop over perturbation files... */
00193  for (iarg = 4; iarg < argc; iarg++) {
00194
00195      /* Read perturbation data... */
00196      if (!(in = fopen(argv[iarg], "r")))
00197          continue;
00198      else {
00199          fclose(in);
00200          read_pert(argv[iarg], pertname, pert);
00201      }
00202
00203      /* Get Cartesian coordinates... */
00204      for (itrack2 = 0; itrack2 < pert->nttrack; itrack2++)

```



```

00205     for (ixtrack2 = 0; ixtrack2 < pert->nxtrack; ixtrack2++)
00206         geo2cart(0, pert->lon[ixtrack2][ixtrack2],
00207                 pert->lat[ixtrack2][ixtrack2], xf[ixtrack2][ixtrack2]);
00208
00209     /* Loop over storms... */
00210     for (istorm = 0; istorm < nstorm; istorm++) {
00211
00212         /* Loop along AIRS center track... */
00213         for (itrack = 0; itrack < pert->ntrack; itrack++) {
00214
00215             /* Get storm position... */
00216             if (get_storm_pos(nobs[istorm], time_wmo[istorm], lon_wmo[istorm],
00217                             lat_wmo[istorm], wind_wmo[istorm], pres_wmo[istorm],
00218                             pert->time[itrack][pert->nxtrack / 2], dt, st, xs,
00219                             &wind, &dwind, &pres, &dpres)) {
00220
00221                 /* Get distance... */
00222                 r2 = DIST2(xs, xf[itrack][pert->nxtrack / 2]);
00223
00224                 /* Find best match... */
00225                 if (r2 < r2best) {
00226
00227                     /* Save position... */
00228                     r2best = r2;
00229                     timebest = pert->time[itrack][pert->nxtrack / 2];
00230                     cart2geo(xs, &z, &lonbest, &latbest);
00231
00232                     /* Save wind... */
00233                     windbest = wind;
00234                     dwindbest = dwind;
00235                     presbest = pres;
00236                     dpresbest = dpres;
00237
00238                     /* Get BT data... */
00239                     n = 0;
00240                     bt8_min = 1e100;
00241                     bt4_mean = 0;
00242                     bt4_var = 0;
00243                     for (itrack2 = GSL_MAX(itrack - ((int) (rmax / 17) + 1), 0);
00244                         itrack2 <= GSL_MIN(itrack + ((int) (rmax / 17) + 1),
00245                                             pert->ntrack - 1); itrack2++)
00246                         for (ixtrack2 = 0; ixtrack2 < pert->nxtrack; ixtrack2++) {
00247
00248                             /* Check data... */
00249                             if (pert->time[ixtrack2][ixtrack2] < 0
00250                                 || pert->lon[ixtrack2][ixtrack2] < -180
00251                                 || pert->lon[ixtrack2][ixtrack2] > 180
00252                                 || pert->lat[ixtrack2][ixtrack2] < -90
00253                                 || pert->lat[ixtrack2][ixtrack2] > 90
00254                                 || pert->pt[ixtrack2][ixtrack2] < -100
00255                                 || pert->pt[ixtrack2][ixtrack2] > 100
00256                                 || !gsl_finite(pert->bt[ixtrack2][ixtrack2])
00257                                 || !gsl_finite(pert->pt[ixtrack2][ixtrack2])
00258                                 || !gsl_finite(pert->var[ixtrack2][ixtrack2])
00259                                 || !gsl_finite(pert->dc[ixtrack2][ixtrack2]))
00260                                 continue;
00261
00262                             /* Check east/west filter... */
00263                             lonsat = pert->lon[ixtrack2][ixtrack2];
00264                             while (lonsat < 20)
00265                                 lonsat += 360;
00266                             lonstorm = lonbest;
00267                             while (lonstorm < 20)
00268                                 lonstorm += 360;
00269                             if ((filter[0] == 'e' || filter[0] == 'E')
00270                                 && lonsat < lonstorm)
00271                                 continue;
00272                             if ((filter[0] == 'w' || filter[0] == 'W')
00273                                 && lonsat > lonstorm)
00274                                 continue;
00275
00276                             /* Get distance... */
00277                             if (DIST2(xs, xf[ixtrack2][ixtrack2]) < rmax * rmax) {
00278                                 bt8_min = GSL_MIN(bt8_min, pert->dc[ixtrack2][ixtrack2]);
00279                                 bt4_mean += pert->bt[ixtrack2][ixtrack2];
00280                                 bt4_var += gsl_pow_2(pert->pt[ixtrack2][ixtrack2]);
00281                                 n++;
00282                             }
00283                         }
00284                     }
00285                 }
00286
00287             /* Output over poles... */
00288             if (fabs(pert->lat[itrack][pert->nxtrack / 2]) > 80.) {
00289
00290                 /* Get and check ascending/descending flag... */
00291                 asc =

```

```

00292         (pert->lat[itrack > 0 ? itrack : itrack + 1][pert->nxtrack / 2]
00293         > pert->lat[itrack >
00294         0 ? itrack - 1 : itrack][pert->nxtrack / 2]);
00295     if ((set[0] == 'f' || set[0] == 'F')
00296         || ((set[0] == 'a' || set[0] == 'A') && asc)
00297         || ((set[0] == 'd' || set[0] == 'D') && !asc)) {
00298
00299         /* Check for match... */
00300         if (r2best < 890. * 890.) {
00301
00302             /* Estimate noise... */
00303             if (dt230 > 0) {
00304                 nesr = planck(230.0 + dt230, nu) - planck(230.0, nu);
00305                 nedt =
00306                     brightness(planck(bt4_mean / n, nu) + nesr,
00307                                nu) - bt4_mean / n;
00308             }
00309
00310             /* Write output... */
00311             if (n > 0)
00312                 fprintf(out,
00313                     "%lu %g %g %g %.2f %g %g %g %g %g %g %g %g %g %d\n",
00314                     istorm, (timebest - time_wmo[istorm][0]) / 3600.,
00315                     (timebest - time_max_wind[istorm]) / 3600.,
00316                     (timebest - time_max_pres[istorm]) / 3600.,
00317                     timebest, lonbest, latbest, sqrt(r2best), windbest,
00318                     dwindbest, presbest, dpresbest, bt8_min, bt4_var / n,
00319                     bt4_var / n - gsl_pow_2(nedt), n);
00320         }
00321     }
00322
00323     /* Reset... */
00324     r2best = 1e100;
00325 }
00326 }
00327 }
00328 }
00329
00330 /* Close file... */
00331 fclose(out);
00332
00333 /* Free... */
00334 free(pert);
00335
00336 return EXIT_SUCCESS;
00337 }
00338
00339 /*****
00340
00341 int get_storm_pos(
00342     int nobs,
00343     double time_wmo[NTIME],
00344     double lon_wmo[NTIME],
00345     double lat_wmo[NTIME],
00346     double wind_wmo[NTIME],
00347     double pres_wmo[NTIME],
00348     double t,
00349     int dt,
00350     int st,
00351     double x[3],
00352     double *wind,
00353     double *dwind,
00354     double *pres,
00355     double *dpres) {
00356
00357     double w, x0[3], x1[3];
00358
00359     int i;
00360
00361     /* Check time range... */
00362     if (t < time_wmo[0] || t > time_wmo[nobs - 1])
00363         return 0;
00364
00365     /* Interpolate position... */
00366     i = locate_irr(time_wmo, nobs, t);
00367     w = (t - time_wmo[i]) / (time_wmo[i + 1] - time_wmo[i]);
00368     geo2cart(0, lon_wmo[i], lat_wmo[i], x0);
00369     geo2cart(0, lon_wmo[i + 1], lat_wmo[i + 1], x1);
00370     x[0] = (1 - w) * x0[0] + w * x1[0];
00371     x[1] = (1 - w) * x0[1] + w * x1[1];
00372     x[2] = (1 - w) * x0[2] + w * x1[2];
00373
00374     /* Interpolate wind and pressure... */
00375     *pres = (1 - w) * pres_wmo[i] + w * pres_wmo[i + 1];
00376     *wind = (1 - w) * wind_wmo[i] + w * wind_wmo[i + 1];
00377
00378     /* Get pressure and wind change... */

```

```

00379  *dpres = (pres_wmo[i + 1 + st] - pres_wmo[GSL_MAX(i - dt + st, 0)])
00380  / (time_wmo[i + 1 + st] - time_wmo[GSL_MAX(i - dt + st, 0)]) * 3600.;
00381  *dwind = (wind_wmo[i + 1 + st] - wind_wmo[GSL_MAX(i - dt + st, 0)])
00382  / (time_wmo[i + 1 + st] - time_wmo[GSL_MAX(i - dt + st, 0)]) * 3600.;
00383
00384  return 1;
00385 }
00386
00387 /*****
00388
00389 void read_var(
00390     int ncid,
00391     const char varname[],
00392     size_t nstorm,
00393     int nobs[NSTORM],
00394     double x[NSTORM][NTIME]) {
00395
00396     int varid;
00397
00398     size_t count[2], istorm, start[2];
00399
00400     /* Read pressure... */
00401     NC(nc_inq_varid(ncid, varname, &varid));
00402     for (istorm = 0; istorm < nstorm; istorm++) {
00403         start[0] = istorm;
00404         start[1] = 0;
00405         count[0] = 1;
00406         count[1] = (size_t) nobs[istorm];
00407         NC(nc_get_vara_double(ncid, varid, start, count, x[istorm]));
00408     }
00409 }

```

## 5.17 island.c File Reference

### Functions

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

#### 5.17.1 Function Documentation

##### 5.17.1.1 `int main ( int argc, char * argv[] )`

Definition at line 3 of file [island.c](#).

```

00005     {
00006
00007     static pert_t *pert;
00008
00009     static wave_t *wave;
00010
00011     static FILE *in, *out;
00012
00013     static char pertname[LEN], ncfile[LEN];
00014
00015     static double gauss_fwhm, var_dh, orblat, lon0, lat0, dlon, dlat, offset,
00016     ebt, emu, enoise, evar, wbt, wmu, wnoise, wvar, etime, wtime,
00017     dt230, nu, nesr, aux;
00018
00019     static int iarg, ix, iy, itrack, itrack2, ixtrack, bg_poly_x, bg_poly_y,
00020     bg_smooth_x, bg_smooth_y, orb, orb_old = -1, en, wn, ncid, dimid[2],
00021     time_varid, track_varid, np_east_varid, var_east_varid,
00022     np_west_varid, var_west_varid, year_varid, doy_varid,
00023     track, year, mon, day, doy,iaux;
00024
00025     static size_t count[2] = { 1, 1 }, start[2];
00026
00027     /* Check arguments... */
00028     if (argc < 4)
00029         ERRMSG("Give parameters: <ctl> <var.tab> <pert1.nc> [<pert2.nc> ...]");
00030
00031     /* Get control parameters... */
00032     scan_ctl(argc, argv, "PERTNAME", -1, "4mu", pertname);
00033     lon0 = scan_ctl(argc, argv, "LON0", -1, "", NULL);
00034     lat0 = scan_ctl(argc, argv, "LAT0", -1, "", NULL);

```

```

00035     dlon = scan_ctl(argc, argv, "DLON", -1, "", NULL);
00036     dlat = scan_ctl(argc, argv, "DLAT", -1, "", NULL);
00037     offset = scan_ctl(argc, argv, "OFFSET", -1, "1", NULL);
00038     bg_poly_x = (int) scan_ctl(argc, argv, "BG_POLY_X", -1, "0", NULL);
00039     bg_poly_y = (int) scan_ctl(argc, argv, "BG_POLY_Y", -1, "0", NULL);
00040     bg_smooth_x = (int) scan_ctl(argc, argv, "BG_SMOOTH_X", -1, "0", NULL);
00041     bg_smooth_y = (int) scan_ctl(argc, argv, "BG_SMOOTH_Y", -1, "0", NULL);
00042     gauss_fwhm = scan_ctl(argc, argv, "GAUSS_FWHM", -1, "0", NULL);
00043     var_dh = scan_ctl(argc, argv, "VAR_DH", -1, "0", NULL);
00044     orblat = scan_ctl(argc, argv, "ORBLAT", -1, "0", NULL);
00045     dt230 = scan_ctl(argc, argv, "DT230", -1, "0.16", NULL);
00046     nu = scan_ctl(argc, argv, "NU", -1, "2345.0", NULL);
00047     scan_ctl(argc, argv, "NCFILE", -1, "-", ncfile);
00048
00049     /* Allocate... */
00050     ALLOC(pert, pert_t, 1);
00051
00052     /* Create file... */
00053     printf("Write variance statistics: %s\n", argv[2]);
00054     if (!(out = fopen(argv[2], "w")))
00055         ERRMSG("Cannot create file!");
00056
00057     /* Write header... */
00058     fprintf(out,
00059         "# $1 = time [s]\n"
00060         "# $2 = orbit number\n"
00061         "# $3 = eastern box: number of footprints\n"
00062         "# $4 = eastern box: variance [K^2]\n"
00063         "# $5 = eastern box: mean background temperature [K]\n"
00064         "# $6 = eastern box: noise estimate [K]\n"
00065         "# $7 = western box: number of footprints\n"
00066         "# $8 = western box: variance [K^2]\n"
00067         "# $9 = western box: mean background temperature [K]\n"
00068         "# $10 = western box: noise estimate [K]\n\n");
00069
00070     /* Create netCDF file... */
00071     if (ncfile[0] != '-') {
00072
00073         /* Create file... */
00074         printf("Write variance statistics: %s\n", ncfile);
00075         NC(nc_create(ncfile, NC_CLOBBER, &ncid));
00076
00077         /* Set dimensions... */
00078         NC(nc_def_dim(ncid, "NP", NC_UNLIMITED, &dimid[0]));
00079
00080         /* Add attributes... */
00081         aux = lon0;
00082         nc_put_att_double(ncid, NC_GLOBAL, "box_east_lon0", NC_DOUBLE, 1, &aux);
00083         aux = lon0 + dlon;
00084         nc_put_att_double(ncid, NC_GLOBAL, "box_east_lon1", NC_DOUBLE, 1, &aux);
00085         aux = lat0 - 0.5 * dlat;
00086         nc_put_att_double(ncid, NC_GLOBAL, "box_east_lat0", NC_DOUBLE, 1, &aux);
00087         aux = lat0 + 0.5 * dlat;
00088         nc_put_att_double(ncid, NC_GLOBAL, "box_east_lat1", NC_DOUBLE, 1, &aux);
00089         aux = lon0 - dlon - offset;
00090         nc_put_att_double(ncid, NC_GLOBAL, "box_west_lon0", NC_DOUBLE, 1, &aux);
00091         aux = lon0 - offset;
00092         nc_put_att_double(ncid, NC_GLOBAL, "box_west_lon1", NC_DOUBLE, 1, &aux);
00093         aux = lat0 - 0.5 * dlat;
00094         nc_put_att_double(ncid, NC_GLOBAL, "box_west_lat0", NC_DOUBLE, 1, &aux);
00095         aux = lat0 + 0.5 * dlat;
00096         nc_put_att_double(ncid, NC_GLOBAL, "box_west_lat1", NC_DOUBLE, 1, &aux);
00097
00098         /* Add variables... */
00099         NC(nc_def_var(ncid, "time", NC_DOUBLE, 1, dimid, &time_varid));
00100         add_att(ncid, time_varid, "s", "time (seconds since 2000-01-01T00:00Z)");
00101         NC(nc_def_var(ncid, "year", NC_INT, 1, dimid, &year_varid));
00102         add_att(ncid, year_varid, "1", "year");
00103         NC(nc_def_var(ncid, "doy", NC_INT, 1, dimid, &doy_varid));
00104         add_att(ncid, doy_varid, "1", "day of year");
00105         NC(nc_def_var(ncid, "track", NC_INT, 1, dimid, &track_varid));
00106         add_att(ncid, track_varid, "1", "along-track index");
00107         NC(nc_def_var(ncid, "var_east", NC_DOUBLE, 1, dimid, &var_east_varid));
00108         add_att(ncid, var_east_varid, "K^2", "BT variance (east)");
00109         NC(nc_def_var(ncid, "var_west", NC_DOUBLE, 1, dimid, &var_west_varid));
00110         add_att(ncid, var_west_varid, "K^2", "BT variance (west)");
00111         NC(nc_def_var(ncid, "np_east", NC_INT, 1, dimid, &np_east_varid));
00112         add_att(ncid, np_east_varid, "1", "number of footprints (east)");
00113         NC(nc_def_var(ncid, "np_west", NC_INT, 1, dimid, &np_west_varid));
00114         add_att(ncid, np_west_varid, "1", "number of footprints (west)");
00115
00116         /* Leave define mode... */
00117         NC(nc_enddef(ncid));
00118     }
00119
00120     /* Loop over perturbation files... */
00121     for (iarg = 3; iarg < argc; iarg++) {

```

```

00122
00123 /* Check filename... */
00124 if (!strcmp(argv[iarg], ncfile))
00125     continue;
00126
00127 /* Initialize... */
00128 orb = 0;
00129
00130 /* Read perturbation data... */
00131 if (!(in = fopen(argv[iarg], "r")))
00132     continue;
00133 else {
00134     fclose(in);
00135     read_pert(argv[iarg], pertname, pert);
00136 }
00137
00138 /* Recalculate background and perturbations... */
00139 if (bg_poly_x > 0 || bg_poly_y > 0 ||
00140     bg_smooth_x > 0 || bg_smooth_y > 0 || gauss_fwhm > 0 || var_dh > 0) {
00141
00142     /* Allocate... */
00143     ALLOC(wave, wave_t, 1);
00144
00145     /* Convert to wave analysis struct... */
00146     pert2wave(pert, wave, 0, pert->ntrack - 1, 0, pert->ntrack - 1);
00147
00148     /* Estimate background... */
00149     background_poly(wave, bg_poly_x, bg_poly_y);
00150     background_smooth(wave, bg_smooth_x, bg_smooth_y);
00151
00152     /* Gaussian filter... */
00153     gauss(wave, gauss_fwhm);
00154
00155     /* Compute variance... */
00156     variance(wave, var_dh);
00157
00158     /* Copy data... */
00159     for (ix = 0; ix < wave->nx; ix++)
00160         for (iy = 0; iy < wave->ny; iy++) {
00161             pert->pt[iy][ix] = wave->pt[ix][iy];
00162             pert->var[iy][ix] = wave->var[ix][iy];
00163         }
00164
00165     /* Free... */
00166     free(wave);
00167 }
00168
00169 /* Detection... */
00170 for (itrack = 0; itrack < pert->ntrack; itrack++)
00171     for (ixtrack = 0; ixtrack < pert->ntrack; ixtrack++) {
00172
00173         /* Check data... */
00174         if (pert->time[itrack][ixtrack] < 0
00175             || pert->lon[itrack][ixtrack] < -180
00176             || pert->lon[itrack][ixtrack] > 180
00177             || pert->lat[itrack][ixtrack] < -90
00178             || pert->lat[itrack][ixtrack] > 90
00179             || pert->pt[itrack][ixtrack] < -100
00180             || pert->pt[itrack][ixtrack] > 100
00181             || !gsl_finite(pert->bt[itrack][ixtrack])
00182             || !gsl_finite(pert->pt[itrack][ixtrack])
00183             || !gsl_finite(pert->var[itrack][ixtrack])
00184             || !gsl_finite(pert->dc[itrack][ixtrack]))
00185             continue;
00186
00187         /* Count orbits... */
00188         if (itrack > 0 && ixtrack == pert->ntrack / 2)
00189             if (pert->lat[itrack - 1][ixtrack] <= orblat
00190                 && pert->lat[itrack][ixtrack] >= orblat)
00191                 orb++;
00192         if (orb != orb_old) {
00193
00194             /* Set orbit index... */
00195             orb_old = orb;
00196
00197             /* Write output... */
00198             if (en > 0 && wn > 0) {
00199
00200                 /* Estimate noise... */
00201                 if (dt230 > 0) {
00202                     nesr = planck(230.0 + dt230, nu) - planck(230.0, nu);
00203                     enoise = brightness(planck(ebt / en, nu) + nesr, nu) - ebt / en;
00204                     wnoise = brightness(planck(wbt / wn, nu) + nesr, nu) - wbt / wn;
00205                 }
00206
00207                 /* Write output... */
00208                 fprintf(out, "%.2f %d %d %g %g %g %d %g %g %g\n", etime / en, orb,

```

```

00209         en, evar / en - gsl_pow_2(emu / en), ebt / en, enoise,
00210         wn, wvar / wn - gsl_pow_2(wmu / wn), wbt / wn, wnoise);
00211
00212     /* Write to netCDF file... */
00213     if (ncfile[0] != '-') {
00214
00215         /* Get year and doy... */
00216         jsec2time(etime / en, &year, &mon, &day, &iaux, &iaux, &iaux,
00217                 &aux);
00218         day2doy(year, mon, day, &doy);
00219
00220         /* Find along-track index... */
00221         track = 0;
00222         for (itrack2 = 0; itrack2 < pert->ntrack; itrack2++)
00223             if (fabs(pert->time[itrack2][0] - etime / en)
00224                 < fabs(pert->time[track][0] - etime / en))
00225                 track = itrack2;
00226
00227         /* Write data... */
00228         aux = etime / en;
00229         NC(nc_put_vara_double(ncid, time_varid, start, count, &aux));
00230         NC(nc_put_vara_int(ncid, year_varid, start, count, &year));
00231         NC(nc_put_vara_int(ncid, doy_varid, start, count, &doy));
00232         NC(nc_put_vara_int(ncid, track_varid, start, count, &track));
00233         NC(nc_put_vara_int(ncid, np_east_varid, start, count, &en));
00234         aux = evar / en - gsl_pow_2(emu / en) - gsl_pow_2(enoise);
00235         NC(nc_put_vara_double
00236            (ncid, var_east_varid, start, count, &aux));
00237         NC(nc_put_vara_int(ncid, np_west_varid, start, count, &wn));
00238         aux = wvar / wn - gsl_pow_2(wmu / wn) - gsl_pow_2(wnoise);
00239         NC(nc_put_vara_double
00240            (ncid, var_west_varid, start, count, &aux));
00241
00242         /* Increment data point counter... */
00243         start[0]++;
00244     }
00245 }
00246
00247 /* Initialize... */
00248 etime = wtime = 0;
00249 evar = wvar = 0;
00250 emu = wmu = 0;
00251 ebt = wbt = 0;
00252 en = wn = 0;
00253 }
00254
00255 /* Check if footprint is in eastern box... */
00256 if (pert->lon[itrack][ixtrack] >= lon0
00257     && pert->lon[itrack][ixtrack] <= lon0 + dlon
00258     && pert->lat[itrack][ixtrack] >= lat0 - dlat / 2.
00259     && pert->lat[itrack][ixtrack] <= lat0 + dlat / 2.) {
00260
00261     etime += pert->time[itrack][ixtrack];
00262     emu += pert->pt[itrack][ixtrack];
00263     evar += gsl_pow_2(pert->pt[itrack][ixtrack]);
00264     ebt += pert->bt[itrack][ixtrack];
00265     en++;
00266 }
00267
00268 /* Check if footprint is in western box... */
00269 if (pert->lon[itrack][ixtrack] >= lon0 - offset - dlon
00270     && pert->lon[itrack][ixtrack] <= lon0 - offset
00271     && pert->lat[itrack][ixtrack] >= lat0 - dlat / 2.
00272     && pert->lat[itrack][ixtrack] <= lat0 + dlat / 2.) {
00273
00274     wtime += pert->time[itrack][ixtrack];
00275     wmu += pert->pt[itrack][ixtrack];
00276     wvar += gsl_pow_2(pert->pt[itrack][ixtrack]);
00277     wbt += pert->bt[itrack][ixtrack];
00278     wn++;
00279 }
00280 }
00281
00282 /* Write output for last orbit... */
00283 if (en > 0 && wn > 0) {
00284
00285     /* Estimate noise... */
00286     if (dt230 > 0) {
00287         nesr = planck(230.0 + dt230, nu) - planck(230.0, nu);
00288         enoise = brightness(planck(ebt / en, nu) + nesr, nu) - ebt / en;
00289         wnoise = brightness(planck(wbt / wn, nu) + nesr, nu) - wbt / wn;
00290     }
00291
00292     /* Write output... */
00293     fprintf(out, "%.2f %d %d %g %g %g %d %g %g %g\n", etime / en, orb,
00294             en, evar / en - gsl_pow_2(emu / en), ebt / en, enoise,
00295             wn, wvar / wn - gsl_pow_2(wmu / wn), wbt / wn, wnoise);

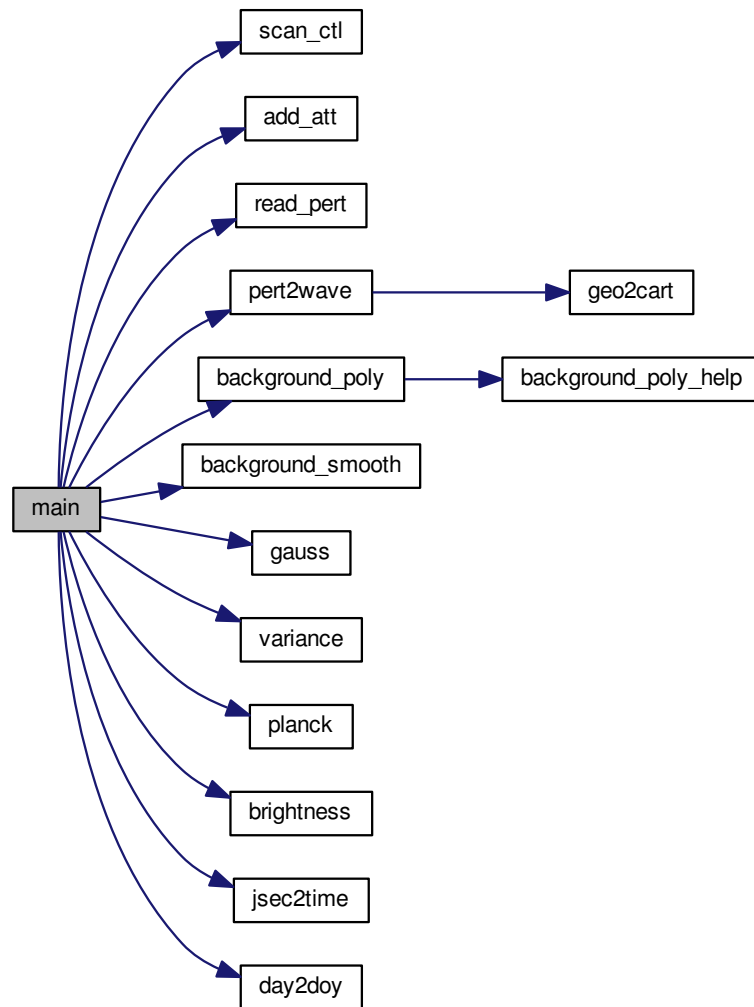
```

```

00296
00297     /* Write to netCDF file... */
00298     if (ncfile[0] != '-') {
00299
00300         /* Get year and doy... */
00301         jsec2time(etime / en, &year, &mon, &day, &iaux, &iaux, &iaux, &aux);
00302         day2doy(year, mon, day, &doy);
00303
00304         /* Find along-track index... */
00305         track = 0;
00306         for (itrack2 = 0; itrack2 < pert->ntrack; itrack2++)
00307             if (fabs(pert->time[itrack2][0] - etime / en)
00308                 < fabs(pert->time[track][0] - etime / en))
00309                 track = itrack2;
00310
00311         /* Write data... */
00312         aux = etime / en;
00313         NC(nc_put_vara_double(ncid, time_varid, start, count, &aux));
00314         NC(nc_put_vara_int(ncid, year_varid, start, count, &year));
00315         NC(nc_put_vara_int(ncid, doy_varid, start, count, &doy));
00316         NC(nc_put_vara_int(ncid, track_varid, start, count, &track));
00317         NC(nc_put_vara_int(ncid, np_east_varid, start, count, &en));
00318         aux = evar / en - gsl_pow_2(emu / en) - gsl_pow_2(enoise);
00319         NC(nc_put_vara_double(ncid, var_east_varid, start, count, &aux));
00320         NC(nc_put_vara_int(ncid, np_west_varid, start, count, &wn));
00321         aux = wvar / wn - gsl_pow_2(wmu / wn) - gsl_pow_2(wnoise);
00322         NC(nc_put_vara_double(ncid, var_west_varid, start, count, &aux));
00323
00324         /* Increment data point counter... */
00325         start[0]++;
00326     }
00327 }
00328 }
00329
00330 /* Close file... */
00331 fclose(out);
00332
00333 /* Close file... */
00334 if (ncfile[0] != '-')
00335     NC(nc_close(ncid));
00336
00337 /* Free... */
00338 free(pert);
00339
00340 return EXIT_SUCCESS;
00341 }

```

Here is the call graph for this function:



## 5.18 island.c

```

00001 #include "libairs.h"
00002
00003 int main(
00004     int argc,
00005     char *argv[]) {
00006
00007     static pert_t *pert;
00008
00009     static wave_t *wave;
00010
00011     static FILE *in, *out;
00012
00013     static char pertname[LEN], ncfile[LEN];
00014
00015     static double gauss_fwhm, var_dh, orblat, lon0, lat0, dlon, dlat, offset,
00016         ebt, emu, enoise, evar, wbt, wmu, wnoise, wvar, etime, wtime,
00017         dt230, nu, nesr, aux;
00018
00019     static int iarg, ix, iy, itrack, itrack2, ixtrack, bg_poly_x, bg_poly_y,
00020         bg_smooth_x, bg_smooth_y, orb, orb_old = -1, en, wn, ncid, dimid[2],

```



```

00021     time_varid, track_varid, np_east_varid, var_east_varid,
00022     np_west_varid, var_west_varid, year_varid, doy_varid,
00023     track, year, mon, day, doy, iaux;
00024
00025     static size_t count[2] = { 1, 1 }, start[2];
00026
00027     /* Check arguments... */
00028     if (argc < 4)
00029         ERRMSG("Give parameters: <ctl> <var.tab> <pert1.nc> [<pert2.nc> ...]");
00030
00031     /* Get control parameters... */
00032     scan_ctl(argc, argv, "PERTNAME", -1, "4mu", pertname);
00033     lon0 = scan_ctl(argc, argv, "LON0", -1, "", NULL);
00034     lat0 = scan_ctl(argc, argv, "LAT0", -1, "", NULL);
00035     dlon = scan_ctl(argc, argv, "DLON", -1, "", NULL);
00036     dlat = scan_ctl(argc, argv, "DLAT", -1, "", NULL);
00037     offset = scan_ctl(argc, argv, "OFFSET", -1, "1", NULL);
00038     bg_poly_x = (int) scan_ctl(argc, argv, "BG_POLY_X", -1, "0", NULL);
00039     bg_poly_y = (int) scan_ctl(argc, argv, "BG_POLY_Y", -1, "0", NULL);
00040     bg_smooth_x = (int) scan_ctl(argc, argv, "BG_SMOOTH_X", -1, "0", NULL);
00041     bg_smooth_y = (int) scan_ctl(argc, argv, "BG_SMOOTH_Y", -1, "0", NULL);
00042     gauss_fwhm = scan_ctl(argc, argv, "GAUSS_FWHM", -1, "0", NULL);
00043     var_dh = scan_ctl(argc, argv, "VAR_DH", -1, "0", NULL);
00044     orblat = scan_ctl(argc, argv, "ORBLAT", -1, "0", NULL);
00045     dt230 = scan_ctl(argc, argv, "DT230", -1, "0.16", NULL);
00046     nu = scan_ctl(argc, argv, "NU", -1, "2345.0", NULL);
00047     scan_ctl(argc, argv, "NCFILE", -1, "-", ncfile);
00048
00049     /* Allocate... */
00050     ALLOC(pert, pert_t, 1);
00051
00052     /* Create file... */
00053     printf("Write variance statistics: %s\n", argv[2]);
00054     if (!(out = fopen(argv[2], "w")))
00055         ERRMSG("Cannot create file!");
00056
00057     /* Write header... */
00058     fprintf(out,
00059         "# $1 = time [s]\n"
00060         "# $2 = orbit number\n"
00061         "# $3 = eastern box: number of footprints\n"
00062         "# $4 = eastern box: variance [K^2]\n"
00063         "# $5 = eastern box: mean background temperature [K]\n"
00064         "# $6 = eastern box: noise estimate [K]\n"
00065         "# $7 = western box: number of footprints\n"
00066         "# $8 = western box: variance [K^2]\n"
00067         "# $9 = western box: mean background temperature [K]\n"
00068         "# $10 = western box: noise estimate [K]\n\n");
00069
00070     /* Create netCDF file... */
00071     if (ncfile[0] != '-') {
00072
00073         /* Create file... */
00074         printf("Write variance statistics: %s\n", ncfile);
00075         NC(nc_create(ncfile, NC_CLOBBER, &ncid));
00076
00077         /* Set dimensions... */
00078         NC(nc_def_dim(ncid, "NP", NC_UNLIMITED, &dimid[0]));
00079
00080         /* Add attributes... */
00081         aux = lon0;
00082         nc_put_att_double(ncid, NC_GLOBAL, "box_east_lon0", NC_DOUBLE, 1, &aux);
00083         aux = lon0 + dlon;
00084         nc_put_att_double(ncid, NC_GLOBAL, "box_east_lon1", NC_DOUBLE, 1, &aux);
00085         aux = lat0 - 0.5 * dlat;
00086         nc_put_att_double(ncid, NC_GLOBAL, "box_east_lat0", NC_DOUBLE, 1, &aux);
00087         aux = lat0 + 0.5 * dlat;
00088         nc_put_att_double(ncid, NC_GLOBAL, "box_east_lat1", NC_DOUBLE, 1, &aux);
00089         aux = lon0 - dlon - offset;
00090         nc_put_att_double(ncid, NC_GLOBAL, "box_west_lon0", NC_DOUBLE, 1, &aux);
00091         aux = lon0 - offset;
00092         nc_put_att_double(ncid, NC_GLOBAL, "box_west_lon1", NC_DOUBLE, 1, &aux);
00093         aux = lat0 - 0.5 * dlat;
00094         nc_put_att_double(ncid, NC_GLOBAL, "box_west_lat0", NC_DOUBLE, 1, &aux);
00095         aux = lat0 + 0.5 * dlat;
00096         nc_put_att_double(ncid, NC_GLOBAL, "box_west_lat1", NC_DOUBLE, 1, &aux);
00097
00098         /* Add variables... */
00099         NC(nc_def_var(ncid, "time", NC_DOUBLE, 1, dimid, &time_varid));
00100         add_att(ncid, time_varid, "s", "time (seconds since 2000-01-01T00:00Z)");
00101         NC(nc_def_var(ncid, "year", NC_INT, 1, dimid, &year_varid));
00102         add_att(ncid, year_varid, "1", "year");
00103         NC(nc_def_var(ncid, "doy", NC_INT, 1, dimid, &doy_varid));
00104         add_att(ncid, doy_varid, "1", "day of year");
00105         NC(nc_def_var(ncid, "track", NC_INT, 1, dimid, &track_varid));
00106         add_att(ncid, track_varid, "1", "along-track index");
00107         NC(nc_def_var(ncid, "var_east", NC_DOUBLE, 1, dimid, &var_east_varid));

```

```

00108     add_att(ncid, var_east_varid, "K^2", "BT variance (east)");
00109     NC(nc_def_var(ncid, "var_west", NC_DOUBLE, 1, dimid, &var_west_varid));
00110     add_att(ncid, var_west_varid, "K^2", "BT variance (west)");
00111     NC(nc_def_var(ncid, "np_east", NC_INT, 1, dimid, &np_east_varid));
00112     add_att(ncid, np_east_varid, "1", "number of footprints (east)");
00113     NC(nc_def_var(ncid, "np_west", NC_INT, 1, dimid, &np_west_varid));
00114     add_att(ncid, np_west_varid, "1", "number of footprints (west)");
00115
00116     /* Leave define mode... */
00117     NC(nc_enddef(ncid));
00118 }
00119
00120 /* Loop over perturbation files... */
00121 for (iarg = 3; iarg < argc; iarg++) {
00122
00123     /* Check filename... */
00124     if (!strcmp(argv[iarg], ncfile))
00125         continue;
00126
00127     /* Initialize... */
00128     orb = 0;
00129
00130     /* Read perturbation data... */
00131     if (!(in = fopen(argv[iarg], "r")))
00132         continue;
00133     else {
00134         fclose(in);
00135         read_pert(argv[iarg], pertname, pert);
00136     }
00137
00138     /* Recalculate background and perturbations... */
00139     if (bg_poly_x > 0 || bg_poly_y > 0 ||
00140         bg_smooth_x > 0 || bg_smooth_y > 0 || gauss_fwhm > 0 || var_dh > 0) {
00141
00142         /* Allocate... */
00143         ALLOC(wave, wave_t, 1);
00144
00145         /* Convert to wave analysis struct... */
00146         pert2wave(pert, wave, 0, pert->ntrack - 1, 0, pert->ntrack - 1);
00147
00148         /* Estimate background... */
00149         background_poly(wave, bg_poly_x, bg_poly_y);
00150         background_smooth(wave, bg_smooth_x, bg_smooth_y);
00151
00152         /* Gaussian filter... */
00153         gauss(wave, gauss_fwhm);
00154
00155         /* Compute variance... */
00156         variance(wave, var_dh);
00157
00158         /* Copy data... */
00159         for (ix = 0; ix < wave->nx; ix++)
00160             for (iy = 0; iy < wave->ny; iy++) {
00161                 pert->pt[ix][iy] = wave->pt[ix][iy];
00162                 pert->var[ix][iy] = wave->var[ix][iy];
00163             }
00164
00165         /* Free... */
00166         free(wave);
00167     }
00168
00169     /* Detection... */
00170     for (itrack = 0; itrack < pert->ntrack; itrack++)
00171         for (ixtrack = 0; ixtrack < pert->ntrack; ixtrack++) {
00172
00173             /* Check data... */
00174             if (pert->time[ixtrack][ixtrack] < 0
00175                 || pert->lon[ixtrack][ixtrack] < -180
00176                 || pert->lon[ixtrack][ixtrack] > 180
00177                 || pert->lat[ixtrack][ixtrack] < -90
00178                 || pert->lat[ixtrack][ixtrack] > 90
00179                 || pert->pt[ixtrack][ixtrack] < -100
00180                 || pert->pt[ixtrack][ixtrack] > 100
00181                 || !gsl_finite(pert->bt[ixtrack][ixtrack])
00182                 || !gsl_finite(pert->pt[ixtrack][ixtrack])
00183                 || !gsl_finite(pert->var[ixtrack][ixtrack])
00184                 || !gsl_finite(pert->dc[ixtrack][ixtrack]))
00185                 continue;
00186
00187             /* Count orbits... */
00188             if (itrack > 0 && ixtrack == pert->ntrack / 2)
00189                 if (pert->lat[itrack - 1][ixtrack] <= orblat
00190                     && pert->lat[ixtrack][ixtrack] >= orblat)
00191                     orb++;
00192             if (orb != orb_old) {
00193
00194                 /* Set orbit index... */

```

```

00195     orb_old = orb;
00196
00197     /* Write output... */
00198     if (en > 0 && wn > 0) {
00199
00200         /* Estimate noise... */
00201         if (dt230 > 0) {
00202             nesr = planck(230.0 + dt230, nu) - planck(230.0, nu);
00203             enoise = brightness(planck(ebt / en, nu) + nesr, nu) - ebt / en;
00204             wnoise = brightness(planck(wbt / wn, nu) + nesr, nu) - wbt / wn;
00205         }
00206
00207         /* Write output... */
00208         fprintf(out, "%.2f %d %d %g %g %g %d %g %g %g\n", etime / en, orb,
00209             en, evar / en - gsl_pow_2(emu / en), ebt / en, enoise,
00210             wn, wvar / wn - gsl_pow_2(wmu / wn), wbt / wn, wnoise);
00211
00212         /* Write to netCDF file... */
00213         if (ncfile[0] != '-') {
00214
00215             /* Get year and doy... */
00216             jsec2time(etime / en, &year, &mon, &day, &iaux, &iaux, &iaux,
00217                 &aux);
00218             day2doy(year, mon, day, &doy);
00219
00220             /* Find along-track index... */
00221             track = 0;
00222             for (itrack2 = 0; itrack2 < pert->ntrack; itrack2++)
00223                 if (fabs(pert->time[itrack2][0] - etime / en)
00224                     < fabs(pert->time[track][0] - etime / en))
00225                     track = itrack2;
00226
00227             /* Write data... */
00228             aux = etime / en;
00229             NC(nc_put_vara_double(ncid, time_varid, start, count, &aux));
00230             NC(nc_put_vara_int(ncid, year_varid, start, count, &year));
00231             NC(nc_put_vara_int(ncid, doy_varid, start, count, &doy));
00232             NC(nc_put_vara_int(ncid, track_varid, start, count, &track));
00233             NC(nc_put_vara_int(ncid, np_east_varid, start, count, &en));
00234             aux = evar / en - gsl_pow_2(emu / en) - gsl_pow_2(enoise);
00235             NC(nc_put_vara_double
00236                 (ncid, var_east_varid, start, count, &aux));
00237             NC(nc_put_vara_int(ncid, np_west_varid, start, count, &wn));
00238             aux = wvar / wn - gsl_pow_2(wmu / wn) - gsl_pow_2(wnoise);
00239             NC(nc_put_vara_double
00240                 (ncid, var_west_varid, start, count, &aux));
00241
00242             /* Increment data point counter... */
00243             start[0]++;
00244         }
00245     }
00246
00247     /* Initialize... */
00248     etime = wtime = 0;
00249     evar = wvar = 0;
00250     emu = wmu = 0;
00251     ebt = wbt = 0;
00252     en = wn = 0;
00253 }
00254
00255 /* Check if footprint is in eastern box... */
00256 if (pert->lon[itrack][ixtrack] >= lon0
00257     && pert->lon[itrack][ixtrack] <= lon0 + dlon
00258     && pert->lat[itrack][ixtrack] >= lat0 - dlat / 2.
00259     && pert->lat[itrack][ixtrack] <= lat0 + dlat / 2.) {
00260
00261     etime += pert->time[itrack][ixtrack];
00262     emu += pert->pt[itrack][ixtrack];
00263     evar += gsl_pow_2(pert->pt[itrack][ixtrack]);
00264     ebt += pert->bt[itrack][ixtrack];
00265     en++;
00266 }
00267
00268 /* Check if footprint is in western box... */
00269 if (pert->lon[itrack][ixtrack] >= lon0 - offset - dlon
00270     && pert->lon[itrack][ixtrack] <= lon0 - offset
00271     && pert->lat[itrack][ixtrack] >= lat0 - dlat / 2.
00272     && pert->lat[itrack][ixtrack] <= lat0 + dlat / 2.) {
00273
00274     wtime += pert->time[itrack][ixtrack];
00275     wmu += pert->pt[itrack][ixtrack];
00276     wvar += gsl_pow_2(pert->pt[itrack][ixtrack]);
00277     wbt += pert->bt[itrack][ixtrack];
00278     wn++;
00279 }
00280 }
00281

```

```

00282      /* Write output for last orbit... */
00283      if (en > 0 && wn > 0) {
00284
00285          /* Estimate noise... */
00286          if (dt230 > 0) {
00287              nesr = planck(230.0 + dt230, nu) - planck(230.0, nu);
00288              enoise = brightness(planck(ebt / en, nu) + nesr, nu) - ebt / en;
00289              wnoise = brightness(planck(wbt / wn, nu) + nesr, nu) - wbt / wn;
00290          }
00291
00292          /* Write output... */
00293          fprintf(out, "%.2f %d %d %g %g %g %d %g %g %g\n", etime / en, orb,
00294                  en, evar / en - gsl_pow_2(emu / en), ebt / en, enoise,
00295                  wn, wvar / wn - gsl_pow_2(wmu / wn), wbt / wn, wnoise);
00296
00297          /* Write to netCDF file... */
00298          if (ncfile[0] != '-') {
00299
00300              /* Get year and doy... */
00301              jsec2time(etime / en, &year, &mon, &day, &iaux, &iaux, &iaux, &aux);
00302              day2doy(year, mon, day, &doy);
00303
00304              /* Find along-track index... */
00305              track = 0;
00306              for (itrack2 = 0; itrack2 < pert->ntrack; itrack2++)
00307                  if (fabs(pert->time[itrack2][0] - etime / en)
00308                      < fabs(pert->time[track][0] - etime / en))
00309                      track = itrack2;
00310
00311              /* Write data... */
00312              aux = etime / en;
00313              NC(nc_put_vara_double(ncid, time_varid, start, count, &aux));
00314              NC(nc_put_vara_int(ncid, year_varid, start, count, &year));
00315              NC(nc_put_vara_int(ncid, doy_varid, start, count, &doy));
00316              NC(nc_put_vara_int(ncid, track_varid, start, count, &track));
00317              NC(nc_put_vara_int(ncid, np_east_varid, start, count, &en));
00318              aux = evar / en - gsl_pow_2(emu / en) - gsl_pow_2(enoise);
00319              NC(nc_put_vara_double(ncid, var_east_varid, start, count, &aux));
00320              NC(nc_put_vara_int(ncid, np_west_varid, start, count, &wn));
00321              aux = wvar / wn - gsl_pow_2(wmu / wn) - gsl_pow_2(wnoise);
00322              NC(nc_put_vara_double(ncid, var_west_varid, start, count, &aux));
00323
00324              /* Increment data point counter... */
00325              start[0]++;
00326          }
00327      }
00328  }
00329
00330  /* Close file... */
00331  fclose(out);
00332
00333  /* Close file... */
00334  if (ncfile[0] != '-')
00335      NC(nc_close(ncid));
00336
00337  /* Free... */
00338  free(pert);
00339
00340  return EXIT_SUCCESS;
00341 }

```

## 5.19 issifm.c File Reference

### Functions

- void [intpol](#) (float ps[NLON][NLAT][NZ], float ts[NLON][NLAT][NZ], float zs[NLON][NLAT][NZ], double lons[NLON], double lats[NLAT], int nz, int nlon, int nlat, double z, double lon, double lat, double \*p, double \*t)

*Interpolation of model data.*

- void [smooth](#) (float ps[NLON][NLAT][NZ], float ts[NLON][NLAT][NZ], float zs[NLON][NLAT][NZ], double lons[NLON], double lats[NLAT], int nz, int nlon, int nlat)

*Smoothing of model data.*

- void [write\\_nc](#) (char \*filename, [wave\\_t](#) \*wave)

*Write wave struct to netCDF file.*

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

### 5.19.1 Function Documentation

**5.19.1.1** `void intpol ( float ps[NLON][NLAT][NZ], float ts[NLON][NLAT][NZ], float zs[NLON][NLAT][NZ], double lons[NLON], double lats[NLAT], int nz, int nlon, int nlat, double z, double lon, double lat, double * p, double * t )`

Interpolation of model data.

Definition at line 504 of file `issifm.c`.

```

00517     {
00518
00519     double p00, p01, p10, p11, t00, t01, t10, t11, zd[NZ];
00520
00521     int iz, ilon, ilat;
00522
00523     /* Adjust longitude... */
00524     if (lons[nlon - 1] > 180)
00525         if (lon < 0)
00526             lon += 360;
00527
00528     /* Check horizontal range... */
00529     if (lon < lons[0]
00530         || lon > lons[nlon - 1]
00531         || lat < GSL_MIN(lats[0], lats[nlat - 1])
00532         || lat > GSL_MAX(lats[0], lats[nlat - 1])) {
00533         *p = GSL_NAN;
00534         *t = GSL_NAN;
00535         return;
00536     }
00537
00538     /* Get indices... */
00539     ilon = locate_irr(lons, nlon, lon);
00540     ilat = locate_irr(lats, nlat, lat);
00541
00542     /* Check data... */
00543     if (!gsl_finite(zs[ilon][ilat][0])
00544         || !gsl_finite(zs[ilon][ilat][nz - 1])
00545         || !gsl_finite(zs[ilon][ilat + 1][nz - 1])
00546         || !gsl_finite(zs[ilon][ilat + 1][nz - 1])
00547         || !gsl_finite(zs[ilon + 1][ilat][nz - 1])
00548         || !gsl_finite(zs[ilon + 1][ilat][nz - 1])
00549         || !gsl_finite(zs[ilon + 1][ilat + 1][nz - 1])
00550         || !gsl_finite(zs[ilon + 1][ilat + 1][nz - 1])) {
00551         *p = GSL_NAN;
00552         *t = GSL_NAN;
00553         return;
00554     }
00555
00556     /* Check vertical range... */
00557     if (z > GSL_MAX(zs[ilon][ilat][0], zs[ilon][ilat][nz - 1])
00558         || z < GSL_MIN(zs[ilon][ilat][0], zs[ilon][ilat][nz - 1])
00559         || z > GSL_MAX(zs[ilon][ilat + 1][0], zs[ilon][ilat + 1][nz - 1])
00560         || z < GSL_MIN(zs[ilon][ilat + 1][0], zs[ilon][ilat + 1][nz - 1])
00561         || z > GSL_MAX(zs[ilon + 1][ilat][0], zs[ilon + 1][ilat][nz - 1])
00562         || z < GSL_MIN(zs[ilon + 1][ilat][0], zs[ilon + 1][ilat][nz - 1])
00563         || z > GSL_MAX(zs[ilon + 1][ilat + 1][0],
00564             zs[ilon + 1][ilat + 1][nz - 1])
00565         || z < GSL_MIN(zs[ilon + 1][ilat + 1][0],
00566             zs[ilon + 1][ilat + 1][nz - 1]))
00567         return;
00568
00569     /* Interpolate vertically... */
00570     for (iz = 0; iz < nz; iz++)
00571         zd[iz] = zs[ilon][ilat][iz];
00572     iz = locate_irr(zd, nz, z);
00573     p00 = LIN(zs[ilon][ilat][iz], ps[ilon][ilat][iz],
00574         zs[ilon][ilat][iz + 1], ps[ilon][ilat][iz + 1], z);
00575     t00 = LIN(zs[ilon][ilat][iz], ts[ilon][ilat][iz],
00576         zs[ilon][ilat][iz + 1], ts[ilon][ilat][iz + 1], z);
00577
00578     for (iz = 0; iz < nz; iz++)
00579         zd[iz] = zs[ilon][ilat + 1][iz];
00580     iz = locate_irr(zd, nz, z);
00581     p01 = LIN(zs[ilon][ilat + 1][iz], ps[ilon][ilat + 1][iz],
00582         zs[ilon][ilat + 1][iz + 1], ps[ilon][ilat + 1][iz + 1], z);
00583     t01 = LIN(zs[ilon][ilat + 1][iz], ts[ilon][ilat + 1][iz],
00584         zs[ilon][ilat + 1][iz + 1], ts[ilon][ilat + 1][iz + 1], z);
00585
00586     for (iz = 0; iz < nz; iz++)
00587         zd[iz] = zs[ilon + 1][ilat][iz];
00588     iz = locate_irr(zd, nz, z);

```

```

00589  p10 = LIN(zs[ilon + 1][ilat][iz], ps[ilon + 1][ilat][iz],
00590           zs[ilon + 1][ilat][iz + 1], ps[ilon + 1][ilat][iz + 1], z);
00591  t10 = LIN(zs[ilon + 1][ilat][iz], ts[ilon + 1][ilat][iz],
00592           zs[ilon + 1][ilat][iz + 1], ts[ilon + 1][ilat][iz + 1], z);
00593
00594  for (iz = 0; iz < nz; iz++)
00595      zd[iz] = zs[ilon + 1][ilat + 1][iz];
00596  iz = locate_irr(zd, nz, z);
00597  p11 = LIN(zs[ilon + 1][ilat + 1][iz], ps[ilon + 1][ilat + 1][iz],
00598           zs[ilon + 1][ilat + 1][iz + 1], ps[ilon + 1][ilat + 1][iz + 1],
00599           z);
00600  t11 = LIN(zs[ilon + 1][ilat + 1][iz], ts[ilon + 1][ilat + 1][iz],
00601           zs[ilon + 1][ilat + 1][iz + 1], ts[ilon + 1][ilat + 1][iz + 1],
00602           z);
00603
00604  /* Interpolate horizontally... */
00605  p00 = LIN(lons[ilon], p00, lons[ilon + 1], p10, lon);
00606  p11 = LIN(lons[ilon], p01, lons[ilon + 1], p11, lon);
00607  *p = LIN(lats[ilat], p00, lats[ilat + 1], p11, lat);
00608
00609  t00 = LIN(lons[ilon], t00, lons[ilon + 1], t10, lon);
00610  t11 = LIN(lons[ilon], t01, lons[ilon + 1], t11, lon);
00611  *t = LIN(lats[ilat], t00, lats[ilat + 1], t11, lat);
00612 }

```

Here is the call graph for this function:



**5.19.1.2** void smooth ( float *ps*[NLON][NLAT][NZ], float *ts*[NLON][NLAT][NZ], float *zs*[NLON][NLAT][NZ], double *lons*[NLON], double *lats*[NLAT], int *nz*, int *nlon*, int *nlat* )

Smoothing of model data.

Definition at line 616 of file [issifm.c](#).

```

00624  {
00625
00626  static float hp[NLON][NLAT], ht[NLON][NLAT], hz[NLON][NLAT], w, wsum;
00627
00628  static double dx, dy, wx[10], wy[10];
00629
00630  int iz, ilon, ilon2, ilat, ilat2, dlon = 3, dlat = 3;
00631
00632  /* Set weights... */
00633  dy = RE * M_PI / 180. * fabs(lats[1] - lats[0]);
00634  for (ilat = 0; ilat <= dlat; ilat++)
00635      wy[ilat] = exp(-0.5 * POW2(ilat * dy * 2.35482 / 20.));
00636
00637  /* Loop over height levels... */
00638  for (iz = 0; iz < nz; iz++) {
00639
00640      /* Write info... */
00641      printf("Smoothing level %d / %d ...\n", iz + 1, nz);
00642
00643      /* Copy data... */
00644      for (ilon = 0; ilon < nlon; ilon++)
00645          for (ilat = 0; ilat < nlat; ilat++) {
00646              hp[ilon][ilat] = ps[ilon][ilat][iz];
00647              ht[ilon][ilat] = ts[ilon][ilat][iz];
00648              hz[ilon][ilat] = zs[ilon][ilat][iz];
00649          }
00650
00651      /* Loop over latitudes... */

```

```

00652     for (ilat = 0; ilat < nlat; ilat++) {
00653
00654         /* Set weights... */
00655         dx = RE * M_PI / 180. * cos(lats[ilat] * M_PI / 180.) *
00656             fabs(lons[1] - lons[0]);
00657         for (ilon = 0; ilon <= dlon; ilon++)
00658             wx[ilon] = exp(-0.5 * POW2(ilon * dx * 2.35482 / 20.));
00659
00660         /* Loop over longitudes... */
00661         for (ilon = 0; ilon < nlon; ilon++) {
00662             wsum = 0;
00663             ps[ilon][ilat][iz] = 0;
00664             ts[ilon][ilat][iz] = 0;
00665             zs[ilon][ilat][iz] = 0;
00666             for (ilon2 = GSL_MAX(ilon - dlon, 0);
00667                 ilon2 <= GSL_MIN(ilon + dlon, nlon - 1); ilon2++)
00668                 for (ilat2 = GSL_MAX(ilat - dlat, 0);
00669                     ilat2 <= GSL_MIN(ilat + dlat, nlat - 1); ilat2++) {
00670                     w = (float) (wx[abs(ilon2 - ilon)] * wy[abs(ilat2 - ilat)]);
00671                     ps[ilon][ilat][iz] += w * hp[ilon2][ilat2];
00672                     ts[ilon][ilat][iz] += w * ht[ilon2][ilat2];
00673                     zs[ilon][ilat][iz] += w * hz[ilon2][ilat2];
00674                     wsum += w;
00675                 }
00676             ps[ilon][ilat][iz] /= wsum;
00677             ts[ilon][ilat][iz] /= wsum;
00678             zs[ilon][ilat][iz] /= wsum;
00679         }
00680     }
00681 }
00682 }

```

### 5.19.1.3 void write\_nc ( char \* filename, wave\_t \* wave )

Write wave struct to netCDF file.

Definition at line 686 of file [issifm.c](#).

```

00688     {
00689
00690         static double help[WX * WY];
00691
00692         int ix, iy, ncid, dimid[10], lon_id, lat_id, bt_id, pt_id, var_id;
00693
00694         /* Create netCDF file... */
00695         NC(nc_create(filename, NC_CLOBBER, &ncid));
00696
00697         /* Set dimensions... */
00698         NC(nc_def_dim(ncid, "NTRACK", (size_t) wave->ny, &dimid[0]));
00699         NC(nc_def_dim(ncid, "NXTRACK", (size_t) wave->nx, &dimid[1]));
00700
00701         /* Add variables... */
00702         NC(nc_def_var(ncid, "lon", NC_DOUBLE, 2, dimid, &lon_id));
00703         add_att(ncid, lon_id, "deg", "footprint longitude");
00704         NC(nc_def_var(ncid, "lat", NC_DOUBLE, 2, dimid, &lat_id));
00705         add_att(ncid, lat_id, "deg", "footprint latitude");
00706         NC(nc_def_var(ncid, "bt", NC_FLOAT, 2, dimid, &bt_id));
00707         add_att(ncid, bt_id, "K", "brightness temperature");
00708         NC(nc_def_var(ncid, "bt_pt", NC_FLOAT, 2, dimid, &pt_id));
00709         add_att(ncid, pt_id, "K", "brightness temperature perturbation");
00710         NC(nc_def_var(ncid, "bt_var", NC_FLOAT, 2, dimid, &var_id));
00711         add_att(ncid, var_id, "K^2", "brightness temperature variance");
00712
00713         /* Leave define mode... */
00714         NC(nc_enddef(ncid));
00715
00716         /* Write data... */
00717         for (ix = 0; ix < wave->nx; ix++)
00718             for (iy = 0; iy < wave->ny; iy++)
00719                 help[iy * wave->nx + ix] = wave->lon[ix][iy];
00720         NC(nc_put_var_double(ncid, lon_id, help));
00721         for (ix = 0; ix < wave->nx; ix++)
00722             for (iy = 0; iy < wave->ny; iy++)
00723                 help[iy * wave->nx + ix] = wave->lat[ix][iy];
00724         NC(nc_put_var_double(ncid, lat_id, help));
00725         for (ix = 0; ix < wave->nx; ix++)
00726             for (iy = 0; iy < wave->ny; iy++)
00727                 help[iy * wave->nx + ix] = wave->temp[ix][iy];
00728         NC(nc_put_var_double(ncid, bt_id, help));
00729         for (ix = 0; ix < wave->nx; ix++)

```

```

00730     for (iy = 0; iy < wave->ny; iy++)
00731         help[iy * wave->nx + ix] = wave->pt[ix][iy];
00732     NC(nc_put_var_double(ncid, pt_id, help));
00733     for (ix = 0; ix < wave->nx; ix++)
00734         for (iy = 0; iy < wave->ny; iy++)
00735             help[iy * wave->nx + ix] = wave->var[ix][iy];
00736     NC(nc_put_var_double(ncid, var_id, help));
00737
00738     /* Close file... */
00739     NC(nc_close(ncid));
00740 }

```

Here is the call graph for this function:



#### 5.19.1.4 int main ( int argc, char \* argv[] )

Definition at line 56 of file [issifm.c](#).

```

00058     {
00059
00060     static ctl_t ctl;
00061
00062     static char kernel[LEN], pertname[LEN];
00063
00064     static double lon[NLON], lat[NLAT], xo[3], xs[3], xm[3], var_dh = 100.,
00065         f, t_ovp, hybm[NZ], hybm[NZ], kz[NSHAPE], kw[NSHAPE], w, wsum;
00066
00067     static float *help, ps[NLON][NLAT], p[NLON][NLAT][NZ], t[NLON][NLAT][NZ],
00068         z[NLON][NLAT][NZ];
00069
00070     static int init, id, itrack, ixtrack, ncid, dimid, varid, slant,
00071         ilon, ilat, iz, nlon, nlat, nz, ip, track0, track1, nk, okay;
00072
00073     static size_t rs;
00074
00075     atm_t *atm;
00076
00077     obs_t *obs;
00078
00079     pert_t *pert;
00080
00081     wave_t *wave;
00082
00083     /* -----
00084     Get control parameters...
00085     ----- */
00086
00087     /* Check arguments... */
00088     if (argc < 8)
00089         ERRMSG("Give parameters: <ctl> <model> <model.nc> <pert.nc>"
00090             " <wave_airs.tab> <wave_model.tab> <wave_airs.nc> <wave_model.nc>");
00091
00092     /* Read control parameters... */
00093     read_ctl(argc, argv, &ctl);
00094     scan_ctl(argc, argv, "PERTNAME", -1, "4mu", pertname);
00095     scan_ctl(argc, argv, "KERNEL", -1, "-", kernel);
00096     slant = (int) scan_ctl(argc, argv, "SLANT", -1, "1", NULL);
00097     t_ovp = scan_ctl(argc, argv, "T_OVP", -1, "", NULL);
00098
00099     /* Set control parameters... */
00100     ctl.write_bbt = 1;
00101
00102     /* -----

```



```

00103     Read model data...
00104     ----- */
00105
00106     /* Allocate... */
00107     ALLOC(help, float,
00108           NLON * NLAT * NZ);
00109
00110     /* Open file... */
00111     printf("Read %s data: %s\n", argv[2], argv[3]);
00112     NC(nc_open(argv[3], NC_NOWRITE, &ncid));
00113
00114     /* Read latitudes... */
00115     if (nc_inq_dimid(ncid, "lat", &dimid) != NC_NOERR)
00116         NC(nc_inq_dimid(ncid, "latitude", &dimid));
00117     NC(nc_inq_dimlen(ncid, dimid, &rs));
00118     nlat = (int) rs;
00119     if (nlat > NLAT)
00120         ERRMSG("Too many latitudes!");
00121     if (nc_inq_varid(ncid, "lat", &varid) != NC_NOERR)
00122         NC(nc_inq_varid(ncid, "latitude", &varid));
00123     NC(nc_get_var_double(ncid, varid, lat));
00124
00125     /* Read longitudes... */
00126     if (nc_inq_dimid(ncid, "lon", &dimid) != NC_NOERR)
00127         NC(nc_inq_dimid(ncid, "longitude", &dimid));
00128     NC(nc_inq_dimlen(ncid, dimid, &rs));
00129     nlon = (int) rs;
00130     if (nlon > NLON)
00131         ERRMSG("Too many longitudes!");
00132     if (nc_inq_varid(ncid, "lon", &varid))
00133         NC(nc_inq_varid(ncid, "longitude", &varid));
00134     NC(nc_get_var_double(ncid, varid, lon));
00135
00136     /* Read ICON data... */
00137     if (strcasecmp(argv[2], "icon") == 0) {
00138
00139         /* Get height levels... */
00140         NC(nc_inq_dimid(ncid, "height", &dimid));
00141         NC(nc_inq_dimlen(ncid, dimid, &rs));
00142         nz = (int) rs;
00143         if (nz > NZ)
00144             ERRMSG("Too many altitudes!");
00145
00146         /* Read height... */
00147         NC(nc_inq_varid(ncid, "z_mc", &varid));
00148         NC(nc_get_var_float(ncid, varid, help));
00149         for (ilon = 0; ilon < nlon; ilon++)
00150             for (ilat = 0; ilat < nlat; ilat++)
00151                 for (iz = 0; iz < nz; iz++)
00152                     z[ilon][ilat][iz] =
00153                         (float) (help[(iz * nlat + ilat) * nlon + ilon] / 1e3);
00154
00155         /* Read temperature... */
00156         NC(nc_inq_varid(ncid, "temp", &varid));
00157         NC(nc_get_var_float(ncid, varid, help));
00158         for (ilon = 0; ilon < nlon; ilon++)
00159             for (ilat = 0; ilat < nlat; ilat++)
00160                 for (iz = 0; iz < nz; iz++)
00161                     t[ilon][ilat][iz] = help[(iz * nlat + ilat) * nlon + ilon];
00162
00163         /* Read pressure... */
00164         NC(nc_inq_varid(ncid, "pres", &varid));
00165         NC(nc_get_var_float(ncid, varid, help));
00166         for (ilon = 0; ilon < nlon; ilon++)
00167             for (ilat = 0; ilat < nlat; ilat++)
00168                 for (iz = 0; iz < nz; iz++)
00169                     p[ilon][ilat][iz] =
00170                         (float) (help[(iz * nlat + ilat) * nlon + ilon] / 1e2);
00171     }
00172
00173     /* Read IFS data... */
00174     else if (strcasecmp(argv[2], "ifs") == 0) {
00175
00176         /* Get height levels... */
00177         NC(nc_inq_dimid(ncid, "lev_2", &dimid));
00178         NC(nc_inq_dimlen(ncid, dimid, &rs));
00179         nz = (int) rs;
00180         if (nz > NZ)
00181             ERRMSG("Too many altitudes!");
00182
00183         /* Read height... */
00184         NC(nc_inq_varid(ncid, "gh", &varid));
00185         NC(nc_get_var_float(ncid, varid, help));
00186         for (ilon = 0; ilon < nlon; ilon++)
00187             for (ilat = 0; ilat < nlat; ilat++)
00188                 for (iz = 0; iz < nz; iz++)
00189                     z[ilon][ilat][iz] =

```

```

00190         (float) (help[(iz * nlat + ilat) * nlon + ilon] / 1e3);
00191
00192     /* Read temperature... */
00193     NC(nc_inq_varid(ncid, "t", &varid));
00194     NC(nc_get_var_float(ncid, varid, help));
00195     for (ilon = 0; ilon < nlon; ilon++)
00196         for (ilat = 0; ilat < nlat; ilat++)
00197             for (iz = 0; iz < nz; iz++)
00198                 t[ilon][ilat][iz] = help[(iz * nlat + ilat) * nlon + ilon];
00199
00200     /* Read surface pressure... */
00201     NC(nc_inq_varid(ncid, "lnsp", &varid));
00202     NC(nc_get_var_float(ncid, varid, help));
00203     for (ilon = 0; ilon < nlon; ilon++)
00204         for (ilat = 0; ilat < nlat; ilat++)
00205             ps[ilon][ilat] = (float) exp(help[ilat * nlon + ilon]);
00206
00207     /* Read grid coefficients... */
00208     NC(nc_inq_varid(ncid, "hyam", &varid));
00209     NC(nc_get_var_double(ncid, varid, hyam));
00210     NC(nc_inq_varid(ncid, "hybm", &varid));
00211     NC(nc_get_var_double(ncid, varid, hybm));
00212
00213     /* Calculate pressure... */
00214     for (ilon = 0; ilon < nlon; ilon++)
00215         for (ilat = 0; ilat < nlat; ilat++)
00216             for (iz = 0; iz < nz; iz++)
00217                 p[ilon][ilat][iz]
00218                     = (float) ((hyam[iz] + hybm[iz] * ps[ilon][ilat]) / 100.);
00219 }
00220
00221 /* Read UM data... */
00222 else if (strcasecmp(argv[2], "um") == 0) {
00223
00224     /* Get height levels... */
00225     if (nc_inq_dimid(ncid, "RHO_TOP_eta_rho", &dimid) != NC_NOERR)
00226         NC(nc_inq_dimid(ncid, "RHO_eta_rho", &dimid));
00227     NC(nc_inq_dimlen(ncid, dimid, &rs));
00228     nz = (int) rs;
00229     if (nz > NZ)
00230         ERRMSG("Too many altitudes!");
00231
00232     /* Read height... */
00233     if (nc_inq_varid(ncid, "STASH_m0ls15i102_2", &varid) != NC_NOERR)
00234         NC(nc_inq_varid(ncid, "STASH_m0ls15i102", &varid));
00235     NC(nc_get_var_float(ncid, varid, help));
00236     for (ilon = 0; ilon < nlon; ilon++)
00237         for (ilat = 0; ilat < nlat; ilat++)
00238             for (iz = 0; iz < nz; iz++)
00239                 z[ilon][ilat][iz] =
00240                     (float) (help[(iz * nlat + ilat) * nlon + ilon] / 1e3);
00241
00242     /* Read temperature... */
00243     NC(nc_inq_varid(ncid, "STASH_m0ls30i004", &varid));
00244     NC(nc_get_var_float(ncid, varid, help));
00245     for (ilon = 0; ilon < nlon; ilon++)
00246         for (ilat = 0; ilat < nlat; ilat++)
00247             for (iz = 0; iz < nz; iz++)
00248                 t[ilon][ilat][iz] = help[(iz * nlat + ilat) * nlon + ilon];
00249
00250     /* Read pressure... */
00251     NC(nc_inq_varid(ncid, "STASH_m0ls00i407", &varid));
00252     NC(nc_get_var_float(ncid, varid, help));
00253     for (ilon = 0; ilon < nlon; ilon++)
00254         for (ilat = 0; ilat < nlat; ilat++)
00255             for (iz = 0; iz < nz; iz++)
00256                 p[ilon][ilat][iz] = 0.01f * help[(iz * nlat + ilat) * nlon + ilon];
00257 }
00258
00259 /* Read WRF data... */
00260 else if (strcasecmp(argv[2], "wrf") == 0) {
00261
00262     /* Get height levels... */
00263     NC(nc_inq_dimid(ncid, "bottom_top", &dimid));
00264     NC(nc_inq_dimlen(ncid, dimid, &rs));
00265     nz = (int) rs;
00266     if (nz > NZ)
00267         ERRMSG("Too many altitudes!");
00268
00269     /* Read height... */
00270     NC(nc_inq_varid(ncid, "z", &varid));
00271     NC(nc_get_var_float(ncid, varid, help));
00272     for (ilon = 0; ilon < nlon; ilon++)
00273         for (ilat = 0; ilat < nlat; ilat++)
00274             for (iz = 0; iz < nz; iz++)
00275                 z[ilon][ilat][iz] =
00276                     (float) (help[(iz * nlat + ilat) * nlon + ilon] / 1e3);

```

```

00277
00278     /* Read temperature... */
00279     NC(nc_inq_varid(ncid, "tk", &varid));
00280     NC(nc_get_var_float(ncid, varid, help));
00281     for (ilon = 0; ilon < nlon; ilon++)
00282         for (ilat = 0; ilat < nlat; ilat++)
00283             for (iz = 0; iz < nz; iz++)
00284                 t[ilon][ilat][iz] = help[(iz * nlat + ilat) * nlon + ilon];
00285
00286     /* Read pressure... */
00287     NC(nc_inq_varid(ncid, "p", &varid));
00288     NC(nc_get_var_float(ncid, varid, help));
00289     for (ilon = 0; ilon < nlon; ilon++)
00290         for (ilat = 0; ilat < nlat; ilat++)
00291             for (iz = 0; iz < nz; iz++)
00292                 p[ilon][ilat][iz] =
00293                     (float) (help[(iz * nlat + ilat) * nlon + ilon] / 1e2);
00294 }
00295
00296 else
00297     ERRMSG("Model type not supported!");
00298
00299 /* Close file... */
00300 NC(nc_close(ncid));
00301
00302 /* Free... */
00303 free(help);
00304
00305 /* Check data... */
00306 for (ilon = 0; ilon < nlon; ilon++)
00307     for (ilat = 0; ilat < nlat; ilat++)
00308         for (iz = 0; iz < nz; iz++)
00309             if (t[ilon][ilat][iz] <= 100 || t[ilon][ilat][iz] >= 400) {
00310                 p[ilon][ilat][iz] = GSL_NAN;
00311                 t[ilon][ilat][iz] = GSL_NAN;
00312                 z[ilon][ilat][iz] = GSL_NAN;
00313             }
00314
00315 /* Smoothing of model data... */
00316 smooth(p, t, z, lon, lat, nz, nlon, nlat);
00317
00318 /* Write info... */
00319 for (iz = 0; iz < nz; iz++)
00320     printf("section_height: %d %g %g %g %g %g\n", iz,
00321           z[nlon / 2][nlat / 2][iz], lon[nlon / 2], lat[nlat / 2],
00322           p[nlon / 2][nlat / 2][iz], t[nlon / 2][nlat / 2][iz]);
00323 for (ilon = 0; ilon < nlon; ilon++)
00324     printf("section_west_east: %d %g %g %g %g %g\n", ilon,
00325           z[ilon][nlat / 2][nz / 2], lon[ilon], lat[nlat / 2],
00326           p[ilon][nlat / 2][nz / 2], t[ilon][nlat / 2][nz / 2]);
00327 for (ilat = 0; ilat < nlat; ilat++)
00328     printf("section_north_south: %d %g %g %g %g %g\n", ilat,
00329           z[nlon / 2][ilat][nz / 2], lon[nlon / 2], lat[ilat],
00330           p[nlon / 2][ilat][nz / 2], t[nlon / 2][ilat][nz / 2]);
00331
00332 /* -----
00333     Read AIRS perturbation data...
00334     ----- */
00335
00336 /* Allocate... */
00337 ALLOC(atm, atm_t, 1);
00338 ALLOC(obs, obs_t, 1);
00339 ALLOC(pert, pert_t, 1);
00340 ALLOC(wave, wave_t, 1);
00341
00342 /* Read perturbation data... */
00343 read_pert(argv[4], pertname, pert);
00344
00345 /* Find track range... */
00346 for (itrack = 0; itrack < pert->ntrack; itrack++) {
00347     if (pert->time[itrack][44] < t_ovp - 720 || itrack == 0)
00348         track0 = itrack;
00349     track1 = itrack;
00350     if (pert->time[itrack][44] > t_ovp + 720)
00351         break;
00352 }
00353
00354 /* Convert to wave analysis struct... */
00355 pert2wave(pert, wave, track0, track1, 0, pert->ntrack - 1);
00356
00357 /* Estimate background... */
00358 background_poly(wave, 5, 0);
00359
00360 /* Compute variance... */
00361 variance(wave, var_dh);
00362
00363 /* Write observation wave struct... */

```

```

00364 write_wave(argv[5], wave);
00365 write_nc(argv[7], wave);
00366
00367 /* -----
00368 Run forward model...
00369 ----- */
00370
00371 /* Loop over AIRS geolocations... */
00372 for (itrack = track0; itrack <= track1; itrack++)
00373     for (ixtrack = 0; ixtrack < pert->nxtrack; ixtrack++) {
00374
00375         /* Write info... */
00376         if (ixtrack == 0)
00377             printf("Compute track %d / %d ...\\n", itrack - track0 + 1,
00378                   track1 - track0 + 1);
00379
00380         /* Set observation data... */
00381         obs->nr = 1;
00382         obs->obsz[0] = 705;
00383         obs->obslon[0] = pert->lon[itrack][44];
00384         obs->obslat[0] = pert->lat[itrack][44];
00385         obs->vpz[0] = 0;
00386         obs->vplon[0] = pert->lon[itrack][ixtrack];
00387         obs->vplat[0] = pert->lat[itrack][ixtrack];
00388
00389         /* Get Cartesian coordinates... */
00390         geo2cart(obs->obsz[0], obs->obslon[0], obs->obslat[0], xo);
00391         geo2cart(obs->vpz[0], obs->vplon[0], obs->vplat[0], xs);
00392
00393         /* Set profile for atmospheric data... */
00394         if (slant) {
00395             atm->np = 0;
00396             for (f = 0.0; f <= 1.0; f += 0.0002) {
00397                 xm[0] = f * xo[0] + (1 - f) * xs[0];
00398                 xm[1] = f * xo[1] + (1 - f) * xs[1];
00399                 xm[2] = f * xo[2] + (1 - f) * xs[2];
00400                 cart2geo(xm, &atm->z[atm->np], &atm->lon[atm->np],
00401                        &atm->lat[atm->np]);
00402                 atm->time[atm->np] = pert->time[itrack][ixtrack];
00403                 if (atm->z[atm->np] < 10)
00404                     continue;
00405                 else if (atm->z[atm->np] > 90)
00406                     break;
00407                 else if ((++atm->np) >= NP)
00408                     ERRMSG("Too many altitudes!");
00409             }
00410         } else {
00411             atm->np = 0;
00412             for (f = 10.0; f <= 90.0; f += 0.2) {
00413                 atm->time[atm->np] = pert->time[itrack][ixtrack];
00414                 atm->z[atm->np] = f;
00415                 atm->lon[atm->np] = pert->lon[itrack][ixtrack];
00416                 atm->lat[atm->np] = pert->lat[itrack][ixtrack];
00417                 if ((++atm->np) >= NP)
00418                     ERRMSG("Too many altitudes!");
00419             }
00420         }
00421
00422         /* Initialize with climatological data... */
00423         climatology(&ctl, atm);
00424
00425         /* Interpolate model data... */
00426         for (ip = 0; ip < atm->np; ip++)
00427             intpol(p, t, z, lon, lat, nz, nlon, nlat, atm->z[ip],
00428                  atm->lon[ip], atm->lat[ip], &atm->p[ip], &atm->t[ip]);
00429
00430         /* Check profile... */
00431         okay = 1;
00432         for (ip = 0; ip < atm->np; ip++)
00433             if (!gsl_finite(atm->p[ip]) || !gsl_finite(atm->t[ip]))
00434                 okay = 0;
00435         if (!okay)
00436             pert->bt[itrack][ixtrack] = GSL_NAN;
00437         else {
00438
00439             /* Use kernel function... */
00440             if (kernel[0] != '-') {
00441
00442                 /* Read kernel function... */
00443                 if (!init) {
00444                     init = 1;
00445                     read_shape(kernel, kz, kw, &nk);
00446                     if (kz[0] > kz[1])
00447                         ERRMSG("Kernel function must be ascending!");
00448                 }
00449
00450                 /* Calculate mean temperature... */

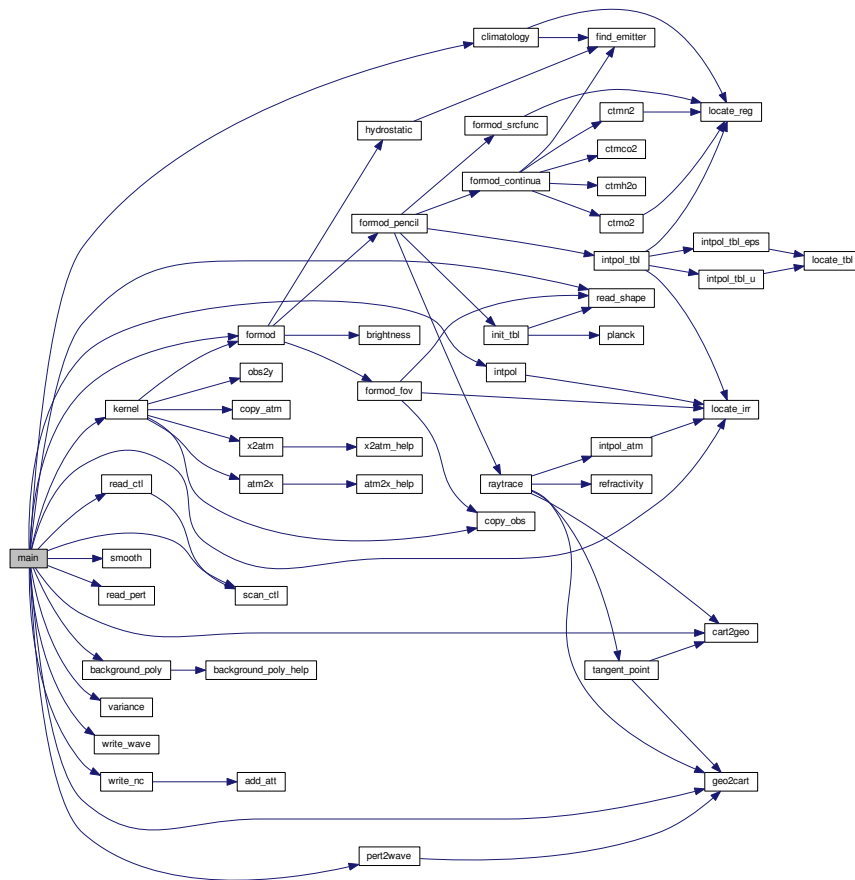
```

```

00451     pert->bt[itrack][ixtrack] = wsum = 0;
00452     for (ip = 0; ip < atm->np; ip++)
00453     {
00454         if (atm->z[ip] >= kz[0] && atm->z[ip] <= kz[nk - 1]) {
00455             iz = locate_irr(kz, nk, atm->z[ip]);
00456             w = LIN(kz[iz], kw[iz], kz[iz + 1], kw[iz + 1], atm->z[ip]);
00457             pert->bt[itrack][ixtrack] += w * atm->t[ip];
00458             wsum += w;
00459         }
00460     }
00461     pert->bt[itrack][ixtrack] /= wsum;
00462 }
00463
00464 /* Use radiative transfer model... */
00465 else {
00466     /* Run forward model... */
00467     formod(&ctl, atm, obs);
00468
00469     /* Get mean brightness temperature... */
00470     pert->bt[itrack][ixtrack] = 0;
00471     for (id = 0; id < ctl.nd; id++)
00472     {
00473         pert->bt[itrack][ixtrack] += obs->rad[id][0] / ctl.nd;
00474     }
00475 }
00476
00477 /* -----
00478    Write model perturbations...
00479    ----- */
00480
00481 /* Convert to wave analysis struct... */
00482 pert2wave(pert, wave, track0, track1, 0, pert->ntrack - 1);
00483
00484 /* Estimate background... */
00485 background_poly(wave, 5, 0);
00486
00487 /* Compute variance... */
00488 variance(wave, var_dh);
00489
00490 /* Write observation wave struct... */
00491 write_wave(argv[6], wave);
00492 write_nc(argv[8], wave);
00493
00494 /* Free... */
00495 free(atm);
00496 free(obs);
00497 free(pert);
00498 free(wave);
00499
00500 return EXIT_SUCCESS;
00501 }

```

Here is the call graph for this function:



## 5.20 issifm.c

```

00001 #include "libairs.h"
00002
00003 /* -----
00004     Dimensions...
00005     ----- */
00006
00008 #define NZ 248
00009
00011 #define NLON 3000
00012
00014 #define NLAT 1208
00015
00016 /* -----
00017     Functions...
00018     ----- */
00019
00021 void intpol(
00022     float ps[NLON][NLAT][NZ],
00023     float ts[NLON][NLAT][NZ],
00024     float zs[NLON][NLAT][NZ],
00025     double lons[NLON],
00026     double lats[NLAT],
00027     int nz,
00028     int nlon,
00029     int nlat,
00030     double z,
00031     double lon,
00032     double lat,
00033     double *p,
00034     double *t);
00035

```

```

00037 void smooth(
00038     float ps[NLON][NLAT][NZ],
00039     float ts[NLON][NLAT][NZ],
00040     float zs[NLON][NLAT][NZ],
00041     double lons[NLON],
00042     double lats[NLAT],
00043     int nz,
00044     int nlon,
00045     int nlat);
00046
00048 void write_nc(
00049     char *filename,
00050     wave_t * wave);
00051
00052 /* -----
00053     Main...
00054     ----- */
00055
00056 int main(
00057     int argc,
00058     char *argv[]) {
00059
00060     static ctl_t ctl;
00061
00062     static char kernel[LEN], pertname[LEN];
00063
00064     static double lon[NLON], lat[NLAT], xo[3], xs[3], xm[3], var_dh = 100.,
00065         f, t_ovp, hyam[NZ], hybm[NZ], kz[NSHAPE], kw[NSHAPE], w, wsum;
00066
00067     static float *help, ps[NLON][NLAT][NZ], p[NLON][NLAT][NZ], t[NLON][NLAT][NZ],
00068         z[NLON][NLAT][NZ];
00069
00070     static int init, id, itrack, ixtrack, ncid, dimid, varid, slant,
00071         ilon, ilat, iz, nlon, nlat, nz, ip, track0, track1, nk, okay;
00072
00073     static size_t rs;
00074
00075     atm_t *atm;
00076
00077     obs_t *obs;
00078
00079     pert_t *pert;
00080
00081     wave_t *wave;
00082
00083     /* -----
00084         Get control parameters...
00085         ----- */
00086
00087     /* Check arguments... */
00088     if (argc < 8)
00089         ERRMSG("Give parameters: <ctl> <model> <model.nc> <pert.nc>"
00090             " <wave_airs.tab> <wave_model.tab> <wave_airs.nc> <wave_model.nc>");
00091
00092     /* Read control parameters... */
00093     read_ctl(argc, argv, &ctl);
00094     scan_ctl(argc, argv, "PERTNAME", -1, "4mu", pertname);
00095     scan_ctl(argc, argv, "KERNEL", -1, "-", kernel);
00096     slant = (int) scan_ctl(argc, argv, "SLANT", -1, "1", NULL);
00097     t_ovp = scan_ctl(argc, argv, "T_OVP", -1, "", NULL);
00098
00099     /* Set control parameters... */
00100     ctl.write_bbt = 1;
00101
00102     /* -----
00103         Read model data...
00104         ----- */
00105
00106     /* Allocate... */
00107     ALLOC(help, float,
00108         NLON * NLAT * NZ);
00109
00110     /* Open file... */
00111     printf("Read %s data: %s\n", argv[2], argv[3]);
00112     NC(nc_open(argv[3], NC_NOWRITE, &ncid));
00113
00114     /* Read latitudes... */
00115     if (nc_inq_dimid(ncid, "lat", &dimid) != NC_NOERR)
00116         NC(nc_inq_dimid(ncid, "latitude", &dimid));
00117     NC(nc_inq_dimlen(ncid, dimid, &rs));
00118     nlat = (int) rs;
00119     if (nlat > NLAT)
00120         ERRMSG("Too many latitudes!");
00121     if (nc_inq_varid(ncid, "lat", &varid) != NC_NOERR)
00122         NC(nc_inq_varid(ncid, "latitude", &varid));
00123     NC(nc_get_var_double(ncid, varid, lat));
00124

```

```

00125  /* Read longitudes... */
00126  if (nc_inq_dimid(ncid, "lon", &dimid) != NC_NOERR)
00127      NC(nc_inq_dimid(ncid, "longitude", &dimid));
00128  NC(nc_inq_dimlen(ncid, dimid, &rs));
00129  nlon = (int) rs;
00130  if (nlon > Nlon)
00131      ERRMSG("Too many longitudes!");
00132  if (nc_inq_varid(ncid, "lon", &varid))
00133      NC(nc_inq_varid(ncid, "longitude", &varid));
00134  NC(nc_get_var_double(ncid, varid, lon));
00135
00136  /* Read ICON data... */
00137  if (strcasecmp(argv[2], "icon") == 0) {
00138
00139      /* Get height levels... */
00140      NC(nc_inq_dimid(ncid, "height", &dimid));
00141      NC(nc_inq_dimlen(ncid, dimid, &rs));
00142      nz = (int) rs;
00143      if (nz > NZ)
00144          ERRMSG("Too many altitudes!");
00145
00146      /* Read height... */
00147      NC(nc_inq_varid(ncid, "z_mc", &varid));
00148      NC(nc_get_var_float(ncid, varid, help));
00149      for (ilon = 0; ilon < nlon; ilon++)
00150          for (ilat = 0; ilat < nlat; ilat++)
00151              for (iz = 0; iz < nz; iz++)
00152                  z[ilon][ilat][iz] =
00153                      (float) (help[(iz * nlat + ilat) * nlon + ilon] / 1e3);
00154
00155      /* Read temperature... */
00156      NC(nc_inq_varid(ncid, "temp", &varid));
00157      NC(nc_get_var_float(ncid, varid, help));
00158      for (ilon = 0; ilon < nlon; ilon++)
00159          for (ilat = 0; ilat < nlat; ilat++)
00160              for (iz = 0; iz < nz; iz++)
00161                  t[ilon][ilat][iz] = help[(iz * nlat + ilat) * nlon + ilon];
00162
00163      /* Read pressure... */
00164      NC(nc_inq_varid(ncid, "pres", &varid));
00165      NC(nc_get_var_float(ncid, varid, help));
00166      for (ilon = 0; ilon < nlon; ilon++)
00167          for (ilat = 0; ilat < nlat; ilat++)
00168              for (iz = 0; iz < nz; iz++)
00169                  p[ilon][ilat][iz] =
00170                      (float) (help[(iz * nlat + ilat) * nlon + ilon] / 1e2);
00171  }
00172
00173  /* Read IFS data... */
00174  else if (strcasecmp(argv[2], "ifs") == 0) {
00175
00176      /* Get height levels... */
00177      NC(nc_inq_dimid(ncid, "lev_2", &dimid));
00178      NC(nc_inq_dimlen(ncid, dimid, &rs));
00179      nz = (int) rs;
00180      if (nz > NZ)
00181          ERRMSG("Too many altitudes!");
00182
00183      /* Read height... */
00184      NC(nc_inq_varid(ncid, "gh", &varid));
00185      NC(nc_get_var_float(ncid, varid, help));
00186      for (ilon = 0; ilon < nlon; ilon++)
00187          for (ilat = 0; ilat < nlat; ilat++)
00188              for (iz = 0; iz < nz; iz++)
00189                  z[ilon][ilat][iz] =
00190                      (float) (help[(iz * nlat + ilat) * nlon + ilon] / 1e3);
00191
00192      /* Read temperature... */
00193      NC(nc_inq_varid(ncid, "t", &varid));
00194      NC(nc_get_var_float(ncid, varid, help));
00195      for (ilon = 0; ilon < nlon; ilon++)
00196          for (ilat = 0; ilat < nlat; ilat++)
00197              for (iz = 0; iz < nz; iz++)
00198                  t[ilon][ilat][iz] = help[(iz * nlat + ilat) * nlon + ilon];
00199
00200      /* Read surface pressure... */
00201      NC(nc_inq_varid(ncid, "lnsp", &varid));
00202      NC(nc_get_var_float(ncid, varid, help));
00203      for (ilon = 0; ilon < nlon; ilon++)
00204          for (ilat = 0; ilat < nlat; ilat++)
00205              ps[ilon][ilat] = (float) exp(help[ilat * nlon + ilon]);
00206
00207      /* Read grid coefficients... */
00208      NC(nc_inq_varid(ncid, "hyam", &varid));
00209      NC(nc_get_var_double(ncid, varid, hyam));
00210      NC(nc_inq_varid(ncid, "hybm", &varid));
00211      NC(nc_get_var_double(ncid, varid, hybm));

```



```

00212
00213     /* Calculate pressure... */
00214     for (ilon = 0; ilon < nlon; ilon++)
00215         for (ilat = 0; ilat < nlat; ilat++)
00216             for (iz = 0; iz < nz; iz++)
00217                 p[ilon][ilat][iz]
00218                     = (float) ((hyam[iz] + hybm[iz] * ps[ilon][ilat]) / 100.);
00219     }
00220
00221     /* Read UM data... */
00222     else if (strcasecmp(argv[2], "um") == 0) {
00223
00224         /* Get height levels... */
00225         if (nc_inq_dimid(ncid, "RHO_TOP_eta_rho", &dimid) != NC_NOERR)
00226             NC(nc_inq_dimid(ncid, "RHO_eta_rho", &dimid));
00227         NC(nc_inq_dimlen(ncid, dimid, &rs));
00228         nz = (int) rs;
00229         if (nz > NZ)
00230             ERRMSG("Too many altitudes!");
00231
00232         /* Read height... */
00233         if (nc_inq_varid(ncid, "STASH_m01s15i102_2", &varid) != NC_NOERR)
00234             NC(nc_inq_varid(ncid, "STASH_m01s15i102", &varid));
00235         NC(nc_get_var_float(ncid, varid, help));
00236         for (ilon = 0; ilon < nlon; ilon++)
00237             for (ilat = 0; ilat < nlat; ilat++)
00238                 for (iz = 0; iz < nz; iz++)
00239                     z[ilon][ilat][iz] =
00240                         (float) (help[(iz * nlat + ilat) * nlon + ilon] / 1e3);
00241
00242         /* Read temperature... */
00243         NC(nc_inq_varid(ncid, "STASH_m01s30i004", &varid));
00244         NC(nc_get_var_float(ncid, varid, help));
00245         for (ilon = 0; ilon < nlon; ilon++)
00246             for (ilat = 0; ilat < nlat; ilat++)
00247                 for (iz = 0; iz < nz; iz++)
00248                     t[ilon][ilat][iz] = help[(iz * nlat + ilat) * nlon + ilon];
00249
00250         /* Read pressure... */
00251         NC(nc_inq_varid(ncid, "STASH_m01s00i407", &varid));
00252         NC(nc_get_var_float(ncid, varid, help));
00253         for (ilon = 0; ilon < nlon; ilon++)
00254             for (ilat = 0; ilat < nlat; ilat++)
00255                 for (iz = 0; iz < nz; iz++)
00256                     p[ilon][ilat][iz] = 0.01f * help[(iz * nlat + ilat) * nlon + ilon];
00257     }
00258
00259     /* Read WRF data... */
00260     else if (strcasecmp(argv[2], "wrf") == 0) {
00261
00262         /* Get height levels... */
00263         NC(nc_inq_dimid(ncid, "bottom_top", &dimid));
00264         NC(nc_inq_dimlen(ncid, dimid, &rs));
00265         nz = (int) rs;
00266         if (nz > NZ)
00267             ERRMSG("Too many altitudes!");
00268
00269         /* Read height... */
00270         NC(nc_inq_varid(ncid, "z", &varid));
00271         NC(nc_get_var_float(ncid, varid, help));
00272         for (ilon = 0; ilon < nlon; ilon++)
00273             for (ilat = 0; ilat < nlat; ilat++)
00274                 for (iz = 0; iz < nz; iz++)
00275                     z[ilon][ilat][iz] =
00276                         (float) (help[(iz * nlat + ilat) * nlon + ilon] / 1e3);
00277
00278         /* Read temperature... */
00279         NC(nc_inq_varid(ncid, "tk", &varid));
00280         NC(nc_get_var_float(ncid, varid, help));
00281         for (ilon = 0; ilon < nlon; ilon++)
00282             for (ilat = 0; ilat < nlat; ilat++)
00283                 for (iz = 0; iz < nz; iz++)
00284                     t[ilon][ilat][iz] = help[(iz * nlat + ilat) * nlon + ilon];
00285
00286         /* Read pressure... */
00287         NC(nc_inq_varid(ncid, "p", &varid));
00288         NC(nc_get_var_float(ncid, varid, help));
00289         for (ilon = 0; ilon < nlon; ilon++)
00290             for (ilat = 0; ilat < nlat; ilat++)
00291                 for (iz = 0; iz < nz; iz++)
00292                     p[ilon][ilat][iz] =
00293                         (float) (help[(iz * nlat + ilat) * nlon + ilon] / 1e2);
00294     }
00295
00296     else
00297         ERRMSG("Model type not supported!");
00298

```

```

00299  /* Close file... */
00300  NC(nc_close(ncid));
00301
00302  /* Free... */
00303  free(help);
00304
00305  /* Check data... */
00306  for (ilon = 0; ilon < nlon; ilon++)
00307      for (ilat = 0; ilat < nlat; ilat++)
00308          for (iz = 0; iz < nz; iz++)
00309              if (t[ilon][ilat][iz] <= 100 || t[ilon][ilat][iz] >= 400) {
00310                  p[ilon][ilat][iz] = GSL_NAN;
00311                  t[ilon][ilat][iz] = GSL_NAN;
00312                  z[ilon][ilat][iz] = GSL_NAN;
00313              }
00314
00315  /* Smoothing of model data... */
00316  smooth(p, t, z, lon, lat, nz, nlon, nlat);
00317
00318  /* Write info... */
00319  for (iz = 0; iz < nz; iz++)
00320      printf("section_height: %d %g %g %g %g %g\n", iz,
00321             z[nlon / 2][nlat / 2][iz], lon[nlon / 2], lat[nlat / 2],
00322             p[nlon / 2][nlat / 2][iz], t[nlon / 2][nlat / 2][iz]);
00323  for (ilon = 0; ilon < nlon; ilon++)
00324      printf("section_west_east: %d %g %g %g %g %g\n", ilon,
00325             z[ilon][nlat / 2][nz / 2], lon[ilon], lat[nlat / 2],
00326             p[ilon][nlat / 2][nz / 2], t[ilon][nlat / 2][nz / 2]);
00327  for (ilat = 0; ilat < nlat; ilat++)
00328      printf("section_north_south: %d %g %g %g %g %g\n", ilat,
00329             z[nlon / 2][ilat][nz / 2], lon[nlon / 2], lat[ilat],
00330             p[nlon / 2][ilat][nz / 2], t[nlon / 2][ilat][nz / 2]);
00331
00332  /* -----
00333  Read AIRS perturbation data...
00334  ----- */
00335
00336  /* Allocate... */
00337  ALLOC(atm, atm_t, 1);
00338  ALLOC(obs, obs_t, 1);
00339  ALLOC(pert, pert_t, 1);
00340  ALLOC(wave, wave_t, 1);
00341
00342  /* Read perturbation data... */
00343  read_pert(argv[4], pertname, pert);
00344
00345  /* Find track range... */
00346  for (itrack = 0; itrack < pert->ntrack; itrack++) {
00347      if (pert->time[itrack][44] < t_ovp - 720 || itrack == 0)
00348          track0 = itrack;
00349      track1 = itrack;
00350      if (pert->time[itrack][44] > t_ovp + 720)
00351          break;
00352  }
00353
00354  /* Convert to wave analysis struct... */
00355  pert2wave(pert, wave, track0, track1, 0, pert->ntrack - 1);
00356
00357  /* Estimate background... */
00358  background_poly(wave, 5, 0);
00359
00360  /* Compute variance... */
00361  variance(wave, var_dh);
00362
00363  /* Write observation wave struct... */
00364  write_wave(argv[5], wave);
00365  write_nc(argv[7], wave);
00366
00367  /* -----
00368  Run forward model...
00369  ----- */
00370
00371  /* Loop over AIRS geolocations... */
00372  for (itrack = track0; itrack <= track1; itrack++)
00373      for (ixtrack = 0; ixtrack < pert->ntrack; ixtrack++) {
00374
00375          /* Write info... */
00376          if (ixtrack == 0)
00377              printf("Compute track %d / %d ...\\n", itrack - track0 + 1,
00378                     track1 - track0 + 1);
00379
00380          /* Set observation data... */
00381          obs->nr = 1;
00382          obs->obsz[0] = 705;
00383          obs->obslon[0] = pert->lon[itrack][44];
00384          obs->obslat[0] = pert->lat[itrack][44];
00385          obs->vpz[0] = 0;

```

```

00386     obs->vplon[0] = pert->lon[itrack][ixtrack];
00387     obs->vplat[0] = pert->lat[itrack][ixtrack];
00388
00389     /* Get Cartesian coordinates... */
00390     geo2cart(obs->obsz[0], obs->obslon[0], obs->obslat[0], xo);
00391     geo2cart(obs->vpz[0], obs->vplon[0], obs->vplat[0], xs);
00392
00393     /* Set profile for atmospheric data... */
00394     if (slant) {
00395         atm->np = 0;
00396         for (f = 0.0; f <= 1.0; f += 0.0002) {
00397             xm[0] = f * xo[0] + (1 - f) * xs[0];
00398             xm[1] = f * xo[1] + (1 - f) * xs[1];
00399             xm[2] = f * xo[2] + (1 - f) * xs[2];
00400             cart2geo(xm, &atm->z[atm->np], &atm->lon[atm->np],
00401                   &atm->lat[atm->np]);
00402             atm->time[atm->np] = pert->time[itrack][ixtrack];
00403             if (atm->z[atm->np] < 10)
00404                 continue;
00405             else if (atm->z[atm->np] > 90)
00406                 break;
00407             else if ((++atm->np) >= NP)
00408                 ERRMSG("Too many altitudes!");
00409         }
00410     } else {
00411         atm->np = 0;
00412         for (f = 10.0; f <= 90.0; f += 0.2) {
00413             atm->time[atm->np] = pert->time[itrack][ixtrack];
00414             atm->z[atm->np] = f;
00415             atm->lon[atm->np] = pert->lon[itrack][ixtrack];
00416             atm->lat[atm->np] = pert->lat[itrack][ixtrack];
00417             if ((++atm->np) >= NP)
00418                 ERRMSG("Too many altitudes!");
00419         }
00420     }
00421
00422     /* Initialize with climatological data... */
00423     climatology(&ctl, atm);
00424
00425     /* Interpolate model data... */
00426     for (ip = 0; ip < atm->np; ip++)
00427         intpol(p, t, z, lon, lat, nz, nlon, nlat, atm->z[ip],
00428              atm->lon[ip], atm->lat[ip], &atm->p[ip], &atm->t[ip]);
00429
00430     /* Check profile... */
00431     okay = 1;
00432     for (ip = 0; ip < atm->np; ip++)
00433         if (!gsl_finite(atm->p[ip]) || !gsl_finite(atm->t[ip]))
00434             okay = 0;
00435     if (!okay)
00436         pert->bt[itrack][ixtrack] = GSL_NAN;
00437     else {
00438
00439         /* Use kernel function... */
00440         if (kernel[0] != '-') {
00441
00442             /* Read kernel function... */
00443             if (!init) {
00444                 init = 1;
00445                 read_shape(kernel, kz, kw, &nk);
00446                 if (kz[0] > kz[1])
00447                     ERRMSG("Kernel function must be ascending!");
00448             }
00449
00450             /* Calculate mean temperature... */
00451             pert->bt[itrack][ixtrack] = wsum = 0;
00452             for (ip = 0; ip < atm->np; ip++)
00453                 if (atm->z[ip] >= kz[0] && atm->z[ip] <= kz[nk - 1]) {
00454                     iz = locate_irr(kz, nk, atm->z[ip]);
00455                     w = LIN(kz[iz], kw[iz], kz[iz + 1], kw[iz + 1], atm->z[ip]);
00456                     pert->bt[itrack][ixtrack] += w * atm->t[ip];
00457                     wsum += w;
00458                 }
00459             pert->bt[itrack][ixtrack] /= wsum;
00460         }
00461
00462         /* Use radiative transfer model... */
00463         else {
00464
00465             /* Run forward model... */
00466             formod(&ctl, atm, obs);
00467
00468             /* Get mean brightness temperature... */
00469             pert->bt[itrack][ixtrack] = 0;
00470             for (id = 0; id < ctl.nd; id++)
00471                 pert->bt[itrack][ixtrack] += obs->rad[id][0] / ctl.nd;
00472         }

```

```

00473     }
00474 }
00475
00476 /* -----
00477    Write model perturbations...
00478    ----- */
00479
00480 /* Convert to wave analysis struct... */
00481 pert2wave(pert, wave, track0, track1, 0, pert->ntrack - 1);
00482
00483 /* Estimate background... */
00484 background_poly(wave, 5, 0);
00485
00486 /* Compute variance... */
00487 variance(wave, var_dh);
00488
00489 /* Write observation wave struct... */
00490 write_wave(argv[6], wave);
00491 write_nc(argv[8], wave);
00492
00493 /* Free... */
00494 free(atm);
00495 free(obs);
00496 free(pert);
00497 free(wave);
00498
00499 return EXIT_SUCCESS;
00500 }
00501
00502 /*****
00503
00504 void intpol(
00505     float ps[NLON][NLAT][NZ],
00506     float ts[NLON][NLAT][NZ],
00507     float zs[NLON][NLAT][NZ],
00508     double lons[NLON],
00509     double lats[NLAT],
00510     int nz,
00511     int nlon,
00512     int nlat,
00513     double z,
00514     double lon,
00515     double lat,
00516     double *p,
00517     double *t) {
00518
00519     double p00, p01, p10, p11, t00, t01, t10, t11, zd[NZ];
00520
00521     int iz, ilon, ilat;
00522
00523     /* Adjust longitude... */
00524     if (lons[nlon - 1] > 180)
00525         if (lon < 0)
00526             lon += 360;
00527
00528     /* Check horizontal range... */
00529     if (lon < lons[0]
00530         || lon > lons[nlon - 1]
00531         || lat < GSL_MIN(lats[0], lats[nlat - 1])
00532         || lat > GSL_MAX(lats[0], lats[nlat - 1])) {
00533         *p = GSL_NAN;
00534         *t = GSL_NAN;
00535         return;
00536     }
00537
00538     /* Get indices... */
00539     ilon = locate_irr(lons, nlon, lon);
00540     ilat = locate_irr(lats, nlat, lat);
00541
00542     /* Check data... */
00543     if (!gsl_finite(zs[ilon][ilat][0])
00544         || !gsl_finite(zs[ilon][ilat][nz - 1])
00545         || !gsl_finite(zs[ilon][ilat + 1][nz - 1])
00546         || !gsl_finite(zs[ilon][ilat + 1][nz - 1])
00547         || !gsl_finite(zs[ilon + 1][ilat][nz - 1])
00548         || !gsl_finite(zs[ilon + 1][ilat][nz - 1])
00549         || !gsl_finite(zs[ilon + 1][ilat + 1][nz - 1])
00550         || !gsl_finite(zs[ilon + 1][ilat + 1][nz - 1])) {
00551         *p = GSL_NAN;
00552         *t = GSL_NAN;
00553         return;
00554     }
00555
00556     /* Check vertical range... */
00557     if (z > GSL_MAX(zs[ilon][ilat][0], zs[ilon][ilat][nz - 1])
00558         || z < GSL_MIN(zs[ilon][ilat][0], zs[ilon][ilat][nz - 1])
00559         || z > GSL_MAX(zs[ilon][ilat + 1][0], zs[ilon][ilat + 1][nz - 1])

```

```

00560     || z < GSL_MIN(zs[ilon][ilat + 1][0], zs[ilon][ilat + 1][nz - 1])
00561     || z > GSL_MAX(zs[ilon + 1][ilat][0], zs[ilon + 1][ilat][nz - 1])
00562     || z < GSL_MIN(zs[ilon + 1][ilat][0], zs[ilon + 1][ilat][nz - 1])
00563     || z > GSL_MAX(zs[ilon + 1][ilat + 1][0],
00564                    zs[ilon + 1][ilat + 1][nz - 1])
00565     || z < GSL_MIN(zs[ilon + 1][ilat + 1][0],
00566                    zs[ilon + 1][ilat + 1][nz - 1]))
00567     return;
00568
00569     /* Interpolate vertically... */
00570     for (iz = 0; iz < nz; iz++)
00571         zd[iz] = zs[ilon][ilat][iz];
00572     iz = locate_irr(zd, nz, z);
00573     p00 = LIN(zs[ilon][ilat][iz], ps[ilon][ilat][iz],
00574               zs[ilon][ilat][iz + 1], ps[ilon][ilat][iz + 1], z);
00575     t00 = LIN(zs[ilon][ilat][iz], ts[ilon][ilat][iz],
00576               zs[ilon][ilat][iz + 1], ts[ilon][ilat][iz + 1], z);
00577
00578     for (iz = 0; iz < nz; iz++)
00579         zd[iz] = zs[ilon][ilat + 1][iz];
00580     iz = locate_irr(zd, nz, z);
00581     p01 = LIN(zs[ilon][ilat + 1][iz], ps[ilon][ilat + 1][iz],
00582               zs[ilon][ilat + 1][iz + 1], ps[ilon][ilat + 1][iz + 1], z);
00583     t01 = LIN(zs[ilon][ilat + 1][iz], ts[ilon][ilat + 1][iz],
00584               zs[ilon][ilat + 1][iz + 1], ts[ilon][ilat + 1][iz + 1], z);
00585
00586     for (iz = 0; iz < nz; iz++)
00587         zd[iz] = zs[ilon + 1][ilat][iz];
00588     iz = locate_irr(zd, nz, z);
00589     p10 = LIN(zs[ilon + 1][ilat][iz], ps[ilon + 1][ilat][iz],
00590               zs[ilon + 1][ilat][iz + 1], ps[ilon + 1][ilat][iz + 1], z);
00591     t10 = LIN(zs[ilon + 1][ilat][iz], ts[ilon + 1][ilat][iz],
00592               zs[ilon + 1][ilat][iz + 1], ts[ilon + 1][ilat][iz + 1], z);
00593
00594     for (iz = 0; iz < nz; iz++)
00595         zd[iz] = zs[ilon + 1][ilat + 1][iz];
00596     iz = locate_irr(zd, nz, z);
00597     p11 = LIN(zs[ilon + 1][ilat + 1][iz], ps[ilon + 1][ilat + 1][iz],
00598               zs[ilon + 1][ilat + 1][iz + 1], ps[ilon + 1][ilat + 1][iz + 1],
00599               z);
00600     t11 = LIN(zs[ilon + 1][ilat + 1][iz], ts[ilon + 1][ilat + 1][iz],
00601               zs[ilon + 1][ilat + 1][iz + 1], ts[ilon + 1][ilat + 1][iz + 1],
00602               z);
00603
00604     /* Interpolate horizontally... */
00605     p00 = LIN(lons[ilon], p00, lons[ilon + 1], p10, lon);
00606     p11 = LIN(lons[ilon], p01, lons[ilon + 1], p11, lon);
00607     *p = LIN(lats[ilat], p00, lats[ilat + 1], p11, lat);
00608
00609     t00 = LIN(lons[ilon], t00, lons[ilon + 1], t10, lon);
00610     t11 = LIN(lons[ilon], t01, lons[ilon + 1], t11, lon);
00611     *t = LIN(lats[ilat], t00, lats[ilat + 1], t11, lat);
00612 }
00613
00614 /*****
00615 void smooth(
00616     float ps[NLON][NLAT][NZ],
00617     float ts[NLON][NLAT][NZ],
00618     float zs[NLON][NLAT][NZ],
00619     double lons[NLON],
00620     double lats[NLAT],
00621     int nz,
00622     int nlon,
00623     int nlat) {
00624
00625     static float hp[NLON][NLAT], ht[NLON][NLAT], hz[NLON][NLAT], w, wsum;
00626
00627     static double dx, dy, wx[10], wy[10];
00628
00629     int iz, ilon, ilon2, ilat, ilat2, dlon = 3, dlat = 3;
00630
00631     /* Set weights... */
00632     dy = RE * M_PI / 180. * fabs(lats[1] - lats[0]);
00633     for (ilat = 0; ilat <= dlat; ilat++)
00634         wy[ilat] = exp(-0.5 * POW2(ilat * dy * 2.35482 / 20.));
00635
00636     /* Loop over height levels... */
00637     for (iz = 0; iz < nz; iz++) {
00638
00639         /* Write info... */
00640         printf("Smoothing level %d / %d ...\n", iz + 1, nz);
00641
00642         /* Copy data... */
00643         for (ilon = 0; ilon < nlon; ilon++)
00644             for (ilat = 0; ilat < nlat; ilat++) {
00645                 hp[ilon][ilat] = ps[ilon][ilat][iz];

```

```

00647         ht[ilon][ilat] = ts[ilon][ilat][iz];
00648         hz[ilon][ilat] = zs[ilon][ilat][iz];
00649     }
00650
00651     /* Loop over latitudes... */
00652     for (ilat = 0; ilat < nlat; ilat++) {
00653
00654         /* Set weights... */
00655         dx = RE * M_PI / 180. * cos(lats[ilat] * M_PI / 180.) *
00656             fabs(lons[1] - lons[0]);
00657         for (ilon = 0; ilon <= dlon; ilon++)
00658             wx[ilon] = exp(-0.5 * POW2(ilon * dx * 2.35482 / 20.));
00659
00660         /* Loop over longitudes... */
00661         for (ilon = 0; ilon < nlon; ilon++) {
00662             wsum = 0;
00663             ps[ilon][ilat][iz] = 0;
00664             ts[ilon][ilat][iz] = 0;
00665             zs[ilon][ilat][iz] = 0;
00666             for (ilon2 = GSL_MAX(ilon - dlon, 0);
00667                 ilon2 <= GSL_MIN(ilon + dlon, nlon - 1); ilon2++)
00668                 for (ilat2 = GSL_MAX(ilat - dlat, 0);
00669                     ilat2 <= GSL_MIN(ilat + dlat, nlat - 1); ilat2++) {
00670                     w = (float) (wx[abs(ilon2 - ilon)] * wy[abs(ilat2 - ilat)]);
00671                     ps[ilon][ilat][iz] += w * hp[ilon2][ilat2];
00672                     ts[ilon][ilat][iz] += w * ht[ilon2][ilat2];
00673                     zs[ilon][ilat][iz] += w * hz[ilon2][ilat2];
00674                     wsum += w;
00675                 }
00676             ps[ilon][ilat][iz] /= wsum;
00677             ts[ilon][ilat][iz] /= wsum;
00678             zs[ilon][ilat][iz] /= wsum;
00679         }
00680     }
00681 }
00682 }
00683
00684 /*****
00685
00686 void write_nc(
00687     char *filename,
00688     wave_t * wave) {
00689
00690     static double help[WX * WY];
00691
00692     int ix, iy, ncid, dimid[10], lon_id, lat_id, bt_id, pt_id, var_id;
00693
00694     /* Create netCDF file... */
00695     NC(nc_create(filename, NC_CLOBBER, &ncid));
00696
00697     /* Set dimensions... */
00698     NC(nc_def_dim(ncid, "NTRACK", (size_t) wave->ny, &dimid[0]));
00699     NC(nc_def_dim(ncid, "NXTRACK", (size_t) wave->nx, &dimid[1]));
00700
00701     /* Add variables... */
00702     NC(nc_def_var(ncid, "lon", NC_DOUBLE, 2, dimid, &lon_id));
00703     add_att(ncid, lon_id, "deg", "Footprint longitude");
00704     NC(nc_def_var(ncid, "lat", NC_DOUBLE, 2, dimid, &lat_id));
00705     add_att(ncid, lat_id, "deg", "Footprint latitude");
00706     NC(nc_def_var(ncid, "bt", NC_FLOAT, 2, dimid, &bt_id));
00707     add_att(ncid, bt_id, "K", "brightness temperature");
00708     NC(nc_def_var(ncid, "bt_pt", NC_FLOAT, 2, dimid, &pt_id));
00709     add_att(ncid, pt_id, "K", "brightness temperature perturbation");
00710     NC(nc_def_var(ncid, "bt_var", NC_FLOAT, 2, dimid, &var_id));
00711     add_att(ncid, var_id, "K^2", "brightness temperature variance");
00712
00713     /* Leave define mode... */
00714     NC(nc_enddef(ncid));
00715
00716     /* Write data... */
00717     for (ix = 0; ix < wave->nx; ix++)
00718         for (iy = 0; iy < wave->ny; iy++)
00719             help[iy * wave->nx + ix] = wave->lon[ix][iy];
00720     NC(nc_put_var_double(ncid, lon_id, help));
00721     for (ix = 0; ix < wave->nx; ix++)
00722         for (iy = 0; iy < wave->ny; iy++)
00723             help[iy * wave->nx + ix] = wave->lat[ix][iy];
00724     NC(nc_put_var_double(ncid, lat_id, help));
00725     for (ix = 0; ix < wave->nx; ix++)
00726         for (iy = 0; iy < wave->ny; iy++)
00727             help[iy * wave->nx + ix] = wave->temp[ix][iy];
00728     NC(nc_put_var_double(ncid, bt_id, help));
00729     for (ix = 0; ix < wave->nx; ix++)
00730         for (iy = 0; iy < wave->ny; iy++)
00731             help[iy * wave->nx + ix] = wave->pt[ix][iy];
00732     NC(nc_put_var_double(ncid, pt_id, help));
00733     for (ix = 0; ix < wave->nx; ix++)

```

```

00734     for (iy = 0; iy < wave->ny; iy++)
00735         help[iy * wave->nx + ix] = wave->var[ix][iy];
00736     NC(nc_put_var_double(ncid, var_id, help));
00737
00738     /* Close file... */
00739     NC(nc_close(ncid));
00740 }

```

## 5.21 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 ctmc02 (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.*
- `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_irr` (double \*xx, int n, double x)

*Find array index for irregular grid.*
- int `locate_reg` (double \*xx, int n, double x)

*Find array index for regular grid.*
- 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.*

### 5.21.1 Detailed Description

JURASSIC library definitions.

Definition in file [jurassic.c](#).

### 5.21.2 Function Documentation

#### 5.21.2.1 `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](#).

```

00034         {
00035
00036     int ig, iw;
00037
00038     size_t n = 0;
00039
00040     /* Add pressure... */
00041     atm2x_help(atm, ctl->retp_zmin, ctl->retp_zmax,
00042               atm->p, IDXP, x, iqa, ipa, &n);
00043
00044     /* Add temperature... */
00045     atm2x_help(atm, ctl->rett_zmin, ctl->rett_zmax,
00046               atm->t, IDXT, x, iqa, ipa, &n);
00047
00048     /* Add volume mixing ratios... */
00049     for (ig = 0; ig < ctl->ng; ig++)
00050         atm2x_help(atm, ctl->retq_zmin[ig], ctl->retq_zmax[ig],
00051                   atm->q[ig], IDXQ(ig), x, iqa, ipa, &n);
00052
00053     /* Add extinction... */
00054     for (iw = 0; iw < ctl->nw; iw++)
00055         atm2x_help(atm, ctl->retk_zmin[iw], ctl->retk_zmax[iw],
00056                   atm->k[iw], IDXK(iw), x, iqa, ipa, &n);
00057
00058     return n;
00059 }

```

Here is the call graph for this function:



**5.21.2.2** `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](#).

```
00072         {
00073
00074     int ip;
00075
00076     /* Add elements to state vector... */
00077     for (ip = 0; ip < atm->np; ip++)
00078         if (atm->z[ip] >= zmin && atm->z[ip] <= zmax) {
00079             if (x != NULL)
00080                 gsl_vector_set(x, *n, value[ip]);
00081             if (iqa != NULL)
00082                 iqa[*n] = val_iqa;
00083             if (ipa != NULL)
00084                 ipa[*n] = ip;
00085             (*n)++;
00086         }
00087 }
```

**5.21.2.3** `double brightness ( double rad, double nu )`

Compute brightness temperature.

Definition at line 91 of file [jurassic.c](#).

```
00093     {
00094
00095     return C2 * nu / gsl_log1p(C1 * POW3(nu) / rad);
00096 }
```

**5.21.2.4** `void cart2geo ( double * x, double * z, double * lon, double * lat )`

Convert Cartesian coordinates to geolocation.

Definition at line 101 of file [jurassic.c](#).

```
00105     {
00106
00107     double radius;
00108
00109     radius = NORM(x);
00110     *lat = asin(x[2] / radius) * 180 / M_PI;
00111     *lon = atan2(x[1], x[0]) * 180 / M_PI;
00112     *z = radius - RE;
00113 }
```

### 5.21.2.5 void climatology ( ctl\_t\* *ctl*, atm\_t\* *atm\_mean* )

Interpolate climatological data.

Definition at line 117 of file [jurassic.c](#).

```

00119         {
00120
00121     static double z[121] = {
00122         0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19,
00123         20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37,
00124         38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55,
00125         56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73,
00126         74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91,
00127         92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107,
00128         108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 120
00129     };
00130
00131     static double pre[121] = {
00132         1017, 901.083, 796.45, 702.227, 617.614, 541.644, 473.437, 412.288,
00133         357.603, 308.96, 265.994, 228.348, 195.619, 167.351, 143.039, 122.198,
00134         104.369, 89.141, 76.1528, 65.0804, 55.641, 47.591, 40.7233, 34.8637,
00135         29.8633, 25.5956, 21.9534, 18.8445, 16.1909, 13.9258, 11.9913,
00136         10.34, 8.92988, 7.72454, 6.6924, 5.80701, 5.04654, 4.39238, 3.82902,
00137         3.34337, 2.92413, 2.56128, 2.2464, 1.97258, 1.73384, 1.52519, 1.34242,
00138         1.18197, 1.04086, 0.916546, 0.806832, 0.709875, 0.624101, 0.548176,
00139         0.480974, 0.421507, 0.368904, 0.322408, 0.281386, 0.245249, 0.213465,
00140         0.185549, 0.161072, 0.139644, 0.120913, 0.104568, 0.0903249, 0.0779269,
00141         0.0671493, 0.0577962, 0.0496902, 0.0426736, 0.0366093, 0.0313743,
00142         0.0268598, 0.0229699, 0.0196206, 0.0167399, 0.0142646, 0.0121397,
00143         0.0103181, 0.00875775, 0.00742226, 0.00628076, 0.00530519, 0.00447183,
00144         0.00376124, 0.00315632, 0.00264248, 0.00220738, 0.00184003, 0.00153095,
00145         0.00127204, 0.00105608, 0.000876652, 0.00072798, 0.00060492,
00146         0.000503201, 0.000419226, 0.000349896, 0.000292659, 0.000245421,
00147         0.000206394, 0.000174125, 0.000147441, 0.000125333, 0.000106985,
00148         9.173e-05, 7.90172e-05, 6.84172e-05, 5.95574e-05, 5.21183e-05,
00149         4.58348e-05, 4.05127e-05, 3.59987e-05, 3.21583e-05, 2.88718e-05,
00150         2.60322e-05, 2.35687e-05, 2.14263e-05, 1.95489e-05
00151     };
00152
00153     static double tem[121] = {
00154         285.14, 279.34, 273.91, 268.3, 263.24, 256.55, 250.2, 242.82, 236.17,
00155         229.87, 225.04, 221.19, 218.85, 217.19, 216.2, 215.68, 215.42, 215.55,
00156         215.92, 216.4, 216.93, 217.45, 218, 218.68, 219.39, 220.25, 221.3,
00157         222.41, 223.88, 225.42, 227.2, 229.52, 231.89, 234.51, 236.85, 239.42,
00158         241.94, 244.57, 247.36, 250.32, 253.34, 255.82, 258.27, 260.39,
00159         262.03, 263.45, 264.2, 264.78, 264.67, 264.38, 263.24, 262.03, 260.02,
00160         258.09, 255.63, 253.28, 250.43, 247.81, 245.26, 242.77, 240.38,
00161         237.94, 235.79, 233.53, 231.5, 229.53, 227.6, 225.62, 223.77, 222.06,
00162         220.33, 218.69, 217.18, 215.64, 214.13, 212.52, 210.86, 209.25,
00163         207.49, 205.81, 204.11, 202.22, 200.32, 198.39, 195.92, 193.46,
00164         190.94, 188.31, 185.82, 183.57, 181.43, 179.74, 178.64, 178.1, 178.25,
00165         178.7, 179.41, 180.67, 182.31, 184.18, 186.6, 189.53, 192.66, 196.54,
00166         201.13, 205.93, 211.73, 217.86, 225, 233.53, 242.57, 252.14, 261.48,
00167         272.97, 285.26, 299.12, 312.2, 324.17, 338.34, 352.56, 365.28
00168     };
00169
00170     static double c2h2[121] = {
00171         1.352e-09, 2.83e-10, 1.269e-10, 6.926e-11, 4.346e-11, 2.909e-11,
00172         2.014e-11, 1.363e-11, 8.71e-12, 5.237e-12, 2.718e-12, 1.375e-12,
00173         5.786e-13, 2.16e-13, 7.317e-14, 2.551e-14, 1.055e-14, 4.758e-15,
00174         2.056e-15, 7.703e-16, 2.82e-16, 1.035e-16, 4.382e-17, 1.946e-17,
00175         9.638e-18, 5.2e-18, 2.811e-18, 1.494e-18, 7.925e-19, 4.213e-19,
00176         1.998e-19, 8.78e-20, 3.877e-20, 1.728e-20, 7.743e-21, 3.536e-21,
00177         1.623e-21, 7.508e-22, 3.508e-22, 1.65e-22, 7.837e-23, 3.733e-23,
00178         1.808e-23, 8.77e-24, 4.285e-24, 2.095e-24, 1.032e-24, 5.082e-25,
00179         2.506e-25, 1.236e-25, 6.088e-26, 2.996e-26, 1.465e-26, 0, 0, 0,
00180         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
00181         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
00182         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
00183     };
00184
00185     static double c2h6[121] = {
00186         2.667e-09, 2.02e-09, 1.658e-09, 1.404e-09, 1.234e-09, 1.109e-09,
00187         1.012e-09, 9.262e-10, 8.472e-10, 7.71e-10, 6.932e-10, 6.216e-10,
00188         5.503e-10, 4.87e-10, 4.342e-10, 3.861e-10, 3.347e-10, 2.772e-10,
00189         2.209e-10, 1.672e-10, 1.197e-10, 8.536e-11, 5.783e-11, 3.846e-11,
00190         2.495e-11, 1.592e-11, 1.017e-11, 6.327e-12, 3.895e-12, 2.403e-12,
00191         1.416e-12, 8.101e-13, 4.649e-13, 2.686e-13, 1.557e-13, 9.14e-14,
00192         5.386e-14, 3.19e-14, 1.903e-14, 1.14e-14, 6.875e-15, 4.154e-15,
00193         2.538e-15, 1.553e-15, 9.548e-16, 5.872e-16, 3.63e-16, 2.244e-16,
00194         1.388e-16, 8.587e-17, 5.308e-17, 3.279e-17, 2.017e-17, 1.238e-17,
00195         7.542e-18, 4.585e-18, 2.776e-18, 1.671e-18, 9.985e-19, 5.937e-19,

```

```

00196     3.518e-19, 2.07e-19, 1.215e-19, 7.06e-20, 4.097e-20, 2.37e-20,
00197     1.363e-20, 7.802e-21, 4.441e-21, 2.523e-21, 1.424e-21, 8.015e-22,
00198     4.497e-22, 2.505e-22, 1.391e-22, 7.691e-23, 4.238e-23, 2.331e-23,
00199     1.274e-23, 6.929e-24, 3.752e-24, 2.02e-24, 1.083e-24, 5.774e-25,
00200     3.041e-25, 1.593e-25, 8.308e-26, 4.299e-26, 2.195e-26, 1.112e-26,
00201     0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
00202     0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
00203 };
00204
00205 static double ccl4[121] = {
00206     1.075e-10, 1.075e-10, 1.075e-10, 1.075e-10, 1.075e-10, 1.075e-10,
00207     1.075e-10, 1.075e-10, 1.075e-10, 1.06e-10, 1.024e-10, 9.69e-11,
00208     8.93e-11, 8.078e-11, 7.213e-11, 6.307e-11, 5.383e-11, 4.49e-11,
00209     3.609e-11, 2.705e-11, 1.935e-11, 1.385e-11, 8.35e-12, 5.485e-12,
00210     3.853e-12, 2.22e-12, 5.875e-13, 3.445e-13, 1.015e-13, 6.075e-14,
00211     4.383e-14, 2.692e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14,
00212     1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14,
00213     1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14,
00214     1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14,
00215     1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14,
00216     1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14,
00217     1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14,
00218     1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14,
00219     1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14,
00220     1e-14, 1e-14, 1e-14
00221 };
00222
00223 static double ch4[121] = {
00224     1.864e-06, 1.835e-06, 1.819e-06, 1.805e-06, 1.796e-06, 1.788e-06,
00225     1.782e-06, 1.776e-06, 1.769e-06, 1.761e-06, 1.749e-06, 1.734e-06,
00226     1.716e-06, 1.692e-06, 1.654e-06, 1.61e-06, 1.567e-06, 1.502e-06,
00227     1.433e-06, 1.371e-06, 1.323e-06, 1.277e-06, 1.232e-06, 1.188e-06,
00228     1.147e-06, 1.108e-06, 1.07e-06, 1.027e-06, 9.854e-07, 9.416e-07,
00229     8.933e-07, 8.478e-07, 7.988e-07, 7.515e-07, 7.07e-07, 6.64e-07,
00230     6.239e-07, 5.864e-07, 5.512e-07, 5.184e-07, 4.87e-07, 4.571e-07,
00231     4.296e-07, 4.04e-07, 3.802e-07, 3.578e-07, 3.383e-07, 3.203e-07,
00232     3.032e-07, 2.889e-07, 2.76e-07, 2.635e-07, 2.519e-07, 2.409e-07,
00233     2.302e-07, 2.219e-07, 2.144e-07, 2.071e-07, 1.999e-07, 1.93e-07,
00234     1.862e-07, 1.795e-07, 1.731e-07, 1.668e-07, 1.607e-07, 1.548e-07,
00235     1.49e-07, 1.434e-07, 1.38e-07, 1.328e-07, 1.277e-07, 1.227e-07,
00236     1.18e-07, 1.134e-07, 1.089e-07, 1.046e-07, 1.004e-07, 9.635e-08,
00237     9.245e-08, 8.867e-08, 8.502e-08, 8.15e-08, 7.809e-08, 7.48e-08,
00238     7.159e-08, 6.849e-08, 6.55e-08, 6.262e-08, 5.98e-08, 5.708e-08,
00239     5.448e-08, 5.194e-08, 4.951e-08, 4.72e-08, 4.5e-08, 4.291e-08,
00240     4.093e-08, 3.905e-08, 3.729e-08, 3.563e-08, 3.408e-08, 3.265e-08,
00241     3.128e-08, 2.996e-08, 2.87e-08, 2.76e-08, 2.657e-08, 2.558e-08,
00242     2.467e-08, 2.385e-08, 2.307e-08, 2.234e-08, 2.168e-08, 2.108e-08,
00243     2.05e-08, 1.998e-08, 1.947e-08, 1.902e-08, 1.86e-08, 1.819e-08,
00244     1.782e-08
00245 };
00246
00247 static double clo[121] = {
00248     7.419e-15, 1.061e-14, 1.518e-14, 2.195e-14, 3.175e-14, 4.666e-14,
00249     6.872e-14, 1.03e-13, 1.553e-13, 2.375e-13, 3.664e-13, 5.684e-13,
00250     8.915e-13, 1.402e-12, 2.269e-12, 4.125e-12, 7.501e-12, 1.257e-11,
00251     2.048e-11, 3.338e-11, 5.44e-11, 8.846e-11, 1.008e-10, 1.082e-10,
00252     1.157e-10, 1.232e-10, 1.312e-10, 1.539e-10, 1.822e-10, 2.118e-10,
00253     2.387e-10, 2.687e-10, 2.875e-10, 3.031e-10, 3.23e-10, 3.648e-10,
00254     4.117e-10, 4.477e-10, 4.633e-10, 4.794e-10, 4.95e-10, 5.104e-10,
00255     5.259e-10, 5.062e-10, 4.742e-10, 4.443e-10, 4.051e-10, 3.659e-10,
00256     3.305e-10, 2.911e-10, 2.54e-10, 2.215e-10, 1.927e-10, 1.675e-10,
00257     1.452e-10, 1.259e-10, 1.09e-10, 9.416e-11, 8.119e-11, 6.991e-11,
00258     6.015e-11, 5.163e-11, 4.43e-11, 3.789e-11, 3.24e-11, 2.769e-11,
00259     2.361e-11, 2.011e-11, 1.71e-11, 1.453e-11, 1.233e-11, 1.045e-11,
00260     8.851e-12, 7.48e-12, 6.316e-12, 5.326e-12, 4.487e-12, 3.778e-12,
00261     3.176e-12, 2.665e-12, 2.234e-12, 1.87e-12, 1.563e-12, 1.304e-12,
00262     1.085e-12, 9.007e-13, 7.468e-13, 6.179e-13, 5.092e-13, 4.188e-13,
00263     3.442e-13, 2.816e-13, 2.304e-13, 1.885e-13, 1.542e-13, 1.263e-13,
00264     1.035e-13, 8.5e-14, 7.004e-14, 5.783e-14, 4.795e-14, 4.007e-14,
00265     3.345e-14, 2.792e-14, 2.33e-14, 1.978e-14, 1.686e-14, 1.438e-14,
00266     1.234e-14, 1.07e-14, 9.312e-15, 8.131e-15, 7.164e-15, 6.367e-15,
00267     5.67e-15, 5.088e-15, 4.565e-15, 4.138e-15, 3.769e-15, 3.432e-15,
00268     3.148e-15
00269 };
00270
00271 static double clono2[121] = {
00272     1.011e-13, 1.515e-13, 2.272e-13, 3.446e-13, 5.231e-13, 8.085e-13,
00273     1.253e-12, 1.979e-12, 3.149e-12, 5.092e-12, 8.312e-12, 1.366e-11,
00274     2.272e-11, 3.791e-11, 6.209e-11, 9.101e-11, 1.334e-10, 1.951e-10,
00275     2.853e-10, 3.94e-10, 4.771e-10, 5.771e-10, 6.675e-10, 7.665e-10,
00276     8.504e-10, 8.924e-10, 9.363e-10, 8.923e-10, 8.411e-10, 7.646e-10,
00277     6.525e-10, 5.576e-10, 4.398e-10, 3.403e-10, 2.612e-10, 1.915e-10,
00278     1.407e-10, 1.028e-10, 7.455e-11, 5.42e-11, 3.708e-11, 2.438e-11,
00279     1.618e-11, 1.075e-11, 7.17e-12, 4.784e-12, 3.205e-12, 2.147e-12,
00280     1.44e-12, 9.654e-13, 6.469e-13, 4.332e-13, 2.891e-13, 1.926e-13,
00281     1.274e-13, 8.422e-14, 5.547e-14, 3.636e-14, 2.368e-14, 1.536e-14,
00282     9.937e-15, 6.39e-15, 4.101e-15, 2.61e-15, 1.659e-15, 1.052e-15,

```

```
00283     6.638e-16, 4.172e-16, 2.61e-16, 1.63e-16, 1.013e-16, 6.275e-17,
00284     3.879e-17, 2.383e-17, 1.461e-17, 8.918e-18, 5.43e-18, 3.301e-18,
00285     1.997e-18, 1.203e-18, 7.216e-19, 4.311e-19, 2.564e-19, 1.519e-19,
00286     8.911e-20, 5.203e-20, 3.026e-20, 1.748e-20, 9.99e-21, 5.673e-21,
00287     3.215e-21, 1.799e-21, 1.006e-21, 5.628e-22, 3.146e-22, 1.766e-22,
00288     9.94e-23, 5.614e-23, 3.206e-23, 1.841e-23, 1.071e-23, 6.366e-24,
00289     3.776e-24, 2.238e-24, 1.326e-24, 8.253e-25, 5.201e-25, 3.279e-25,
00290     2.108e-25, 1.395e-25, 9.326e-26, 6.299e-26, 4.365e-26, 3.104e-26,
00291     2.219e-26, 1.621e-26, 1.185e-26, 8.92e-27, 6.804e-27, 5.191e-27,
00292     4.041e-27
00293 };
00294
00295 static double co[121] = {
00296     1.907e-07, 1.553e-07, 1.362e-07, 1.216e-07, 1.114e-07, 1.036e-07,
00297     9.737e-08, 9.152e-08, 8.559e-08, 7.966e-08, 7.277e-08, 6.615e-08,
00298     5.884e-08, 5.22e-08, 4.699e-08, 4.284e-08, 3.776e-08, 3.274e-08,
00299     2.845e-08, 2.479e-08, 2.246e-08, 2.054e-08, 1.991e-08, 1.951e-08,
00300     1.94e-08, 2.009e-08, 2.1e-08, 2.201e-08, 2.322e-08, 2.45e-08,
00301     2.602e-08, 2.73e-08, 2.867e-08, 2.998e-08, 3.135e-08, 3.255e-08,
00302     3.352e-08, 3.426e-08, 3.484e-08, 3.53e-08, 3.593e-08, 3.671e-08,
00303     3.759e-08, 3.945e-08, 4.192e-08, 4.49e-08, 5.03e-08, 5.703e-08,
00304     6.538e-08, 7.878e-08, 9.644e-08, 1.196e-07, 1.498e-07, 1.904e-07,
00305     2.422e-07, 3.055e-07, 3.804e-07, 4.747e-07, 5.899e-07, 7.272e-07,
00306     8.91e-07, 1.071e-06, 1.296e-06, 1.546e-06, 1.823e-06, 2.135e-06,
00307     2.44e-06, 2.714e-06, 2.967e-06, 3.189e-06, 3.391e-06, 3.58e-06,
00308     3.773e-06, 4.022e-06, 4.346e-06, 4.749e-06, 5.199e-06, 5.668e-06,
00309     6.157e-06, 6.688e-06, 7.254e-06, 7.867e-06, 8.539e-06, 9.26e-06,
00310     1.009e-05, 1.119e-05, 1.228e-05, 1.365e-05, 1.506e-05, 1.641e-05,
00311     1.784e-05, 1.952e-05, 2.132e-05, 2.323e-05, 2.531e-05, 2.754e-05,
00312     3.047e-05, 3.459e-05, 3.922e-05, 4.439e-05, 4.825e-05, 5.077e-05,
00313     5.34e-05, 5.618e-05, 5.909e-05, 6.207e-05, 6.519e-05, 6.845e-05,
00314     6.819e-05, 6.726e-05, 6.622e-05, 6.512e-05, 6.671e-05, 6.862e-05,
00315     7.048e-05, 7.264e-05, 7.3e-05, 7.3e-05, 7.3e-05, 7.3e-05, 7.3e-05,
00316 };
00317
00318 static double cof2[121] = {
00319     7.5e-14, 1.055e-13, 1.485e-13, 2.111e-13, 3.001e-13, 4.333e-13,
00320     6.269e-13, 9.221e-13, 1.364e-12, 2.046e-12, 3.093e-12, 4.703e-12,
00321     7.225e-12, 1.113e-11, 1.66e-11, 2.088e-11, 2.626e-11, 3.433e-11,
00322     4.549e-11, 5.886e-11, 7.21e-11, 8.824e-11, 1.015e-10, 1.155e-10,
00323     1.288e-10, 1.388e-10, 1.497e-10, 1.554e-10, 1.606e-10, 1.639e-10,
00324     1.64e-10, 1.64e-10, 1.596e-10, 1.542e-10, 1.482e-10, 1.382e-10,
00325     1.289e-10, 1.198e-10, 1.109e-10, 1.026e-10, 9.484e-11, 8.75e-11,
00326     8.086e-11, 7.49e-11, 6.948e-11, 6.446e-11, 5.961e-11, 5.505e-11,
00327     5.085e-11, 4.586e-11, 4.1e-11, 3.665e-11, 3.235e-11, 2.842e-11,
00328     2.491e-11, 2.11e-11, 1.769e-11, 1.479e-11, 1.197e-11, 9.631e-12,
00329     7.74e-12, 6.201e-12, 4.963e-12, 3.956e-12, 3.151e-12, 2.507e-12,
00330     1.99e-12, 1.576e-12, 1.245e-12, 9.83e-13, 7.742e-13, 6.088e-13,
00331     4.782e-13, 3.745e-13, 2.929e-13, 2.286e-13, 1.782e-13, 1.388e-13,
00332     1.079e-13, 8.362e-14, 6.471e-14, 4.996e-14, 3.85e-14, 2.96e-14,
00333     2.265e-14, 1.729e-14, 1.317e-14, 9.998e-15, 7.549e-15, 5.683e-15,
00334     4.273e-15, 3.193e-15, 2.385e-15, 1.782e-15, 1.331e-15, 9.957e-16,
00335     7.461e-16, 5.601e-16, 4.228e-16, 3.201e-16, 2.438e-16, 1.878e-16,
00336     1.445e-16, 1.111e-16, 8.544e-17, 6.734e-17, 5.341e-17, 4.237e-17,
00337     3.394e-17, 2.759e-17, 2.254e-17, 1.851e-17, 1.54e-17, 1.297e-17,
00338     1.096e-17, 9.365e-18, 8e-18, 6.938e-18, 6.056e-18, 5.287e-18,
00339     4.662e-18
00340 };
00341
00342 static double f11[121] = {
00343     2.65e-10, 2.65e-10, 2.65e-10, 2.65e-10, 2.65e-10, 2.65e-10, 2.65e-10,
00344     2.65e-10, 2.65e-10, 2.65e-10, 2.65e-10, 2.65e-10, 2.635e-10, 2.536e-10,
00345     2.44e-10, 2.348e-10, 2.258e-10, 2.153e-10, 2.046e-10, 1.929e-10,
00346     1.782e-10, 1.648e-10, 1.463e-10, 1.291e-10, 1.1e-10, 8.874e-11,
00347     7.165e-11, 5.201e-11, 3.744e-11, 2.577e-11, 1.64e-11, 1.048e-11,
00348     5.993e-12, 3.345e-12, 1.839e-12, 9.264e-13, 4.688e-13, 2.329e-13,
00349     1.129e-13, 5.505e-14, 2.825e-14, 1.492e-14, 7.997e-15, 5.384e-15,
00350     3.988e-15, 2.955e-15, 2.196e-15, 1.632e-15, 1.214e-15, 9.025e-16,
00351     6.708e-16, 4.984e-16, 3.693e-16, 2.733e-16, 2.013e-16, 1.481e-16,
00352     1.087e-16, 7.945e-17, 5.782e-17, 4.195e-17, 3.038e-17, 2.19e-17,
00353     1.577e-17, 1.128e-17, 8.063e-18, 5.753e-18, 4.09e-18, 2.899e-18,
00354     2.048e-18, 1.444e-18, 1.015e-18, 7.12e-19, 4.985e-19, 3.474e-19,
00355     2.417e-19, 1.677e-19, 1.161e-19, 8.029e-20, 5.533e-20, 3.799e-20,
00356     2.602e-20, 1.776e-20, 1.209e-20, 8.202e-21, 5.522e-21, 3.707e-21,
00357     2.48e-21, 1.652e-21, 1.091e-21, 7.174e-22, 4.709e-22, 3.063e-22,
00358     1.991e-22, 1.294e-22, 8.412e-23, 5.483e-23, 3.581e-23, 2.345e-23,
00359     1.548e-23, 1.027e-23, 6.869e-24, 4.673e-24, 3.173e-24, 2.153e-24,
00360     1.461e-24, 1.028e-24, 7.302e-25, 5.188e-25, 3.739e-25, 2.753e-25,
00361     2.043e-25, 1.528e-25, 1.164e-25, 9.041e-26, 7.051e-26, 5.587e-26,
00362     4.428e-26, 3.588e-26, 2.936e-26, 2.402e-26, 1.995e-26
00363 };
00364
00365 static double f12[121] = {
00366     5.45e-10, 5.45e-10, 5.45e-10, 5.45e-10, 5.45e-10, 5.45e-10, 5.45e-10,
00367     5.45e-10, 5.45e-10, 5.45e-10, 5.45e-10, 5.45e-10, 5.429e-10, 5.291e-10,
00368     5.155e-10, 5.022e-10, 4.893e-10, 4.772e-10, 4.655e-10, 4.497e-10,
00369     4.249e-10, 4.015e-10, 3.632e-10, 3.261e-10, 2.858e-10, 2.408e-10,
```

```
00370     2.03e-10, 1.685e-10, 1.4e-10, 1.163e-10, 9.65e-11, 8.02e-11, 6.705e-11,
00371     5.624e-11, 4.764e-11, 4.249e-11, 3.792e-11, 3.315e-11, 2.819e-11,
00372     2.4e-11, 1.999e-11, 1.64e-11, 1.352e-11, 1.14e-11, 9.714e-12,
00373     8.28e-12, 7.176e-12, 6.251e-12, 5.446e-12, 4.72e-12, 4.081e-12,
00374     3.528e-12, 3.08e-12, 2.699e-12, 2.359e-12, 2.111e-12, 1.901e-12,
00375     1.709e-12, 1.534e-12, 1.376e-12, 1.233e-12, 1.103e-12, 9.869e-13,
00376     8.808e-13, 7.859e-13, 7.008e-13, 6.241e-13, 5.553e-13, 4.935e-13,
00377     4.383e-13, 3.889e-13, 3.447e-13, 3.054e-13, 2.702e-13, 2.389e-13,
00378     2.11e-13, 1.862e-13, 1.643e-13, 1.448e-13, 1.274e-13, 1.121e-13,
00379     9.844e-14, 8.638e-14, 7.572e-14, 6.62e-14, 5.782e-14, 5.045e-14,
00380     4.394e-14, 3.817e-14, 3.311e-14, 2.87e-14, 2.48e-14, 2.142e-14,
00381     1.851e-14, 1.599e-14, 1.383e-14, 1.196e-14, 1.036e-14, 9e-15,
00382     7.828e-15, 6.829e-15, 5.992e-15, 5.254e-15, 4.606e-15, 4.037e-15,
00383     3.583e-15, 3.19e-15, 2.841e-15, 2.542e-15, 2.291e-15, 2.07e-15,
00384     1.875e-15, 1.71e-15, 1.57e-15, 1.442e-15, 1.333e-15, 1.232e-15,
00385     1.147e-15, 1.071e-15, 1.001e-15, 9.396e-16
00386 };
00387
00388 static double f14[121] = {
00389     9e-11, 9e-11, 9e-11, 9e-11, 9e-11, 9e-11, 9e-11, 9e-11, 9e-11, 9e-11,
00390     9e-11, 9e-11, 9e-11, 9e-11, 9e-11, 8.91e-11, 8.73e-11, 8.46e-11,
00391     8.19e-11, 7.92e-11, 7.74e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11,
00392     7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11,
00393     7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11,
00394     7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11,
00395     7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11,
00396     7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11,
00397     7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11,
00398     7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11,
00399     7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11,
00400     7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11,
00401     7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11,
00402     7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11,
00403     7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11,
00404     7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11,
00405     7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11
00406 };
00407
00408 static double f22[121] = {
00409     1.4e-10, 1.4e-10, 1.4e-10, 1.4e-10, 1.4e-10, 1.4e-10, 1.4e-10, 1.4e-10,
00410     1.4e-10, 1.4e-10, 1.4e-10, 1.372e-10, 1.317e-10, 1.235e-10, 1.153e-10,
00411     1.075e-10, 1.002e-10, 9.332e-11, 8.738e-11, 8.194e-11, 7.7e-11,
00412     7.165e-11, 6.753e-11, 6.341e-11, 5.971e-11, 5.6e-11, 5.229e-11,
00413     4.859e-11, 4.488e-11, 4.118e-11, 3.83e-11, 3.568e-11, 3.308e-11,
00414     3.047e-11, 2.82e-11, 2.594e-11, 2.409e-11, 2.237e-11, 2.065e-11,
00415     1.894e-11, 1.771e-11, 1.647e-11, 1.532e-11, 1.416e-11, 1.332e-11,
00416     1.246e-11, 1.161e-11, 1.087e-11, 1.017e-11, 9.471e-12, 8.853e-12,
00417     8.235e-12, 7.741e-12, 7.247e-12, 6.836e-12, 6.506e-12, 6.176e-12,
00418     5.913e-12, 5.65e-12, 5.419e-12, 5.221e-12, 5.024e-12, 4.859e-12,
00419     4.694e-12, 4.546e-12, 4.414e-12, 4.282e-12, 4.15e-12, 4.019e-12,
00420     3.903e-12, 3.805e-12, 3.706e-12, 3.607e-12, 3.508e-12, 3.41e-12,
00421     3.31e-12, 3.212e-12, 3.129e-12, 3.047e-12, 2.964e-12, 2.882e-12,
00422     2.8e-12, 2.734e-12, 2.668e-12, 2.602e-12, 2.537e-12, 2.471e-12,
00423     2.421e-12, 2.372e-12, 2.322e-12, 2.273e-12, 2.224e-12, 2.182e-12,
00424     2.141e-12, 2.1e-12, 2.059e-12, 2.018e-12, 1.977e-12, 1.935e-12,
00425     1.894e-12, 1.853e-12, 1.812e-12, 1.77e-12, 1.73e-12, 1.688e-12,
00426     1.647e-12, 1.606e-12, 1.565e-12, 1.524e-12, 1.483e-12, 1.441e-12,
00427     1.4e-12, 1.359e-12, 1.317e-12, 1.276e-12, 1.235e-12, 1.194e-12,
00428     1.153e-12, 1.112e-12, 1.071e-12, 1.029e-12, 9.883e-13
00429 };
00430
00431 static double h2o[121] = {
00432     0.01166, 0.008269, 0.005742, 0.003845, 0.00277, 0.001897, 0.001272,
00433     0.000827, 0.000539, 0.0003469, 0.0001579, 3.134e-05, 1.341e-05,
00434     6.764e-06, 4.498e-06, 3.703e-06, 3.724e-06, 3.899e-06, 4.002e-06,
00435     4.122e-06, 4.277e-06, 4.438e-06, 4.558e-06, 4.673e-06, 4.763e-06,
00436     4.809e-06, 4.856e-06, 4.936e-06, 5.021e-06, 5.114e-06, 5.222e-06,
00437     5.331e-06, 5.414e-06, 5.488e-06, 5.563e-06, 5.633e-06, 5.704e-06,
00438     5.767e-06, 5.819e-06, 5.872e-06, 5.914e-06, 5.949e-06, 5.984e-06,
00439     6.015e-06, 6.044e-06, 6.073e-06, 6.104e-06, 6.136e-06, 6.167e-06,
00440     6.189e-06, 6.208e-06, 6.226e-06, 6.212e-06, 6.185e-06, 6.158e-06,
00441     6.114e-06, 6.066e-06, 6.018e-06, 5.877e-06, 5.728e-06, 5.582e-06,
00442     5.437e-06, 5.296e-06, 5.156e-06, 5.02e-06, 4.886e-06, 4.754e-06,
00443     4.625e-06, 4.498e-06, 4.374e-06, 4.242e-06, 4.096e-06, 3.955e-06,
00444     3.817e-06, 3.683e-06, 3.491e-06, 3.204e-06, 2.94e-06, 2.696e-06,
00445     2.47e-06, 2.252e-06, 2.019e-06, 1.808e-06, 1.618e-06, 1.445e-06,
00446     1.285e-06, 1.105e-06, 9.489e-07, 8.121e-07, 6.938e-07, 5.924e-07,
00447     5.04e-07, 4.288e-07, 3.648e-07, 3.103e-07, 2.642e-07, 2.252e-07,
00448     1.921e-07, 1.643e-07, 1.408e-07, 1.211e-07, 1.048e-07, 9.063e-08,
00449     7.835e-08, 6.774e-08, 5.936e-08, 5.221e-08, 4.592e-08, 4.061e-08,
00450     3.62e-08, 3.236e-08, 2.902e-08, 2.62e-08, 2.383e-08, 2.171e-08,
00451     1.989e-08, 1.823e-08, 1.684e-08, 1.562e-08, 1.449e-08, 1.351e-08
00452 };
00453
00454 static double h2o2[121] = {
00455     1.779e-10, 7.938e-10, 8.953e-10, 8.032e-10, 6.564e-10, 5.159e-10,
00456     4.003e-10, 3.026e-10, 2.222e-10, 1.58e-10, 1.044e-10, 6.605e-11,
```

```
00457 3.413e-11, 1.453e-11, 1.062e-11, 1.009e-11, 9.597e-12, 1.175e-11,
00458 1.572e-11, 2.091e-11, 2.746e-11, 3.603e-11, 4.791e-11, 6.387e-11,
00459 8.239e-11, 1.007e-10, 1.23e-10, 1.363e-10, 1.489e-10, 1.585e-10,
00460 1.608e-10, 1.632e-10, 1.576e-10, 1.502e-10, 1.423e-10, 1.302e-10,
00461 1.192e-10, 1.085e-10, 9.795e-11, 8.854e-11, 8.057e-11, 7.36e-11,
00462 6.736e-11, 6.362e-11, 6.087e-11, 5.825e-11, 5.623e-11, 5.443e-11,
00463 5.27e-11, 5.098e-11, 4.931e-11, 4.769e-11, 4.611e-11, 4.458e-11,
00464 4.308e-11, 4.102e-11, 3.887e-11, 3.682e-11, 3.521e-11, 3.369e-11,
00465 3.224e-11, 3.082e-11, 2.946e-11, 2.814e-11, 2.687e-11, 2.566e-11,
00466 2.449e-11, 2.336e-11, 2.227e-11, 2.123e-11, 2.023e-11, 1.927e-11,
00467 1.835e-11, 1.746e-11, 1.661e-11, 1.58e-11, 1.502e-11, 1.428e-11,
00468 1.357e-11, 1.289e-11, 1.224e-11, 1.161e-11, 1.102e-11, 1.045e-11,
00469 9.895e-12, 9.369e-12, 8.866e-12, 8.386e-12, 7.922e-12, 7.479e-12,
00470 7.06e-12, 6.656e-12, 6.274e-12, 5.914e-12, 5.575e-12, 5.257e-12,
00471 4.959e-12, 4.679e-12, 4.42e-12, 4.178e-12, 3.954e-12, 3.75e-12,
00472 3.557e-12, 3.372e-12, 3.198e-12, 3.047e-12, 2.908e-12, 2.775e-12,
00473 2.653e-12, 2.544e-12, 2.442e-12, 2.346e-12, 2.26e-12, 2.183e-12,
00474 2.11e-12, 2.044e-12, 1.98e-12, 1.924e-12, 1.871e-12, 1.821e-12,
00475 1.775e-12
00476 };
00477
00478 static double hcn[121] = {
00479 5.5e-10, 5.5e-10, 5.5e-10, 5.5e-10, 5.5e-10, 5.5e-10, 5.5e-10,
00480 5.5e-10, 5.5e-10, 5.5e-10, 5.5e-10, 5.498e-10, 5.495e-10, 5.493e-10,
00481 5.49e-10, 5.488e-10, 4.717e-10, 3.946e-10, 3.174e-10, 2.4e-10,
00482 1.626e-10, 1.619e-10, 1.612e-10, 1.602e-10, 1.593e-10, 1.582e-10,
00483 1.572e-10, 1.56e-10, 1.549e-10, 1.539e-10, 1.53e-10, 1.519e-10,
00484 1.506e-10, 1.487e-10, 1.467e-10, 1.449e-10, 1.43e-10, 1.413e-10,
00485 1.397e-10, 1.382e-10, 1.368e-10, 1.354e-10, 1.337e-10, 1.315e-10,
00486 1.292e-10, 1.267e-10, 1.241e-10, 1.215e-10, 1.19e-10, 1.165e-10,
00487 1.141e-10, 1.118e-10, 1.096e-10, 1.072e-10, 1.047e-10, 1.021e-10,
00488 9.968e-11, 9.739e-11, 9.539e-11, 9.339e-11, 9.135e-11, 8.898e-11,
00489 8.664e-11, 8.439e-11, 8.249e-11, 8.075e-11, 7.904e-11, 7.735e-11,
00490 7.565e-11, 7.399e-11, 7.245e-11, 7.109e-11, 6.982e-11, 6.863e-11,
00491 6.755e-11, 6.657e-11, 6.587e-11, 6.527e-11, 6.476e-11, 6.428e-11,
00492 6.382e-11, 6.343e-11, 6.307e-11, 6.272e-11, 6.238e-11, 6.205e-11,
00493 6.17e-11, 6.137e-11, 6.102e-11, 6.072e-11, 6.046e-11, 6.03e-11,
00494 6.018e-11, 6.01e-11, 6.001e-11, 5.992e-11, 5.984e-11, 5.975e-11,
00495 5.967e-11, 5.958e-11, 5.95e-11, 5.941e-11, 5.933e-11, 5.925e-11,
00496 5.916e-11, 5.908e-11, 5.899e-11, 5.891e-11, 5.883e-11, 5.874e-11,
00497 5.866e-11, 5.858e-11, 5.85e-11, 5.841e-11, 5.833e-11, 5.825e-11,
00498 5.817e-11, 5.808e-11, 5.8e-11, 5.792e-11, 5.784e-11
00499 };
00500
00501 static double hno3[121] = {
00502 1.809e-10, 7.234e-10, 5.899e-10, 4.342e-10, 3.277e-10, 2.661e-10,
00503 2.35e-10, 2.267e-10, 2.389e-10, 2.651e-10, 3.255e-10, 4.099e-10,
00504 5.42e-10, 6.978e-10, 8.807e-10, 1.112e-09, 1.405e-09, 2.04e-09,
00505 3.111e-09, 4.5e-09, 5.762e-09, 7.37e-09, 7.852e-09, 8.109e-09,
00506 8.067e-09, 7.554e-09, 7.076e-09, 6.268e-09, 5.524e-09, 4.749e-09,
00507 3.909e-09, 3.223e-09, 2.517e-09, 1.942e-09, 1.493e-09, 1.122e-09,
00508 8.449e-10, 6.361e-10, 4.787e-10, 3.611e-10, 2.804e-10, 2.215e-10,
00509 1.758e-10, 1.441e-10, 1.197e-10, 9.953e-11, 8.505e-11, 7.334e-11,
00510 6.325e-11, 5.625e-11, 5.058e-11, 4.548e-11, 4.122e-11, 3.748e-11,
00511 3.402e-11, 3.088e-11, 2.8e-11, 2.536e-11, 2.293e-11, 2.072e-11,
00512 1.871e-11, 1.687e-11, 1.52e-11, 1.368e-11, 1.23e-11, 1.105e-11,
00513 9.922e-12, 8.898e-12, 7.972e-12, 7.139e-12, 6.385e-12, 5.708e-12,
00514 5.099e-12, 4.549e-12, 4.056e-12, 3.613e-12, 3.216e-12, 2.862e-12,
00515 2.544e-12, 2.259e-12, 2.004e-12, 1.776e-12, 1.572e-12, 1.391e-12,
00516 1.227e-12, 1.082e-12, 9.528e-13, 8.379e-13, 7.349e-13, 6.436e-13,
00517 5.634e-13, 4.917e-13, 4.291e-13, 3.745e-13, 3.267e-13, 2.854e-13,
00518 2.494e-13, 2.181e-13, 1.913e-13, 1.68e-13, 1.479e-13, 1.31e-13,
00519 1.159e-13, 1.025e-13, 9.067e-14, 8.113e-14, 7.281e-14, 6.535e-14,
00520 5.892e-14, 5.348e-14, 4.867e-14, 4.439e-14, 4.073e-14, 3.76e-14,
00521 3.476e-14, 3.229e-14, 3e-14, 2.807e-14, 2.635e-14, 2.473e-14,
00522 2.332e-14
00523 };
00524
00525 static double hno4[121] = {
00526 6.118e-12, 3.594e-12, 2.807e-12, 3.04e-12, 4.458e-12, 7.986e-12,
00527 1.509e-11, 2.661e-11, 3.738e-11, 4.652e-11, 4.429e-11, 3.992e-11,
00528 3.347e-11, 3.005e-11, 3.173e-11, 4.055e-11, 5.812e-11, 8.489e-11,
00529 1.19e-10, 1.482e-10, 1.766e-10, 2.103e-10, 2.35e-10, 2.598e-10,
00530 2.801e-10, 2.899e-10, 3e-10, 2.817e-10, 2.617e-10, 2.332e-10,
00531 1.933e-10, 1.605e-10, 1.232e-10, 9.285e-11, 6.941e-11, 4.951e-11,
00532 3.539e-11, 2.402e-11, 1.522e-11, 9.676e-12, 6.056e-12, 3.745e-12,
00533 2.34e-12, 1.463e-12, 9.186e-13, 5.769e-13, 3.322e-13, 1.853e-13,
00534 1.035e-13, 7.173e-14, 5.382e-14, 4.036e-14, 3.401e-14, 2.997e-14,
00535 2.635e-14, 2.316e-14, 2.034e-14, 1.783e-14, 1.56e-14, 1.363e-14,
00536 1.19e-14, 1.037e-14, 9.032e-15, 7.846e-15, 6.813e-15, 5.912e-15,
00537 5.121e-15, 4.431e-15, 3.829e-15, 3.306e-15, 2.851e-15, 2.456e-15,
00538 2.114e-15, 1.816e-15, 1.559e-15, 1.337e-15, 1.146e-15, 9.811e-16,
00539 8.389e-16, 7.162e-16, 6.109e-16, 5.203e-16, 4.425e-16, 3.76e-16,
00540 3.184e-16, 2.692e-16, 2.274e-16, 1.917e-16, 1.61e-16, 1.35e-16,
00541 1.131e-16, 9.437e-17, 7.874e-17, 6.57e-17, 5.481e-17, 4.579e-17,
00542 3.828e-17, 3.204e-17, 2.691e-17, 2.264e-17, 1.912e-17, 1.626e-17,
00543 1.382e-17, 1.174e-17, 9.972e-18, 8.603e-18, 7.45e-18, 6.453e-18,
```

```
00544     5.623e-18, 4.944e-18, 4.361e-18, 3.859e-18, 3.443e-18, 3.096e-18,
00545     2.788e-18, 2.528e-18, 2.293e-18, 2.099e-18, 1.929e-18, 1.773e-18,
00546     1.64e-18
00547 };
00548
00549 static double hoc1[121] = {
00550     1.056e-12, 1.194e-12, 1.35e-12, 1.531e-12, 1.737e-12, 1.982e-12,
00551     2.263e-12, 2.599e-12, 2.991e-12, 3.459e-12, 4.012e-12, 4.662e-12,
00552     5.438e-12, 6.35e-12, 7.425e-12, 8.686e-12, 1.016e-11, 1.188e-11,
00553     1.389e-11, 1.659e-11, 2.087e-11, 2.621e-11, 3.265e-11, 4.064e-11,
00554     4.859e-11, 5.441e-11, 6.09e-11, 6.373e-11, 6.611e-11, 6.94e-11,
00555     7.44e-11, 7.97e-11, 8.775e-11, 9.722e-11, 1.064e-10, 1.089e-10,
00556     1.114e-10, 1.106e-10, 1.053e-10, 1.004e-10, 9.006e-11, 7.778e-11,
00557     6.739e-11, 5.636e-11, 4.655e-11, 3.845e-11, 3.042e-11, 2.368e-11,
00558     1.845e-11, 1.442e-11, 1.127e-11, 8.814e-12, 6.544e-12, 4.763e-12,
00559     3.449e-12, 2.612e-12, 1.999e-12, 1.526e-12, 1.16e-12, 8.793e-13,
00560     6.655e-13, 5.017e-13, 3.778e-13, 2.829e-13, 2.117e-13, 1.582e-13,
00561     1.178e-13, 8.755e-14, 6.486e-14, 4.799e-14, 3.54e-14, 2.606e-14,
00562     1.916e-14, 1.403e-14, 1.026e-14, 7.48e-15, 5.446e-15, 3.961e-15,
00563     2.872e-15, 2.076e-15, 1.498e-15, 1.077e-15, 7.726e-16, 5.528e-16,
00564     3.929e-16, 2.785e-16, 1.969e-16, 1.386e-16, 9.69e-17, 6.747e-17,
00565     4.692e-17, 3.236e-17, 2.232e-17, 1.539e-17, 1.061e-17, 7.332e-18,
00566     5.076e-18, 3.522e-18, 2.461e-18, 1.726e-18, 1.22e-18, 8.75e-19,
00567     6.264e-19, 4.482e-19, 3.207e-19, 2.368e-19, 1.762e-19, 1.312e-19,
00568     9.891e-20, 7.595e-20, 5.87e-20, 4.567e-20, 3.612e-20, 2.904e-20,
00569     2.343e-20, 1.917e-20, 1.568e-20, 1.308e-20, 1.1e-20, 9.25e-21,
00570     7.881e-21
00571 };
00572
00573 static double n2o[121] = {
00574     3.17e-07, 3.17e-07, 3.17e-07, 3.17e-07, 3.17e-07, 3.17e-07, 3.17e-07,
00575     3.17e-07, 3.17e-07, 3.17e-07, 3.124e-07, 3.077e-07, 3.03e-07,
00576     2.984e-07, 2.938e-07, 2.892e-07, 2.847e-07, 2.779e-07, 2.705e-07,
00577     2.631e-07, 2.557e-07, 2.484e-07, 2.345e-07, 2.201e-07, 2.01e-07,
00578     1.754e-07, 1.532e-07, 1.329e-07, 1.154e-07, 1.003e-07, 8.735e-08,
00579     7.617e-08, 6.512e-08, 5.547e-08, 4.709e-08, 3.915e-08, 3.259e-08,
00580     2.738e-08, 2.327e-08, 1.98e-08, 1.711e-08, 1.493e-08, 1.306e-08,
00581     1.165e-08, 1.049e-08, 9.439e-09, 8.375e-09, 7.391e-09, 6.525e-09,
00582     5.759e-09, 5.083e-09, 4.485e-09, 3.953e-09, 3.601e-09, 3.27e-09,
00583     2.975e-09, 2.757e-09, 2.556e-09, 2.37e-09, 2.195e-09, 2.032e-09,
00584     1.912e-09, 1.79e-09, 1.679e-09, 1.572e-09, 1.482e-09, 1.402e-09,
00585     1.326e-09, 1.254e-09, 1.187e-09, 1.127e-09, 1.071e-09, 1.02e-09,
00586     9.673e-10, 9.193e-10, 8.752e-10, 8.379e-10, 8.017e-10, 7.66e-10,
00587     7.319e-10, 7.004e-10, 6.721e-10, 6.459e-10, 6.199e-10, 5.942e-10,
00588     5.703e-10, 5.488e-10, 5.283e-10, 5.082e-10, 4.877e-10, 4.696e-10,
00589     4.52e-10, 4.355e-10, 4.198e-10, 4.039e-10, 3.888e-10, 3.754e-10,
00590     3.624e-10, 3.499e-10, 3.381e-10, 3.267e-10, 3.163e-10, 3.058e-10,
00591     2.959e-10, 2.864e-10, 2.77e-10, 2.686e-10, 2.604e-10, 2.534e-10,
00592     2.462e-10, 2.386e-10, 2.318e-10, 2.247e-10, 2.189e-10, 2.133e-10,
00593     2.071e-10, 2.014e-10, 1.955e-10, 1.908e-10, 1.86e-10, 1.817e-10
00594 };
00595
00596 static double n2o5[121] = {
00597     1.231e-11, 3.035e-12, 1.702e-12, 9.877e-13, 8.081e-13, 9.039e-13,
00598     1.169e-12, 1.474e-12, 1.651e-12, 1.795e-12, 1.998e-12, 2.543e-12,
00599     4.398e-12, 7.698e-12, 1.28e-11, 2.131e-11, 3.548e-11, 5.894e-11,
00600     7.645e-11, 1.089e-10, 1.391e-10, 1.886e-10, 2.386e-10, 2.986e-10,
00601     3.487e-10, 3.994e-10, 4.5e-10, 4.6e-10, 4.591e-10, 4.1e-10, 3.488e-10,
00602     2.846e-10, 2.287e-10, 1.696e-10, 1.011e-10, 6.428e-11, 4.324e-11,
00603     2.225e-11, 6.214e-12, 3.608e-12, 8.793e-13, 4.491e-13, 1.04e-13,
00604     6.1e-14, 3.436e-14, 6.671e-15, 1.171e-15, 5.848e-16, 1.212e-16,
00605     1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16,
00606     1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16,
00607     1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16,
00608     1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16,
00609     1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16,
00610     1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16,
00611     1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16,
00612     1e-16, 1e-16
00613 };
00614
00615 static double nh3[121] = {
00616     1e-10, 1e-10, 1e-10, 1e-10, 1e-10, 1e-10, 1e-10, 1e-10, 1e-10, 1e-10,
00617     1e-10, 1e-10, 1e-10, 9.444e-11, 8.488e-11, 7.241e-11, 5.785e-11,
00618     4.178e-11, 3.018e-11, 2.18e-11, 1.574e-11, 1.137e-11, 8.211e-12,
00619     5.973e-12, 4.327e-12, 3.118e-12, 2.234e-12, 1.573e-12, 1.04e-12,
00620     6.762e-13, 4.202e-13, 2.406e-13, 1.335e-13, 6.938e-14, 3.105e-14,
00621     1.609e-14, 1.033e-14, 6.432e-15, 4.031e-15, 2.555e-15, 1.656e-15,
00622     1.115e-15, 7.904e-16, 5.63e-16, 4.048e-16, 2.876e-16, 2.004e-16,
00623     1.356e-16, 9.237e-17, 6.235e-17, 4.223e-17, 3.009e-17, 2.328e-17,
00624     2.002e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17,
00625     1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17,
00626     1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17,
00627     1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17,
00628     1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17,
00629     1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17,
00630     1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17,
```



```
00631    1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17,
00632    1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17,
00633    1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17,
00634    1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17,
00635    1.914e-17
00636 };
00637
00638 static double no[121] = {
00639    2.586e-10, 4.143e-11, 1.566e-11, 9.591e-12, 8.088e-12, 8.462e-12,
00640    1.013e-11, 1.328e-11, 1.855e-11, 2.678e-11, 3.926e-11, 5.464e-11,
00641    7.012e-11, 8.912e-11, 1.127e-10, 1.347e-10, 1.498e-10, 1.544e-10,
00642    1.602e-10, 1.824e-10, 2.078e-10, 2.366e-10, 2.691e-10, 5.141e-10,
00643    8.259e-10, 1.254e-09, 1.849e-09, 2.473e-09, 3.294e-09, 4.16e-09,
00644    5.095e-09, 6.11e-09, 6.93e-09, 7.888e-09, 8.903e-09, 9.713e-09,
00645    1.052e-08, 1.115e-08, 1.173e-08, 1.21e-08, 1.228e-08, 1.239e-08,
00646    1.231e-08, 1.213e-08, 1.192e-08, 1.138e-08, 1.085e-08, 1.008e-08,
00647    9.224e-09, 8.389e-09, 7.262e-09, 6.278e-09, 5.335e-09, 4.388e-09,
00648    3.589e-09, 2.761e-09, 2.129e-09, 1.633e-09, 1.243e-09, 9.681e-10,
00649    8.355e-10, 7.665e-10, 7.442e-10, 8.584e-10, 9.732e-10, 1.063e-09,
00650    1.163e-09, 1.286e-09, 1.472e-09, 1.707e-09, 2.032e-09, 2.474e-09,
00651    2.977e-09, 3.506e-09, 4.102e-09, 5.013e-09, 6.493e-09, 8.414e-09,
00652    1.077e-08, 1.367e-08, 1.777e-08, 2.625e-08, 3.926e-08, 5.545e-08,
00653    7.195e-08, 9.464e-08, 1.404e-07, 2.183e-07, 3.329e-07, 4.535e-07,
00654    6.158e-07, 8.187e-07, 1.075e-06, 1.422e-06, 1.979e-06, 2.71e-06,
00655    3.58e-06, 4.573e-06, 5.951e-06, 7.999e-06, 1.072e-05, 1.372e-05,
00656    1.697e-05, 2.112e-05, 2.643e-05, 3.288e-05, 3.994e-05, 4.794e-05,
00657    5.606e-05, 6.383e-05, 7.286e-05, 8.156e-05, 8.883e-05, 9.469e-05,
00658    9.848e-05, 0.0001023, 0.0001066, 0.0001115, 0.0001145, 0.0001142,
00659    0.0001133
00660 };
00661
00662 static double no2[121] = {
00663    3.036e-09, 2.945e-10, 9.982e-11, 5.069e-11, 3.485e-11, 2.982e-11,
00664    2.947e-11, 3.164e-11, 3.714e-11, 4.586e-11, 6.164e-11, 8.041e-11,
00665    9.982e-11, 1.283e-10, 1.73e-10, 2.56e-10, 3.909e-10, 5.959e-10,
00666    9.081e-10, 1.384e-09, 1.788e-09, 2.189e-09, 2.686e-09, 3.091e-09,
00667    3.49e-09, 3.796e-09, 4.2e-09, 5.103e-09, 6.005e-09, 6.3e-09, 6.706e-09,
00668    7.07e-09, 7.434e-09, 7.663e-09, 7.788e-09, 7.8e-09, 7.597e-09,
00669    7.482e-09, 7.227e-09, 6.403e-09, 5.585e-09, 4.606e-09, 3.703e-09,
00670    2.984e-09, 2.183e-09, 1.48e-09, 8.441e-10, 5.994e-10, 3.799e-10,
00671    2.751e-10, 1.927e-10, 1.507e-10, 1.102e-10, 6.971e-11, 5.839e-11,
00672    3.904e-11, 3.087e-11, 2.176e-11, 1.464e-11, 1.209e-11, 8.497e-12,
00673    6.477e-12, 4.371e-12, 2.914e-12, 2.424e-12, 1.753e-12, 1.35e-12,
00674    9.417e-13, 6.622e-13, 5.148e-13, 3.841e-13, 3.446e-13, 3.01e-13,
00675    2.551e-13, 2.151e-13, 1.829e-13, 1.64e-13, 1.475e-13, 1.352e-13,
00676    1.155e-13, 9.963e-14, 9.771e-14, 9.577e-14, 9.384e-14, 9.186e-14,
00677    9e-14, 9e-14, 9e-14, 9e-14, 9e-14, 9e-14, 9e-14, 9e-14, 9e-14, 9e-14,
00678    9e-14, 9e-14, 9e-14, 9e-14, 9e-14, 9e-14, 9e-14, 9e-14, 9e-14,
00679    9e-14, 9e-14, 9e-14, 9e-14, 9e-14, 9e-14, 9e-14, 9e-14, 9e-14,
00680    9e-14, 9e-14, 9e-14, 9e-14, 9e-14, 9e-14
00681 };
00682
00683 static double o3[121] = {
00684    2.218e-08, 3.394e-08, 3.869e-08, 4.219e-08, 4.501e-08, 4.778e-08,
00685    5.067e-08, 5.402e-08, 5.872e-08, 6.521e-08, 7.709e-08, 9.461e-08,
00686    1.269e-07, 1.853e-07, 2.723e-07, 3.964e-07, 5.773e-07, 8.2e-07,
00687    1.155e-06, 1.59e-06, 2.076e-06, 2.706e-06, 3.249e-06, 3.848e-06,
00688    4.459e-06, 4.986e-06, 5.573e-06, 5.958e-06, 6.328e-06, 6.661e-06,
00689    6.9e-06, 7.146e-06, 7.276e-06, 7.374e-06, 7.447e-06, 7.383e-06,
00690    7.321e-06, 7.161e-06, 6.879e-06, 6.611e-06, 6.216e-06, 5.765e-06,
00691    5.355e-06, 4.905e-06, 4.471e-06, 4.075e-06, 3.728e-06, 3.413e-06,
00692    3.125e-06, 2.856e-06, 2.607e-06, 2.379e-06, 2.17e-06, 1.978e-06,
00693    1.8e-06, 1.646e-06, 1.506e-06, 1.376e-06, 1.233e-06, 1.102e-06,
00694    9.839e-07, 8.771e-07, 7.814e-07, 6.947e-07, 6.102e-07, 5.228e-07,
00695    4.509e-07, 3.922e-07, 3.501e-07, 3.183e-07, 2.909e-07, 2.686e-07,
00696    2.476e-07, 2.284e-07, 2.109e-07, 2.003e-07, 2.013e-07, 2.022e-07,
00697    2.032e-07, 2.042e-07, 2.097e-07, 2.361e-07, 2.656e-07, 2.989e-07,
00698    3.37e-07, 3.826e-07, 4.489e-07, 5.26e-07, 6.189e-07, 7.312e-07,
00699    8.496e-07, 8.444e-07, 8.392e-07, 8.339e-07, 8.286e-07, 8.234e-07,
00700    8.181e-07, 8.129e-07, 8.077e-07, 8.026e-07, 6.918e-07, 5.176e-07,
00701    3.865e-07, 2.885e-07, 2.156e-07, 1.619e-07, 1.219e-07, 9.161e-08,
00702    6.972e-08, 5.399e-08, 3.498e-08, 2.111e-08, 1.322e-08, 8.482e-09,
00703    5.527e-09, 3.423e-09, 2.071e-09, 1.314e-09, 8.529e-10, 5.503e-10,
00704    3.665e-10
00705 };
00706
00707 static double ocs[121] = {
00708    6e-10, 6e-10, 6e-10, 6e-10, 6e-10, 6e-10, 6e-10, 6e-10, 6e-10, 5.997e-10,
00709    5.989e-10, 5.881e-10, 5.765e-10, 5.433e-10, 5.074e-10, 4.567e-10,
00710    4.067e-10, 3.601e-10, 3.093e-10, 2.619e-10, 2.232e-10, 1.805e-10,
00711    1.46e-10, 1.187e-10, 8.03e-11, 5.435e-11, 3.686e-11, 2.217e-11,
00712    1.341e-11, 8.756e-12, 4.511e-12, 2.37e-12, 1.264e-12, 8.28e-13,
00713    5.263e-13, 3.209e-13, 1.717e-13, 9.068e-14, 4.709e-14, 2.389e-14,
00714    1.236e-14, 1.127e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14,
00715    1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14,
00716    1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14,
00717    1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14,
```

```

00718     1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14,
00719     1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14,
00720     1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14,
00721     1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14,
00722     1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14,
00723     1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14,
00724     1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14,
00725     1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14,
00726     1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14,
00727     1.091e-14, 1.091e-14, 1.091e-14
00728 };
00729
00730 static double sf6[121] = {
00731     4.103e-12, 4.103e-12, 4.103e-12, 4.103e-12, 4.103e-12, 4.103e-12,
00732     4.103e-12, 4.103e-12, 4.103e-12, 4.087e-12, 4.064e-12, 4.023e-12,
00733     3.988e-12, 3.941e-12, 3.884e-12, 3.755e-12, 3.622e-12, 3.484e-12,
00734     3.32e-12, 3.144e-12, 2.978e-12, 2.811e-12, 2.653e-12, 2.489e-12,
00735     2.332e-12, 2.199e-12, 2.089e-12, 2.013e-12, 1.953e-12, 1.898e-12,
00736     1.859e-12, 1.826e-12, 1.798e-12, 1.776e-12, 1.757e-12, 1.742e-12,
00737     1.728e-12, 1.717e-12, 1.707e-12, 1.698e-12, 1.691e-12, 1.685e-12,
00738     1.679e-12, 1.675e-12, 1.671e-12, 1.668e-12, 1.665e-12, 1.663e-12,
00739     1.661e-12, 1.659e-12, 1.658e-12, 1.657e-12, 1.656e-12, 1.655e-12,
00740     1.654e-12, 1.653e-12, 1.653e-12, 1.652e-12, 1.652e-12, 1.652e-12,
00741     1.651e-12, 1.651e-12, 1.651e-12, 1.651e-12, 1.651e-12, 1.651e-12,
00742     1.651e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12,
00743     1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12,
00744     1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12,
00745     1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12,
00746     1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12,
00747     1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12,
00748     1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12,
00749     1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12
00750 };
00751
00752 static double so2[121] = {
00753     1e-10, 1e-10, 1e-10, 1e-10, 1e-10, 1e-10, 1e-10, 1e-10, 1e-10, 1e-10,
00754     1e-10, 1e-10, 9.867e-11, 9.537e-11, 9e-11, 8.404e-11, 7.799e-11,
00755     7.205e-11, 6.616e-11, 6.036e-11, 5.475e-11, 5.007e-11, 4.638e-11,
00756     4.346e-11, 4.055e-11, 3.763e-11, 3.471e-11, 3.186e-11, 2.905e-11,
00757     2.631e-11, 2.358e-11, 2.415e-11, 2.949e-11, 3.952e-11, 5.155e-11,
00758     6.76e-11, 8.741e-11, 1.099e-10, 1.278e-10, 1.414e-10, 1.512e-10,
00759     1.607e-10, 1.699e-10, 1.774e-10, 1.832e-10, 1.871e-10, 1.907e-10,
00760     1.943e-10, 1.974e-10, 1.993e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10,
00761     2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10,
00762     2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10,
00763     2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10,
00764     2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10,
00765     2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10,
00766     2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10,
00767     2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10
00768 };
00769
00770 static int ig_co2 = -999;
00771
00772 double co2, *q[NG] = { NULL };
00773
00774 int ig, ip, iw, iz;
00775
00776 /* Find emitter index of CO2... */
00777 if (ig_co2 == -999)
00778     ig_co2 = find_emitter(ctl, "CO2");
00779
00780 /* Identify variable... */
00781 for (ig = 0; ig < ctl->ng; ig++) {
00782     q[ig] = NULL;
00783     if (strcasecmp(ctl->emitter[ig], "C2H2") == 0)
00784         q[ig] = c2h2;
00785     if (strcasecmp(ctl->emitter[ig], "C2H6") == 0)
00786         q[ig] = c2h6;
00787     if (strcasecmp(ctl->emitter[ig], "CCl4") == 0)
00788         q[ig] = ccl4;
00789     if (strcasecmp(ctl->emitter[ig], "CH4") == 0)
00790         q[ig] = ch4;
00791     if (strcasecmp(ctl->emitter[ig], "ClO") == 0)
00792         q[ig] = clo;
00793     if (strcasecmp(ctl->emitter[ig], "ClONO2") == 0)
00794         q[ig] = clono2;
00795     if (strcasecmp(ctl->emitter[ig], "CO") == 0)
00796         q[ig] = co;
00797     if (strcasecmp(ctl->emitter[ig], "COF2") == 0)
00798         q[ig] = cof2;
00799     if (strcasecmp(ctl->emitter[ig], "F11") == 0)
00800         q[ig] = f11;
00801     if (strcasecmp(ctl->emitter[ig], "F12") == 0)
00802         q[ig] = f12;
00803     if (strcasecmp(ctl->emitter[ig], "F14") == 0)
00804         q[ig] = f14;

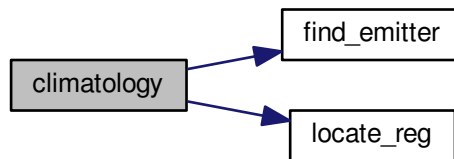
```

```

00805     if (strcasecmp(ctl->emitter[ig], "F22") == 0)
00806         q[ig] = f22;
00807     if (strcasecmp(ctl->emitter[ig], "H2O") == 0)
00808         q[ig] = h2o;
00809     if (strcasecmp(ctl->emitter[ig], "H2O2") == 0)
00810         q[ig] = h2o2;
00811     if (strcasecmp(ctl->emitter[ig], "HCN") == 0)
00812         q[ig] = hcn;
00813     if (strcasecmp(ctl->emitter[ig], "HNO3") == 0)
00814         q[ig] = hno3;
00815     if (strcasecmp(ctl->emitter[ig], "HNO4") == 0)
00816         q[ig] = hno4;
00817     if (strcasecmp(ctl->emitter[ig], "HOCl") == 0)
00818         q[ig] = hocl;
00819     if (strcasecmp(ctl->emitter[ig], "N2O") == 0)
00820         q[ig] = n2o;
00821     if (strcasecmp(ctl->emitter[ig], "N2O5") == 0)
00822         q[ig] = n2o5;
00823     if (strcasecmp(ctl->emitter[ig], "NH3") == 0)
00824         q[ig] = nh3;
00825     if (strcasecmp(ctl->emitter[ig], "NO") == 0)
00826         q[ig] = no;
00827     if (strcasecmp(ctl->emitter[ig], "NO2") == 0)
00828         q[ig] = no2;
00829     if (strcasecmp(ctl->emitter[ig], "O3") == 0)
00830         q[ig] = o3;
00831     if (strcasecmp(ctl->emitter[ig], "OCS") == 0)
00832         q[ig] = ocs;
00833     if (strcasecmp(ctl->emitter[ig], "SF6") == 0)
00834         q[ig] = sf6;
00835     if (strcasecmp(ctl->emitter[ig], "SO2") == 0)
00836         q[ig] = so2;
00837 }
00838
00839 /* Loop over atmospheric data points... */
00840 for (ip = 0; ip < atm->np; ip++) {
00841
00842     /* Get altitude index... */
00843     iz = locate_reg(z, 121, atm->z[ip]);
00844
00845     /* Interpolate pressure... */
00846     atm->p[ip] = EXP(z[iz], pre[iz], z[iz + 1], pre[iz + 1], atm->z[ip]);
00847
00848     /* Interpolate temperature... */
00849     atm->t[ip] = LIN(z[iz], tem[iz], z[iz + 1], tem[iz + 1], atm->z[ip]);
00850
00851     /* Interpolate trace gases... */
00852     for (ig = 0; ig < ctl->ng; ig++)
00853         if (q[ig] != NULL)
00854             atm->q[ig][ip] =
00855                 LIN(z[iz], q[ig][iz], z[iz + 1], q[ig][iz + 1], atm->z[ip]);
00856         else
00857             atm->q[ig][ip] = 0;
00858
00859     /* Set CO2... */
00860     if (ig_co2 >= 0) {
00861         co2 =
00862             371.789948e-6 + 2.026214e-6 * (atm->time[ip] - 63158400.) / 31557600.;
00863         atm->q[ig_co2][ip] = co2;
00864     }
00865
00866     /* Set extinction to zero... */
00867     for (iw = 0; iw < ctl->nw; iw++)
00868         atm->k[iw][ip] = 0;
00869 }
00870 }

```

Here is the call graph for this function:



#### 5.21.2.6 double ctmco2 ( double *nu*, double *p*, double *t*, double *u* )

Compute carbon dioxide continuum (optical depth).

Definition at line 874 of file [jurassic.c](#).

```

00878     {
00879
00880     static double co2296[2001] = { 9.3388e-5, 9.7711e-5, 1.0224e-4, 1.0697e-4,
00881     1.1193e-4, 1.1712e-4, 1.2255e-4, 1.2824e-4, 1.3419e-4, 1.4043e-4,
00882     1.4695e-4, 1.5378e-4, 1.6094e-4, 1.6842e-4, 1.7626e-4, 1.8447e-4,
00883     1.9307e-4, 2.0207e-4, 2.1149e-4, 2.2136e-4, 2.3169e-4, 2.4251e-4,
00884     2.5384e-4, 2.657e-4, 2.7813e-4, 2.9114e-4, 3.0477e-4, 3.1904e-4,
00885     3.3399e-4, 3.4965e-4, 3.6604e-4, 3.8322e-4, 4.0121e-4, 4.2006e-4,
00886     4.398e-4, 4.6047e-4, 4.8214e-4, 5.0483e-4, 5.286e-4, 5.535e-4,
00887     5.7959e-4, 6.0693e-4, 6.3557e-4, 6.6558e-4, 6.9702e-4, 7.2996e-4,
00888     7.6449e-4, 8.0066e-4, 8.3856e-4, 8.7829e-4, 9.1991e-4, 9.6354e-4,
00889     .0010093, .0010572, .0011074, .00116, .0012152, .001273,
00890     .0013336, .0013972, .0014638, .0015336, .0016068, .0016835,
00891     .001764, .0018483, .0019367, .0020295, .0021267, .0022286,
00892     .0023355, .0024476, .0025652, .0026885, .0028178, .0029534,
00893     .0030956, .0032448, .0034012, .0035654, .0037375, .0039181,
00894     .0041076, .0043063, .0045148, .0047336, .0049632, .005204,
00895     .0054567, .0057219, .0060002, .0062923, .0065988, .0069204,
00896     .007258, .0076123, .0079842, .0083746, .0087844, .0092146,
00897     .0096663, .01014, .010638, .011161, .01171, .012286, .012891,
00898     .013527, .014194, .014895, .015631, .016404, .017217, .01807,
00899     .018966, .019908, .020897, .021936, .023028, .024176, .025382,
00900     .026649, .027981, .02938, .030851, .032397, .034023, .035732,
00901     .037528, .039416, .041402, .04349, .045685, .047994, .050422,
00902     .052975, .055661, .058486, .061458, .064584, .067873, .071334,
00903     .074975, .078807, .082839, .087082, .091549, .096249, .1012,
00904     .10641, .11189, .11767, .12375, .13015, .13689, .14399, .15147,
00905     .15935, .16765, .17639, .18561, .19531, .20554, .21632, .22769,
00906     .23967, .25229, .2656, .27964, .29443, .31004, .3265, .34386,
00907     .36218, .3815, .40188, .42339, .44609, .47004, .49533, .52202,
00908     .5502, .57995, .61137, .64455, .6796, .71663, .75574, .79707,
00909     .84075, .88691, .9357, .98728, 1.0418, 1.0995, 1.1605, 1.225,
00910     1.2932, 1.3654, 1.4418, 1.5227, 1.6083, 1.6989, 1.7948, 1.8964,
00911     2.004, 2.118, 2.2388, 2.3668, 2.5025, 2.6463, 2.7988, 2.9606,
00912     3.1321, 3.314, 3.5071, 3.712, 3.9296, 4.1605, 4.4058, 4.6663,
00913     4.9431, 5.2374, 5.5501, 5.8818, 6.2353, 6.6114, 7.0115, 7.4372,
00914     7.8905, 8.3731, 8.8871, 9.4349, 10.019, 10.641, 11.305, 12.013,
00915     12.769, 13.576, 14.437, 15.358, 16.342, 17.39, 18.513, 19.716,
00916     21.003, 22.379, 23.854, 25.436, 27.126, 28.942, 30.89, 32.973,
00917     35.219, 37.634, 40.224, 43.021, 46.037, 49.29, 52.803, 56.447,
00918     60.418, 64.792, 69.526, 74.637, 80.182, 86.193, 92.713, 99.786,
00919     107.47, 115.84, 124.94, 134.86, 145.69, 157.49, 170.3, 184.39,
00920     199.83, 216.4, 234.55, 254.72, 276.82, 299.85, 326.16, 354.99,
00921     386.51, 416.68, 449.89, 490.12, 534.35, 578.25, 632.26, 692.61,
00922     756.43, 834.75, 924.11, 1016.9, 996.96, 1102.7, 1219.2, 1351.9,
00923     1494.3, 1654.1, 1826.5, 2027.9, 2249., 2453.8, 2714.4, 2999.4,
00924     3209.5, 3509., 3840.4, 3907.5, 4190.7, 4533.5, 4648.3, 5059.1,
00925     5561.6, 6191.4, 6820.8, 7905.9, 9362.2, 2431.3, 2211.3, 2046.8,
00926     2023.8, 1985.9, 1905.9, 1491.1, 1369.8, 1262.2, 1200.7, 887.74,
00927     820.25, 885.23, 887.21, 816.73, 1126.9, 1216.2, 1272.4, 1579.5,
00928     1634.2, 1656.3, 1657.9, 1789.5, 1670.8, 1509.5, 8474.6, 7489.2,
00929     6793.6, 6117., 5574.1, 5141.2, 5084.6, 4745.1, 4413.2, 4102.8,
  
```

00930 4024.7, 3715., 3398.6, 3100.8, 2900.4, 2629.2, 2374., 2144.7,  
00931 1955.8, 1760.8, 1591.2, 1435.2, 1296.2, 1174., 1065.1, 967.76,  
00932 999.48, 897.45, 809.23, 732.77, 670.26, 611.93, 560.11, 518.77,  
00933 476.84, 438.8, 408.48, 380.21, 349.24, 322.71, 296.65, 272.85,  
00934 251.96, 232.04, 213.88, 197.69, 182.41, 168.41, 155.79, 144.05,  
00935 133.31, 123.48, 114.5, 106.21, 98.591, 91.612, 85.156, 79.204,  
00936 73.719, 68.666, 63.975, 59.637, 56.35, 52.545, 49.042, 45.788,  
00937 42.78, 39.992, 37.441, 35.037, 32.8, 30.744, 28.801, 26.986,  
00938 25.297, 23.731, 22.258, 20.883, 19.603, 18.403, 17.295, 16.249,  
00939 15.271, 14.356, 13.501, 12.701, 11.954, 11.254, 10.6, 9.9864,  
00940 9.4118, 8.8745, 8.3714, 7.8997, 7.4578, 7.0446, 6.6573, 6.2949,  
00941 5.9577, 5.6395, 5.3419, 5.063, 4.8037, 4.5608, 4.3452, 4.1364,  
00942 3.9413, 3.7394, 3.562, 3.3932, 3.2325, 3.0789, 2.9318, 2.7898,  
00943 2.6537, 2.5225, 2.3958, 2.2305, 2.1215, 2.0245, 1.9427, 1.8795,  
00944 1.8336, 1.7604, 1.7016, 1.6419, 1.5282, 1.4611, 1.3443, 1.27,  
00945 1.1675, 1.0824, 1.0534, .99833, .95854, .92981, .90887, .89346,  
00946 .88113, .87068, .86102, .85096, .88262, .86151, .83565, .80518,  
00947 .77045, .73736, .74744, .74954, .75773, .82267, .83493, .89402,  
00948 .89725, .93426, .95564, .94045, .94174, .93404, .92035, .90456,  
00949 .88621, .86673, .78117, .7515, .72056, .68822, .65658, .62764,  
00950 .55984, .55598, .57407, .60963, .63763, .66198, .61132, .60972,  
00951 .52496, .50649, .41872, .3964, .32422, .27276, .24048, .23772,  
00952 .2286, .22711, .23999, .32038, .34371, .36621, .38561, .39953,  
00953 .40636, .44913, .42716, .3919, .35477, .33935, .3351, .39746,  
00954 .40993, .49398, .49956, .56157, .54742, .57295, .57386, .55417,  
00955 .50745, .471, .43446, .39102, .34993, .31269, .27888, .24912,  
00956 .22291, .19994, .17972, .16197, .14633, .13252, .12029, .10942,  
00957 .099745, .091118, .083404, .076494, .070292, .064716, .059697,  
00958 .055173, .051093, .047411, .044089, .041092, .038392, .035965,  
00959 .033789, .031846, .030122, .028607, .02729, .026169, .025209,  
00960 .024405, .023766, .023288, .022925, .022716, .022681, .022685,  
00961 .022768, .023133, .023325, .023486, .024004, .024126, .024083,  
00962 .023785, .024023, .023029, .021649, .021108, .019454, .017809,  
00963 .017292, .016635, .017037, .018068, .018977, .018756, .017847,  
00964 .016557, .016142, .014459, .012869, .012381, .010875, .0098701,  
00965 .009285, .0091698, .0091701, .0096145, .010553, .01106, .012613,  
00966 .014362, .015017, .016507, .017741, .01768, .017784, .0171,  
00967 .016357, .016172, .017257, .018978, .020935, .021741, .023567,  
00968 .025183, .025589, .026732, .027648, .028278, .028215, .02856,  
00969 .029015, .029062, .028851, .028497, .027825, .027801, .026523,  
00970 .02487, .022967, .022168, .020194, .018605, .017903, .018439,  
00971 .019697, .020311, .020855, .020057, .018608, .016738, .015963,  
00972 .013844, .011801, .011134, .0097573, .0086007, .0086226,  
00973 .0083721, .0090978, .0097616, .0098426, .011317, .012853, .01447,  
00974 .014657, .015771, .016351, .016079, .014829, .013431, .013185,  
00975 .013207, .01448, .016176, .017971, .018265, .019526, .020455,  
00976 .019797, .019802, .0194, .018176, .017505, .016197, .015339,  
00977 .014401, .013213, .012203, .011186, .010236, .0093288, .0084854,  
00978 .0076837, .0069375, .0062614, .0056628, .0051153, .0046015,  
00979 .0041501, .003752, .0033996, .0030865, .0028077, .0025586,  
00980 .0023355, .0021353, .0019553, .0017931, .0016466, .0015141,  
00981 .0013941, .0012852, .0011862, .0010962, .0010142, 9.3935e-4,  
00982 8.71e-4, 8.0851e-4, 7.5132e-4, 6.9894e-4, 6.5093e-4, 6.0689e-4,  
00983 5.6647e-4, 5.2935e-4, 4.9525e-4, 4.6391e-4, 4.3509e-4, 4.086e-4,  
00984 3.8424e-4, 3.6185e-4, 3.4126e-4, 3.2235e-4, 3.0498e-4, 2.8904e-4,  
00985 2.7444e-4, 2.6106e-4, 2.4883e-4, 2.3766e-4, 2.275e-4, 2.1827e-4,  
00986 2.0992e-4, 2.0239e-4, 1.9563e-4, 1.896e-4, 1.8427e-4, 1.796e-4,  
00987 1.7555e-4, 1.7209e-4, 1.692e-4, 1.6687e-4, 1.6505e-4, 1.6375e-4,  
00988 1.6294e-4, 1.6261e-4, 1.6274e-4, 1.6334e-4, 1.6438e-4, 1.6587e-4,  
00989 1.678e-4, 1.7017e-4, 1.7297e-4, 1.762e-4, 1.7988e-4, 1.8399e-4,  
00990 1.8855e-4, 1.9355e-4, 1.9902e-4, 2.0494e-4, 2.1134e-4, 2.1823e-4,  
00991 2.2561e-4, 2.335e-4, 2.4192e-4, 2.5088e-4, 2.604e-4, 2.705e-4,  
00992 2.8119e-4, 2.9251e-4, 3.0447e-4, 3.171e-4, 3.3042e-4, 3.4447e-4,  
00993 3.5927e-4, 3.7486e-4, 3.9127e-4, 4.0854e-4, 4.267e-4, 4.4579e-4,  
00994 4.6586e-4, 4.8696e-4, 5.0912e-4, 5.324e-4, 5.5685e-4, 5.8253e-4,  
00995 6.0949e-4, 6.378e-4, 6.6753e-4, 6.9873e-4, 7.3149e-4, 7.6588e-4,  
00996 8.0198e-4, 8.3987e-4, 8.7964e-4, 9.2139e-4, 9.6522e-4, .0010112,  
00997 .0010595, .0011102, .0011634, .0012193, .001278, .0013396,  
00998 .0014043, .0014722, .0015436, .0016185, .0016972, .0017799,  
00999 .0018668, .001958, .0020539, .0021547, .0022606, .0023719,  
01000 .002489, .002612, .0027414, .0028775, .0030206, .0031712,  
01001 .0033295, .0034962, .0036716, .0038563, .0040506, .0042553,  
01002 .0044709, .004698, .0049373, .0051894, .0054552, .0057354,  
01003 .006031, .0063427, .0066717, .0070188, .0073854, .0077726,  
01004 .0081816, .0086138, .0090709, .0095543, .010066, .010607,  
01005 .011181, .011789, .012433, .013116, .013842, .014613, .015432,  
01006 .016304, .017233, .018224, .019281, .020394, .021574, .022836,  
01007 .024181, .025594, .027088, .028707, .030401, .032245, .034219,  
01008 .036262, .038539, .040987, .043578, .04641, .04949, .052726,  
01009 .056326, .0602, .064093, .068521, .073278, .077734, .083064,  
01010 .088731, .093885, .1003, .1072, .11365, .12187, .13078, .13989,  
01011 .15095, .16299, .17634, .19116, .20628, .22419, .24386, .26587,  
01012 .28811, .31399, .34321, .36606, .39675, .42742, .44243, .47197,  
01013 .49993, .49027, .51147, .52803, .48931, .49729, .5026, .43854,  
01014 .441, .44766, .43414, .46151, .50029, .55247, .43855, .32115,  
01015 .32607, .3431, .36119, .38029, .41179, .43996, .47144, .51853,  
01016 .55362, .59122, .66338, .69877, .74001, .82923, .86907, .90361,

```

01017 1.0025, 1.031, 1.0559, 1.104, 1.1178, 1.1341, 1.1547, 1.351,
01018 1.4772, 1.4812, 1.4907, 1.512, 1.5442, 1.5853, 1.6358, 1.6963,
01019 1.7674, 1.8474, 1.9353, 2.0335, 2.143, 2.2592, 2.3853, 2.5217,
01020 2.6686, 2.8273, 2.9998, 3.183, 3.3868, 3.6109, 3.8564, 4.1159,
01021 4.4079, 4.7278, 5.0497, 5.3695, 5.758, 6.0834, 6.4976, 6.9312,
01022 7.38, 7.5746, 7.9833, 8.3791, 8.3956, 8.7501, 9.1067, 9.072,
01023 9.4649, 9.9112, 10.402, 10.829, 11.605, 12.54, 12.713, 10.443,
01024 10.825, 11.375, 11.955, 12.623, 13.326, 14.101, 15.041, 15.547,
01025 16.461, 17.439, 18.716, 19.84, 21.036, 22.642, 23.901, 25.244,
01026 27.03, 28.411, 29.871, 31.403, 33.147, 34.744, 36.456, 39.239,
01027 43.605, 45.162, 47.004, 49.093, 51.391, 53.946, 56.673, 59.629,
01028 63.167, 66.576, 70.254, 74.222, 78.477, 83.034, 87.914, 93.18,
01029 98.77, 104.74, 111.15, 117.95, 125.23, 133.01, 141.33, 150.21,
01030 159.71, 169.89, 180.93, 192.54, 204.99, 218.34, 232.65, 248.,
01031 264.47, 282.14, 301.13, 321.53, 343.48, 367.08, 392.5, 419.88,
01032 449.4, 481.26, 515.64, 552.79, 592.99, 636.48, 683.61, 734.65,
01033 789.99, 850.02, 915.14, 985.81, 1062.5, 1147.1, 1237.8, 1336.4,
01034 1443.2, 1558.9, 1684.2, 1819.2, 1965.2, 2122.6, 2291.7, 2470.8,
01035 2665.7, 2874.9, 3099.4, 3337.9, 3541., 3813.3, 4111.9, 4439.3,
01036 4798.9, 5196., 5639.2, 6087.5, 6657.7, 7306.7, 8040.7, 8845.5,
01037 9702.2, 10670., 11739., 12842., 14141., 15498., 17068., 18729.,
01038 20557., 22559., 25248., 27664., 30207., 32915., 35611., 38081.,
01039 40715., 43191., 41651., 42750., 43785., 44353., 44366., 44189.,
01040 43618., 42862., 41878., 35133., 35215., 36383., 39420., 44055.,
01041 44155., 45850., 46853., 39197., 38274., 29942., 28553., 21792.,
01042 21228., 17106., 14955., 18181., 19557., 21427., 23728., 26301.,
01043 28584., 30775., 32536., 33867., 40089., 39204., 37329., 34452.,
01044 31373., 33921., 34800., 36043., 44415., 45162., 52181., 50895.,
01045 54140., 50840., 50468., 48302., 44915., 40910., 36754., 32755.,
01046 29093., 25860., 22962., 20448., 18247., 16326., 14645., 13165.,
01047 11861., 10708., 9686.9, 8779.7, 7971.9, 7250.8, 6605.7, 6027.2,
01048 5507.3, 5039.1, 4616.6, 4234.8, 3889., 3575.4, 3290.5, 3031.3,
01049 2795.2, 2579.9, 2383.1, 2203.3, 2038.6, 1887.6, 1749.1, 1621.9,
01050 1505., 1397.4, 1298.3, 1207., 1122.8, 1045., 973.1, 906.64,
01051 845.16, 788.22, 735.48, 686.57, 641.21, 599.1, 559.99, 523.64,
01052 489.85, 458.42, 429.16, 401.92, 376.54, 352.88, 330.82, 310.24,
01053 291.03, 273.09, 256.34, 240.69, 226.05, 212.37, 199.57, 187.59,
01054 176.37, 165.87, 156.03, 146.82, 138.17, 130.07, 122.47, 115.34,
01055 108.65, 102.37, 96.473, 90.934, 85.73, 80.84, 76.243, 71.922,
01056 67.858, 64.034, 60.438, 57.052, 53.866, 50.866, 48.04, 45.379,
01057 42.872, 40.51, 38.285, 36.188, 34.211, 32.347, 30.588, 28.929,
01058 27.362, 25.884, 24.489, 23.171, 21.929, 20.755, 19.646, 18.599,
01059 17.61, 16.677, 15.795, 14.961, 14.174, 13.43, 12.725, 12.06,
01060 11.431, 10.834, 10.27, 9.7361, 9.2302, 8.7518, 8.2997, 7.8724,
01061 7.4674, 7.0848, 6.7226, 6.3794, 6.054, 5.745, 5.4525, 5.1752,
01062 4.9121, 4.6625, 4.4259, 4.2015, 3.9888, 3.7872, 3.5961, 3.4149,
01063 3.2431, 3.0802, 2.9257, 2.7792, 2.6402, 2.5084, 2.3834, 2.2648,
01064 2.1522, 2.0455, 1.9441, 1.848, 1.7567, 1.6701, 1.5878, 1.5097,
01065 1.4356, 1.3651, 1.2981, 1.2345, 1.174, 1.1167, 1.062, 1.0101,
01066 .96087, .91414, .86986, .82781, .78777, .74971, .71339, .67882,
01067 .64604, .61473, .58507, .55676, .52987, .5044, .48014, .45715,
01068 .43527, .41453, .3948, .37609, .35831, .34142, .32524, .30995,
01069 .29536, .28142, .26807, .25527, .24311, .23166, .22077, .21053,
01070 .20081, .19143, .18261, .17407, .16603, .15833, .15089, .14385,
01071 .13707, .13065, .12449, .11865, .11306, .10774, .10266, .097818,
01072 .093203, .088815, .084641, .080671, .076892, .073296, .069873,
01073 .066613, .06351, .060555, .05774, .055058, .052504, .050071,
01074 .047752, .045543, .043438, .041432, .039521, .037699, .035962,
01075 .034307, .032729, .031225, .029791, .028423, .02712, .025877,
01076 .024692, .023563, .022485, .021458, .020478, .019543, .018652,
01077 .017802, .016992, .016219, .015481, .014778, .014107, .013467,
01078 .012856, .012274, .011718, .011188, .010682, .0102, .0097393,
01079 .0093001, .008881, .0084812, .0080997, .0077358, .0073885,
01080 .0070571, .0067409, .0064393, .0061514, .0058768, .0056147,
01081 .0053647, .0051262, .0048987, .0046816, .0044745, .0042769,
01082 .0040884, .0039088, .0037373, .0035739, .003418, .0032693,
01083 .0031277, .0029926, .0028639, .0027413, .0026245, .0025133,
01084 .0024074, .0023066, .0022108, .0021196, .002033, .0019507,
01085 .0018726, .0017985, .0017282, .0016617, .0015988, .0015394,
01086 .0014834, .0014306, .0013811, .0013346, .0012911, .0012506,
01087 .0012131, .0011784, .0011465, .0011175, .0010912, .0010678,
01088 .0010472, .0010295, .0010147, .001003, 9.9428e-4, 9.8883e-4,
01089 9.8673e-4, 9.8821e-4, 9.9343e-4, .0010027, .0010164, .0010348,
01090 .0010586, .0010882, .0011245, .0011685, .0012145, .0012666,
01091 .0013095, .0013688, .0014048, .0014663, .0015309, .0015499,
01092 .0016144, .0016312, .001705, .0017892, .0018499, .0019715,
01093 .0021102, .0022442, .0024284, .0025893, .0027703, .0029445,
01094 .0031193, .003346, .0034552, .0036906, .0037584, .0040084,
01095 .0041934, .0044587, .0047093, .0049759, .0053421, .0055134,
01096 .0059048, .0058663, .0061036, .0063259, .0059657, .0060653,
01097 .0060972, .0055539, .0055653, .0055772, .005331, .0054953,
01098 .0055919, .0058684, .006183, .0066675, .0069808, .0075142,
01099 .0078536, .0084282, .0089454, .0094625, .0093703, .0095857,
01100 .0099283, .010063, .010521, .0097778, .0098175, .010379, .010447,
01101 .0105, .010617, .010706, .01078, .011177, .011212, .011304,
01102 .011446, .011603, .011816, .012165, .012545, .013069, .013539,
01103 .01411, .014776, .016103, .017016, .017994, .018978, .01998,

```

```

01104 .021799, .022745, .023681, .024627, .025562, .026992, .027958,
01105 .029013, .030154, .031402, .03228, .033651, .035272, .037088,
01106 .039021, .041213, .043597, .045977, .04877, .051809, .054943,
01107 .058064, .061528, .06537, .069309, .071928, .075752, .079589,
01108 .083352, .084096, .087497, .090817, .091198, .094966, .099045,
01109 .10429, .10867, .11518, .12269, .13126, .14087, .15161, .16388,
01110 .16423, .1759, .18721, .19994, .21275, .22513, .23041, .24231,
01111 .25299, .25396, .26396, .27696, .27929, .2908, .30595, .31433,
01112 .3282, .3429, .35944, .37467, .39277, .41245, .43326, .45649,
01113 .48152, .51897, .54686, .57877, .61263, .64962, .68983, .73945,
01114 .78619, .83537, .89622, .95002, 1.0067, 1.0742, 1.1355, 1.2007,
01115 1.2738, 1.347, 1.4254, 1.5094, 1.6009, 1.6976, 1.8019, 1.9148,
01116 2.0357, 2.166, 2.3066, 2.4579, 2.6208, 2.7966, 2.986, 3.188,
01117 3.4081, 3.6456, 3.9, 4.1747, 4.4712, 4.7931, 5.1359, 5.5097,
01118 5.9117, 6.3435, 6.8003, 7.3001, 7.8385, 8.3945, 9.011, 9.6869,
01119 10.392, 11.18, 12.036, 12.938, 13.944, 14.881, 16.029, 17.255,
01120 18.574, 19.945, 21.38, 22.9, 24.477, 26.128, 27.87, 29.037,
01121 30.988, 33.145, 35.506, 37.76, 40.885, 44.487, 48.505, 52.911,
01122 57.56, 61.964, 67.217, 72.26, 78.343, 85.08, 91.867, 99.435,
01123 107.68, 116.97, 127.12, 138.32, 150.26, 163.04, 174.81, 189.26,
01124 205.61, 224.68, 240.98, 261.88, 285.1, 307.58, 334.35, 363.53,
01125 394.68, 427.85, 458.85, 489.25, 472.87, 486.93, 496.27, 501.52,
01126 501.57, 497.14, 488.09, 476.32, 393.76, 388.51, 393.42, 414.45,
01127 455.12, 514.62, 520.38, 547.42, 562.6, 487.47, 480.83, 391.06,
01128 376.92, 303.7, 295.91, 256.03, 236.73, 280.38, 310.71, 335.53,
01129 367.88, 401.94, 435.52, 469.13, 497.94, 588.82, 597.94, 597.2,
01130 588.28, 571.2, 555.75, 603.56, 638.15, 680.75, 801.72, 848.01,
01131 962.15, 990.06, 1068.1, 1076.2, 1115.3, 1134.2, 1136.6, 1119.1,
01132 1108.9, 1090.6, 1068.7, 1041.9, 1005.4, 967.98, 927.08, 780.1,
01133 751.41, 733.12, 742.65, 785.56, 855.16, 852.45, 878.1, 784.59,
01134 777.81, 765.13, 622.93, 498.09, 474.89, 386.9, 378.48, 336.17,
01135 322.04, 329.57, 350.5, 383.38, 420.02, 462.39, 499.71, 531.98,
01136 654.99, 653.43, 639.99, 605.16, 554.16, 504.42, 540.64, 552.33,
01137 679.46, 699.51, 713.91, 832.17, 919.91, 884.96, 907.57, 846.56,
01138 818.56, 768.93, 706.71, 642.17, 575.95, 515.38, 459.07, 409.02,
01139 364.61, 325.46, 291.1, 260.89, 234.39, 211.01, 190.38, 172.11,
01140 155.91, 141.49, 128.63, 117.13, 106.84, 97.584, 89.262, 81.756,
01141 74.975, 68.842, 63.28, 58.232, 53.641, 49.46, 45.649, 42.168,
01142 38.991, 36.078, 33.409, 30.96, 28.71, 26.642, 24.737, 22.985,
01143 21.37, 19.882, 18.512, 17.242, 16.073, 14.987, 13.984, 13.05,
01144 12.186, 11.384, 10.637, 9.9436, 9.2988, 8.6991, 8.141, 7.6215,
01145 7.1378, 6.6872, 6.2671, 5.8754, 5.51, 5.1691, 4.851, 4.5539,
01146 4.2764, 4.0169, 3.7742, 3.5472, 3.3348, 3.1359, 2.9495, 2.7749,
01147 2.6113, 2.4578, 2.3139, 2.1789, 2.0523, 1.9334, 1.8219, 1.7171,
01148 1.6188, 1.5263, 1.4395, 1.3579, 1.2812, 1.209, 1.1411, 1.0773,
01149 1.0171, .96048, .90713, .85684, .80959, .76495, .72282, .68309,
01150 .64563, .61035, .57707, .54573, .51622, .48834, .46199, .43709,
01151 .41359, .39129, .37034, .35064, .33198, .31442, .29784, .28218,
01152 .26732, .25337, .24017, .22774, .21601, .20479, .19426
01153 };
01154
01155 static double co2260[2001] = { 5.7971e-5, 6.0733e-5, 6.3628e-5, 6.6662e-5,
01156 6.9843e-5, 7.3176e-5, 7.6671e-5, 8.0334e-5, 8.4175e-5, 8.8201e-5,
01157 9.2421e-5, 9.6846e-5, 1.0149e-4, 1.0635e-4, 1.1145e-4, 1.1679e-4,
01158 1.224e-4, 1.2828e-4, 1.3444e-4, 1.409e-4, 1.4768e-4, 1.5479e-4,
01159 1.6224e-4, 1.7006e-4, 1.7826e-4, 1.8685e-4, 1.9587e-4, 2.0532e-4,
01160 2.1524e-4, 2.2565e-4, 2.3656e-4, 2.48e-4, 2.6001e-4, 2.7261e-4,
01161 2.8582e-4, 2.9968e-4, 3.1422e-4, 3.2948e-4, 3.4548e-4, 3.6228e-4,
01162 3.799e-4, 3.9838e-4, 4.1778e-4, 4.3814e-4, 4.595e-4, 4.8191e-4,
01163 5.0543e-4, 5.3012e-4, 5.5603e-4, 5.8321e-4, 6.1175e-4, 6.417e-4,
01164 6.7314e-4, 7.0614e-4, 7.4078e-4, 7.7714e-4, 8.1531e-4, 8.5538e-4,
01165 8.9745e-4, 9.4162e-4, 9.8798e-4, .0010367, .0010878, .0011415,
01166 .0011978, .001257, .0013191, .0013844, .001453, .0015249,
01167 .0016006, .00168, .0017634, .001851, .001943, .0020397, .0021412,
01168 .0022479, .00236, .0024778, .0026015, .0027316, .0028682,
01169 .0030117, .0031626, .0033211, .0034877, .0036628, .0038469,
01170 .0040403, .0042436, .0044574, .004682, .0049182, .0051665,
01171 .0054276, .0057021, .0059907, .0062942, .0066133, .0069489,
01172 .0073018, .0076729, .0080632, .0084738, .0089056, .0093599,
01173 .0098377, .01034, .010869, .011426, .012011, .012627, .013276,
01174 .013958, .014676, .015431, .016226, .017063, .017944, .018872,
01175 .019848, .020876, .021958, .023098, .024298, .025561, .026892,
01176 .028293, .029769, .031323, .032961, .034686, .036503, .038418,
01177 .040435, .042561, .044801, .047161, .049649, .052271, .055035,
01178 .057948, .061019, .064256, .06767, .07127, .075066, .079069,
01179 .083291, .087744, .092441, .097396, .10262, .10814, .11396,
01180 .1201, .12658, .13342, .14064, .14826, .1563, .1648, .17376,
01181 .18323, .19324, .2038, .21496, .22674, .23919, .25234, .26624,
01182 .28093, .29646, .31287, .33021, .34855, .36794, .38844, .41012,
01183 .43305, .45731, .48297, .51011, .53884, .56924, .60141, .63547,
01184 .67152, .70969, .75012, .79292, .83826, .8863, .93718, .99111,
01185 1.0482, 1.1088, 1.173, 1.2411, 1.3133, 1.3898, 1.471, 1.5571,
01186 1.6485, 1.7455, 1.8485, 1.9577, 2.0737, 2.197, 2.3278, 2.4668,
01187 2.6145, 2.7715, 2.9383, 3.1156, 3.3042, 3.5047, 3.7181, 3.9451,
01188 4.1866, 4.4437, 4.7174, 5.0089, 5.3192, 5.65, 6.0025, 6.3782,
01189 6.7787, 7.206, 7.6617, 8.1479, 8.6669, 9.221, 9.8128, 10.445,
01190 11.12, 11.843, 12.615, 13.441, 14.325, 15.271, 16.283, 17.367,

```

```
01191 18.529, 19.776, 21.111, 22.544, 24.082, 25.731, 27.504, 29.409,
01192 31.452, 33.654, 36.024, 38.573, 41.323, 44.29, 47.492, 50.951,
01193 54.608, 58.588, 62.929, 67.629, 72.712, 78.226, 84.207, 90.699,
01194 97.749, 105.42, 113.77, 122.86, 132.78, 143.61, 155.44, 168.33,
01195 182.48, 198.01, 214.87, 233.39, 253.86, 276.34, 300.3, 327.28,
01196 356.89, 389.48, 422.29, 458.99, 501.39, 548.13, 595.62, 652.74,
01197 716.54, 784.57, 866.78, 960.59, 1062.8, 1072.5, 1189.5, 1319.4,
01198 1467.6, 1630.2, 1813.7, 2016.9, 2253., 2515.3, 2773.5, 3092.8,
01199 3444.4, 3720.4, 4104.3, 4527.5, 4645.9, 5021.7, 5462.2, 5597.,
01200 6110.6, 6732.5, 7513.8, 8270.6, 9640.6, 11487., 2796.1, 2680.1,
01201 2441.6, 2404.2, 2334.8, 2215.2, 1642.5, 1477.9, 1328.1, 1223.5,
01202 843.34, 766.96, 831.65, 834.84, 774.85, 1156.3, 1275.6, 1366.1,
01203 1795.6, 1885., 1936.5, 1953.4, 2154.4, 2002.7, 1789.8, 10381.,
01204 9040., 8216.5, 7384.7, 6721.9, 6187.7, 6143.8, 5703.9, 5276.6,
01205 4873.1, 4736., 4325.3, 3927., 3554.1, 3286.1, 2950.1, 2642.4,
01206 2368.7, 2138.9, 1914., 1719.6, 1543.9, 1388.6, 1252.1, 1132.2,
01207 1024.1, 1025.4, 920.58, 829.59, 750.54, 685.01, 624.25, 570.14,
01208 525.81, 481.85, 441.95, 408.71, 377.23, 345.86, 318.51, 292.26,
01209 268.34, 247.04, 227.14, 209.02, 192.69, 177.59, 163.78, 151.26,
01210 139.73, 129.19, 119.53, 110.7, 102.57, 95.109, 88.264, 81.948,
01211 76.13, 70.768, 65.827, 61.251, 57.022, 53.495, 49.824, 46.443,
01212 43.307, 40.405, 37.716, 35.241, 32.923, 30.77, 28.78, 26.915,
01213 25.177, 23.56, 22.059, 20.654, 19.345, 18.126, 16.988, 15.93,
01214 14.939, 14.014, 13.149, 12.343, 11.589, 10.884, 10.225, 9.6093,
01215 9.0327, 8.4934, 7.9889, 7.5166, 7.0744, 6.6604, 6.2727, 5.9098,
01216 5.5701, 5.2529, 4.955, 4.676, 4.4148, 4.171, 3.9426, 3.7332,
01217 3.5347, 3.3493, 3.1677, 3.0025, 2.8466, 2.6994, 2.5601, 2.4277,
01218 2.3016, 2.1814, 2.0664, 1.9564, 1.8279, 1.7311, 1.6427, 1.5645,
01219 1.4982, 1.443, 1.374, 1.3146, 1.2562, 1.17, 1.1105, 1.0272,
01220 .96863, .89718, .83654, .80226, .75908, .72431, .69573, .67174,
01221 .65126, .63315, .61693, .60182, .58715, .59554, .57649, .55526,
01222 .53177, .50622, .48176, .4813, .47642, .47492, .50273, .50293,
01223 .52687, .52239, .53419, .53814, .52626, .52211, .51492, .50622,
01224 .49746, .48841, .4792, .43534, .41999, .40349, .38586, .36799,
01225 .35108, .31089, .30803, .3171, .33599, .35041, .36149, .32924,
01226 .32462, .27309, .25961, .20922, .19504, .15683, .13098, .11588,
01227 .11478, .11204, .11363, .12135, .16423, .17785, .19094, .20236,
01228 .21084, .2154, .24108, .22848, .20871, .18797, .17963, .17834,
01229 .21552, .22284, .26945, .27052, .30108, .28977, .29772, .29224,
01230 .27658, .24956, .22777, .20654, .18392, .16338, .1452, .12916,
01231 .1152, .10304, .092437, .083163, .075031, .067878, .061564,
01232 .055976, .051018, .046609, .042679, .03917, .036032, .033223,
01233 .030706, .02845, .026428, .024617, .022998, .021554, .02027,
01234 .019136, .018141, .017278, .016541, .015926, .015432, .015058,
01235 .014807, .014666, .014635, .014728, .014947, .01527, .015728,
01236 .016345, .017026, .017798, .018839, .019752, .020636, .021886,
01237 .022695, .02327, .023478, .024292, .023544, .022222, .021932,
01238 .020052, .018143, .017722, .017031, .017782, .01938, .020734,
01239 .020476, .019255, .017477, .016878, .014617, .012489, .011765,
01240 .0099077, .0086446, .0079446, .0078644, .0079763, .008671,
01241 .01001, .0108, .012933, .015349, .016341, .018484, .020254,
01242 .020254, .020478, .019591, .018595, .018385, .019913, .022254,
01243 .024847, .025809, .028053, .029924, .030212, .031367, .03222,
01244 .032739, .032537, .03286, .033344, .033507, .033499, .033339,
01245 .032809, .033041, .031723, .029837, .027511, .026603, .024032,
01246 .021914, .020948, .021701, .023425, .024259, .024987, .023818,
01247 .021768, .019223, .018144, .015282, .012604, .01163, .0097907,
01248 .008336, .0082473, .0079582, .0088077, .009779, .010129, .012145,
01249 .014378, .016761, .01726, .018997, .019998, .019809, .01819,
01250 .016358, .016099, .01617, .017939, .020223, .022521, .02277,
01251 .024279, .025247, .024222, .023989, .023224, .021493, .020362,
01252 .018596, .017309, .015975, .014466, .013171, .011921, .01078,
01253 .0097229, .0087612, .0078729, .0070682, .0063494, .0057156,
01254 .0051459, .0046273, .0041712, .0037686, .0034119, .003095,
01255 .0028126, .0025603, .0023342, .0021314, .0019489, .0017845,
01256 .001636, .0015017, .00138, .0012697, .0011694, .0010782,
01257 9.9507e-4, 9.1931e-4, 8.5013e-4, 7.869e-4, 7.2907e-4, 6.7611e-4,
01258 6.2758e-4, 5.8308e-4, 5.4223e-4, 5.0473e-4, 4.7027e-4, 4.3859e-4,
01259 4.0946e-4, 3.8265e-4, 3.5798e-4, 3.3526e-4, 3.1436e-4, 2.9511e-4,
01260 2.7739e-4, 2.6109e-4, 2.4609e-4, 2.3229e-4, 2.1961e-4, 2.0797e-4,
01261 1.9729e-4, 1.875e-4, 1.7855e-4, 1.7038e-4, 1.6294e-4, 1.5619e-4,
01262 1.5007e-4, 1.4456e-4, 1.3961e-4, 1.3521e-4, 1.3131e-4, 1.2789e-4,
01263 1.2494e-4, 1.2242e-4, 1.2032e-4, 1.1863e-4, 1.1733e-4, 1.1641e-4,
01264 1.1585e-4, 1.1565e-4, 1.158e-4, 1.1629e-4, 1.1712e-4, 1.1827e-4,
01265 1.1976e-4, 1.2158e-4, 1.2373e-4, 1.262e-4, 1.2901e-4, 1.3214e-4,
01266 1.3562e-4, 1.3944e-4, 1.4361e-4, 1.4814e-4, 1.5303e-4, 1.5829e-4,
01267 1.6394e-4, 1.6999e-4, 1.7644e-4, 1.8332e-4, 1.9063e-4, 1.984e-4,
01268 2.0663e-4, 2.1536e-4, 2.246e-4, 2.3436e-4, 2.4468e-4, 2.5558e-4,
01269 2.6708e-4, 2.7921e-4, 2.92e-4, 3.0548e-4, 3.1968e-4, 3.3464e-4,
01270 3.5039e-4, 3.6698e-4, 3.8443e-4, 4.0281e-4, 4.2214e-4, 4.4248e-4,
01271 4.6389e-4, 4.864e-4, 5.1009e-4, 5.3501e-4, 5.6123e-4, 5.888e-4,
01272 6.1781e-4, 6.4833e-4, 6.8043e-4, 7.142e-4, 7.4973e-4, 7.8711e-4,
01273 8.2644e-4, 8.6783e-4, 9.1137e-4, 9.5721e-4, .0010054, .0010562,
01274 .0011096, .0011659, .0012251, .0012875, .0013532, .0014224,
01275 .0014953, .001572, .0016529, .0017381, .0018279, .0019226,
01276 .0020224, .0021277, .0022386, .0023557, .0024792, .0026095,
01277 .002747, .0028921, .0030453, .0032071, .003378, .0035586,
```



01278 .0037494, .003951, .0041642, .0043897, .0046282, .0048805,  
 01279 .0051476, .0054304, .00573, .0060473, .0063837, .0067404,  
 01280 .0071188, .0075203, .0079466, .0083994, .0088806, .0093922,  
 01281 .0099366, .010516, .011134, .011792, .012494, .013244, .014046,  
 01282 .014898, .015808, .016781, .017822, .018929, .020108, .02138,  
 01283 .022729, .02419, .02576, .027412, .029233, .031198, .033301,  
 01284 .035594, .038092, .040767, .04372, .046918, .050246, .053974,  
 01285 .058009, .061976, .066586, .071537, .076209, .081856, .087998,  
 01286 .093821, .10113, .10913, .11731, .12724, .13821, .15025, .1639,  
 01287 .17807, .19472, .21356, .23496, .25758, .28387, .31389, .34104,  
 01288 .37469, .40989, .43309, .46845, .5042, .5023, .52981, .55275,  
 01289 .51075, .51976, .52457, .44779, .44721, .4503, .4243, .45244,  
 01290 .49491, .55399, .39021, .24802, .2501, .2618, .27475, .28879,  
 01291 .31317, .33643, .36257, .4018, .43275, .46525, .53333, .56599,  
 01292 .60557, .70142, .74194, .77736, .88567, .91182, .93294, .98407,  
 01293 .98772, .99176, .9995, 1.2405, 1.3602, 1.338, 1.3255, 1.3267,  
 01294 1.3404, 1.3634, 1.3967, 1.4407, 1.4961, 1.5603, 1.6328, 1.7153,  
 01295 1.8094, 1.9091, 2.018, 2.1367, 2.264, 2.4035, 2.5562, 2.7179,  
 01296 2.9017, 3.1052, 3.3304, 3.5731, 3.8488, 4.1553, 4.4769, 4.7818,  
 01297 5.1711, 5.5204, 5.9516, 6.4097, 6.8899, 7.1118, 7.5469, 7.9735,  
 01298 7.9511, 8.3014, 8.6418, 8.4757, 8.8256, 9.2294, 9.6923, 10.033,  
 01299 10.842, 11.851, 11.78, 8.8435, 9.1381, 9.5956, 10.076, 10.629,  
 01300 11.22, 11.883, 12.69, 13.163, 13.974, 14.846, 16.027, 17.053,  
 01301 18.148, 19.715, 20.907, 22.163, 23.956, 25.235, 26.566, 27.94,  
 01302 29.576, 30.956, 32.432, 35.337, 39.911, 41.128, 42.625, 44.386,  
 01303 46.369, 48.619, 51.031, 53.674, 56.825, 59.921, 63.286, 66.929,  
 01304 70.859, 75.081, 79.618, 84.513, 89.739, 95.335, 101.35, 107.76,  
 01305 114.63, 121.98, 129.87, 138.3, 147.34, 157.04, 167.56, 178.67,  
 01306 190.61, 203.43, 217.19, 231.99, 247.88, 264.98, 283.37, 303.17,  
 01307 324.49, 347.47, 372.25, 398.98, 427.85, 459.06, 492.8, 529.31,  
 01308 568.89, 611.79, 658.35, 708.91, 763.87, 823.65, 888.72, 959.58,  
 01309 1036.8, 1121.8, 1213.9, 1314.3, 1423.8, 1543., 1672.8, 1813.4,  
 01310 1966.1, 2131.4, 2309.5, 2499.3, 2705., 2925.7, 3161.6, 3411.3,  
 01311 3611.5, 3889.2, 4191.1, 4519.3, 4877.9, 5272.9, 5712.9, 6142.7,  
 01312 6719.6, 7385., 8145., 8977.7, 9831.9, 10827., 11934., 13063.,  
 01313 14434., 15878., 17591., 19435., 21510., 23835., 26835., 29740.,  
 01314 32878., 36305., 39830., 43273., 46931., 50499., 49586., 51598.,  
 01315 53429., 54619., 55081., 55102., 54485., 53487., 52042., 42689.,  
 01316 42607., 44020., 47994., 54169., 53916., 55808., 56642., 46049.,  
 01317 44243., 32929., 30658., 21963., 20835., 15962., 13679., 17652.,  
 01318 19680., 22388., 25625., 29184., 32520., 35720., 38414., 40523.,  
 01319 49228., 48173., 45678., 41768., 37600., 41313., 42654., 44465.,  
 01320 55736., 56630., 65409., 63308., 66572., 61845., 60379., 56777.,  
 01321 51920., 46601., 41367., 36529., 32219., 28470., 25192., 22362.,  
 01322 19907., 17772., 15907., 14273., 12835., 11567., 10445., 9450.2,  
 01323 8565.1, 7776., 7070.8, 6439.2, 5872.3, 5362.4, 4903., 4488.3,  
 01324 4113.4, 3773.8, 3465.8, 3186.1, 2931.7, 2700.1, 2488.8, 2296.,  
 01325 2119.8, 1958.6, 1810.9, 1675.6, 1551.4, 1437.3, 1332.4, 1236.,  
 01326 1147.2, 1065.3, 989.86, 920.22, 855.91, 796.48, 741.53, 690.69,  
 01327 643.62, 600.02, 559.6, 522.13, 487.35, 455.06, 425.08, 397.21,  
 01328 371.3, 347.2, 324.78, 303.9, 284.46, 266.34, 249.45, 233.7,  
 01329 219.01, 205.3, 192.5, 180.55, 169.38, 158.95, 149.2, 140.07,  
 01330 131.54, 123.56, 116.09, 109.09, 102.54, 96.405, 90.655, 85.266,  
 01331 80.213, 75.475, 71.031, 66.861, 62.948, 59.275, 55.827, 52.587,  
 01332 49.544, 46.686, 43.998, 41.473, 39.099, 36.867, 34.768, 32.795,  
 01333 30.939, 29.192, 27.546, 25.998, 24.539, 23.164, 21.869, 20.65,  
 01334 19.501, 18.419, 17.399, 16.438, 15.532, 14.678, 13.874, 13.115,  
 01335 12.4, 11.726, 11.088, 10.488, 9.921, 9.3846, 8.8784, 8.3996,  
 01336 7.9469, 7.5197, 7.1174, 6.738, 6.379, 6.0409, 5.7213, 5.419,  
 01337 5.1327, 4.8611, 4.6046, 4.3617, 4.1316, 3.9138, 3.7077, 3.5125,  
 01338 3.3281, 3.1536, 2.9885, 2.8323, 2.6846, 2.5447, 2.4124, 2.2871,  
 01339 2.1686, 2.0564, 1.9501, 1.8495, 1.7543, 1.6641, 1.5787, 1.4978,  
 01340 1.4212, 1.3486, 1.2799, 1.2147, 1.1529, 1.0943, 1.0388, .98602,  
 01341 .93596, .8886, .84352, .80078, .76029, .722, .68585, .65161,  
 01342 .61901, .58808, .55854, .53044, .5039, .47853, .45459, .43173,  
 01343 .41008, .38965, .37021, .35186, .33444, .31797, .30234, .28758,  
 01344 .2736, .26036, .24764, .2357, .22431, .21342, .20295, .19288,  
 01345 .18334, .17444, .166, .15815, .15072, .14348, .13674, .13015,  
 01346 .12399, .11807, .11231, .10689, .10164, .096696, .091955,  
 01347 .087476, .083183, .079113, .075229, .071536, .068026, .064698,  
 01348 .06154, .058544, .055699, .052997, .050431, .047993, .045676,  
 01349 .043475, .041382, .039392, .037501, .035702, .033991, .032364,  
 01350 .030817, .029345, .027945, .026613, .025345, .024139, .022991,  
 01351 .021899, .02086, .019871, .018929, .018033, .01718, .016368,  
 01352 .015595, .014859, .014158, .013491, .012856, .012251, .011675,  
 01353 .011126, .010604, .010107, .0096331, .009182, .0087523, .0083431,  
 01354 .0079533, .0075821, .0072284, .0068915, .0065706, .0062649,  
 01355 .0059737, .0056963, .005432, .0051802, .0049404, .0047118,  
 01356 .0044941, .0042867, .0040891, .0039009, .0037216, .0035507,  
 01357 .003388, .0032329, .0030852, .0029445, .0028105, .0026829,  
 01358 .0025613, .0024455, .0023353, .0022303, .0021304, .0020353,  
 01359 .0019448, .0018587, .0017767, .0016988, .0016247, .0015543,  
 01360 .0014874, .0014238, .0013635, .0013062, .0012519, .0012005,  
 01361 .0011517, .0011057, .0010621, .001021, 9.8233e-4, 9.4589e-4,  
 01362 9.1167e-4, 8.7961e-4, 8.4964e-4, 8.2173e-4, 7.9582e-4, 7.7189e-4,  
 01363 7.499e-4, 7.2983e-4, 7.1167e-4, 6.9542e-4, 6.8108e-4, 6.6866e-4,  
 01364 6.5819e-4, 6.4971e-4, 6.4328e-4, 6.3895e-4, 6.3681e-4, 6.3697e-4,

```

01365     6.3956e-4, 6.4472e-4, 6.5266e-4, 6.6359e-4, 6.778e-4, 6.9563e-4,
01366     7.1749e-4, 7.4392e-4, 7.7556e-4, 8.1028e-4, 8.4994e-4, 8.8709e-4,
01367     9.3413e-4, 9.6953e-4, .0010202, .0010738, .0010976, .0011507,
01368     .0011686, .0012264, .001291, .0013346, .0014246, .0015293,
01369     .0016359, .0017824, .0019255, .0020854, .002247, .0024148,
01370     .0026199, .0027523, .0029704, .0030702, .0033047, .0035013,
01371     .0037576, .0040275, .0043089, .0046927, .0049307, .0053486,
01372     .0053809, .0056699, .0059325, .0055488, .005634, .0056392,
01373     .004946, .0048855, .0048208, .0044386, .0045498, .0046377,
01374     .0048939, .0052396, .0057324, .0060859, .0066906, .0071148,
01375     .0077224, .0082687, .008769, .0084471, .008572, .0087729,
01376     .008775, .0090742, .0080704, .0080288, .0085747, .0086087,
01377     .0086408, .0088752, .0089381, .0089757, .0093532, .0092824,
01378     .0092566, .0092645, .0092735, .009342, .0095806, .0097991,
01379     .010213, .010611, .011129, .011756, .013237, .01412, .015034,
01380     .015936, .01682, .018597, .019315, .019995, .020658, .021289,
01381     .022363, .022996, .023716, .024512, .025434, .026067, .027118,
01382     .028396, .029865, .031442, .033253, .03525, .037296, .039701,
01383     .042356, .045154, .048059, .051294, .054893, .058636, .061407,
01384     .065172, .068974, .072676, .073379, .076547, .079556, .079134,
01385     .082308, .085739, .090192, .09359, .099599, .10669, .11496,
01386     .1244, .13512, .14752, .14494, .15647, .1668, .17863, .19029,
01387     .20124, .20254, .21179, .21982, .21625, .22364, .23405, .23382,
01388     .2434, .25708, .26406, .27621, .28909, .30395, .31717, .33271,
01389     .3496, .36765, .38774, .40949, .446, .46985, .49846, .5287, .562,
01390     .59841, .64598, .68834, .7327, .78978, .8373, .88708, .94744,
01391     1.0006, 1.0574, 1.1215, 1.1856, 1.2546, 1.3292, 1.4107, 1.4974,
01392     1.5913, 1.6931, 1.8028, 1.9212, 2.0492, 2.1874, 2.3365, 2.4978,
01393     2.6718, 2.8588, 3.062, 3.2818, 3.5188, 3.7752, 4.0527, 4.3542,
01394     4.6782, 5.0312, 5.4123, 5.8246, 6.2639, 6.7435, 7.2636, 7.8064,
01395     8.4091, 9.0696, 9.7677, 10.548, 11.4, 12.309, 13.324, 14.284,
01396     15.445, 16.687, 18.019, 19.403, 20.847, 22.366, 23.925, 25.537,
01397     27.213, 28.069, 29.864, 31.829, 33.988, 35.856, 38.829, 42.321,
01398     46.319, 50.606, 55.126, 59.126, 64.162, 68.708, 74.615, 81.176,
01399     87.739, 95.494, 103.83, 113.38, 123.99, 135.8, 148.7, 162.58,
01400     176.32, 192.6, 211.47, 232.7, 252.64, 277.41, 305.38, 333.44,
01401     366.42, 402.66, 442.14, 484.53, 526.42, 568.15, 558.78, 582.6,
01402     600.98, 613.94, 619.44, 618.24, 609.84, 595.96, 484.86, 475.59,
01403     478.49, 501.56, 552.19, 628.44, 630.39, 658.92, 671.96, 562.7,
01404     545.88, 423.43, 400.14, 306.59, 294.13, 246.8, 226.51, 278.21,
01405     314.39, 347.22, 389.13, 433.16, 477.48, 521.67, 560.54, 683.6,
01406     696.37, 695.91, 683.1, 658.24, 634.89, 698.85, 742.87, 796.66,
01407     954.49, 1009.5, 1150.5, 1179.1, 1267.9, 1272.4, 1312.7, 1330.4,
01408     1331.6, 1315.8, 1308.3, 1293.3, 1274.6, 1249.5, 1213.2, 1172.1,
01409     1124.4, 930.33, 893.36, 871.27, 883.54, 940.76, 1036., 1025.6,
01410     1053.1, 914.51, 894.15, 865.03, 670.63, 508.41, 475.15, 370.85,
01411     361.06, 319.38, 312.75, 331.87, 367.13, 415., 467.94, 525.49,
01412     578.41, 624.66, 794.82, 796.97, 780.29, 736.49, 670.18, 603.75,
01413     659.67, 679.8, 857.12, 884.05, 900.65, 1046.1, 1141.9, 1083.,
01414     1089.2, 1e3, 947.08, 872.31, 787.91, 704.75, 624.93, 553.68,
01415     489.91, 434.21, 385.64, 343.3, 306.42, 274.18, 245.94, 221.11,
01416     199.23, 179.88, 162.73, 147.48, 133.88, 121.73, 110.86, 101.1,
01417     92.323, 84.417, 77.281, 70.831, 64.991, 59.694, 54.884, 50.509,
01418     46.526, 42.893, 39.58, 36.549, 33.776, 31.236, 28.907, 26.77,
01419     24.805, 23., 21.339, 19.81, 18.404, 17.105, 15.909, 14.801,
01420     13.778, 12.83, 11.954, 11.142, 10.389, 9.691, 9.0434, 8.4423,
01421     7.8842, 7.3657, 6.8838, 6.4357, 6.0189, 5.6308, 5.2696, 4.9332,
01422     4.6198, 4.3277, 4.0553, 3.8012, 3.5639, 3.3424, 3.1355, 2.9422,
01423     2.7614, 2.5924, 2.4343, 2.2864, 2.148, 2.0184, 1.8971, 1.7835,
01424     1.677, 1.5773, 1.4838, 1.3961, 1.3139, 1.2369, 1.1645, 1.0966,
01425     1.0329, .97309, .91686, .86406, .81439, .76767, .72381, .68252,
01426     .64359, .60695, .57247, .54008, .50957, .48092, .45401, .42862,
01427     .40465, .38202, .36072, .34052, .3216, .30386, .28711, .27135,
01428     .25651, .24252, .2293, .21689, .20517, .19416, .18381, .17396,
01429     .16469
01430 };
01431
01432 static double co2230[2001] = { 2.743e-5, 2.8815e-5, 3.027e-5, 3.1798e-5,
01433     3.3405e-5, 3.5094e-5, 3.6869e-5, 3.8734e-5, 4.0694e-5, 4.2754e-5,
01434     4.492e-5, 4.7196e-5, 4.9588e-5, 5.2103e-5, 5.4747e-5, 5.7525e-5,
01435     6.0446e-5, 6.3516e-5, 6.6744e-5, 7.0137e-5, 7.3704e-5, 7.7455e-5,
01436     8.1397e-5, 8.5543e-5, 8.9901e-5, 9.4484e-5, 9.9302e-5, 1.0437e-4,
01437     1.097e-4, 1.153e-4, 1.2119e-4, 1.2738e-4, 1.3389e-4, 1.4074e-4,
01438     1.4795e-4, 1.5552e-4, 1.6349e-4, 1.7187e-4, 1.8068e-4, 1.8995e-4,
01439     1.997e-4, 2.0996e-4, 2.2075e-4, 2.321e-4, 2.4403e-4, 2.5659e-4,
01440     2.698e-4, 2.837e-4, 2.9832e-4, 3.137e-4, 3.2988e-4, 3.4691e-4,
01441     3.6483e-4, 3.8368e-4, 4.0351e-4, 4.2439e-4, 4.4635e-4, 4.6947e-4,
01442     4.9379e-4, 5.1939e-4, 5.4633e-4, 5.7468e-4, 6.0452e-4, 6.3593e-4,
01443     6.69e-4, 7.038e-4, 7.4043e-4, 7.79e-4, 8.1959e-4, 8.6233e-4,
01444     9.0732e-4, 9.5469e-4, .0010046, .0010571, .0011124, .0011706,
01445     .0012319, .0012964, .0013644, .001436, .0015114, .0015908,
01446     .0016745, .0017625, .0018553, .0019531, .002056, .0021645,
01447     .0022788, .0023992, .002526, .0026596, .0028004, .0029488,
01448     .0031052, .0032699, .0034436, .0036265, .0038194, .0040227,
01449     .0042369, .0044628, .0047008, .0049518, .0052164, .0054953,
01450     .0057894, .0060995, .0064265, .0067713, .007135, .0075184,
01451     .0079228, .0083494, .0087993, .0092738, .0097745, .010303,

```

01452 .01086, .011448, .012068, .012722, .013413, .014142, .014911,  
 01453 .015723, .01658, .017484, .018439, .019447, .020511, .021635,  
 01454 .022821, .024074, .025397, .026794, .02827, .029829, .031475,  
 01455 .033215, .035052, .036994, .039045, .041213, .043504, .045926,  
 01456 .048485, .05119, .05405, .057074, .060271, .063651, .067225,  
 01457 .071006, .075004, .079233, .083708, .088441, .093449, .098749,  
 01458 .10436, .11029, .11657, .12322, .13026, .13772, .14561, .15397,  
 01459 .16282, .1722, .18214, .19266, .20381, .21563, .22816, .24143,  
 01460 .2555, .27043, .28625, .30303, .32082, .3397, .35972, .38097,  
 01461 .40352, .42746, .45286, .47983, .50847, .53888, .57119, .6055,  
 01462 .64196, .6807, .72187, .76564, .81217, .86165, .91427, .97025,  
 01463 1.0298, 1.0932, 1.1606, 1.2324, 1.3088, 1.3902, 1.477, 1.5693,  
 01464 1.6678, 1.7727, 1.8845, 2.0038, 2.131, 2.2666, 2.4114, 2.5659,  
 01465 2.7309, 2.907, 3.0951, 3.2961, 3.5109, 3.7405, 3.986, 4.2485,  
 01466 4.5293, 4.8299, 5.1516, 5.4961, 5.8651, 6.2605, 6.6842, 7.1385,  
 01467 7.6256, 8.1481, 8.7089, 9.3109, 9.9573, 10.652, 11.398, 12.2,  
 01468 13.063, 13.992, 14.99, 16.064, 17.222, 18.469, 19.813, 21.263,  
 01469 22.828, 24.516, 26.34, 28.31, 30.437, 32.738, 35.226, 37.914,  
 01470 40.824, 43.974, 47.377, 51.061, 55.011, 59.299, 63.961, 69.013,  
 01471 74.492, 80.444, 86.919, 93.836, 101.23, 109.25, 117.98, 127.47,  
 01472 137.81, 149.07, 161.35, 174.75, 189.42, 205.49, 223.02, 242.26,  
 01473 263.45, 286.75, 311.94, 340.01, 370.86, 404.92, 440.44, 480.27,  
 01474 525.17, 574.71, 626.22, 686.8, 754.38, 827.07, 913.38, 1011.7,  
 01475 1121.5, 1161.6, 1289.5, 1432.2, 1595.4, 1777., 1983.3, 2216.1,  
 01476 2485.7, 2788.3, 3101.5, 3481., 3902.1, 4257.1, 4740., 5272.8,  
 01477 5457.9, 5946.2, 6505.3, 6668.4, 7302.4, 8061.6, 9015.8, 9908.3,  
 01478 11613., 13956., 3249.6, 3243., 2901.5, 2841.3, 2729.6, 2558.2,  
 01479 1797.8, 1583.2, 1386., 1233.5, 787.74, 701.46, 761.66, 767.21,  
 01480 722.83, 1180.6, 1332.1, 1461.6, 2032.9, 2166., 2255.9, 2294.7,  
 01481 2587.2, 2396.5, 2122.4, 12553., 10784., 9832.5, 8827.3, 8029.1,  
 01482 7377.9, 7347.1, 6783.8, 6239.1, 5721.1, 5503., 4975.1, 4477.8,  
 01483 4021.3, 3676.8, 3275.3, 2914.9, 2597.4, 2328.2, 2075.4, 1857.6,  
 01484 1663.6, 1493.3, 1343.8, 1213.3, 1095.6, 1066.5, 958.91, 865.15,  
 01485 783.31, 714.35, 650.77, 593.98, 546.2, 499.9, 457.87, 421.75,  
 01486 387.61, 355.25, 326.62, 299.7, 275.21, 253.17, 232.83, 214.31,  
 01487 197.5, 182.08, 167.98, 155.12, 143.32, 132.5, 122.58, 113.48,  
 01488 105.11, 97.415, 90.182, 83.463, 77.281, 71.587, 66.341, 61.493,  
 01489 57.014, 53.062, 49.21, 45.663, 42.38, 39.348, 36.547, 33.967,  
 01490 31.573, 29.357, 27.314, 25.415, 23.658, 22.03, 20.524, 19.125,  
 01491 17.829, 16.627, 15.511, 14.476, 13.514, 12.618, 11.786, 11.013,  
 01492 10.294, 9.6246, 9.0018, 8.4218, 7.8816, 7.3783, 6.9092, 6.4719,  
 01493 6.0641, 5.6838, 5.3289, 4.998, 4.6893, 4.4014, 4.1325, 3.8813,  
 01494 3.6469, 3.4283, 3.2241, 3.035, 2.8576, 2.6922, 2.5348, 2.3896,  
 01495 2.2535, 2.1258, 2.0059, 1.8929, 1.7862, 1.6854, 1.5898, 1.4992,  
 01496 1.4017, 1.3218, 1.2479, 1.1809, 1.1215, 1.0693, 1.0116, .96016,  
 01497 .9105, .84859, .80105, .74381, .69982, .65127, .60899, .57843,  
 01498 .54592, .51792, .49336, .47155, .45201, .43426, .41807, .40303,  
 01499 .38876, .3863, .37098, .35492, .33801, .32032, .30341, .29874,  
 01500 .29193, .28689, .29584, .29155, .29826, .29195, .29287, .2904,  
 01501 .28199, .27709, .27162, .26622, .26133, .25676, .25235, .23137,  
 01502 .22365, .21519, .20597, .19636, .18699, .16485, .16262, .16643,  
 01503 .17542, .18198, .18631, .16759, .16338, .13505, .1267, .10053,  
 01504 .092554, .074093, .062159, .055523, .054849, .05401, .05528,  
 01505 .058982, .07952, .08647, .093244, .099285, .10393, .10661,  
 01506 .12072, .11417, .10396, .093265, .089137, .088909, .10902,  
 01507 .11277, .13625, .13565, .14907, .14167, .1428, .13744, .12768,  
 01508 .11382, .10244, .091686, .08109, .071739, .063616, .056579,  
 01509 .050504, .045251, .040689, .036715, .033237, .030181, .027488,  
 01510 .025107, .022998, .021125, .01946, .017979, .016661, .015489,  
 01511 .014448, .013526, .012712, .011998, .011375, .010839, .010384,  
 01512 .010007, .0097053, .0094783, .0093257, .0092489, .0092504,  
 01513 .0093346, .0095077, .0097676, .01012, .01058, .011157, .011844,  
 01514 .012672, .013665, .014766, .015999, .017509, .018972, .020444,  
 01515 .022311, .023742, .0249, .025599, .026981, .026462, .025143,  
 01516 .025066, .022814, .020458, .020026, .019142, .020189, .022371,  
 01517 .024163, .023728, .02199, .019506, .018591, .015576, .012784,  
 01518 .011744, .0094777, .0079148, .0070652, .006986, .0071758,  
 01519 .008086, .0098025, .01087, .013609, .016764, .018137, .021061,  
 01520 .023498, .023576, .023965, .022828, .021519, .021283, .023364,  
 01521 .026457, .029782, .030856, .033486, .035515, .035543, .036558,  
 01522 .037198, .037472, .037045, .037284, .03777, .038085, .038366,  
 01523 .038526, .038282, .038915, .037697, .035667, .032941, .031959,  
 01524 .028692, .025918, .024596, .025592, .027873, .028935, .02984,  
 01525 .028148, .025305, .021912, .020454, .016732, .013357, .01205,  
 01526 .009731, .0079881, .0077704, .0074387, .0083895, .0096776,  
 01527 .010326, .01293, .015955, .019247, .020145, .02267, .024231,  
 01528 .024184, .022131, .019784, .01955, .01971, .022119, .025116,  
 01529 .027978, .028107, .029808, .030701, .029164, .028551, .027286,  
 01530 .024946, .023259, .020982, .019221, .017471, .015643, .014074,  
 01531 .01261, .011301, .010116, .0090582, .0081036, .0072542, .0065034,  
 01532 .0058436, .0052571, .0047321, .0042697, .0038607, .0034977,  
 01533 .0031747, .0028864, .0026284, .002397, .002189, .0020017,  
 01534 .0018326, .0016798, .0015414, .0014159, .0013019, .0011983,  
 01535 .0011039, .0010177, 9.391e-4, 8.6717e-4, 8.0131e-4, 7.4093e-4,  
 01536 6.8553e-4, 6.3464e-4, 5.8787e-4, 5.4487e-4, 5.0533e-4, 4.69e-4,  
 01537 4.3556e-4, 4.0474e-4, 3.7629e-4, 3.5e-4, 3.2569e-4, 3.032e-4,  
 01538 2.8239e-4, 2.6314e-4, 2.4535e-4, 2.2891e-4, 2.1374e-4, 1.9975e-4,

```
01539 1.8685e-4, 1.7498e-4, 1.6406e-4, 1.5401e-4, 1.4479e-4, 1.3633e-4,
01540 1.2858e-4, 1.2148e-4, 1.1499e-4, 1.0907e-4, 1.0369e-4, 9.8791e-5,
01541 9.4359e-5, 9.0359e-5, 8.6766e-5, 8.3555e-5, 8.0703e-5, 7.8192e-5,
01542 7.6003e-5, 7.4119e-5, 7.2528e-5, 7.1216e-5, 7.0171e-5, 6.9385e-5,
01543 6.8848e-5, 6.8554e-5, 6.8496e-5, 6.8669e-5, 6.9069e-5, 6.9694e-5,
01544 7.054e-5, 7.1608e-5, 7.2896e-5, 7.4406e-5, 7.6139e-5, 7.8097e-5,
01545 8.0283e-5, 8.2702e-5, 8.5357e-5, 8.8255e-5, 9.1402e-5, 9.4806e-5,
01546 9.8473e-5, 1.0241e-4, 1.0664e-4, 1.1115e-4, 1.1598e-4, 1.2112e-4,
01547 1.2659e-4, 1.3241e-4, 1.3859e-4, 1.4515e-4, 1.521e-4, 1.5947e-4,
01548 1.6728e-4, 1.7555e-4, 1.8429e-4, 1.9355e-4, 2.0334e-4, 2.1369e-4,
01549 2.2463e-4, 2.3619e-4, 2.4841e-4, 2.6132e-4, 2.7497e-4, 2.8938e-4,
01550 3.0462e-4, 3.2071e-4, 3.3771e-4, 3.5567e-4, 3.7465e-4, 3.947e-4,
01551 4.1588e-4, 4.3828e-4, 4.6194e-4, 4.8695e-4, 5.1338e-4, 5.4133e-4,
01552 5.7087e-4, 6.0211e-4, 6.3515e-4, 6.701e-4, 7.0706e-4, 7.4617e-4,
01553 7.8756e-4, 8.3136e-4, 8.7772e-4, 9.2681e-4, 9.788e-4, .0010339,
01554 .0010922, .001154, .0012195, .0012889, .0013626, .0014407,
01555 .0015235, .0016114, .0017048, .0018038, .001909, .0020207,
01556 .0021395, .0022657, .0023998, .0025426, .0026944, .002856,
01557 .0030281, .0032114, .0034068, .003615, .0038371, .004074,
01558 .004327, .0045971, .0048857, .0051942, .0055239, .0058766,
01559 .0062538, .0066573, .0070891, .007551, .0080455, .0085747,
01560 .0091412, .0097481, .010397, .011092, .011837, .012638, .013495,
01561 .014415, .01541, .016475, .017621, .018857, .020175, .02162,
01562 .023185, .024876, .02672, .028732, .030916, .033319, .035939,
01563 .038736, .041847, .04524, .048715, .052678, .056977, .061203,
01564 .066184, .07164, .076952, .083477, .090674, .098049, .10697,
01565 .1169, .1277, .14011, .15323, .1684, .18601, .20626, .22831,
01566 .25417, .28407, .31405, .34957, .38823, .41923, .46026, .50409,
01567 .51227, .54805, .57976, .53818, .55056, .557, .46741, .46403,
01568 .4636, .42265, .45166, .49852, .56663, .34306, .17779, .17697,
01569 .18346, .19129, .20014, .21778, .23604, .25649, .28676, .31238,
01570 .33856, .39998, .4288, .46568, .56654, .60786, .64473, .76466,
01571 .7897, .80778, .86443, .85736, .84798, .84157, 1.1385, 1.2446,
01572 1.1923, 1.1552, 1.1338, 1.1266, 1.1292, 1.1431, 1.1683, 1.2059,
01573 1.2521, 1.3069, 1.3712, 1.4471, 1.5275, 1.6165, 1.7145, 1.8189,
01574 1.9359, 2.065, 2.2007, 2.3591, 2.5362, 2.7346, 2.9515, 3.2021,
01575 3.4851, 3.7935, 4.0694, 4.4463, 4.807, 5.2443, 5.7178, 6.2231,
01576 6.4796, 6.9461, 7.4099, 7.3652, 7.7182, 8.048, 7.7373, 8.0363,
01577 8.3855, 8.8044, 9.0257, 9.8574, 10.948, 10.563, 6.8979, 7.0744,
01578 7.4121, 7.7663, 8.1768, 8.6243, 9.1437, 9.7847, 10.182, 10.849,
01579 11.572, 12.602, 13.482, 14.431, 15.907, 16.983, 18.11, 19.884,
01580 21.02, 22.18, 23.355, 24.848, 25.954, 27.13, 30.186, 34.893,
01581 35.682, 36.755, 38.111, 39.703, 41.58, 43.606, 45.868, 48.573,
01582 51.298, 54.291, 57.559, 61.116, 64.964, 69.124, 73.628, 78.471,
01583 83.683, 89.307, 95.341, 101.84, 108.83, 116.36, 124.46, 133.18,
01584 142.57, 152.79, 163.69, 175.43, 188.11, 201.79, 216.55, 232.51,
01585 249.74, 268.38, 288.54, 310.35, 333.97, 359.55, 387.26, 417.3,
01586 449.88, 485.2, 523.54, 565.14, 610.28, 659.31, 712.56, 770.43,
01587 833.36, 901.82, 976.36, 1057.6, 1146.8, 1243.8, 1350., 1466.3,
01588 1593.6, 1732.7, 1884.1, 2049.1, 2228.2, 2421.9, 2629.4, 2853.7,
01589 3094.4, 3351.1, 3622.3, 3829.8, 4123.1, 4438.3, 4777.2, 5144.1,
01590 5545.4, 5990.5, 6404.5, 6996.8, 7687.6, 8482.9, 9349.4, 10203.,
01591 11223., 12358., 13493., 14916., 16416., 18236., 20222., 22501.,
01592 25102., 28358., 31707., 35404., 39538., 43911., 48391., 53193.,
01593 58028., 58082., 61276., 64193., 66294., 67480., 67921., 67423.,
01594 66254., 64341., 51737., 51420., 53072., 58145., 66195., 65358.,
01595 67377., 67869., 53509., 50553., 35737., 32425., 21704., 19974.,
01596 14457., 12142., 16798., 19489., 23049., 27270., 31910., 36457.,
01597 40877., 44748., 47876., 59793., 58626., 55454., 50337., 44893.,
01598 50228., 52216., 54747., 69541., 70455., 81014., 77694., 80533.,
01599 73953., 70927., 65539., 59002., 52281., 45953., 40292., 35360.,
01600 31124., 27478., 24346., 21647., 19308., 17271., 15491., 13927.,
01601 12550., 11331., 10250., 9288.8, 8431.4, 7664.9, 6978.3, 6361.8,
01602 5807.4, 5307.7, 4856.8, 4449., 4079.8, 3744.9, 3440.8, 3164.2,
01603 2912.3, 2682.7, 2473., 2281.4, 2106., 1945.3, 1797.9, 1662.5,
01604 1538.1, 1423.6, 1318.1, 1221., 1131.5, 1049., 972.99, 902.87,
01605 838.01, 777.95, 722.2, 670.44, 622.35, 577.68, 536.21, 497.76,
01606 462.12, 429.13, 398.61, 370.39, 344.29, 320.16, 297.85, 277.2,
01607 258.08, 240.38, 223.97, 208.77, 194.66, 181.58, 169.43, 158.15,
01608 147.67, 137.92, 128.86, 120.44, 112.6, 105.3, 98.499, 92.166,
01609 86.264, 80.763, 75.632, 70.846, 66.381, 62.213, 58.321, 54.685,
01610 51.288, 48.114, 45.145, 42.368, 39.772, 37.341, 35.065, 32.937,
01611 30.943, 29.077, 27.33, 25.693, 24.158, 22.717, 21.367, 20.099,
01612 18.909, 17.792, 16.744, 15.761, 14.838, 13.971, 13.157, 12.393,
01613 11.676, 11.003, 10.369, 9.775, 9.2165, 8.6902, 8.1963, 7.7314,
01614 7.2923, 6.8794, 6.4898, 6.122, 5.7764, 5.4525, 5.1484, 4.8611,
01615 4.5918, 4.3379, 4.0982, 3.8716, 3.6567, 3.4545, 3.2634, 3.0828,
01616 2.9122, 2.7512, 2.5993, 2.4561, 2.3211, 2.1938, 2.0737, 1.9603,
01617 1.8534, 1.7525, 1.6572, 1.5673, 1.4824, 1.4022, 1.3265, 1.2551,
01618 1.1876, 1.1239, 1.0637, 1.0069, .9532, .90248, .85454, .80921,
01619 .76631, .72569, .6872, .65072, .61635, .5836, .55261, .52336,
01620 .49581, .46998, .44559, .42236, .40036, .37929, .35924, .34043,
01621 .32238, .30547, .28931, .27405, .25975, .24616, .23341, .22133,
01622 .20997, .19924, .18917, .17967, .17075, .16211, .15411, .14646,
01623 .13912, .13201, .12509, .11857, .11261, .10698, .10186, .097039,
01624 .092236, .087844, .083443, .07938, .075452, .071564, .067931,
01625 .064389, .061078, .057901, .054921, .052061, .049364, .046789,
```

```

01626 .04435, .042044, .039866, .037808, .035863, .034023, .032282,
01627 .030634, .029073, .027595, .026194, .024866, .023608, .022415,
01628 .021283, .02021, .019193, .018228, .017312, .016443, .015619,
01629 .014837, .014094, .01339, .012721, .012086, .011483, .010911,
01630 .010368, .009852, .0093623, .0088972, .0084556, .0080362,
01631 .0076379, .0072596, .0069003, .006559, .0062349, .0059269,
01632 .0056344, .0053565, .0050925, .0048417, .0046034, .004377,
01633 .0041618, .0039575, .0037633, .0035788, .0034034, .0032368,
01634 .0030785, .002928, .0027851, .0026492, .0025201, .0023975,
01635 .0022809, .0021701, .0020649, .0019649, .0018699, .0017796,
01636 .0016938, .0016122, .0015348, .0014612, .0013913, .001325,
01637 .0012619, .0012021, .0011452, .0010913, .0010401, 9.9149e-4,
01638 9.454e-4, 9.0169e-4, 8.6024e-4, 8.2097e-4, 7.8377e-4, 7.4854e-4,
01639 7.1522e-4, 6.8371e-4, 6.5393e-4, 6.2582e-4, 5.9932e-4, 5.7435e-4,
01640 5.5087e-4, 5.2882e-4, 5.0814e-4, 4.8881e-4, 4.7076e-4, 4.5398e-4,
01641 4.3843e-4, 4.2407e-4, 4.109e-4, 3.9888e-4, 3.88e-4, 3.7826e-4,
01642 3.6963e-4, 3.6213e-4, 3.5575e-4, 3.505e-4, 3.464e-4, 3.4346e-4,
01643 3.4173e-4, 3.4125e-4, 3.4206e-4, 3.4424e-4, 3.4787e-4, 3.5303e-4,
01644 3.5986e-4, 3.6847e-4, 3.7903e-4, 3.9174e-4, 4.0681e-4, 4.2455e-4,
01645 4.4527e-4, 4.6942e-4, 4.9637e-4, 5.2698e-4, 5.5808e-4, 5.9514e-4,
01646 6.2757e-4, 6.689e-4, 7.1298e-4, 7.3955e-4, 7.8403e-4, 8.0449e-4,
01647 8.5131e-4, 9.0256e-4, 9.3692e-4, .0010051, .0010846, .0011678,
01648 .001282, .0014016, .0015355, .0016764, .0018272, .0020055,
01649 .0021455, .0023421, .0024615, .0026786, .0028787, .0031259,
01650 .0034046, .0036985, .0040917, .0043902, .0048349, .0049531,
01651 .0052989, .0056148, .0052452, .0053357, .005333, .0045069,
01652 .0043851, .004253, .003738, .0038084, .0039013, .0041505,
01653 .0045372, .0050569, .0054507, .0061267, .0066122, .0072449,
01654 .0078012, .0082651, .0076538, .0076573, .0076806, .0075227,
01655 .0076269, .0063758, .006254, .0067749, .0067909, .0068231,
01656 .0072143, .0072762, .0072954, .007679, .0075107, .0073658,
01657 .0072441, .0071074, .0070378, .007176, .0072472, .0075844,
01658 .0079291, .008412, .0090165, .010688, .011535, .012375, .013166,
01659 .013895, .015567, .016011, .016392, .016737, .017043, .017731,
01660 .018031, .018419, .018877, .019474, .019868, .020604, .021538,
01661 .022653, .023869, .025288, .026879, .028547, .030524, .03274,
01662 .035132, .03769, .040567, .043793, .047188, .049962, .053542,
01663 .057205, .060776, .061489, .064419, .067124, .065945, .068487,
01664 .071209, .074783, .077039, .082444, .08902, .09692, .10617,
01665 .11687, .12952, .12362, .13498, .14412, .15492, .16519, .1744,
01666 .17096, .17714, .18208, .17363, .17813, .18564, .18295, .19045,
01667 .20252, .20815, .21844, .22929, .24229, .25321, .26588, .2797,
01668 .29465, .31136, .32961, .36529, .38486, .41027, .43694, .4667,
01669 .49943, .54542, .58348, .62303, .67633, .71755, .76054, .81371,
01670 .85934, .90841, .96438, 1.0207, 1.0821, 1.1491, 1.2226, 1.3018,
01671 1.388, 1.4818, 1.5835, 1.6939, 1.8137, 1.9435, 2.0843, 2.237,
01672 2.4026, 2.5818, 2.7767, 2.9885, 3.2182, 3.4679, 3.7391, 4.0349,
01673 4.3554, 4.7053, 5.0849, 5.4986, 5.9436, 6.4294, 6.9598, 7.5203,
01674 8.143, 8.8253, 9.5568, 10.371, 11.267, 12.233, 13.31, 14.357,
01675 15.598, 16.93, 18.358, 19.849, 21.408, 23.04, 24.706, 26.409,
01676 28.153, 28.795, 30.549, 32.43, 34.49, 36.027, 38.955, 42.465,
01677 46.565, 50.875, 55.378, 59.002, 63.882, 67.949, 73.693, 80.095,
01678 86.403, 94.264, 102.65, 112.37, 123.3, 135.54, 149.14, 163.83,
01679 179.17, 196.89, 217.91, 240.94, 264.13, 292.39, 324.83, 358.21,
01680 397.16, 440.5, 488.6, 541.04, 595.3, 650.43, 652.03, 688.74,
01681 719.47, 743.54, 757.68, 762.35, 756.43, 741.42, 595.43, 580.97,
01682 580.83, 605.68, 667.88, 764.49, 759.93, 789.12, 798.17, 645.66,
01683 615.65, 455.05, 421.09, 306.45, 289.14, 235.7, 215.52, 274.57,
01684 316.53, 357.73, 409.89, 465.06, 521.84, 579.02, 630.64, 794.46,
01685 813., 813.56, 796.25, 761.57, 727.97, 812.14, 866.75, 932.5,
01686 1132.8, 1194.8, 1362.2, 1387.2, 1482.3, 1479.7, 1517.9, 1533.1,
01687 1534.2, 1523.3, 1522.5, 1515.5, 1505.2, 1486.5, 1454., 1412.,
01688 1358.8, 1107.8, 1060.9, 1033.5, 1048.2, 1122.4, 1248.9, 1227.1,
01689 1255.4, 1058.9, 1020.7, 970.59, 715.24, 512.56, 468.47, 349.3,
01690 338.26, 299.22, 301.26, 332.38, 382.08, 445.49, 515.87, 590.85,
01691 662.3, 726.05, 955.59, 964.11, 945.17, 891.48, 807.11, 720.9,
01692 803.36, 834.46, 1073.9, 1107.1, 1123.6, 1296., 1393.7, 1303.1,
01693 1284.3, 1161.8, 1078.8, 976.13, 868.72, 767.4, 674.72, 593.73,
01694 523.12, 462.24, 409.75, 364.34, 325., 290.73, 260.76, 234.46,
01695 211.28, 190.78, 172.61, 156.44, 142.01, 129.12, 117.57, 107.2,
01696 97.877, 89.47, 81.882, 75.021, 68.807, 63.171, 58.052, 53.396,
01697 49.155, 45.288, 41.759, 38.531, 35.576, 32.868, 30.384, 28.102,
01698 26.003, 24.071, 22.293, 20.655, 19.147, 17.756, 16.476, 15.292,
01699 14.198, 13.183, 12.241, 11.367, 10.554, 9.7989, 9.0978, 8.4475,
01700 7.845, 7.2868, 6.7704, 6.2927, 5.8508, 5.4421, 5.064, 4.714,
01701 4.3902, 4.0902, 3.8121, 3.5543, 3.315, 3.093, 2.8869, 2.6953,
01702 2.5172, 2.3517, 2.1977, 2.0544, 1.9211, 1.7969, 1.6812, 1.5735,
01703 1.4731, 1.3794, 1.2921, 1.2107, 1.1346, 1.0637, .99744, .93554,
01704 .87771, .82368, .77313, .72587, .6816, .64014, .60134, .565,
01705 .53086, .49883, .46881, .44074, .4144, .38979, .36679, .34513,
01706 .32474, .30552, .28751, .27045, .25458, .23976, .22584, .21278,
01707 .20051, .18899, .17815, .16801, .15846, .14954, .14117, .13328,
01708 .12584
01709 };
01710
01711 double xw, dw, ew, cw296, cw260, cw230, dt230, dt260, dt296, ctw, ctmph;
01712

```

```

01713     int iw;
01714
01715     /* Get CO2 continuum absorption... */
01716     xw = nu / 2 + 1;
01717     if (xw >= 1 && xw < 2001) {
01718         iw = (int) xw;
01719         dw = xw - iw;
01720         ew = 1 - dw;
01721         cw296 = ew * co2296[iw - 1] + dw * co2296[iw];
01722         cw260 = ew * co2260[iw - 1] + dw * co2260[iw];
01723         cw230 = ew * co2230[iw - 1] + dw * co2230[iw];
01724         dt230 = t - 230;
01725         dt260 = t - 260;
01726         dt296 = t - 296;
01727         ctw = dt260 * 5.050505e-4 * dt296 * cw230 - dt230 * 9.259259e-4
01728             * dt296 * cw260 + dt230 * 4.208754e-4 * dt260 * cw296;
01729         ctmph = u / NA / 1000 * p / P0 * ctw;
01730     } else
01731         ctmph = 0;
01732     return ctmph;
01733 }

```

### 5.21.2.7 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](#).

```

01742     {
01743
01744     static double h2o296[2001] = { .17, .1695, .172, .168, .1687, .1624, .1606,
01745     .1508, .1447, .1344, .1214, .1133, .1009, .09217, .08297, .06989,
01746     .06513, .05469, .05056, .04417, .03779, .03484, .02994, .0272,
01747     .02325, .02063, .01818, .01592, .01405, .01251, .0108, .009647,
01748     .008424, .007519, .006555, .00588, .005136, .004511, .003989,
01749     .003509, .003114, .00274, .002446, .002144, .001895, .001676,
01750     .001486, .001312, .001164, .001031, 9.129e-4, 8.106e-4, 7.213e-4,
01751     6.4e-4, 5.687e-4, 5.063e-4, 4.511e-4, 4.029e-4, 3.596e-4,
01752     3.22e-4, 2.889e-4, 2.597e-4, 2.337e-4, 2.108e-4, 1.907e-4,
01753     1.728e-4, 1.57e-4, 1.43e-4, 1.305e-4, 1.195e-4, 1.097e-4,
01754     1.009e-4, 9.307e-5, 8.604e-5, 7.971e-5, 7.407e-5, 6.896e-5,
01755     6.433e-5, 6.013e-5, 5.631e-5, 5.283e-5, 4.963e-5, 4.669e-5,
01756     4.398e-5, 4.148e-5, 3.917e-5, 3.702e-5, 3.502e-5, 3.316e-5,
01757     3.142e-5, 2.978e-5, 2.825e-5, 2.681e-5, 2.546e-5, 2.419e-5,
01758     2.299e-5, 2.186e-5, 2.079e-5, 1.979e-5, 1.884e-5, 1.795e-5,
01759     1.711e-5, 1.633e-5, 1.559e-5, 1.49e-5, 1.426e-5, 1.367e-5,
01760     1.312e-5, 1.263e-5, 1.218e-5, 1.178e-5, 1.143e-5, 1.112e-5,
01761     1.088e-5, 1.07e-5, 1.057e-5, 1.05e-5, 1.051e-5, 1.059e-5,
01762     1.076e-5, 1.1e-5, 1.133e-5, 1.18e-5, 1.237e-5, 1.308e-5,
01763     1.393e-5, 1.483e-5, 1.614e-5, 1.758e-5, 1.93e-5, 2.123e-5,
01764     2.346e-5, 2.647e-5, 2.93e-5, 3.279e-5, 3.745e-5, 4.152e-5,
01765     4.813e-5, 5.477e-5, 6.203e-5, 7.331e-5, 8.056e-5, 9.882e-5,
01766     1.05e-4, 1.21e-4, 1.341e-4, 1.572e-4, 1.698e-4, 1.968e-4,
01767     2.175e-4, 2.431e-4, 2.735e-4, 2.867e-4, 3.19e-4, 3.371e-4,
01768     3.554e-4, 3.726e-4, 3.837e-4, 3.878e-4, 3.864e-4, 3.858e-4,
01769     3.841e-4, 3.852e-4, 3.815e-4, 3.762e-4, 3.618e-4, 3.579e-4,
01770     3.45e-4, 3.202e-4, 3.018e-4, 2.785e-4, 2.602e-4, 2.416e-4,
01771     2.097e-4, 1.939e-4, 1.689e-4, 1.498e-4, 1.308e-4, 1.17e-4,
01772     1.011e-4, 9.237e-5, 7.909e-5, 7.006e-5, 6.112e-5, 5.401e-5,
01773     4.914e-5, 4.266e-5, 3.963e-5, 3.316e-5, 3.037e-5, 2.598e-5,
01774     2.294e-5, 2.066e-5, 1.813e-5, 1.583e-5, 1.423e-5, 1.247e-5,
01775     1.116e-5, 9.76e-6, 8.596e-6, 7.72e-6, 6.825e-6, 6.108e-6,
01776     5.366e-6, 4.733e-6, 4.229e-6, 3.731e-6, 3.346e-6, 2.972e-6,
01777     2.628e-6, 2.356e-6, 2.102e-6, 1.878e-6, 1.678e-6, 1.507e-6,
01778     1.348e-6, 1.21e-6, 1.089e-6, 9.806e-7, 8.857e-7, 8.004e-7,
01779     7.261e-7, 6.599e-7, 6.005e-7, 5.479e-7, 5.011e-7, 4.595e-7,
01780     4.219e-7, 3.885e-7, 3.583e-7, 3.314e-7, 3.071e-7, 2.852e-7,
01781     2.654e-7, 2.474e-7, 2.311e-7, 2.162e-7, 2.026e-7, 1.902e-7,
01782     1.788e-7, 1.683e-7, 1.587e-7, 1.497e-7, 1.415e-7, 1.338e-7,
01783     1.266e-7, 1.2e-7, 1.138e-7, 1.08e-7, 1.027e-7, 9.764e-8,
01784     9.296e-8, 8.862e-8, 8.458e-8, 8.087e-8, 7.744e-8, 7.429e-8,
01785     7.145e-8, 6.893e-8, 6.664e-8, 6.468e-8, 6.322e-8, 6.162e-8,
01786     6.07e-8, 5.992e-8, 5.913e-8, 5.841e-8, 5.796e-8, 5.757e-8,
01787     5.746e-8, 5.731e-8, 5.679e-8, 5.577e-8, 5.671e-8, 5.656e-8,
01788     5.594e-8, 5.593e-8, 5.602e-8, 5.62e-8, 5.693e-8, 5.725e-8,
01789     5.858e-8, 6.037e-8, 6.249e-8, 6.535e-8, 6.899e-8, 7.356e-8,
01790     7.918e-8, 8.618e-8, 9.385e-8, 1.039e-7, 1.158e-7, 1.29e-7,
01791     1.437e-7, 1.65e-7, 1.871e-7, 2.121e-7, 2.427e-7, 2.773e-7,
01792     3.247e-7, 3.677e-7, 4.037e-7, 4.776e-7, 5.101e-7, 6.214e-7,
01793     6.936e-7, 7.581e-7, 8.486e-7, 9.355e-7, 9.942e-7, 1.063e-6,

```

01794 1.123e-6, 1.191e-6, 1.215e-6, 1.247e-6, 1.26e-6, 1.271e-6,  
 01795 1.284e-6, 1.317e-6, 1.323e-6, 1.349e-6, 1.353e-6, 1.362e-6,  
 01796 1.344e-6, 1.329e-6, 1.336e-6, 1.327e-6, 1.325e-6, 1.359e-6,  
 01797 1.374e-6, 1.415e-6, 1.462e-6, 1.526e-6, 1.619e-6, 1.735e-6,  
 01798 1.863e-6, 2.034e-6, 2.265e-6, 2.482e-6, 2.756e-6, 3.103e-6,  
 01799 3.466e-6, 3.832e-6, 4.378e-6, 4.913e-6, 5.651e-6, 6.311e-6,  
 01800 7.169e-6, 8.057e-6, 9.253e-6, 1.047e-5, 1.212e-5, 1.36e-5,  
 01801 1.569e-5, 1.776e-5, 2.02e-5, 2.281e-5, 2.683e-5, 2.994e-5,  
 01802 3.488e-5, 3.896e-5, 4.499e-5, 5.175e-5, 6.035e-5, 6.34e-5,  
 01803 7.281e-5, 7.923e-5, 8.348e-5, 9.631e-5, 1.044e-4, 1.102e-4,  
 01804 1.176e-4, 1.244e-4, 1.283e-4, 1.326e-4, 1.4e-4, 1.395e-4,  
 01805 1.387e-4, 1.363e-4, 1.314e-4, 1.241e-4, 1.228e-4, 1.148e-4,  
 01806 1.086e-4, 1.018e-4, 8.89e-5, 8.316e-5, 7.292e-5, 6.452e-5,  
 01807 5.625e-5, 5.045e-5, 4.38e-5, 3.762e-5, 3.29e-5, 2.836e-5,  
 01808 2.485e-5, 2.168e-5, 1.895e-5, 1.659e-5, 1.453e-5, 1.282e-5,  
 01809 1.132e-5, 1.001e-5, 8.836e-6, 7.804e-6, 6.922e-6, 6.116e-6,  
 01810 5.429e-6, 4.824e-6, 4.278e-6, 3.788e-6, 3.371e-6, 2.985e-6,  
 01811 2.649e-6, 2.357e-6, 2.09e-6, 1.858e-6, 1.647e-6, 1.462e-6,  
 01812 1.299e-6, 1.155e-6, 1.028e-6, 9.142e-7, 8.132e-7, 7.246e-7,  
 01813 6.451e-7, 5.764e-7, 5.151e-7, 4.603e-7, 4.121e-7, 3.694e-7,  
 01814 3.318e-7, 2.985e-7, 2.69e-7, 2.428e-7, 2.197e-7, 1.992e-7,  
 01815 1.81e-7, 1.649e-7, 1.506e-7, 1.378e-7, 1.265e-7, 1.163e-7,  
 01816 1.073e-7, 9.918e-8, 9.191e-8, 8.538e-8, 7.949e-8, 7.419e-8,  
 01817 6.94e-8, 6.508e-8, 6.114e-8, 5.761e-8, 5.437e-8, 5.146e-8,  
 01818 4.89e-8, 4.636e-8, 4.406e-8, 4.201e-8, 4.015e-8, 3.84e-8,  
 01819 3.661e-8, 3.51e-8, 3.377e-8, 3.242e-8, 3.13e-8, 3.015e-8,  
 01820 2.918e-8, 2.83e-8, 2.758e-8, 2.707e-8, 2.656e-8, 2.619e-8,  
 01821 2.609e-8, 2.615e-8, 2.63e-8, 2.675e-8, 2.745e-8, 2.842e-8,  
 01822 2.966e-8, 3.125e-8, 3.318e-8, 3.565e-8, 3.85e-8, 4.191e-8,  
 01823 4.59e-8, 5.059e-8, 5.607e-8, 6.239e-8, 6.958e-8, 7.796e-8,  
 01824 8.773e-8, 9.88e-8, 1.114e-7, 1.258e-7, 1.422e-7, 1.61e-7,  
 01825 1.822e-7, 2.06e-7, 2.337e-7, 2.645e-7, 2.996e-7, 3.393e-7,  
 01826 3.843e-7, 4.363e-7, 4.935e-7, 5.607e-7, 6.363e-7, 7.242e-7,  
 01827 8.23e-7, 9.411e-7, 1.071e-6, 1.232e-6, 1.402e-6, 1.6e-6, 1.82e-6,  
 01828 2.128e-6, 2.386e-6, 2.781e-6, 3.242e-6, 3.653e-6, 4.323e-6,  
 01829 4.747e-6, 5.321e-6, 5.919e-6, 6.681e-6, 7.101e-6, 7.983e-6,  
 01830 8.342e-6, 8.741e-6, 9.431e-6, 9.952e-6, 1.026e-5, 1.055e-5,  
 01831 1.095e-5, 1.095e-5, 1.087e-5, 1.056e-5, 1.026e-5, 9.715e-6,  
 01832 9.252e-6, 8.452e-6, 7.958e-6, 7.268e-6, 6.295e-6, 6.003e-6, 5e-6,  
 01833 4.591e-6, 3.983e-6, 3.479e-6, 3.058e-6, 2.667e-6, 2.293e-6,  
 01834 1.995e-6, 1.747e-6, 1.517e-6, 1.335e-6, 1.165e-6, 1.028e-6,  
 01835 9.007e-7, 7.956e-7, 7.015e-7, 6.192e-7, 5.491e-7, 4.859e-7,  
 01836 4.297e-7, 3.799e-7, 3.38e-7, 3.002e-7, 2.659e-7, 2.366e-7,  
 01837 2.103e-7, 1.861e-7, 1.655e-7, 1.469e-7, 1.309e-7, 1.162e-7,  
 01838 1.032e-7, 9.198e-8, 8.181e-8, 7.294e-8, 6.516e-8, 5.787e-8,  
 01839 5.163e-8, 4.612e-8, 4.119e-8, 3.695e-8, 3.308e-8, 2.976e-8,  
 01840 2.67e-8, 2.407e-8, 2.171e-8, 1.965e-8, 1.78e-8, 1.617e-8,  
 01841 1.47e-8, 1.341e-8, 1.227e-8, 1.125e-8, 1.033e-8, 9.524e-9,  
 01842 8.797e-9, 8.162e-9, 7.565e-9, 7.04e-9, 6.56e-9, 6.129e-9,  
 01843 5.733e-9, 5.376e-9, 5.043e-9, 4.75e-9, 4.466e-9, 4.211e-9,  
 01844 3.977e-9, 3.759e-9, 3.558e-9, 3.373e-9, 3.201e-9, 3.043e-9,  
 01845 2.895e-9, 2.76e-9, 2.635e-9, 2.518e-9, 2.411e-9, 2.314e-9,  
 01846 2.23e-9, 2.151e-9, 2.087e-9, 2.035e-9, 1.988e-9, 1.946e-9,  
 01847 1.927e-9, 1.916e-9, 1.916e-9, 1.933e-9, 1.966e-9, 2.018e-9,  
 01848 2.09e-9, 2.182e-9, 2.299e-9, 2.442e-9, 2.623e-9, 2.832e-9,  
 01849 3.079e-9, 3.368e-9, 3.714e-9, 4.104e-9, 4.567e-9, 5.091e-9,  
 01850 5.701e-9, 6.398e-9, 7.194e-9, 8.127e-9, 9.141e-9, 1.035e-8,  
 01851 1.177e-8, 1.338e-8, 1.508e-8, 1.711e-8, 1.955e-8, 2.216e-8,  
 01852 2.534e-8, 2.871e-8, 3.291e-8, 3.711e-8, 4.285e-8, 4.868e-8,  
 01853 5.509e-8, 6.276e-8, 7.262e-8, 8.252e-8, 9.4e-8, 1.064e-7,  
 01854 1.247e-7, 1.411e-7, 1.626e-7, 1.827e-7, 2.044e-7, 2.284e-7,  
 01855 2.452e-7, 2.854e-7, 3.026e-7, 3.278e-7, 3.474e-7, 3.693e-7,  
 01856 3.93e-7, 4.104e-7, 4.22e-7, 4.439e-7, 4.545e-7, 4.778e-7,  
 01857 4.812e-7, 5.018e-7, 4.899e-7, 5.075e-7, 5.073e-7, 5.171e-7,  
 01858 5.131e-7, 5.25e-7, 5.617e-7, 5.846e-7, 6.239e-7, 6.696e-7,  
 01859 7.398e-7, 8.073e-7, 9.15e-7, 1.009e-6, 1.116e-6, 1.264e-6,  
 01860 1.439e-6, 1.644e-6, 1.856e-6, 2.147e-6, 2.317e-6, 2.713e-6,  
 01861 2.882e-6, 2.99e-6, 3.489e-6, 3.581e-6, 4.033e-6, 4.26e-6,  
 01862 4.543e-6, 4.84e-6, 4.826e-6, 5.013e-6, 5.252e-6, 5.277e-6,  
 01863 5.306e-6, 5.236e-6, 5.123e-6, 5.171e-6, 4.843e-6, 4.615e-6,  
 01864 4.385e-6, 3.97e-6, 3.693e-6, 3.231e-6, 2.915e-6, 2.495e-6,  
 01865 2.144e-6, 1.91e-6, 1.639e-6, 1.417e-6, 1.226e-6, 1.065e-6,  
 01866 9.29e-7, 8.142e-7, 7.161e-7, 6.318e-7, 5.581e-7, 4.943e-7,  
 01867 4.376e-7, 3.884e-7, 3.449e-7, 3.06e-7, 2.712e-7, 2.412e-7,  
 01868 2.139e-7, 1.903e-7, 1.689e-7, 1.499e-7, 1.331e-7, 1.183e-7,  
 01869 1.05e-7, 9.362e-8, 8.306e-8, 7.403e-8, 6.578e-8, 5.853e-8,  
 01870 5.216e-8, 4.632e-8, 4.127e-8, 3.678e-8, 3.279e-8, 2.923e-8,  
 01871 2.612e-8, 2.339e-8, 2.094e-8, 1.877e-8, 1.686e-8, 1.516e-8,  
 01872 1.366e-8, 1.234e-8, 1.114e-8, 1.012e-8, 9.182e-9, 8.362e-9,  
 01873 7.634e-9, 6.981e-9, 6.406e-9, 5.888e-9, 5.428e-9, 5.021e-9,  
 01874 4.65e-9, 4.326e-9, 4.033e-9, 3.77e-9, 3.536e-9, 3.327e-9,  
 01875 3.141e-9, 2.974e-9, 2.825e-9, 2.697e-9, 2.584e-9, 2.488e-9,  
 01876 2.406e-9, 2.34e-9, 2.292e-9, 2.259e-9, 2.244e-9, 2.243e-9,  
 01877 2.272e-9, 2.31e-9, 2.378e-9, 2.454e-9, 2.618e-9, 2.672e-9,  
 01878 2.831e-9, 3.05e-9, 3.225e-9, 3.425e-9, 3.677e-9, 3.968e-9,  
 01879 4.221e-9, 4.639e-9, 4.96e-9, 5.359e-9, 5.649e-9, 6.23e-9,  
 01880 6.716e-9, 7.218e-9, 7.746e-9, 7.988e-9, 8.627e-9, 8.999e-9,

```
01881 9.442e-9, 9.82e-9, 1.015e-8, 1.06e-8, 1.079e-8, 1.109e-8,
01882 1.137e-8, 1.186e-8, 1.18e-8, 1.187e-8, 1.194e-8, 1.192e-8,
01883 1.224e-8, 1.24e-8, 1.246e-8, 1.318e-8, 1.377e-8, 1.471e-8,
01884 1.582e-8, 1.713e-8, 1.853e-8, 2.063e-8, 2.27e-8, 2.567e-8,
01885 2.891e-8, 3.264e-8, 3.744e-8, 4.286e-8, 4.915e-8, 5.623e-8,
01886 6.336e-8, 7.293e-8, 8.309e-8, 9.319e-8, 1.091e-7, 1.243e-7,
01887 1.348e-7, 1.449e-7, 1.62e-7, 1.846e-7, 1.937e-7, 2.04e-7,
01888 2.179e-7, 2.298e-7, 2.433e-7, 2.439e-7, 2.464e-7, 2.611e-7,
01889 2.617e-7, 2.582e-7, 2.453e-7, 2.401e-7, 2.349e-7, 2.203e-7,
01890 2.066e-7, 1.939e-7, 1.78e-7, 1.558e-7, 1.391e-7, 1.203e-7,
01891 1.048e-7, 9.464e-8, 8.306e-8, 7.239e-8, 6.317e-8, 5.52e-8,
01892 4.847e-8, 4.282e-8, 3.796e-8, 3.377e-8, 2.996e-8, 2.678e-8,
01893 2.4e-8, 2.134e-8, 1.904e-8, 1.705e-8, 1.523e-8, 1.35e-8,
01894 1.204e-8, 1.07e-8, 9.408e-9, 8.476e-9, 7.47e-9, 6.679e-9,
01895 5.929e-9, 5.267e-9, 4.711e-9, 4.172e-9, 3.761e-9, 3.288e-9,
01896 2.929e-9, 2.609e-9, 2.315e-9, 2.042e-9, 1.844e-9, 1.64e-9,
01897 1.47e-9, 1.31e-9, 1.176e-9, 1.049e-9, 9.377e-10, 8.462e-10,
01898 7.616e-10, 6.854e-10, 6.191e-10, 5.596e-10, 5.078e-10, 4.611e-10,
01899 4.197e-10, 3.83e-10, 3.505e-10, 3.215e-10, 2.956e-10, 2.726e-10,
01900 2.521e-10, 2.338e-10, 2.173e-10, 2.026e-10, 1.895e-10, 1.777e-10,
01901 1.672e-10, 1.579e-10, 1.496e-10, 1.423e-10, 1.358e-10, 1.302e-10,
01902 1.254e-10, 1.216e-10, 1.187e-10, 1.163e-10, 1.147e-10, 1.145e-10,
01903 1.15e-10, 1.17e-10, 1.192e-10, 1.25e-10, 1.298e-10, 1.345e-10,
01904 1.405e-10, 1.538e-10, 1.648e-10, 1.721e-10, 1.872e-10, 1.968e-10,
01905 2.089e-10, 2.172e-10, 2.317e-10, 2.389e-10, 2.503e-10, 2.585e-10,
01906 2.686e-10, 2.8e-10, 2.895e-10, 3.019e-10, 3.037e-10, 3.076e-10,
01907 3.146e-10, 3.198e-10, 3.332e-10, 3.397e-10, 3.54e-10, 3.667e-10,
01908 3.895e-10, 4.071e-10, 4.565e-10, 4.983e-10, 5.439e-10, 5.968e-10,
01909 6.676e-10, 7.456e-10, 8.405e-10, 9.478e-10, 1.064e-9, 1.218e-9,
01910 1.386e-9, 1.581e-9, 1.787e-9, 2.032e-9, 2.347e-9, 2.677e-9,
01911 3.008e-9, 3.544e-9, 4.056e-9, 4.687e-9, 5.331e-9, 6.227e-9,
01912 6.854e-9, 8.139e-9, 9.945e-9, 9.865e-9, 1.125e-8, 1.178e-8,
01913 1.364e-8, 1.436e-8, 1.54e-8, 1.672e-8, 1.793e-8, 1.906e-8,
01914 2.036e-8, 2.144e-8, 2.292e-8, 2.371e-8, 2.493e-8, 2.606e-8,
01915 2.706e-8, 2.866e-8, 3.036e-8, 3.136e-8, 3.405e-8, 3.665e-8,
01916 3.837e-8, 4.229e-8, 4.748e-8, 5.32e-8, 5.763e-8, 6.677e-8,
01917 7.216e-8, 7.716e-8, 8.958e-8, 9.419e-8, 1.036e-7, 1.108e-7,
01918 1.189e-7, 1.246e-7, 1.348e-7, 1.31e-7, 1.361e-7, 1.364e-7,
01919 1.363e-7, 1.343e-7, 1.293e-7, 1.254e-7, 1.235e-7, 1.158e-7,
01920 1.107e-7, 9.961e-8, 9.011e-8, 7.91e-8, 6.916e-8, 6.338e-8,
01921 5.564e-8, 4.827e-8, 4.198e-8, 3.695e-8, 3.276e-8, 2.929e-8,
01922 2.633e-8, 2.391e-8, 2.192e-8, 2.021e-8, 1.89e-8, 1.772e-8,
01923 1.667e-8, 1.603e-8, 1.547e-8, 1.537e-8, 1.492e-8, 1.515e-8,
01924 1.479e-8, 1.45e-8, 1.513e-8, 1.495e-8, 1.529e-8, 1.565e-8,
01925 1.564e-8, 1.553e-8, 1.569e-8, 1.584e-8, 1.57e-8, 1.538e-8,
01926 1.513e-8, 1.472e-8, 1.425e-8, 1.349e-8, 1.328e-8, 1.249e-8,
01927 1.17e-8, 1.077e-8, 9.514e-9, 8.614e-9, 7.46e-9, 6.621e-9,
01928 5.775e-9, 5.006e-9, 4.308e-9, 3.747e-9, 3.24e-9, 2.84e-9,
01929 2.481e-9, 2.184e-9, 1.923e-9, 1.71e-9, 1.504e-9, 1.334e-9,
01930 1.187e-9, 1.053e-9, 9.367e-10, 8.306e-10, 7.419e-10, 6.63e-10,
01931 5.918e-10, 5.277e-10, 4.717e-10, 4.222e-10, 3.783e-10, 3.39e-10,
01932 3.036e-10, 2.729e-10, 2.455e-10, 2.211e-10, 1.995e-10, 1.804e-10,
01933 1.635e-10, 1.485e-10, 1.355e-10, 1.24e-10, 1.139e-10, 1.051e-10,
01934 9.757e-11, 9.114e-11, 8.577e-11, 8.139e-11, 7.792e-11, 7.52e-11,
01935 7.39e-11, 7.311e-11, 7.277e-11, 7.482e-11, 7.698e-11, 8.162e-11,
01936 8.517e-11, 8.968e-11, 9.905e-11, 1.075e-10, 1.187e-10, 1.291e-10,
01937 1.426e-10, 1.573e-10, 1.734e-10, 1.905e-10, 2.097e-10, 2.28e-10,
01938 2.473e-10, 2.718e-10, 2.922e-10, 3.128e-10, 3.361e-10, 3.641e-10,
01939 3.91e-10, 4.196e-10, 4.501e-10, 4.932e-10, 5.258e-10, 5.755e-10,
01940 6.253e-10, 6.664e-10, 7.344e-10, 7.985e-10, 8.877e-10, 1.005e-9,
01941 1.118e-9, 1.251e-9, 1.428e-9, 1.61e-9, 1.888e-9, 2.077e-9,
01942 2.331e-9, 2.751e-9, 3.061e-9, 3.522e-9, 3.805e-9, 4.181e-9,
01943 4.575e-9, 5.167e-9, 5.634e-9, 6.007e-9, 6.501e-9, 6.829e-9,
01944 7.211e-9, 7.262e-9, 7.696e-9, 7.832e-9, 7.799e-9, 7.651e-9,
01945 7.304e-9, 7.15e-9, 6.977e-9, 6.603e-9, 6.209e-9, 5.69e-9,
01946 5.432e-9, 4.764e-9, 4.189e-9, 3.64e-9, 3.203e-9, 2.848e-9,
01947 2.51e-9, 2.194e-9, 1.946e-9, 1.75e-9, 1.567e-9, 1.426e-9,
01948 1.302e-9, 1.197e-9, 1.109e-9, 1.035e-9, 9.719e-10, 9.207e-10,
01949 8.957e-10, 8.578e-10, 8.262e-10, 8.117e-10, 7.987e-10, 7.875e-10,
01950 7.741e-10, 7.762e-10, 7.537e-10, 7.424e-10, 7.474e-10, 7.294e-10,
01951 7.216e-10, 7.233e-10, 7.075e-10, 6.892e-10, 6.618e-10, 6.314e-10,
01952 6.208e-10, 5.689e-10, 5.55e-10, 4.984e-10, 4.6e-10, 4.078e-10,
01953 3.879e-10, 3.459e-10, 2.982e-10, 2.626e-10, 2.329e-10, 1.988e-10,
01954 1.735e-10, 1.487e-10, 1.297e-10, 1.133e-10, 9.943e-11, 8.736e-11,
01955 7.726e-11, 6.836e-11, 6.053e-11, 5.384e-11, 4.789e-11, 4.267e-11,
01956 3.804e-11, 3.398e-11, 3.034e-11, 2.71e-11, 2.425e-11, 2.173e-11,
01957 1.95e-11, 1.752e-11, 1.574e-11, 1.418e-11, 1.278e-11, 1.154e-11,
01958 1.044e-11, 9.463e-12, 8.602e-12, 7.841e-12, 7.171e-12, 6.584e-12,
01959 6.073e-12, 5.631e-12, 5.254e-12, 4.937e-12, 4.679e-12, 4.476e-12,
01960 4.328e-12, 4.233e-12, 4.194e-12, 4.211e-12, 4.286e-12, 4.424e-12,
01961 4.628e-12, 4.906e-12, 5.262e-12, 5.708e-12, 6.254e-12, 6.914e-12,
01962 7.714e-12, 8.677e-12, 9.747e-12, 1.101e-11, 1.256e-11, 1.409e-11,
01963 1.597e-11, 1.807e-11, 2.034e-11, 2.316e-11, 2.622e-11, 2.962e-11,
01964 3.369e-11, 3.819e-11, 4.329e-11, 4.932e-11, 5.589e-11, 6.364e-11,
01965 7.284e-11, 8.236e-11, 9.447e-11, 1.078e-10, 1.229e-10, 1.417e-10,
01966 1.614e-10, 1.843e-10, 2.107e-10, 2.406e-10, 2.728e-10, 3.195e-10,
01967 3.595e-10, 4.153e-10, 4.736e-10, 5.41e-10, 6.088e-10, 6.769e-10,
```



01968 7.691e-10, 8.545e-10, 9.621e-10, 1.047e-9, 1.161e-9, 1.296e-9,  
01969 1.424e-9, 1.576e-9, 1.739e-9, 1.893e-9, 2.08e-9, 2.336e-9,  
01970 2.604e-9, 2.76e-9, 3.001e-9, 3.365e-9, 3.55e-9, 3.895e-9,  
01971 4.183e-9, 4.614e-9, 4.846e-9, 5.068e-9, 5.427e-9, 5.541e-9,  
01972 5.864e-9, 5.997e-9, 5.997e-9, 6.061e-9, 5.944e-9, 5.855e-9,  
01973 5.661e-9, 5.523e-9, 5.374e-9, 4.94e-9, 4.688e-9, 4.17e-9,  
01974 3.913e-9, 3.423e-9, 2.997e-9, 2.598e-9, 2.253e-9, 1.946e-9,  
01975 1.71e-9, 1.507e-9, 1.336e-9, 1.19e-9, 1.068e-9, 9.623e-10,  
01976 8.772e-10, 8.007e-10, 7.42e-10, 6.884e-10, 6.483e-10, 6.162e-10,  
01977 5.922e-10, 5.688e-10, 5.654e-10, 5.637e-10, 5.701e-10, 5.781e-10,  
01978 5.874e-10, 6.268e-10, 6.357e-10, 6.525e-10, 7.137e-10, 7.441e-10,  
01979 8.024e-10, 8.485e-10, 9.143e-10, 9.536e-10, 9.717e-10, 1.018e-9,  
01980 1.042e-9, 1.054e-9, 1.092e-9, 1.079e-9, 1.064e-9, 1.043e-9,  
01981 1.02e-9, 9.687e-10, 9.273e-10, 9.208e-10, 9.068e-10, 7.687e-10,  
01982 7.385e-10, 6.595e-10, 5.87e-10, 5.144e-10, 4.417e-10, 3.804e-10,  
01983 3.301e-10, 2.866e-10, 2.509e-10, 2.202e-10, 1.947e-10, 1.719e-10,  
01984 1.525e-10, 1.361e-10, 1.21e-10, 1.084e-10, 9.8e-11, 8.801e-11,  
01985 7.954e-11, 7.124e-11, 6.335e-11, 5.76e-11, 5.132e-11, 4.601e-11,  
01986 4.096e-11, 3.657e-11, 3.25e-11, 2.909e-11, 2.587e-11, 2.297e-11,  
01987 2.05e-11, 1.828e-11, 1.632e-11, 1.462e-11, 1.314e-11, 1.185e-11,  
01988 1.073e-11, 9.76e-12, 8.922e-12, 8.206e-12, 7.602e-12, 7.1e-12,  
01989 6.694e-12, 6.378e-12, 6.149e-12, 6.004e-12, 5.941e-12, 5.962e-12,  
01990 6.069e-12, 6.265e-12, 6.551e-12, 6.935e-12, 7.457e-12, 8.074e-12,  
01991 8.811e-12, 9.852e-12, 1.086e-11, 1.207e-11, 1.361e-11, 1.553e-11,  
01992 1.737e-11, 1.93e-11, 2.175e-11, 2.41e-11, 2.706e-11, 3.023e-11,  
01993 3.313e-11, 3.657e-11, 4.118e-11, 4.569e-11, 5.025e-11, 5.66e-11,  
01994 6.231e-11, 6.881e-11, 7.996e-11, 8.526e-11, 9.694e-11, 1.106e-10,  
01995 1.222e-10, 1.355e-10, 1.525e-10, 1.775e-10, 1.924e-10, 2.181e-10,  
01996 2.379e-10, 2.662e-10, 2.907e-10, 3.154e-10, 3.366e-10, 3.579e-10,  
01997 3.858e-10, 4.046e-10, 4.196e-10, 4.166e-10, 4.457e-10, 4.466e-10,  
01998 4.404e-10, 4.337e-10, 4.15e-10, 4.083e-10, 3.91e-10, 3.723e-10,  
01999 3.514e-10, 3.303e-10, 2.847e-10, 2.546e-10, 2.23e-10, 1.994e-10,  
02000 1.733e-10, 1.488e-10, 1.297e-10, 1.144e-10, 1.004e-10, 8.741e-11,  
02001 7.928e-11, 7.034e-11, 6.323e-11, 5.754e-11, 5.25e-11, 4.85e-11,  
02002 4.502e-11, 4.286e-11, 4.028e-11, 3.899e-11, 3.824e-11, 3.761e-11,  
02003 3.804e-11, 3.839e-11, 3.845e-11, 4.244e-11, 4.382e-11, 4.582e-11,  
02004 4.847e-11, 5.209e-11, 5.384e-11, 5.887e-11, 6.371e-11, 6.737e-11,  
02005 7.168e-11, 7.415e-11, 7.827e-11, 8.037e-11, 8.12e-11, 8.071e-11,  
02006 8.008e-11, 7.851e-11, 7.544e-11, 7.377e-11, 7.173e-11, 6.801e-11,  
02007 6.267e-11, 5.727e-11, 5.288e-11, 4.853e-11, 4.082e-11, 3.645e-11,  
02008 3.136e-11, 2.672e-11, 2.304e-11, 1.986e-11, 1.725e-11, 1.503e-11,  
02009 1.315e-11, 1.153e-11, 1.014e-11, 8.942e-12, 7.901e-12, 6.993e-12,  
02010 6.199e-12, 5.502e-12, 4.89e-12, 4.351e-12, 3.878e-12, 3.461e-12,  
02011 3.094e-12, 2.771e-12, 2.488e-12, 2.241e-12, 2.025e-12, 1.838e-12,  
02012 1.677e-12, 1.541e-12, 1.427e-12, 1.335e-12, 1.262e-12, 1.209e-12,  
02013 1.176e-12, 1.161e-12, 1.165e-12, 1.189e-12, 1.234e-12, 1.3e-12,  
02014 1.389e-12, 1.503e-12, 1.644e-12, 1.814e-12, 2.017e-12, 2.255e-12,  
02015 2.534e-12, 2.858e-12, 3.231e-12, 3.661e-12, 4.153e-12, 4.717e-12,  
02016 5.36e-12, 6.094e-12, 6.93e-12, 7.882e-12, 8.966e-12, 1.02e-11,  
02017 1.162e-11, 1.324e-11, 1.51e-11, 1.72e-11, 1.965e-11, 2.237e-11,  
02018 2.56e-11, 2.927e-11, 3.371e-11, 3.842e-11, 4.429e-11, 5.139e-11,  
02019 5.798e-11, 6.697e-11, 7.626e-11, 8.647e-11, 1.022e-10, 1.136e-10,  
02020 1.3e-10, 1.481e-10, 1.672e-10, 1.871e-10, 2.126e-10, 2.357e-10,  
02021 2.583e-10, 2.997e-10, 3.289e-10, 3.702e-10, 4.012e-10, 4.319e-10,  
02022 4.527e-10, 5.001e-10, 5.448e-10, 5.611e-10, 5.76e-10, 5.965e-10,  
02023 6.079e-10, 6.207e-10, 6.276e-10, 6.222e-10, 6.137e-10, 6e-10,  
02024 5.814e-10, 5.393e-10, 5.35e-10, 4.947e-10, 4.629e-10, 4.117e-10,  
02025 3.712e-10, 3.372e-10, 2.923e-10, 2.55e-10, 2.232e-10, 1.929e-10,  
02026 1.679e-10, 1.46e-10, 1.289e-10, 1.13e-10, 9.953e-11, 8.763e-11,  
02027 7.76e-11, 6.9e-11, 6.16e-11, 5.525e-11, 4.958e-11, 4.489e-11,  
02028 4.072e-11, 3.728e-11, 3.438e-11, 3.205e-11, 3.006e-11, 2.848e-11,  
02029 2.766e-11, 2.688e-11, 2.664e-11, 2.67e-11, 2.696e-11, 2.786e-11,  
02030 2.861e-11, 3.009e-11, 3.178e-11, 3.389e-11, 3.587e-11, 3.819e-11,  
02031 4.054e-11, 4.417e-11, 4.703e-11, 5.137e-11, 5.46e-11, 6.055e-11,  
02032 6.333e-11, 6.771e-11, 7.219e-11, 7.717e-11, 8.131e-11, 8.491e-11,  
02033 8.574e-11, 9.01e-11, 9.017e-11, 8.999e-11, 8.959e-11, 8.838e-11,  
02034 8.579e-11, 8.162e-11, 8.098e-11, 7.472e-11, 7.108e-11, 6.559e-11,  
02035 5.994e-11, 5.172e-11, 4.424e-11, 3.951e-11, 3.34e-11, 2.902e-11,  
02036 2.541e-11, 2.215e-11, 1.945e-11, 1.716e-11, 1.503e-11, 1.339e-11,  
02037 1.185e-11, 1.05e-11, 9.336e-12, 8.307e-12, 7.312e-12, 6.55e-12,  
02038 5.836e-12, 5.178e-12, 4.6e-12, 4.086e-12, 3.639e-12, 3.247e-12,  
02039 2.904e-12, 2.604e-12, 2.341e-12, 2.112e-12, 1.914e-12, 1.744e-12,  
02040 1.598e-12, 1.476e-12, 1.374e-12, 1.293e-12, 1.23e-12, 1.185e-12,  
02041 1.158e-12, 1.147e-12, 1.154e-12, 1.177e-12, 1.219e-12, 1.28e-12,  
02042 1.36e-12, 1.463e-12, 1.591e-12, 1.75e-12, 1.94e-12, 2.156e-12,  
02043 2.43e-12, 2.748e-12, 3.052e-12, 3.533e-12, 3.967e-12, 4.471e-12,  
02044 5.041e-12, 5.86e-12, 6.664e-12, 7.522e-12, 8.342e-12, 9.412e-12,  
02045 1.072e-11, 1.213e-11, 1.343e-11, 1.496e-11, 1.664e-11, 1.822e-11,  
02046 2.029e-11, 2.233e-11, 2.457e-11, 2.709e-11, 2.928e-11, 3.115e-11,  
02047 3.356e-11, 3.592e-11, 3.818e-11, 3.936e-11, 4.061e-11, 4.149e-11,  
02048 4.299e-11, 4.223e-11, 4.251e-11, 4.287e-11, 4.177e-11, 4.094e-11,  
02049 3.942e-11, 3.772e-11, 3.614e-11, 3.394e-11, 3.222e-11, 2.791e-11,  
02050 2.665e-11, 2.309e-11, 2.032e-11, 1.74e-11, 1.535e-11, 1.323e-11,  
02051 1.151e-11, 9.803e-12, 8.65e-12, 7.54e-12, 6.619e-12, 5.832e-12,  
02052 5.113e-12, 4.503e-12, 3.975e-12, 3.52e-12, 3.112e-12, 2.797e-12,  
02053 2.5e-12, 2.24e-12, 2.013e-12, 1.819e-12, 1.653e-12, 1.513e-12,  
02054 1.395e-12, 1.299e-12, 1.225e-12, 1.168e-12, 1.124e-12, 1.148e-12,

```
02055     1.107e-12, 1.128e-12, 1.169e-12, 1.233e-12, 1.307e-12, 1.359e-12,
02056     1.543e-12, 1.686e-12, 1.794e-12, 2.028e-12, 2.21e-12, 2.441e-12,
02057     2.653e-12, 2.828e-12, 3.093e-12, 3.28e-12, 3.551e-12, 3.677e-12,
02058     3.803e-12, 3.844e-12, 4.068e-12, 4.093e-12, 4.002e-12, 3.904e-12,
02059     3.624e-12, 3.633e-12, 3.622e-12, 3.443e-12, 3.184e-12, 2.934e-12,
02060     2.476e-12, 2.212e-12, 1.867e-12, 1.594e-12, 1.37e-12, 1.192e-12,
02061     1.045e-12, 9.211e-13, 8.17e-13, 7.29e-13, 6.55e-13, 5.929e-13,
02062     5.415e-13, 4.995e-13, 4.661e-13, 4.406e-13, 4.225e-13, 4.116e-13,
02063     4.075e-13, 4.102e-13, 4.198e-13, 4.365e-13, 4.606e-13, 4.925e-13,
02064     5.326e-13, 5.818e-13, 6.407e-13, 7.104e-13, 7.92e-13, 8.868e-13,
02065     9.964e-13, 1.123e-12, 1.268e-12, 1.434e-12, 1.626e-12, 1.848e-12,
02066     2.107e-12, 2.422e-12, 2.772e-12, 3.145e-12, 3.704e-12, 4.27e-12,
02067     4.721e-12, 5.361e-12, 6.083e-12, 7.095e-12, 7.968e-12, 9.228e-12,
02068     1.048e-11, 1.187e-11, 1.336e-11, 1.577e-11, 1.772e-11, 2.017e-11,
02069     2.25e-11, 2.63e-11, 2.911e-11, 3.356e-11, 3.82e-11, 4.173e-11,
02070     4.811e-11, 5.254e-11, 5.839e-11, 6.187e-11, 6.805e-11, 7.118e-11,
02071     7.369e-11, 7.664e-11, 7.794e-11, 7.947e-11, 8.036e-11, 7.954e-11,
02072     7.849e-11, 7.518e-11, 7.462e-11, 6.926e-11, 6.531e-11, 6.197e-11,
02073     5.421e-11, 4.777e-11, 4.111e-11, 3.679e-11, 3.166e-11, 2.786e-11,
02074     2.436e-11, 2.144e-11, 1.859e-11, 1.628e-11, 1.414e-11, 1.237e-11,
02075     1.093e-11, 9.558e-12
02076 };
02077
02078 static double h2o260[2001] = { .2752, .2732, .2749, .2676, .2667, .2545,
02079     .2497, .2327, .2218, .2036, .1825, .1694, .1497, .1353, .121,
02080     .1014, .09405, .07848, .07195, .06246, .05306, .04853, .04138,
02081     .03735, .03171, .02785, .02431, .02111, .01845, .0164, .01405,
02082     .01255, .01098, .009797, .008646, .007779, .006898, .006099,
02083     .005453, .004909, .004413, .003959, .003581, .003199, .002871,
02084     .002583, .00233, .002086, .001874, .001684, .001512, .001361,
02085     .001225, .0011, 9.89e-4, 8.916e-4, 8.039e-4, 7.256e-4, 6.545e-4,
02086     5.918e-4, 5.359e-4, 4.867e-4, 4.426e-4, 4.033e-4, 3.682e-4,
02087     3.366e-4, 3.085e-4, 2.833e-4, 2.605e-4, 2.403e-4, 2.221e-4,
02088     2.055e-4, 1.908e-4, 1.774e-4, 1.653e-4, 1.544e-4, 1.443e-4,
02089     1.351e-4, 1.267e-4, 1.19e-4, 1.119e-4, 1.053e-4, 9.922e-5,
02090     9.355e-5, 8.831e-5, 8.339e-5, 7.878e-5, 7.449e-5, 7.043e-5,
02091     6.664e-5, 6.307e-5, 5.969e-5, 5.654e-5, 5.357e-5, 5.075e-5,
02092     4.81e-5, 4.56e-5, 4.322e-5, 4.102e-5, 3.892e-5, 3.696e-5,
02093     3.511e-5, 3.339e-5, 3.177e-5, 3.026e-5, 2.886e-5, 2.756e-5,
02094     2.636e-5, 2.527e-5, 2.427e-5, 2.337e-5, 2.257e-5, 2.185e-5,
02095     2.127e-5, 2.08e-5, 2.041e-5, 2.013e-5, 2e-5, 1.997e-5, 2.009e-5,
02096     2.031e-5, 2.068e-5, 2.124e-5, 2.189e-5, 2.267e-5, 2.364e-5,
02097     2.463e-5, 2.618e-5, 2.774e-5, 2.937e-5, 3.144e-5, 3.359e-5,
02098     3.695e-5, 4.002e-5, 4.374e-5, 4.947e-5, 5.431e-5, 6.281e-5,
02099     7.169e-5, 8.157e-5, 9.728e-5, 1.079e-4, 1.337e-4, 1.442e-4,
02100     1.683e-4, 1.879e-4, 2.223e-4, 2.425e-4, 2.838e-4, 3.143e-4,
02101     3.527e-4, 4.012e-4, 4.237e-4, 4.747e-4, 5.057e-4, 5.409e-4,
02102     5.734e-4, 5.944e-4, 6.077e-4, 6.175e-4, 6.238e-4, 6.226e-4,
02103     6.248e-4, 6.192e-4, 6.098e-4, 5.818e-4, 5.709e-4, 5.465e-4,
02104     5.043e-4, 4.699e-4, 4.294e-4, 3.984e-4, 3.672e-4, 3.152e-4,
02105     2.883e-4, 2.503e-4, 2.211e-4, 1.92e-4, 1.714e-4, 1.485e-4,
02106     1.358e-4, 1.156e-4, 1.021e-4, 8.887e-5, 7.842e-5, 7.12e-5,
02107     6.186e-5, 5.73e-5, 4.792e-5, 4.364e-5, 3.72e-5, 3.28e-5,
02108     2.946e-5, 2.591e-5, 2.261e-5, 2.048e-5, 1.813e-5, 1.63e-5,
02109     1.447e-5, 1.282e-5, 1.167e-5, 1.041e-5, 9.449e-6, 8.51e-6,
02110     7.596e-6, 6.961e-6, 6.272e-6, 5.728e-6, 5.198e-6, 4.667e-6,
02111     4.288e-6, 3.897e-6, 3.551e-6, 3.235e-6, 2.952e-6, 2.688e-6,
02112     2.449e-6, 2.241e-6, 2.05e-6, 1.879e-6, 1.722e-6, 1.582e-6,
02113     1.456e-6, 1.339e-6, 1.236e-6, 1.144e-6, 1.06e-6, 9.83e-7,
02114     9.149e-7, 8.535e-7, 7.973e-7, 7.466e-7, 6.999e-7, 6.574e-7,
02115     6.18e-7, 5.821e-7, 5.487e-7, 5.18e-7, 4.896e-7, 4.631e-7,
02116     4.386e-7, 4.16e-7, 3.945e-7, 3.748e-7, 3.562e-7, 3.385e-7,
02117     3.222e-7, 3.068e-7, 2.922e-7, 2.788e-7, 2.659e-7, 2.539e-7,
02118     2.425e-7, 2.318e-7, 2.219e-7, 2.127e-7, 2.039e-7, 1.958e-7,
02119     1.885e-7, 1.818e-7, 1.758e-7, 1.711e-7, 1.662e-7, 1.63e-7,
02120     1.605e-7, 1.58e-7, 1.559e-7, 1.545e-7, 1.532e-7, 1.522e-7,
02121     1.51e-7, 1.495e-7, 1.465e-7, 1.483e-7, 1.469e-7, 1.448e-7,
02122     1.444e-7, 1.436e-7, 1.426e-7, 1.431e-7, 1.425e-7, 1.445e-7,
02123     1.477e-7, 1.515e-7, 1.567e-7, 1.634e-7, 1.712e-7, 1.802e-7,
02124     1.914e-7, 2.024e-7, 2.159e-7, 2.295e-7, 2.461e-7, 2.621e-7,
02125     2.868e-7, 3.102e-7, 3.394e-7, 3.784e-7, 4.223e-7, 4.864e-7,
02126     5.501e-7, 6.039e-7, 7.193e-7, 7.728e-7, 9.514e-7, 1.073e-6,
02127     1.18e-6, 1.333e-6, 1.472e-6, 1.566e-6, 1.677e-6, 1.784e-6,
02128     1.904e-6, 1.953e-6, 2.02e-6, 2.074e-6, 2.128e-6, 2.162e-6,
02129     2.219e-6, 2.221e-6, 2.249e-6, 2.239e-6, 2.235e-6, 2.185e-6,
02130     2.141e-6, 2.124e-6, 2.09e-6, 2.068e-6, 2.1e-6, 2.104e-6,
02131     2.142e-6, 2.181e-6, 2.257e-6, 2.362e-6, 2.5e-6, 2.664e-6,
02132     2.884e-6, 3.189e-6, 3.48e-6, 3.847e-6, 4.313e-6, 4.79e-6,
02133     5.25e-6, 5.989e-6, 6.692e-6, 7.668e-6, 8.52e-6, 9.606e-6,
02134     1.073e-5, 1.225e-5, 1.377e-5, 1.582e-5, 1.761e-5, 2.029e-5,
02135     2.284e-5, 2.602e-5, 2.94e-5, 3.483e-5, 3.928e-5, 4.618e-5,
02136     5.24e-5, 6.132e-5, 7.183e-5, 8.521e-5, 9.111e-5, 1.07e-4,
02137     1.184e-4, 1.264e-4, 1.475e-4, 1.612e-4, 1.704e-4, 1.818e-4,
02138     1.924e-4, 1.994e-4, 2.061e-4, 2.18e-4, 2.187e-4, 2.2e-4,
02139     2.196e-4, 2.131e-4, 2.015e-4, 1.988e-4, 1.847e-4, 1.729e-4,
02140     1.597e-4, 1.373e-4, 1.262e-4, 1.087e-4, 9.439e-5, 8.061e-5,
02141     7.093e-5, 6.049e-5, 5.12e-5, 4.435e-5, 3.817e-5, 3.34e-5,
```

02142 2.927e-5, 2.573e-5, 2.291e-5, 2.04e-5, 1.827e-5, 1.636e-5,  
 02143 1.463e-5, 1.309e-5, 1.17e-5, 1.047e-5, 9.315e-6, 8.328e-6,  
 02144 7.458e-6, 6.665e-6, 5.94e-6, 5.316e-6, 4.752e-6, 4.252e-6,  
 02145 3.825e-6, 3.421e-6, 3.064e-6, 2.746e-6, 2.465e-6, 2.216e-6,  
 02146 1.99e-6, 1.79e-6, 1.609e-6, 1.449e-6, 1.306e-6, 1.177e-6,  
 02147 1.063e-6, 9.607e-7, 8.672e-7, 7.855e-7, 7.118e-7, 6.46e-7,  
 02148 5.871e-7, 5.34e-7, 4.868e-7, 4.447e-7, 4.068e-7, 3.729e-7,  
 02149 3.423e-7, 3.151e-7, 2.905e-7, 2.686e-7, 2.484e-7, 2.306e-7,  
 02150 2.142e-7, 1.995e-7, 1.86e-7, 1.738e-7, 1.626e-7, 1.522e-7,  
 02151 1.427e-7, 1.338e-7, 1.258e-7, 1.183e-7, 1.116e-7, 1.056e-7,  
 02152 9.972e-8, 9.46e-8, 9.007e-8, 8.592e-8, 8.195e-8, 7.816e-8,  
 02153 7.483e-8, 7.193e-8, 6.892e-8, 6.642e-8, 6.386e-8, 6.154e-8,  
 02154 5.949e-8, 5.764e-8, 5.622e-8, 5.479e-8, 5.364e-8, 5.301e-8,  
 02155 5.267e-8, 5.263e-8, 5.313e-8, 5.41e-8, 5.55e-8, 5.745e-8,  
 02156 6.003e-8, 6.311e-8, 6.713e-8, 7.173e-8, 7.724e-8, 8.368e-8,  
 02157 9.121e-8, 9.986e-8, 1.097e-7, 1.209e-7, 1.338e-7, 1.486e-7,  
 02158 1.651e-7, 1.837e-7, 2.048e-7, 2.289e-7, 2.557e-7, 2.857e-7,  
 02159 3.195e-7, 3.587e-7, 4.015e-7, 4.497e-7, 5.049e-7, 5.665e-7,  
 02160 6.366e-7, 7.121e-7, 7.996e-7, 8.946e-7, 1.002e-6, 1.117e-6,  
 02161 1.262e-6, 1.416e-6, 1.611e-6, 1.807e-6, 2.056e-6, 2.351e-6,  
 02162 2.769e-6, 3.138e-6, 3.699e-6, 4.386e-6, 5.041e-6, 6.074e-6,  
 02163 6.812e-6, 7.79e-6, 8.855e-6, 1.014e-5, 1.095e-5, 1.245e-5,  
 02164 1.316e-5, 1.39e-5, 1.504e-5, 1.583e-5, 1.617e-5, 1.652e-5,  
 02165 1.713e-5, 1.724e-5, 1.715e-5, 1.668e-5, 1.629e-5, 1.552e-5,  
 02166 1.478e-5, 1.34e-5, 1.245e-5, 1.121e-5, 9.575e-6, 8.956e-6,  
 02167 7.345e-6, 6.597e-6, 5.612e-6, 4.818e-6, 4.165e-6, 3.579e-6,  
 02168 3.041e-6, 2.623e-6, 2.29e-6, 1.984e-6, 1.748e-6, 1.534e-6,  
 02169 1.369e-6, 1.219e-6, 1.092e-6, 9.8e-7, 8.762e-7, 7.896e-7,  
 02170 7.104e-7, 6.364e-7, 5.691e-7, 5.107e-7, 4.575e-7, 4.09e-7,  
 02171 3.687e-7, 3.287e-7, 2.931e-7, 2.633e-7, 2.356e-7, 2.111e-7,  
 02172 1.895e-7, 1.697e-7, 1.525e-7, 1.369e-7, 1.233e-7, 1.114e-7,  
 02173 9.988e-8, 9.004e-8, 8.149e-8, 7.352e-8, 6.662e-8, 6.03e-8,  
 02174 5.479e-8, 4.974e-8, 4.532e-8, 4.129e-8, 3.781e-8, 3.462e-8,  
 02175 3.176e-8, 2.919e-8, 2.687e-8, 2.481e-8, 2.292e-8, 2.119e-8,  
 02176 1.967e-8, 1.828e-8, 1.706e-8, 1.589e-8, 1.487e-8, 1.393e-8,  
 02177 1.307e-8, 1.228e-8, 1.156e-8, 1.089e-8, 1.028e-8, 9.696e-9,  
 02178 9.159e-9, 8.658e-9, 8.187e-9, 7.746e-9, 7.34e-9, 6.953e-9,  
 02179 6.594e-9, 6.259e-9, 5.948e-9, 5.66e-9, 5.386e-9, 5.135e-9,  
 02180 4.903e-9, 4.703e-9, 4.515e-9, 4.362e-9, 4.233e-9, 4.117e-9,  
 02181 4.017e-9, 3.962e-9, 3.924e-9, 3.905e-9, 3.922e-9, 3.967e-9,  
 02182 4.046e-9, 4.165e-9, 4.32e-9, 4.522e-9, 4.769e-9, 5.083e-9,  
 02183 5.443e-9, 5.872e-9, 6.366e-9, 6.949e-9, 7.601e-9, 8.371e-9,  
 02184 9.22e-9, 1.02e-8, 1.129e-8, 1.251e-8, 1.393e-8, 1.542e-8,  
 02185 1.72e-8, 1.926e-8, 2.152e-8, 2.392e-8, 2.678e-8, 3.028e-8,  
 02186 3.39e-8, 3.836e-8, 4.309e-8, 4.9e-8, 5.481e-8, 6.252e-8,  
 02187 7.039e-8, 7.883e-8, 8.849e-8, 1.012e-7, 1.142e-7, 1.3e-7,  
 02188 1.475e-7, 1.732e-7, 1.978e-7, 2.304e-7, 2.631e-7, 2.988e-7,  
 02189 3.392e-7, 3.69e-7, 4.355e-7, 4.672e-7, 5.11e-7, 5.461e-7,  
 02190 5.828e-7, 6.233e-7, 6.509e-7, 6.672e-7, 6.969e-7, 7.104e-7,  
 02191 7.439e-7, 7.463e-7, 7.708e-7, 7.466e-7, 7.668e-7, 7.549e-7,  
 02192 7.586e-7, 7.384e-7, 7.439e-7, 7.785e-7, 7.915e-7, 8.31e-7,  
 02193 8.745e-7, 9.558e-7, 1.038e-6, 1.173e-6, 1.304e-6, 1.452e-6,  
 02194 1.671e-6, 1.931e-6, 2.239e-6, 2.578e-6, 3.032e-6, 3.334e-6,  
 02195 3.98e-6, 4.3e-6, 4.518e-6, 5.321e-6, 5.508e-6, 6.211e-6, 6.59e-6,  
 02196 7.046e-6, 7.555e-6, 7.558e-6, 7.875e-6, 8.319e-6, 8.433e-6,  
 02197 8.59e-6, 8.503e-6, 8.304e-6, 8.336e-6, 7.739e-6, 7.301e-6,  
 02198 6.827e-6, 6.078e-6, 5.551e-6, 4.762e-6, 4.224e-6, 3.538e-6,  
 02199 2.984e-6, 2.619e-6, 2.227e-6, 1.923e-6, 1.669e-6, 1.462e-6,  
 02200 1.294e-6, 1.155e-6, 1.033e-6, 9.231e-7, 8.238e-7, 7.36e-7,  
 02201 6.564e-7, 5.869e-7, 5.236e-7, 4.673e-7, 4.174e-7, 3.736e-7,  
 02202 3.33e-7, 2.976e-7, 2.657e-7, 2.367e-7, 2.106e-7, 1.877e-7,  
 02203 1.671e-7, 1.494e-7, 1.332e-7, 1.192e-7, 1.065e-7, 9.558e-8,  
 02204 8.586e-8, 7.717e-8, 6.958e-8, 6.278e-8, 5.666e-8, 5.121e-8,  
 02205 4.647e-8, 4.213e-8, 3.815e-8, 3.459e-8, 3.146e-8, 2.862e-8,  
 02206 2.604e-8, 2.375e-8, 2.162e-8, 1.981e-8, 1.817e-8, 1.67e-8,  
 02207 1.537e-8, 1.417e-8, 1.31e-8, 1.215e-8, 1.128e-8, 1.05e-8,  
 02208 9.793e-9, 9.158e-9, 8.586e-9, 8.068e-9, 7.595e-9, 7.166e-9,  
 02209 6.778e-9, 6.427e-9, 6.108e-9, 5.826e-9, 5.571e-9, 5.347e-9,  
 02210 5.144e-9, 4.968e-9, 4.822e-9, 4.692e-9, 4.589e-9, 4.506e-9,  
 02211 4.467e-9, 4.44e-9, 4.466e-9, 4.515e-9, 4.718e-9, 4.729e-9,  
 02212 4.937e-9, 5.249e-9, 5.466e-9, 5.713e-9, 6.03e-9, 6.436e-9,  
 02213 6.741e-9, 7.33e-9, 7.787e-9, 8.414e-9, 8.908e-9, 9.868e-9,  
 02214 1.069e-8, 1.158e-8, 1.253e-8, 1.3e-8, 1.409e-8, 1.47e-8,  
 02215 1.548e-8, 1.612e-8, 1.666e-8, 1.736e-8, 1.763e-8, 1.812e-8,  
 02216 1.852e-8, 1.923e-8, 1.897e-8, 1.893e-8, 1.888e-8, 1.868e-8,  
 02217 1.895e-8, 1.899e-8, 1.876e-8, 1.96e-8, 2.02e-8, 2.121e-8,  
 02218 2.239e-8, 2.379e-8, 2.526e-8, 2.766e-8, 2.994e-8, 3.332e-8,  
 02219 3.703e-8, 4.158e-8, 4.774e-8, 5.499e-8, 6.355e-8, 7.349e-8,  
 02220 8.414e-8, 9.846e-8, 1.143e-7, 1.307e-7, 1.562e-7, 1.817e-7,  
 02221 2.011e-7, 2.192e-7, 2.485e-7, 2.867e-7, 3.035e-7, 3.223e-7,  
 02222 3.443e-7, 3.617e-7, 3.793e-7, 3.793e-7, 3.839e-7, 4.081e-7,  
 02223 4.117e-7, 4.085e-7, 3.92e-7, 3.851e-7, 3.754e-7, 3.49e-7,  
 02224 3.229e-7, 2.978e-7, 2.691e-7, 2.312e-7, 2.029e-7, 1.721e-7,  
 02225 1.472e-7, 1.308e-7, 1.132e-7, 9.736e-8, 8.458e-8, 7.402e-8,  
 02226 6.534e-8, 5.811e-8, 5.235e-8, 4.762e-8, 4.293e-8, 3.896e-8,  
 02227 3.526e-8, 3.165e-8, 2.833e-8, 2.551e-8, 2.288e-8, 2.036e-8,  
 02228 1.82e-8, 1.626e-8, 1.438e-8, 1.299e-8, 1.149e-8, 1.03e-8,

```
02229 9.148e-9, 8.122e-9, 7.264e-9, 6.425e-9, 5.777e-9, 5.06e-9,
02230 4.502e-9, 4.013e-9, 3.567e-9, 3.145e-9, 2.864e-9, 2.553e-9,
02231 2.311e-9, 2.087e-9, 1.886e-9, 1.716e-9, 1.556e-9, 1.432e-9,
02232 1.311e-9, 1.202e-9, 1.104e-9, 1.013e-9, 9.293e-10, 8.493e-10,
02233 7.79e-10, 7.185e-10, 6.642e-10, 6.141e-10, 5.684e-10, 5.346e-10,
02234 5.032e-10, 4.725e-10, 4.439e-10, 4.176e-10, 3.93e-10, 3.714e-10,
02235 3.515e-10, 3.332e-10, 3.167e-10, 3.02e-10, 2.887e-10, 2.769e-10,
02236 2.665e-10, 2.578e-10, 2.503e-10, 2.436e-10, 2.377e-10, 2.342e-10,
02237 2.305e-10, 2.296e-10, 2.278e-10, 2.321e-10, 2.355e-10, 2.402e-10,
02238 2.478e-10, 2.67e-10, 2.848e-10, 2.982e-10, 3.263e-10, 3.438e-10,
02239 3.649e-10, 3.829e-10, 4.115e-10, 4.264e-10, 4.473e-10, 4.63e-10,
02240 4.808e-10, 4.995e-10, 5.142e-10, 5.313e-10, 5.318e-10, 5.358e-10,
02241 5.452e-10, 5.507e-10, 5.698e-10, 5.782e-10, 5.983e-10, 6.164e-10,
02242 6.532e-10, 6.811e-10, 7.624e-10, 8.302e-10, 9.067e-10, 9.937e-10,
02243 1.104e-9, 1.221e-9, 1.361e-9, 1.516e-9, 1.675e-9, 1.883e-9,
02244 2.101e-9, 2.349e-9, 2.614e-9, 2.92e-9, 3.305e-9, 3.724e-9,
02245 4.142e-9, 4.887e-9, 5.614e-9, 6.506e-9, 7.463e-9, 8.817e-9,
02246 9.849e-9, 1.187e-8, 1.321e-8, 1.474e-8, 1.698e-8, 1.794e-8,
02247 2.09e-8, 2.211e-8, 2.362e-8, 2.556e-8, 2.729e-8, 2.88e-8,
02248 3.046e-8, 3.167e-8, 3.367e-8, 3.457e-8, 3.59e-8, 3.711e-8,
02249 3.826e-8, 4.001e-8, 4.211e-8, 4.315e-8, 4.661e-8, 5.01e-8,
02250 5.249e-8, 5.84e-8, 6.628e-8, 7.512e-8, 8.253e-8, 9.722e-8,
02251 1.067e-7, 1.153e-7, 1.347e-7, 1.428e-7, 1.577e-7, 1.694e-7,
02252 1.833e-7, 1.938e-7, 2.108e-7, 2.059e-7, 2.157e-7, 2.185e-7,
02253 2.208e-7, 2.182e-7, 2.093e-7, 2.014e-7, 1.962e-7, 1.819e-7,
02254 1.713e-7, 1.51e-7, 1.34e-7, 1.154e-7, 9.89e-8, 8.88e-8, 7.673e-8,
02255 6.599e-8, 5.73e-8, 5.081e-8, 4.567e-8, 4.147e-8, 3.773e-8,
02256 3.46e-8, 3.194e-8, 2.953e-8, 2.759e-8, 2.594e-8, 2.442e-8,
02257 2.355e-8, 2.283e-8, 2.279e-8, 2.231e-8, 2.279e-8, 2.239e-8,
02258 2.21e-8, 2.309e-8, 2.293e-8, 2.352e-8, 2.415e-8, 2.43e-8,
02259 2.426e-8, 2.465e-8, 2.5e-8, 2.496e-8, 2.465e-8, 2.445e-8,
02260 2.383e-8, 2.299e-8, 2.165e-8, 2.113e-8, 1.968e-8, 1.819e-8,
02261 1.644e-8, 1.427e-8, 1.27e-8, 1.082e-8, 9.428e-9, 8.091e-9,
02262 6.958e-9, 5.988e-9, 5.246e-9, 4.601e-9, 4.098e-9, 3.664e-9,
02263 3.287e-9, 2.942e-9, 2.656e-9, 2.364e-9, 2.118e-9, 1.903e-9,
02264 1.703e-9, 1.525e-9, 1.365e-9, 1.229e-9, 1.107e-9, 9.96e-10,
02265 8.945e-10, 8.08e-10, 7.308e-10, 6.616e-10, 5.994e-10, 5.422e-10,
02266 4.929e-10, 4.478e-10, 4.07e-10, 3.707e-10, 3.379e-10, 3.087e-10,
02267 2.823e-10, 2.592e-10, 2.385e-10, 2.201e-10, 2.038e-10, 1.897e-10,
02268 1.774e-10, 1.667e-10, 1.577e-10, 1.502e-10, 1.437e-10, 1.394e-10,
02269 1.358e-10, 1.324e-10, 1.329e-10, 1.324e-10, 1.36e-10, 1.39e-10,
02270 1.424e-10, 1.544e-10, 1.651e-10, 1.817e-10, 1.984e-10, 2.195e-10,
02271 2.438e-10, 2.7e-10, 2.991e-10, 3.322e-10, 3.632e-10, 3.957e-10,
02272 4.36e-10, 4.701e-10, 5.03e-10, 5.381e-10, 5.793e-10, 6.19e-10,
02273 6.596e-10, 7.004e-10, 7.561e-10, 7.934e-10, 8.552e-10, 9.142e-10,
02274 9.57e-10, 1.027e-9, 1.097e-9, 1.193e-9, 1.334e-9, 1.47e-9,
02275 1.636e-9, 1.871e-9, 2.122e-9, 2.519e-9, 2.806e-9, 3.203e-9,
02276 3.846e-9, 4.362e-9, 5.114e-9, 5.643e-9, 6.305e-9, 6.981e-9,
02277 7.983e-9, 8.783e-9, 9.419e-9, 1.017e-8, 1.063e-8, 1.121e-8,
02278 1.13e-8, 1.201e-8, 1.225e-8, 1.232e-8, 1.223e-8, 1.177e-8,
02279 1.151e-8, 1.116e-8, 1.047e-8, 9.698e-9, 8.734e-9, 8.202e-9,
02280 7.041e-9, 6.074e-9, 5.172e-9, 4.468e-9, 3.913e-9, 3.414e-9,
02281 2.975e-9, 2.65e-9, 2.406e-9, 2.173e-9, 2.009e-9, 1.861e-9,
02282 1.727e-9, 1.612e-9, 1.514e-9, 1.43e-9, 1.362e-9, 1.333e-9,
02283 1.288e-9, 1.249e-9, 1.238e-9, 1.228e-9, 1.217e-9, 1.202e-9,
02284 1.209e-9, 1.177e-9, 1.157e-9, 1.165e-9, 1.142e-9, 1.131e-9,
02285 1.138e-9, 1.117e-9, 1.1e-9, 1.069e-9, 1.023e-9, 1.005e-9,
02286 9.159e-10, 8.863e-10, 7.865e-10, 7.153e-10, 6.247e-10, 5.846e-10,
02287 5.133e-10, 4.36e-10, 3.789e-10, 3.335e-10, 2.833e-10, 2.483e-10,
02288 2.155e-10, 1.918e-10, 1.709e-10, 1.529e-10, 1.374e-10, 1.235e-10,
02289 1.108e-10, 9.933e-11, 8.932e-11, 8.022e-11, 7.224e-11, 6.52e-11,
02290 5.896e-11, 5.328e-11, 4.813e-11, 4.365e-11, 3.961e-11, 3.594e-11,
02291 3.266e-11, 2.967e-11, 2.701e-11, 2.464e-11, 2.248e-11, 2.054e-11,
02292 1.878e-11, 1.721e-11, 1.579e-11, 1.453e-11, 1.341e-11, 1.241e-11,
02293 1.154e-11, 1.078e-11, 1.014e-11, 9.601e-12, 9.167e-12, 8.838e-12,
02294 8.614e-12, 8.493e-12, 8.481e-12, 8.581e-12, 8.795e-12, 9.131e-12,
02295 9.601e-12, 1.021e-11, 1.097e-11, 1.191e-11, 1.303e-11, 1.439e-11,
02296 1.601e-11, 1.778e-11, 1.984e-11, 2.234e-11, 2.474e-11, 2.766e-11,
02297 3.085e-11, 3.415e-11, 3.821e-11, 4.261e-11, 4.748e-11, 5.323e-11,
02298 5.935e-11, 6.619e-11, 7.418e-11, 8.294e-11, 9.26e-11, 1.039e-10,
02299 1.156e-10, 1.297e-10, 1.46e-10, 1.641e-10, 1.858e-10, 2.1e-10,
02300 2.383e-10, 2.724e-10, 3.116e-10, 3.538e-10, 4.173e-10, 4.727e-10,
02301 5.503e-10, 6.337e-10, 7.32e-10, 8.298e-10, 9.328e-10, 1.059e-9,
02302 1.176e-9, 1.328e-9, 1.445e-9, 1.593e-9, 1.77e-9, 1.954e-9,
02303 2.175e-9, 2.405e-9, 2.622e-9, 2.906e-9, 3.294e-9, 3.713e-9,
02304 3.98e-9, 4.384e-9, 4.987e-9, 5.311e-9, 5.874e-9, 6.337e-9,
02305 7.027e-9, 7.39e-9, 7.769e-9, 8.374e-9, 8.605e-9, 9.165e-9,
02306 9.415e-9, 9.511e-9, 9.704e-9, 9.588e-9, 9.45e-9, 9.086e-9,
02307 8.798e-9, 8.469e-9, 7.697e-9, 7.168e-9, 6.255e-9, 5.772e-9,
02308 4.97e-9, 4.271e-9, 3.653e-9, 3.154e-9, 2.742e-9, 2.435e-9,
02309 2.166e-9, 1.936e-9, 1.731e-9, 1.556e-9, 1.399e-9, 1.272e-9,
02310 1.157e-9, 1.066e-9, 9.844e-10, 9.258e-10, 8.787e-10, 8.421e-10,
02311 8.083e-10, 8.046e-10, 8.067e-10, 8.181e-10, 8.325e-10, 8.517e-10,
02312 9.151e-10, 9.351e-10, 9.677e-10, 1.071e-9, 1.126e-9, 1.219e-9,
02313 1.297e-9, 1.408e-9, 1.476e-9, 1.517e-9, 1.6e-9, 1.649e-9,
02314 1.678e-9, 1.746e-9, 1.742e-9, 1.728e-9, 1.699e-9, 1.655e-9,
02315 1.561e-9, 1.48e-9, 1.451e-9, 1.411e-9, 1.171e-9, 1.106e-9,
```

02316 9.714e-10, 8.523e-10, 7.346e-10, 6.241e-10, 5.371e-10, 4.704e-10,  
02317 4.144e-10, 3.683e-10, 3.292e-10, 2.942e-10, 2.62e-10, 2.341e-10,  
02318 2.104e-10, 1.884e-10, 1.7e-10, 1.546e-10, 1.394e-10, 1.265e-10,  
02319 1.14e-10, 1.019e-10, 9.279e-11, 8.283e-11, 7.458e-11, 6.668e-11,  
02320 5.976e-11, 5.33e-11, 4.794e-11, 4.289e-11, 3.841e-11, 3.467e-11,  
02321 3.13e-11, 2.832e-11, 2.582e-11, 2.356e-11, 2.152e-11, 1.97e-11,  
02322 1.808e-11, 1.664e-11, 1.539e-11, 1.434e-11, 1.344e-11, 1.269e-11,  
02323 1.209e-11, 1.162e-11, 1.129e-11, 1.108e-11, 1.099e-11, 1.103e-11,  
02324 1.119e-11, 1.148e-11, 1.193e-11, 1.252e-11, 1.329e-11, 1.421e-11,  
02325 1.555e-11, 1.685e-11, 1.839e-11, 2.054e-11, 2.317e-11, 2.571e-11,  
02326 2.839e-11, 3.171e-11, 3.49e-11, 3.886e-11, 4.287e-11, 4.645e-11,  
02327 5.047e-11, 5.592e-11, 6.109e-11, 6.628e-11, 7.381e-11, 8.088e-11,  
02328 8.966e-11, 1.045e-10, 1.12e-10, 1.287e-10, 1.486e-10, 1.662e-10,  
02329 1.866e-10, 2.133e-10, 2.524e-10, 2.776e-10, 3.204e-10, 3.559e-10,  
02330 4.028e-10, 4.448e-10, 4.882e-10, 5.244e-10, 5.605e-10, 6.018e-10,  
02331 6.328e-10, 6.579e-10, 6.541e-10, 7.024e-10, 7.074e-10, 7.068e-10,  
02332 7.009e-10, 6.698e-10, 6.545e-10, 6.209e-10, 5.834e-10, 5.412e-10,  
02333 5.001e-10, 4.231e-10, 3.727e-10, 3.211e-10, 2.833e-10, 2.447e-10,  
02334 2.097e-10, 1.843e-10, 1.639e-10, 1.449e-10, 1.27e-10, 1.161e-10,  
02335 1.033e-10, 9.282e-11, 8.407e-11, 7.639e-11, 7.023e-11, 6.474e-11,  
02336 6.142e-11, 5.76e-11, 5.568e-11, 5.472e-11, 5.39e-11, 5.455e-11,  
02337 5.54e-11, 5.587e-11, 6.23e-11, 6.49e-11, 6.868e-11, 7.382e-11,  
02338 8.022e-11, 8.372e-11, 9.243e-11, 1.004e-10, 1.062e-10, 1.13e-10,  
02339 1.176e-10, 1.244e-10, 1.279e-10, 1.298e-10, 1.302e-10, 1.312e-10,  
02340 1.295e-10, 1.244e-10, 1.211e-10, 1.167e-10, 1.098e-10, 9.927e-11,  
02341 8.854e-11, 8.011e-11, 7.182e-11, 5.923e-11, 5.212e-11, 4.453e-11,  
02342 3.832e-11, 3.371e-11, 2.987e-11, 2.651e-11, 2.354e-11, 2.093e-11,  
02343 1.863e-11, 1.662e-11, 1.486e-11, 1.331e-11, 1.193e-11, 1.071e-11,  
02344 9.628e-12, 8.66e-12, 7.801e-12, 7.031e-12, 6.347e-12, 5.733e-12,  
02345 5.182e-12, 4.695e-12, 4.26e-12, 3.874e-12, 3.533e-12, 3.235e-12,  
02346 2.979e-12, 2.76e-12, 2.579e-12, 2.432e-12, 2.321e-12, 2.246e-12,  
02347 2.205e-12, 2.196e-12, 2.223e-12, 2.288e-12, 2.387e-12, 2.525e-12,  
02348 2.704e-12, 2.925e-12, 3.191e-12, 3.508e-12, 3.876e-12, 4.303e-12,  
02349 4.793e-12, 5.347e-12, 5.978e-12, 6.682e-12, 7.467e-12, 8.34e-12,  
02350 9.293e-12, 1.035e-11, 1.152e-11, 1.285e-11, 1.428e-11, 1.586e-11,  
02351 1.764e-11, 1.972e-11, 2.214e-11, 2.478e-11, 2.776e-11, 3.151e-11,  
02352 3.591e-11, 4.103e-11, 4.66e-11, 5.395e-11, 6.306e-11, 7.172e-11,  
02353 8.358e-11, 9.67e-11, 1.11e-10, 1.325e-10, 1.494e-10, 1.736e-10,  
02354 2.007e-10, 2.296e-10, 2.608e-10, 3.004e-10, 3.361e-10, 3.727e-10,  
02355 4.373e-10, 4.838e-10, 5.483e-10, 6.006e-10, 6.535e-10, 6.899e-10,  
02356 7.687e-10, 8.444e-10, 8.798e-10, 9.135e-10, 9.532e-10, 9.757e-10,  
02357 9.968e-10, 1.006e-9, 9.949e-10, 9.789e-10, 9.564e-10, 9.215e-10,  
02358 8.51e-10, 8.394e-10, 7.707e-10, 7.152e-10, 6.274e-10, 5.598e-10,  
02359 5.028e-10, 4.3e-10, 3.71e-10, 3.245e-10, 2.809e-10, 2.461e-10,  
02360 2.154e-10, 1.91e-10, 1.685e-10, 1.487e-10, 1.313e-10, 1.163e-10,  
02361 1.031e-10, 9.172e-11, 8.221e-11, 7.382e-11, 6.693e-11, 6.079e-11,  
02362 5.581e-11, 5.167e-11, 4.811e-11, 4.506e-11, 4.255e-11, 4.083e-11,  
02363 3.949e-11, 3.881e-11, 3.861e-11, 3.858e-11, 3.951e-11, 4.045e-11,  
02364 4.24e-11, 4.487e-11, 4.806e-11, 5.133e-11, 5.518e-11, 5.919e-11,  
02365 6.533e-11, 7.031e-11, 7.762e-11, 8.305e-11, 9.252e-11, 9.727e-11,  
02366 1.045e-10, 1.117e-10, 1.2e-10, 1.275e-10, 1.341e-10, 1.362e-10,  
02367 1.438e-10, 1.45e-10, 1.455e-10, 1.455e-10, 1.434e-10, 1.381e-10,  
02368 1.301e-10, 1.276e-10, 1.163e-10, 1.089e-10, 9.911e-11, 8.943e-11,  
02369 7.618e-11, 6.424e-11, 5.717e-11, 4.866e-11, 4.257e-11, 3.773e-11,  
02370 3.331e-11, 2.958e-11, 2.629e-11, 2.316e-11, 2.073e-11, 1.841e-11,  
02371 1.635e-11, 1.464e-11, 1.31e-11, 1.16e-11, 1.047e-11, 9.408e-12,  
02372 8.414e-12, 7.521e-12, 6.705e-12, 5.993e-12, 5.371e-12, 4.815e-12,  
02373 4.338e-12, 3.921e-12, 3.567e-12, 3.265e-12, 3.01e-12, 2.795e-12,  
02374 2.613e-12, 2.464e-12, 2.346e-12, 2.256e-12, 2.195e-12, 2.165e-12,  
02375 2.166e-12, 2.198e-12, 2.262e-12, 2.364e-12, 2.502e-12, 2.682e-12,  
02376 2.908e-12, 3.187e-12, 3.533e-12, 3.946e-12, 4.418e-12, 5.013e-12,  
02377 5.708e-12, 6.379e-12, 7.43e-12, 8.39e-12, 9.51e-12, 1.078e-11,  
02378 1.259e-11, 1.438e-11, 1.63e-11, 1.814e-11, 2.055e-11, 2.348e-11,  
02379 2.664e-11, 2.956e-11, 3.3e-11, 3.677e-11, 4.032e-11, 4.494e-11,  
02380 4.951e-11, 5.452e-11, 6.014e-11, 6.5e-11, 6.915e-11, 7.45e-11,  
02381 7.971e-11, 8.468e-11, 8.726e-11, 8.995e-11, 9.182e-11, 9.509e-11,  
02382 9.333e-11, 9.386e-11, 9.457e-11, 9.21e-11, 9.019e-11, 8.68e-11,  
02383 8.298e-11, 7.947e-11, 7.46e-11, 7.082e-11, 6.132e-11, 5.855e-11,  
02384 5.073e-11, 4.464e-11, 3.825e-11, 3.375e-11, 2.911e-11, 2.535e-11,  
02385 2.16e-11, 1.907e-11, 1.665e-11, 1.463e-11, 1.291e-11, 1.133e-11,  
02386 9.997e-12, 8.836e-12, 7.839e-12, 6.943e-12, 6.254e-12, 5.6e-12,  
02387 5.029e-12, 4.529e-12, 4.102e-12, 3.737e-12, 3.428e-12, 3.169e-12,  
02388 2.959e-12, 2.798e-12, 2.675e-12, 2.582e-12, 2.644e-12, 2.557e-12,  
02389 2.614e-12, 2.717e-12, 2.874e-12, 3.056e-12, 3.187e-12, 3.631e-12,  
02390 3.979e-12, 4.248e-12, 4.817e-12, 5.266e-12, 5.836e-12, 6.365e-12,  
02391 6.807e-12, 7.47e-12, 7.951e-12, 8.636e-12, 8.972e-12, 9.314e-12,  
02392 9.445e-12, 1.003e-11, 1.013e-11, 9.937e-12, 9.729e-12, 9.064e-12,  
02393 9.119e-12, 9.124e-12, 8.704e-12, 8.078e-12, 7.47e-12, 6.329e-12,  
02394 5.674e-12, 4.808e-12, 4.119e-12, 3.554e-12, 3.103e-12, 2.731e-12,  
02395 2.415e-12, 2.15e-12, 1.926e-12, 1.737e-12, 1.578e-12, 1.447e-12,  
02396 1.34e-12, 1.255e-12, 1.191e-12, 1.146e-12, 1.121e-12, 1.114e-12,  
02397 1.126e-12, 1.156e-12, 1.207e-12, 1.278e-12, 1.372e-12, 1.49e-12,  
02398 1.633e-12, 1.805e-12, 2.01e-12, 2.249e-12, 2.528e-12, 2.852e-12,  
02399 3.228e-12, 3.658e-12, 4.153e-12, 4.728e-12, 5.394e-12, 6.176e-12,  
02400 7.126e-12, 8.188e-12, 9.328e-12, 1.103e-11, 1.276e-11, 1.417e-11,  
02401 1.615e-11, 1.84e-11, 2.155e-11, 2.429e-11, 2.826e-11, 3.222e-11,  
02402 3.664e-11, 4.14e-11, 4.906e-11, 5.536e-11, 6.327e-11, 7.088e-11,

```
02403      8.316e-11, 9.242e-11, 1.07e-10, 1.223e-10, 1.341e-10, 1.553e-10,
02404      1.703e-10, 1.9e-10, 2.022e-10, 2.233e-10, 2.345e-10, 2.438e-10,
02405      2.546e-10, 2.599e-10, 2.661e-10, 2.703e-10, 2.686e-10, 2.662e-10,
02406      2.56e-10, 2.552e-10, 2.378e-10, 2.252e-10, 2.146e-10, 1.885e-10,
02407      1.668e-10, 1.441e-10, 1.295e-10, 1.119e-10, 9.893e-11, 8.687e-11,
02408      7.678e-11, 6.685e-11, 5.879e-11, 5.127e-11, 4.505e-11, 3.997e-11,
02409      3.511e-11
02410  };
02411
02412  static double h2ofrn[2001] = { .01095, .01126, .01205, .01322, .0143,
02413      .01506, .01548, .01534, .01486, .01373, .01262, .01134, .01001,
02414      .008702, .007475, .006481, .00548, .0046, .003833, .00311,
02415      .002543, .002049, .00168, .001374, .001046, 8.193e-4, 6.267e-4,
02416      4.968e-4, 3.924e-4, 2.983e-4, 2.477e-4, 1.997e-4, 1.596e-4,
02417      1.331e-4, 1.061e-4, 8.942e-5, 7.168e-5, 5.887e-5, 4.848e-5,
02418      3.817e-5, 3.17e-5, 2.579e-5, 2.162e-5, 1.768e-5, 1.49e-5,
02419      1.231e-5, 1.013e-5, 8.555e-6, 7.328e-6, 6.148e-6, 5.207e-6,
02420      4.387e-6, 3.741e-6, 3.22e-6, 2.753e-6, 2.346e-6, 1.985e-6,
02421      1.716e-6, 1.475e-6, 1.286e-6, 1.122e-6, 9.661e-7, 8.284e-7,
02422      7.057e-7, 6.119e-7, 5.29e-7, 4.571e-7, 3.948e-7, 3.432e-7,
02423      2.983e-7, 2.589e-7, 2.265e-7, 1.976e-7, 1.704e-7, 1.456e-7,
02424      1.26e-7, 1.101e-7, 9.648e-8, 8.415e-8, 7.34e-8, 6.441e-8,
02425      5.643e-8, 4.94e-8, 4.276e-8, 3.703e-8, 3.227e-8, 2.825e-8,
02426      2.478e-8, 2.174e-8, 1.898e-8, 1.664e-8, 1.458e-8, 1.278e-8,
02427      1.126e-8, 9.891e-9, 8.709e-9, 7.652e-9, 6.759e-9, 5.975e-9,
02428      5.31e-9, 4.728e-9, 4.214e-9, 3.792e-9, 3.463e-9, 3.226e-9,
02429      2.992e-9, 2.813e-9, 2.749e-9, 2.809e-9, 2.913e-9, 3.037e-9,
02430      3.413e-9, 3.738e-9, 4.189e-9, 4.808e-9, 5.978e-9, 7.088e-9,
02431      8.071e-9, 9.61e-9, 1.21e-8, 1.5e-8, 1.764e-8, 2.221e-8, 2.898e-8,
02432      3.948e-8, 5.068e-8, 6.227e-8, 7.898e-8, 1.033e-7, 1.437e-7,
02433      1.889e-7, 2.589e-7, 3.59e-7, 4.971e-7, 7.156e-7, 9.983e-7,
02434      1.381e-6, 1.929e-6, 2.591e-6, 3.453e-6, 4.57e-6, 5.93e-6,
02435      7.552e-6, 9.556e-6, 1.183e-5, 1.425e-5, 1.681e-5, 1.978e-5,
02436      2.335e-5, 2.668e-5, 3.022e-5, 3.371e-5, 3.715e-5, 3.967e-5,
02437      4.06e-5, 4.01e-5, 3.809e-5, 3.491e-5, 3.155e-5, 2.848e-5,
02438      2.678e-5, 2.66e-5, 2.811e-5, 3.071e-5, 3.294e-5, 3.459e-5,
02439      3.569e-5, 3.56e-5, 3.434e-5, 3.186e-5, 2.916e-5, 2.622e-5,
02440      2.275e-5, 1.918e-5, 1.62e-5, 1.373e-5, 1.182e-5, 1.006e-5,
02441      8.556e-6, 7.26e-6, 6.107e-6, 5.034e-6, 4.211e-6, 3.426e-6,
02442      2.865e-6, 2.446e-6, 1.998e-6, 1.628e-6, 1.242e-6, 1.005e-6,
02443      7.853e-7, 6.21e-7, 5.071e-7, 4.156e-7, 3.548e-7, 2.825e-7,
02444      2.261e-7, 1.916e-7, 1.51e-7, 1.279e-7, 1.059e-7, 9.14e-8,
02445      7.707e-8, 6.17e-8, 5.311e-8, 4.263e-8, 3.518e-8, 2.961e-8,
02446      2.457e-8, 2.119e-8, 1.712e-8, 1.439e-8, 1.201e-8, 1.003e-8,
02447      8.564e-9, 7.199e-9, 6.184e-9, 5.206e-9, 4.376e-9, 3.708e-9,
02448      3.157e-9, 2.725e-9, 2.361e-9, 2.074e-9, 1.797e-9, 1.562e-9,
02449      1.364e-9, 1.196e-9, 1.042e-9, 8.862e-10, 7.648e-10, 6.544e-10,
02450      5.609e-10, 4.791e-10, 4.108e-10, 3.531e-10, 3.038e-10, 2.618e-10,
02451      2.268e-10, 1.969e-10, 1.715e-10, 1.496e-10, 1.308e-10, 1.147e-10,
02452      1.008e-10, 8.894e-11, 7.885e-11, 7.031e-11, 6.355e-11, 5.854e-11,
02453      5.534e-11, 5.466e-11, 5.725e-11, 6.447e-11, 7.943e-11, 1.038e-10,
02454      1.437e-10, 2.04e-10, 2.901e-10, 4.051e-10, 5.556e-10, 7.314e-10,
02455      9.291e-10, 1.134e-9, 1.321e-9, 1.482e-9, 1.596e-9, 1.669e-9,
02456      1.715e-9, 1.762e-9, 1.817e-9, 1.828e-9, 1.848e-9, 1.873e-9,
02457      1.902e-9, 1.894e-9, 1.864e-9, 1.841e-9, 1.797e-9, 1.704e-9,
02458      1.559e-9, 1.382e-9, 1.187e-9, 1.001e-9, 8.468e-10, 7.265e-10,
02459      6.521e-10, 6.381e-10, 6.66e-10, 7.637e-10, 9.705e-10, 1.368e-9,
02460      1.856e-9, 2.656e-9, 3.954e-9, 5.96e-9, 8.72e-9, 1.247e-8,
02461      1.781e-8, 2.491e-8, 3.311e-8, 4.272e-8, 5.205e-8, 6.268e-8,
02462      7.337e-8, 8.277e-8, 9.185e-8, 1.004e-7, 1.091e-7, 1.159e-7,
02463      1.188e-7, 1.175e-7, 1.124e-7, 1.033e-7, 9.381e-8, 8.501e-8,
02464      7.956e-8, 7.894e-8, 8.331e-8, 9.102e-8, 9.836e-8, 1.035e-7,
02465      1.064e-7, 1.06e-7, 1.032e-7, 9.808e-8, 9.139e-8, 8.442e-8,
02466      7.641e-8, 6.881e-8, 6.161e-8, 5.404e-8, 4.804e-8, 4.446e-8,
02467      4.328e-8, 4.259e-8, 4.421e-8, 4.673e-8, 4.985e-8, 5.335e-8,
02468      5.796e-8, 6.542e-8, 7.714e-8, 8.827e-8, 1.04e-7, 1.238e-7,
02469      1.499e-7, 1.829e-7, 2.222e-7, 2.689e-7, 3.303e-7, 3.981e-7,
02470      4.84e-7, 5.91e-7, 7.363e-7, 9.087e-7, 1.139e-6, 1.455e-6,
02471      1.866e-6, 2.44e-6, 3.115e-6, 3.941e-6, 4.891e-6, 5.992e-6,
02472      7.111e-6, 8.296e-6, 9.21e-6, 9.987e-6, 1.044e-5, 1.073e-5,
02473      1.092e-5, 1.106e-5, 1.138e-5, 1.171e-5, 1.186e-5, 1.186e-5,
02474      1.179e-5, 1.166e-5, 1.151e-5, 1.16e-5, 1.197e-5, 1.241e-5,
02475      1.268e-5, 1.26e-5, 1.184e-5, 1.063e-5, 9.204e-6, 7.584e-6,
02476      6.053e-6, 4.482e-6, 3.252e-6, 2.337e-6, 1.662e-6, 1.18e-6,
02477      8.15e-7, 5.95e-7, 4.354e-7, 3.302e-7, 2.494e-7, 1.93e-7,
02478      1.545e-7, 1.25e-7, 1.039e-7, 8.602e-8, 7.127e-8, 5.897e-8,
02479      4.838e-8, 4.018e-8, 3.28e-8, 2.72e-8, 2.307e-8, 1.972e-8,
02480      1.654e-8, 1.421e-8, 1.174e-8, 1.004e-8, 8.739e-9, 7.358e-9,
02481      6.242e-9, 5.303e-9, 4.567e-9, 3.94e-9, 3.375e-9, 2.864e-9,
02482      2.422e-9, 2.057e-9, 1.75e-9, 1.505e-9, 1.294e-9, 1.101e-9,
02483      9.401e-10, 8.018e-10, 6.903e-10, 5.965e-10, 5.087e-10, 4.364e-10,
02484      3.759e-10, 3.247e-10, 2.809e-10, 2.438e-10, 2.123e-10, 1.853e-10,
02485      1.622e-10, 1.426e-10, 1.26e-10, 1.125e-10, 1.022e-10, 9.582e-11,
02486      9.388e-11, 9.801e-11, 1.08e-10, 1.276e-10, 1.551e-10, 1.903e-10,
02487      2.291e-10, 2.724e-10, 3.117e-10, 3.4e-10, 3.562e-10, 3.625e-10,
02488      3.619e-10, 3.429e-10, 3.221e-10, 2.943e-10, 2.645e-10, 2.338e-10,
02489      2.062e-10, 1.901e-10, 1.814e-10, 1.827e-10, 1.906e-10, 1.984e-10,
```

02490 2.04e-10, 2.068e-10, 2.075e-10, 2.018e-10, 1.959e-10, 1.897e-10,  
02491 1.852e-10, 1.791e-10, 1.696e-10, 1.634e-10, 1.598e-10, 1.561e-10,  
02492 1.518e-10, 1.443e-10, 1.377e-10, 1.346e-10, 1.342e-10, 1.375e-10,  
02493 1.525e-10, 1.767e-10, 2.108e-10, 2.524e-10, 2.981e-10, 3.477e-10,  
02494 4.262e-10, 5.326e-10, 6.646e-10, 8.321e-10, 1.069e-9, 1.386e-9,  
02495 1.743e-9, 2.216e-9, 2.808e-9, 3.585e-9, 4.552e-9, 5.907e-9,  
02496 7.611e-9, 9.774e-9, 1.255e-8, 1.666e-8, 2.279e-8, 3.221e-8,  
02497 4.531e-8, 6.4e-8, 9.187e-8, 1.295e-7, 1.825e-7, 2.431e-7,  
02498 3.181e-7, 4.009e-7, 4.941e-7, 5.88e-7, 6.623e-7, 7.155e-7,  
02499 7.451e-7, 7.594e-7, 7.541e-7, 7.467e-7, 7.527e-7, 7.935e-7,  
02500 8.461e-7, 8.954e-7, 9.364e-7, 9.843e-7, 1.024e-6, 1.05e-6,  
02501 1.059e-6, 1.074e-6, 1.072e-6, 1.043e-6, 9.789e-7, 8.803e-7,  
02502 7.662e-7, 6.378e-7, 5.133e-7, 3.958e-7, 2.914e-7, 2.144e-7,  
02503 1.57e-7, 1.14e-7, 8.47e-8, 6.2e-8, 4.657e-8, 3.559e-8, 2.813e-8,  
02504 2.222e-8, 1.769e-8, 1.391e-8, 1.125e-8, 9.186e-9, 7.704e-9,  
02505 6.447e-9, 5.381e-9, 4.442e-9, 3.669e-9, 3.057e-9, 2.564e-9,  
02506 2.153e-9, 1.784e-9, 1.499e-9, 1.281e-9, 1.082e-9, 9.304e-10,  
02507 8.169e-10, 6.856e-10, 5.866e-10, 5.043e-10, 4.336e-10, 3.731e-10,  
02508 3.175e-10, 2.745e-10, 2.374e-10, 2.007e-10, 1.737e-10, 1.508e-10,  
02509 1.302e-10, 1.13e-10, 9.672e-11, 8.375e-11, 7.265e-11, 6.244e-11,  
02510 5.343e-11, 4.654e-11, 3.975e-11, 3.488e-11, 3.097e-11, 2.834e-11,  
02511 2.649e-11, 2.519e-11, 2.462e-11, 2.443e-11, 2.44e-11, 2.398e-11,  
02512 2.306e-11, 2.183e-11, 2.021e-11, 1.821e-11, 1.599e-11, 1.403e-11,  
02513 1.196e-11, 1.023e-11, 8.728e-12, 7.606e-12, 6.941e-12, 6.545e-12,  
02514 6.484e-12, 6.6e-12, 6.718e-12, 6.785e-12, 6.746e-12, 6.724e-12,  
02515 6.764e-12, 6.995e-12, 7.144e-12, 7.32e-12, 7.33e-12, 7.208e-12,  
02516 6.789e-12, 6.09e-12, 5.337e-12, 4.62e-12, 4.037e-12, 3.574e-12,  
02517 3.311e-12, 3.346e-12, 3.566e-12, 3.836e-12, 4.076e-12, 4.351e-12,  
02518 4.691e-12, 5.114e-12, 5.427e-12, 6.167e-12, 7.436e-12, 8.842e-12,  
02519 1.038e-11, 1.249e-11, 1.54e-11, 1.915e-11, 2.48e-11, 3.256e-11,  
02520 4.339e-11, 5.611e-11, 7.519e-11, 1.037e-10, 1.409e-10, 1.883e-10,  
02521 2.503e-10, 3.38e-10, 4.468e-10, 5.801e-10, 7.335e-10, 8.98e-10,  
02522 1.11e-9, 1.363e-9, 1.677e-9, 2.104e-9, 2.681e-9, 3.531e-9,  
02523 4.621e-9, 6.106e-9, 8.154e-9, 1.046e-8, 1.312e-8, 1.607e-8,  
02524 1.948e-8, 2.266e-8, 2.495e-8, 2.655e-8, 2.739e-8, 2.739e-8,  
02525 2.662e-8, 2.589e-8, 2.59e-8, 2.664e-8, 2.833e-8, 3.023e-8,  
02526 3.305e-8, 3.558e-8, 3.793e-8, 3.961e-8, 4.056e-8, 4.102e-8,  
02527 4.025e-8, 3.917e-8, 3.706e-8, 3.493e-8, 3.249e-8, 3.096e-8,  
02528 3.011e-8, 3.11e-8, 3.395e-8, 3.958e-8, 4.875e-8, 6.066e-8,  
02529 7.915e-8, 1.011e-7, 1.3e-7, 1.622e-7, 2.003e-7, 2.448e-7,  
02530 2.863e-7, 3.317e-7, 3.655e-7, 3.96e-7, 4.098e-7, 4.168e-7,  
02531 4.198e-7, 4.207e-7, 4.289e-7, 4.384e-7, 4.471e-7, 4.524e-7,  
02532 4.574e-7, 4.633e-7, 4.785e-7, 5.028e-7, 5.371e-7, 5.727e-7,  
02533 5.955e-7, 5.998e-7, 5.669e-7, 5.082e-7, 4.397e-7, 3.596e-7,  
02534 2.814e-7, 2.074e-7, 1.486e-7, 1.057e-7, 7.25e-8, 4.946e-8,  
02535 3.43e-8, 2.447e-8, 1.793e-8, 1.375e-8, 1.096e-8, 9.091e-9,  
02536 7.709e-9, 6.631e-9, 5.714e-9, 4.886e-9, 4.205e-9, 3.575e-9,  
02537 3.07e-9, 2.631e-9, 2.284e-9, 2.002e-9, 1.745e-9, 1.509e-9,  
02538 1.284e-9, 1.084e-9, 9.163e-10, 7.663e-10, 6.346e-10, 5.283e-10,  
02539 4.354e-10, 3.59e-10, 2.982e-10, 2.455e-10, 2.033e-10, 1.696e-10,  
02540 1.432e-10, 1.211e-10, 1.02e-10, 8.702e-11, 7.38e-11, 6.293e-11,  
02541 5.343e-11, 4.532e-11, 3.907e-11, 3.365e-11, 2.945e-11, 2.558e-11,  
02542 2.192e-11, 1.895e-11, 1.636e-11, 1.42e-11, 1.228e-11, 1.063e-11,  
02543 9.348e-12, 8.2e-12, 7.231e-12, 6.43e-12, 5.702e-12, 5.052e-12,  
02544 4.469e-12, 4e-12, 3.679e-12, 3.387e-12, 3.197e-12, 3.158e-12,  
02545 3.327e-12, 3.675e-12, 4.292e-12, 5.437e-12, 7.197e-12, 1.008e-11,  
02546 1.437e-11, 2.035e-11, 2.905e-11, 4.062e-11, 5.528e-11, 7.177e-11,  
02547 9.064e-11, 1.109e-10, 1.297e-10, 1.473e-10, 1.652e-10, 1.851e-10,  
02548 2.079e-10, 2.313e-10, 2.619e-10, 2.958e-10, 3.352e-10, 3.796e-10,  
02549 4.295e-10, 4.923e-10, 5.49e-10, 5.998e-10, 6.388e-10, 6.645e-10,  
02550 6.712e-10, 6.549e-10, 6.38e-10, 6.255e-10, 6.253e-10, 6.459e-10,  
02551 6.977e-10, 7.59e-10, 8.242e-10, 8.92e-10, 9.403e-10, 9.701e-10,  
02552 9.483e-10, 9.135e-10, 8.617e-10, 7.921e-10, 7.168e-10, 6.382e-10,  
02553 5.677e-10, 5.045e-10, 4.572e-10, 4.312e-10, 4.145e-10, 4.192e-10,  
02554 4.541e-10, 5.368e-10, 6.771e-10, 8.962e-10, 1.21e-9, 1.659e-9,  
02555 2.33e-9, 3.249e-9, 4.495e-9, 5.923e-9, 7.642e-9, 9.607e-9,  
02556 1.178e-8, 1.399e-8, 1.584e-8, 1.73e-8, 1.816e-8, 1.87e-8,  
02557 1.868e-8, 1.87e-8, 1.884e-8, 1.99e-8, 2.15e-8, 2.258e-8,  
02558 2.364e-8, 2.473e-8, 2.602e-8, 2.689e-8, 2.731e-8, 2.816e-8,  
02559 2.859e-8, 2.839e-8, 2.703e-8, 2.451e-8, 2.149e-8, 1.787e-8,  
02560 1.449e-8, 1.111e-8, 8.282e-9, 6.121e-9, 4.494e-9, 3.367e-9,  
02561 2.487e-9, 1.885e-9, 1.503e-9, 1.249e-9, 1.074e-9, 9.427e-10,  
02562 8.439e-10, 7.563e-10, 6.772e-10, 6.002e-10, 5.254e-10, 4.588e-10,  
02563 3.977e-10, 3.449e-10, 3.003e-10, 2.624e-10, 2.335e-10, 2.04e-10,  
02564 1.771e-10, 1.534e-10, 1.296e-10, 1.097e-10, 9.173e-11, 7.73e-11,  
02565 6.547e-11, 5.191e-11, 4.198e-11, 3.361e-11, 2.732e-11, 2.244e-11,  
02566 1.791e-11, 1.509e-11, 1.243e-11, 1.035e-11, 8.969e-12, 7.394e-12,  
02567 6.323e-12, 5.282e-12, 4.543e-12, 3.752e-12, 3.14e-12, 2.6e-12,  
02568 2.194e-12, 1.825e-12, 1.511e-12, 1.245e-12, 1.024e-12, 8.539e-13,  
02569 7.227e-13, 6.102e-13, 5.189e-13, 4.43e-13, 3.774e-13, 3.236e-13,  
02570 2.8e-13, 2.444e-13, 2.156e-13, 1.932e-13, 1.775e-13, 1.695e-13,  
02571 1.672e-13, 1.704e-13, 1.825e-13, 2.087e-13, 2.614e-13, 3.377e-13,  
02572 4.817e-13, 6.989e-13, 1.062e-12, 1.562e-12, 2.288e-12, 3.295e-12,  
02573 4.55e-12, 5.965e-12, 7.546e-12, 9.395e-12, 1.103e-11, 1.228e-11,  
02574 1.318e-11, 1.38e-11, 1.421e-11, 1.39e-11, 1.358e-11, 1.336e-11,  
02575 1.342e-11, 1.356e-11, 1.424e-11, 1.552e-11, 1.73e-11, 1.951e-11,  
02576 2.128e-11, 2.249e-11, 2.277e-11, 2.226e-11, 2.111e-11, 1.922e-11,

```
02577 1.775e-11, 1.661e-11, 1.547e-11, 1.446e-11, 1.323e-11, 1.21e-11,
02578 1.054e-11, 9.283e-12, 8.671e-12, 8.67e-12, 9.429e-12, 1.062e-11,
02579 1.255e-11, 1.506e-11, 1.818e-11, 2.26e-11, 2.831e-11, 3.723e-11,
02580 5.092e-11, 6.968e-11, 9.826e-11, 1.349e-10, 1.87e-10, 2.58e-10,
02581 3.43e-10, 4.424e-10, 5.521e-10, 6.812e-10, 8.064e-10, 9.109e-10,
02582 9.839e-10, 1.028e-9, 1.044e-9, 1.029e-9, 1.005e-9, 1.002e-9,
02583 1.038e-9, 1.122e-9, 1.233e-9, 1.372e-9, 1.524e-9, 1.665e-9,
02584 1.804e-9, 1.908e-9, 2.015e-9, 2.117e-9, 2.219e-9, 2.336e-9,
02585 2.531e-9, 2.805e-9, 3.189e-9, 3.617e-9, 4.208e-9, 4.911e-9,
02586 5.619e-9, 6.469e-9, 7.188e-9, 7.957e-9, 8.503e-9, 9.028e-9,
02587 9.571e-9, 9.99e-9, 1.055e-8, 1.102e-8, 1.132e-8, 1.141e-8,
02588 1.145e-8, 1.145e-8, 1.176e-8, 1.224e-8, 1.304e-8, 1.388e-8,
02589 1.445e-8, 1.453e-8, 1.368e-8, 1.22e-8, 1.042e-8, 8.404e-9,
02590 6.403e-9, 4.643e-9, 3.325e-9, 2.335e-9, 1.638e-9, 1.19e-9,
02591 9.161e-10, 7.412e-10, 6.226e-10, 5.516e-10, 5.068e-10, 4.831e-10,
02592 4.856e-10, 5.162e-10, 5.785e-10, 6.539e-10, 7.485e-10, 8.565e-10,
02593 9.534e-10, 1.052e-9, 1.115e-9, 1.173e-9, 1.203e-9, 1.224e-9,
02594 1.243e-9, 1.248e-9, 1.261e-9, 1.265e-9, 1.25e-9, 1.217e-9,
02595 1.176e-9, 1.145e-9, 1.153e-9, 1.199e-9, 1.278e-9, 1.366e-9,
02596 1.426e-9, 1.444e-9, 1.365e-9, 1.224e-9, 1.051e-9, 8.539e-10,
02597 6.564e-10, 4.751e-10, 3.404e-10, 2.377e-10, 1.631e-10, 1.114e-10,
02598 7.87e-11, 5.793e-11, 4.284e-11, 3.3e-11, 2.62e-11, 2.152e-11,
02599 1.777e-11, 1.496e-11, 1.242e-11, 1.037e-11, 8.725e-12, 7.004e-12,
02600 5.718e-12, 4.769e-12, 3.952e-12, 3.336e-12, 2.712e-12, 2.213e-12,
02601 1.803e-12, 1.492e-12, 1.236e-12, 1.006e-12, 8.384e-13, 7.063e-13,
02602 5.879e-13, 4.93e-13, 4.171e-13, 3.569e-13, 3.083e-13, 2.688e-13,
02603 2.333e-13, 2.035e-13, 1.82e-13, 1.682e-13, 1.635e-13, 1.674e-13,
02604 1.769e-13, 2.022e-13, 2.485e-13, 3.127e-13, 4.25e-13, 5.928e-13,
02605 8.514e-13, 1.236e-12, 1.701e-12, 2.392e-12, 3.231e-12, 4.35e-12,
02606 5.559e-12, 6.915e-12, 8.519e-12, 1.013e-11, 1.146e-11, 1.24e-11,
02607 1.305e-11, 1.333e-11, 1.318e-11, 1.263e-11, 1.238e-11, 1.244e-11,
02608 1.305e-11, 1.432e-11, 1.623e-11, 1.846e-11, 2.09e-11, 2.328e-11,
02609 2.526e-11, 2.637e-11, 2.702e-11, 2.794e-11, 2.889e-11, 2.989e-11,
02610 3.231e-11, 3.68e-11, 4.375e-11, 5.504e-11, 7.159e-11, 9.502e-11,
02611 1.279e-10, 1.645e-10, 2.098e-10, 2.618e-10, 3.189e-10, 3.79e-10,
02612 4.303e-10, 4.753e-10, 5.027e-10, 5.221e-10, 5.293e-10, 5.346e-10,
02613 5.467e-10, 5.796e-10, 6.2e-10, 6.454e-10, 6.705e-10, 6.925e-10,
02614 7.233e-10, 7.35e-10, 7.538e-10, 7.861e-10, 8.077e-10, 8.132e-10,
02615 7.749e-10, 7.036e-10, 6.143e-10, 5.093e-10, 4.089e-10, 3.092e-10,
02616 2.299e-10, 1.705e-10, 1.277e-10, 9.723e-11, 7.533e-11, 6.126e-11,
02617 5.154e-11, 4.428e-11, 3.913e-11, 3.521e-11, 3.297e-11, 3.275e-11,
02618 3.46e-11, 3.798e-11, 4.251e-11, 4.745e-11, 5.232e-11, 5.606e-11,
02619 5.82e-11, 5.88e-11, 5.79e-11, 5.661e-11, 5.491e-11, 5.366e-11,
02620 5.341e-11, 5.353e-11, 5.336e-11, 5.293e-11, 5.248e-11, 5.235e-11,
02621 5.208e-11, 5.322e-11, 5.521e-11, 5.725e-11, 5.827e-11, 5.685e-11,
02622 5.245e-11, 4.612e-11, 3.884e-11, 3.129e-11, 2.404e-11, 1.732e-11,
02623 1.223e-11, 8.574e-12, 5.888e-12, 3.986e-12, 2.732e-12, 1.948e-12,
02624 1.414e-12, 1.061e-12, 8.298e-13, 6.612e-13, 5.413e-13, 4.472e-13,
02625 3.772e-13, 3.181e-13, 2.645e-13, 2.171e-13, 1.778e-13, 1.464e-13,
02626 1.183e-13, 9.637e-14, 7.991e-14, 6.668e-14, 5.57e-14, 4.663e-14,
02627 3.848e-14, 3.233e-14, 2.706e-14, 2.284e-14, 1.944e-14, 1.664e-14,
02628 1.43e-14, 1.233e-14, 1.066e-14, 9.234e-15, 8.023e-15, 6.993e-15,
02629 6.119e-15, 5.384e-15, 4.774e-15, 4.283e-15, 3.916e-15, 3.695e-15,
02630 3.682e-15, 4.004e-15, 4.912e-15, 6.853e-15, 1.056e-14, 1.712e-14,
02631 2.804e-14, 4.516e-14, 7.113e-14, 1.084e-13, 1.426e-13, 1.734e-13,
02632 1.978e-13, 2.194e-13, 2.388e-13, 2.489e-13, 2.626e-13, 2.865e-13,
02633 3.105e-13, 3.387e-13, 3.652e-13, 3.984e-13, 4.398e-13, 4.906e-13,
02634 5.55e-13, 6.517e-13, 7.813e-13, 9.272e-13, 1.164e-12, 1.434e-12,
02635 1.849e-12, 2.524e-12, 3.328e-12, 4.523e-12, 6.108e-12, 8.207e-12,
02636 1.122e-11, 1.477e-11, 1.9e-11, 2.412e-11, 2.984e-11, 3.68e-11,
02637 4.353e-11, 4.963e-11, 5.478e-11, 5.903e-11, 6.233e-11, 6.483e-11,
02638 6.904e-11, 7.569e-11, 8.719e-11, 1.048e-10, 1.278e-10, 1.557e-10,
02639 1.869e-10, 2.218e-10, 2.61e-10, 2.975e-10, 3.371e-10, 3.746e-10,
02640 4.065e-10, 4.336e-10, 4.503e-10, 4.701e-10, 4.8e-10, 4.917e-10,
02641 5.038e-10, 5.128e-10, 5.143e-10, 5.071e-10, 5.019e-10, 5.025e-10,
02642 5.183e-10, 5.496e-10, 5.877e-10, 6.235e-10, 6.42e-10, 6.234e-10,
02643 5.698e-10, 4.916e-10, 4.022e-10, 3.126e-10, 2.282e-10, 1.639e-10,
02644 1.142e-10, 7.919e-11, 5.69e-11, 4.313e-11, 3.413e-11, 2.807e-11,
02645 2.41e-11, 2.166e-11, 2.024e-11, 1.946e-11, 1.929e-11, 1.963e-11,
02646 2.035e-11, 2.162e-11, 2.305e-11, 2.493e-11, 2.748e-11, 3.048e-11,
02647 3.413e-11, 3.754e-11, 4.155e-11, 4.635e-11, 5.11e-11, 5.734e-11,
02648 6.338e-11, 6.99e-11, 7.611e-11, 8.125e-11, 8.654e-11, 8.951e-11,
02649 9.182e-11, 9.31e-11, 9.273e-11, 9.094e-11, 8.849e-11, 8.662e-11,
02650 8.67e-11, 8.972e-11, 9.566e-11, 1.025e-10, 1.083e-10, 1.111e-10,
02651 1.074e-10, 9.771e-11, 8.468e-11, 6.958e-11, 5.47e-11, 4.04e-11,
02652 2.94e-11, 2.075e-11, 1.442e-11, 1.01e-11, 7.281e-12, 5.409e-12,
02653 4.138e-12, 3.304e-12, 2.784e-12, 2.473e-12, 2.273e-12, 2.186e-12,
02654 2.118e-12, 2.066e-12, 1.958e-12, 1.818e-12, 1.675e-12, 1.509e-12,
02655 1.349e-12, 1.171e-12, 9.838e-13, 8.213e-13, 6.765e-13, 5.378e-13,
02656 4.161e-13, 3.119e-13, 2.279e-13, 1.637e-13, 1.152e-13, 8.112e-14,
02657 5.919e-14, 4.47e-14, 3.492e-14, 2.811e-14, 2.319e-14, 1.948e-14,
02658 1.66e-14, 1.432e-14, 1.251e-14, 1.109e-14, 1.006e-14, 9.45e-15,
02659 9.384e-15, 1.012e-14, 1.216e-14, 1.636e-14, 2.305e-14, 3.488e-14,
02660 5.572e-14, 8.479e-14, 1.265e-13, 1.905e-13, 2.73e-13, 3.809e-13,
02661 4.955e-13, 6.303e-13, 7.861e-13, 9.427e-13, 1.097e-12, 1.212e-12,
02662 1.328e-12, 1.415e-12, 1.463e-12, 1.495e-12, 1.571e-12, 1.731e-12,
02663 1.981e-12, 2.387e-12, 2.93e-12, 3.642e-12, 4.584e-12, 5.822e-12,
```



```

02664 7.278e-12, 9.193e-12, 1.135e-11, 1.382e-11, 1.662e-11, 1.958e-11,
02665 2.286e-11, 2.559e-11, 2.805e-11, 2.988e-11, 3.106e-11, 3.182e-11,
02666 3.2e-11, 3.258e-11, 3.362e-11, 3.558e-11, 3.688e-11, 3.8e-11,
02667 3.929e-11, 4.062e-11, 4.186e-11, 4.293e-11, 4.48e-11, 4.643e-11,
02668 4.704e-11, 4.571e-11, 4.206e-11, 3.715e-11, 3.131e-11, 2.541e-11,
02669 1.978e-11, 1.508e-11, 1.146e-11, 8.7e-12, 6.603e-12, 5.162e-12,
02670 4.157e-12, 3.408e-12, 2.829e-12, 2.405e-12, 2.071e-12, 1.826e-12,
02671 1.648e-12, 1.542e-12, 1.489e-12, 1.485e-12, 1.493e-12, 1.545e-12,
02672 1.637e-12, 1.814e-12, 2.061e-12, 2.312e-12, 2.651e-12, 3.03e-12,
02673 3.46e-12, 3.901e-12, 4.306e-12, 4.721e-12, 5.008e-12, 5.281e-12,
02674 5.541e-12, 5.791e-12, 6.115e-12, 6.442e-12, 6.68e-12, 6.791e-12,
02675 6.831e-12, 6.839e-12, 6.946e-12, 7.128e-12, 7.537e-12, 8.036e-12,
02676 8.392e-12, 8.526e-12, 8.11e-12, 7.325e-12, 6.329e-12, 5.183e-12,
02677 4.081e-12, 2.985e-12, 2.141e-12, 1.492e-12, 1.015e-12, 6.684e-13,
02678 4.414e-13, 2.987e-13, 2.038e-13, 1.391e-13, 9.86e-14, 7.24e-14,
02679 5.493e-14, 4.288e-14, 3.427e-14, 2.787e-14, 2.296e-14, 1.909e-14,
02680 1.598e-14, 1.344e-14, 1.135e-14, 9.616e-15, 8.169e-15, 6.957e-15,
02681 5.938e-15, 5.08e-15, 4.353e-15, 3.738e-15, 3.217e-15, 2.773e-15,
02682 2.397e-15, 2.077e-15, 1.805e-15, 1.575e-15, 1.382e-15, 1.221e-15,
02683 1.09e-15, 9.855e-16, 9.068e-16, 8.537e-16, 8.27e-16, 8.29e-16,
02684 8.634e-16, 9.359e-16, 1.055e-15, 1.233e-15, 1.486e-15, 1.839e-15,
02685 2.326e-15, 2.998e-15, 3.934e-15, 5.256e-15, 7.164e-15, 9.984e-15,
02686 1.427e-14, 2.099e-14, 3.196e-14, 5.121e-14, 7.908e-14, 1.131e-13,
02687 1.602e-13, 2.239e-13, 3.075e-13, 4.134e-13, 5.749e-13, 7.886e-13,
02688 1.071e-12, 1.464e-12, 2.032e-12, 2.8e-12, 3.732e-12, 4.996e-12,
02689 6.483e-12, 8.143e-12, 1.006e-11, 1.238e-11, 1.484e-11, 1.744e-11,
02690 2.02e-11, 2.274e-11, 2.562e-11, 2.848e-11, 3.191e-11, 3.617e-11,
02691 4.081e-11, 4.577e-11, 4.937e-11, 5.204e-11, 5.401e-11, 5.462e-11,
02692 5.507e-11, 5.51e-11, 5.605e-11, 5.686e-11, 5.739e-11, 5.766e-11,
02693 5.74e-11, 5.754e-11, 5.761e-11, 5.777e-11, 5.712e-11, 5.51e-11,
02694 5.088e-11, 4.438e-11, 3.728e-11, 2.994e-11, 2.305e-11, 1.715e-11,
02695 1.256e-11, 9.208e-12, 6.745e-12, 5.014e-12, 3.785e-12, 2.9e-12,
02696 2.239e-12, 1.757e-12, 1.414e-12, 1.142e-12, 9.482e-13, 8.01e-13,
02697 6.961e-13, 6.253e-13, 5.735e-13, 5.433e-13, 5.352e-13, 5.493e-13,
02698 5.706e-13, 6.068e-13, 6.531e-13, 7.109e-13, 7.767e-13, 8.59e-13,
02699 9.792e-13, 1.142e-12, 1.371e-12, 1.65e-12, 1.957e-12, 2.302e-12,
02700 2.705e-12, 3.145e-12, 3.608e-12, 4.071e-12, 4.602e-12, 5.133e-12,
02701 5.572e-12, 5.987e-12, 6.248e-12, 6.533e-12, 6.757e-12, 6.935e-12,
02702 7.224e-12, 7.422e-12, 7.538e-12, 7.547e-12, 7.495e-12, 7.543e-12,
02703 7.725e-12, 8.139e-12, 8.627e-12, 9.146e-12, 9.443e-12, 9.318e-12,
02704 8.649e-12, 7.512e-12, 6.261e-12, 4.915e-12, 3.647e-12, 2.597e-12,
02705 1.785e-12, 1.242e-12, 8.66e-13, 6.207e-13, 4.61e-13, 3.444e-13,
02706 2.634e-13, 2.1e-13, 1.725e-13, 1.455e-13, 1.237e-13, 1.085e-13,
02707 9.513e-14, 7.978e-14, 6.603e-14, 5.288e-14, 4.084e-14, 2.952e-14,
02708 2.157e-14, 1.593e-14, 1.199e-14, 9.267e-15, 7.365e-15, 6.004e-15,
02709 4.995e-15, 4.218e-15, 3.601e-15, 3.101e-15, 2.692e-15, 2.36e-15,
02710 2.094e-15, 1.891e-15, 1.755e-15, 1.699e-15, 1.755e-15, 1.987e-15,
02711 2.506e-15, 3.506e-15, 5.289e-15, 8.311e-15, 1.325e-14, 2.129e-14,
02712 3.237e-14, 4.595e-14, 6.441e-14, 8.433e-14, 1.074e-13, 1.383e-13,
02713 1.762e-13, 2.281e-13, 2.831e-13, 3.523e-13, 4.38e-13, 5.304e-13,
02714 6.29e-13, 7.142e-13, 8.032e-13, 8.934e-13, 9.888e-13, 1.109e-12,
02715 1.261e-12, 1.462e-12, 1.74e-12, 2.099e-12, 2.535e-12, 3.008e-12,
02716 3.462e-12, 3.856e-12, 4.098e-12, 4.239e-12, 4.234e-12, 4.132e-12,
02717 3.986e-12, 3.866e-12, 3.829e-12, 3.742e-12, 3.705e-12, 3.694e-12,
02718 3.765e-12, 3.849e-12, 3.929e-12, 4.056e-12, 4.092e-12, 4.047e-12,
02719 3.792e-12, 3.407e-12, 2.953e-12, 2.429e-12, 1.931e-12, 1.46e-12,
02720 1.099e-12, 8.199e-13, 6.077e-13, 4.449e-13, 3.359e-13, 2.524e-13,
02721 1.881e-13, 1.391e-13, 1.02e-13, 7.544e-14, 5.555e-14, 4.22e-14,
02722 3.321e-14, 2.686e-14, 2.212e-14, 1.78e-14, 1.369e-14, 1.094e-14,
02723 9.13e-15, 8.101e-15, 7.828e-15, 8.393e-15, 1.012e-14, 1.259e-14,
02724 1.538e-14, 1.961e-14, 2.619e-14, 3.679e-14, 5.049e-14, 6.917e-14,
02725 8.88e-14, 1.115e-13, 1.373e-13, 1.619e-13, 1.878e-13, 2.111e-13,
02726 2.33e-13, 2.503e-13, 2.613e-13, 2.743e-13, 2.826e-13, 2.976e-13,
02727 3.162e-13, 3.36e-13, 3.491e-13, 3.541e-13, 3.595e-13, 3.608e-13,
02728 3.709e-13, 3.869e-13, 4.12e-13, 4.366e-13, 4.504e-13, 4.379e-13,
02729 3.955e-13, 3.385e-13, 2.741e-13, 2.089e-13, 1.427e-13, 9.294e-14,
02730 5.775e-14, 3.565e-14, 2.21e-14, 1.398e-14, 9.194e-15, 6.363e-15,
02731 4.644e-15, 3.55e-15, 2.808e-15, 2.274e-15, 1.871e-15, 1.557e-15,
02732 1.308e-15, 1.108e-15, 9.488e-16, 8.222e-16, 7.238e-16, 6.506e-16,
02733 6.008e-16, 5.742e-16, 5.724e-16, 5.991e-16, 6.625e-16, 7.775e-16,
02734 9.734e-16, 1.306e-15, 1.88e-15, 2.879e-15, 4.616e-15, 7.579e-15,
02735 1.248e-14, 2.03e-14, 3.244e-14, 5.171e-14, 7.394e-14, 9.676e-14,
02736 1.199e-13, 1.467e-13, 1.737e-13, 2.02e-13, 2.425e-13, 3.016e-13,
02737 3.7e-13, 4.617e-13, 5.949e-13, 7.473e-13, 9.378e-13, 1.191e-12,
02738 1.481e-12, 1.813e-12, 2.232e-12, 2.722e-12, 3.254e-12, 3.845e-12,
02739 4.458e-12, 5.048e-12, 5.511e-12, 5.898e-12, 6.204e-12, 6.293e-12,
02740 6.386e-12, 6.467e-12, 6.507e-12, 6.466e-12, 6.443e-12, 6.598e-12,
02741 6.873e-12, 7.3e-12, 7.816e-12, 8.368e-12, 8.643e-12, 8.466e-12,
02742 7.871e-12, 6.853e-12, 5.714e-12, 4.482e-12, 3.392e-12, 2.613e-12,
02743 2.008e-12, 1.562e-12, 1.228e-12, 9.888e-13, 7.646e-13, 5.769e-13,
02744 4.368e-13, 3.324e-13, 2.508e-13, 1.916e-13
02745 };
02746
02747 static double xfcrev[15] =
02748 { 1.003, 1.009, 1.015, 1.023, 1.029, 1.033, 1.037,
02749 1.039, 1.04, 1.046, 1.036, 1.027, 1.01, 1.002, 1.
02750 };

```

```

02751
02752 double a1, a2, a3, dw, ew, dx, xw, xx, vf2, vf6, cw260, cw296,
02753 sfac, fscal, cwfrn, ctmph, ctwfrn, ctwsf;
02754
02755 int iw, ix;
02756
02757 /* Get H2O continuum absorption... */
02758 xw = nu / 10 + 1;
02759 if (xw >= 1 && xw < 2001) {
02760     iw = (int) xw;
02761     dw = xw - iw;
02762     ew = 1 - dw;
02763     cw296 = ew * h2o296[iw - 1] + dw * h2o296[iw];
02764     cw260 = ew * h2o260[iw - 1] + dw * h2o260[iw];
02765     cwfrn = ew * h2ofrn[iw - 1] + dw * h2ofrn[iw];
02766     if (nu <= 820 || nu >= 960) {
02767         sfac = 1;
02768     } else {
02769         xx = (nu - 820) / 10;
02770         ix = (int) xx;
02771         dx = xx - ix;
02772         sfac = (1 - dx) * xfcrev[ix] + dx * xfcrev[ix + 1];
02773     }
02774     ctwsf = sfac * cw296 * pow(cw260 / cw296, (296 - t) / (296 - 260));
02775     vf2 = POW2(nu - 370);
02776     vf6 = POW3(vf2);
02777     fscal = 36100 / (vf2 + vf6 * 1e-8 + 36100) * -.25 + 1;
02778     ctwfrn = cwfrn * fscal;
02779     a1 = nu * u * tanh(.7193876 / t * nu);
02780     a2 = 296 / t;
02781     a3 = p / P0 * (q * ctwsf + (1 - q) * ctwfrn) * 1e-20;
02782     ctmph = a1 * a2 * a3;
02783 } else
02784     ctmph = 0;
02785 return ctmph;
02786 }

```

### 5.21.2.8 double ctmn2 ( double nu, double p, double t )

Compute nitrogen continuum (absorption coefficient).

Definition at line 2790 of file [jurassic.c](#).

```

02793     {
02794
02795 static double ba[98] = { 0., 4.45e-8, 5.22e-8, 6.46e-8, 7.75e-8, 9.03e-8,
02796     1.06e-7, 1.21e-7, 1.37e-7, 1.57e-7, 1.75e-7, 2.01e-7, 2.3e-7,
02797     2.59e-7, 2.95e-7, 3.26e-7, 3.66e-7, 4.05e-7, 4.47e-7, 4.92e-7,
02798     5.34e-7, 5.84e-7, 6.24e-7, 6.67e-7, 7.14e-7, 7.26e-7, 7.54e-7,
02799     7.84e-7, 8.09e-7, 8.42e-7, 8.62e-7, 8.87e-7, 9.11e-7, 9.36e-7,
02800     9.76e-7, 1.03e-6, 1.11e-6, 1.23e-6, 1.39e-6, 1.61e-6, 1.76e-6,
02801     1.94e-6, 1.97e-6, 1.87e-6, 1.75e-6, 1.56e-6, 1.42e-6, 1.35e-6,
02802     1.32e-6, 1.29e-6, 1.29e-6, 1.29e-6, 1.3e-6, 1.32e-6, 1.33e-6,
02803     1.34e-6, 1.35e-6, 1.33e-6, 1.31e-6, 1.29e-6, 1.24e-6, 1.2e-6,
02804     1.16e-6, 1.1e-6, 1.04e-6, 9.96e-7, 9.38e-7, 8.63e-7, 7.98e-7,
02805     7.26e-7, 6.55e-7, 5.94e-7, 5.35e-7, 4.74e-7, 4.24e-7, 3.77e-7,
02806     3.33e-7, 2.96e-7, 2.63e-7, 2.34e-7, 2.08e-7, 1.85e-7, 1.67e-7,
02807     1.47e-7, 1.32e-7, 1.2e-7, 1.09e-7, 9.85e-8, 9.08e-8, 8.18e-8,
02808     7.56e-8, 6.85e-8, 6.14e-8, 5.83e-8, 5.77e-8, 5e-8, 4.32e-8, 0.
02809 };
02810
02811 static double betaa[98] = { 802., 802., 761., 722., 679., 646., 609., 562.,
02812     511., 472., 436., 406., 377., 355., 338., 319., 299., 278., 255.,
02813     233., 208., 184., 149., 107., 66., 25., -13., -49., -82., -104.,
02814     -119., -130., -139., -144., -146., -146., -147., -148., -150.,
02815     -153., -160., -169., -181., -189., -195., -200., -205., -209.,
02816     -211., -210., -210., -209., -205., -199., -190., -180., -168.,
02817     -157., -143., -126., -108., -89., -63., -32., 1., 35., 65., 95.,
02818     121., 141., 152., 161., 164., 164., 161., 155., 148., 143., 137.,
02819     133., 131., 133., 139., 150., 165., 187., 213., 248., 284., 321.,
02820     372., 449., 514., 569., 609., 642., 673., 673.
02821 };
02822
02823 static double nua[98] = { 2120., 2125., 2130., 2135., 2140., 2145., 2150.,
02824     2155., 2160., 2165., 2170., 2175., 2180., 2185., 2190., 2195.,
02825     2200., 2205., 2210., 2215., 2220., 2225., 2230., 2235., 2240.,
02826     2245., 2250., 2255., 2260., 2265., 2270., 2275., 2280., 2285.,
02827     2290., 2295., 2300., 2305., 2310., 2315., 2320., 2325., 2330.,
02828     2335., 2340., 2345., 2350., 2355., 2360., 2365., 2370., 2375.,
02829     2380., 2385., 2390., 2395., 2400., 2405., 2410., 2415., 2420.,

```

```

02830     2425., 2430., 2435., 2440., 2445., 2450., 2455., 2460., 2465.,
02831     2470., 2475., 2480., 2485., 2490., 2495., 2500., 2505., 2510.,
02832     2515., 2520., 2525., 2530., 2535., 2540., 2545., 2550., 2555.,
02833     2560., 2565., 2570., 2575., 2580., 2585., 2590., 2595., 2600., 2605.
02834 };
02835
02836 double b, beta, q_n2 = 0.79, t0 = 273, tr = 296;
02837
02838 int idx;
02839
02840 /* Check wavenumber range... */
02841 if (nu < nua[0] || nu > nua[97])
02842     return 0;
02843
02844 /* Interpolate B and beta... */
02845 idx = locate_reg(nua, 98, nu);
02846 b = LIN(nua[idx], ba[idx], nua[idx + 1], ba[idx + 1], nu);
02847 beta = LIN(nua[idx], betaa[idx], nua[idx + 1], betaa[idx + 1], nu);
02848
02849 /* Compute absorption coefficient... */
02850 return 0.1 * POW2(p / P0 * t0 / t) * exp(beta * (1 / tr - 1 / t))
02851     * q_n2 * b * (q_n2 + (1 - q_n2) * (1.294 - 0.4545 * t / tr));
02852 }

```

Here is the call graph for this function:



#### 5.21.2.9 double ctmo2 ( double nu, double p, double t )

Compute oxygen continuum (absorption coefficient).

Definition at line 2856 of file [jurassic.c](#).

```

02859     {
02860
02861     static double ba[90] = { 0., .061, .074, .084, .096, .12, .162, .208, .246,
02862         .285, .314, .38, .444, .5, .571, .673, .768, .853, .966, 1.097,
02863         1.214, 1.333, 1.466, 1.591, 1.693, 1.796, 1.922, 2.037, 2.154,
02864         2.264, 2.375, 2.508, 2.671, 2.847, 3.066, 3.417, 3.828, 4.204,
02865         4.453, 4.599, 4.528, 4.284, 3.955, 3.678, 3.477, 3.346, 3.29,
02866         3.251, 3.231, 3.226, 3.212, 3.192, 3.108, 3.033, 2.911, 2.798,
02867         2.646, 2.508, 2.322, 2.13, 1.928, 1.757, 1.588, 1.417, 1.253,
02868         1.109, .99, .888, .791, .678, .587, .524, .464, .403, .357, .32,
02869         .29, .267, .242, .215, .182, .16, .146, .128, .103, .087, .081,
02870         .071, .064, 0.
02871     };
02872
02873     static double betaa[90] = { 467., 467., 400., 315., 379., 368., 475., 521.,
02874         531., 512., 442., 444., 430., 381., 335., 324., 296., 248., 215.,
02875         193., 158., 127., 101., 71., 31., -6., -26., -47., -63., -79.,
02876         -88., -88., -87., -90., -98., -99., -109., -134., -160., -167.,
02877         -164., -158., -153., -151., -156., -166., -168., -173., -170.,
02878         -161., -145., -126., -108., -84., -59., -29., 4., 41., 73., 97.,
02879         123., 159., 198., 220., 242., 256., 281., 311., 334., 319., 313.,
02880         321., 323., 310., 315., 320., 335., 361., 378., 373., 338., 319.,
02881         346., 322., 291., 290., 350., 371., 504., 504.
02882     };
02883
02884     static double nua[90] = { 1360., 1365., 1370., 1375., 1380., 1385., 1390.,
02885         1395., 1400., 1405., 1410., 1415., 1420., 1425., 1430., 1435.,
02886         1440., 1445., 1450., 1455., 1460., 1465., 1470., 1475., 1480.,
02887         1485., 1490., 1495., 1500., 1505., 1510., 1515., 1520., 1525.,
02888         1530., 1535., 1540., 1545., 1550., 1555., 1560., 1565., 1570.,

```

```

02889     1575., 1580., 1585., 1590., 1595., 1600., 1605., 1610., 1615.,
02890     1620., 1625., 1630., 1635., 1640., 1645., 1650., 1655., 1660.,
02891     1665., 1670., 1675., 1680., 1685., 1690., 1695., 1700., 1705.,
02892     1710., 1715., 1720., 1725., 1730., 1735., 1740., 1745., 1750.,
02893     1755., 1760., 1765., 1770., 1775., 1780., 1785., 1790., 1795.,
02894     1800., 1805.
02895 };
02896
02897 double b, beta, q_o2 = 0.21, t0 = 273, tr = 296;
02898
02899 int idx;
02900
02901 /* Check wavenumber range... */
02902 if (nu < nua[0] || nu > nua[89])
02903     return 0;
02904
02905 /* Interpolate B and beta... */
02906 idx = locate_reg(nua, 90, nu);
02907 b = LIN(nua[idx], ba[idx], nua[idx + 1], ba[idx + 1], nu);
02908 beta = LIN(nua[idx], betaa[idx], nua[idx + 1], betaa[idx + 1], nu);
02909
02910 /* Compute absorption coefficient... */
02911 return 0.1 * POW2(p / P0 * t0 / t) * exp(beta * (1 / tr - 1 / t)) * q_o2 *
02912     b;
02913 }

```

Here is the call graph for this function:



#### 5.21.2.10 void copy\_atm ( ctl\_t \*ctl, atm\_t \*atm\_dest, atm\_t \*atm\_src, int init )

Copy and initialize atmospheric data.

Definition at line 2917 of file [jurassic.c](#).

```

02921     {
02922
02923     int ig, ip, iw;
02924
02925     size_t s;
02926
02927     /* Data size... */
02928     s = (size_t) atm_src->np * sizeof(double);
02929
02930     /* Copy data... */
02931     atm_dest->np = atm_src->np;
02932     memcpy(atm_dest->time, atm_src->time, s);
02933     memcpy(atm_dest->z, atm_src->z, s);
02934     memcpy(atm_dest->lon, atm_src->lon, s);
02935     memcpy(atm_dest->lat, atm_src->lat, s);
02936     memcpy(atm_dest->p, atm_src->p, s);
02937     memcpy(atm_dest->t, atm_src->t, s);
02938     for (ig = 0; ig < ctl->ng; ig++)
02939         memcpy(atm_dest->q[ig], atm_src->q[ig], s);
02940     for (iw = 0; iw < ctl->nw; iw++)
02941         memcpy(atm_dest->k[iw], atm_src->k[iw], s);
02942
02943     /* Initialize... */
02944     if (init)
02945         for (ip = 0; ip < atm_dest->np; ip++) {
02946             atm_dest->p[ip] = 0;
02947             atm_dest->t[ip] = 0;
02948             for (ig = 0; ig < ctl->ng; ig++)
02949                 atm_dest->q[ig][ip] = 0;
02950             for (iw = 0; iw < ctl->nw; iw++)
02951                 atm_dest->k[iw][ip] = 0;
02952         }
02953 }

```

### 5.21.2.11 void copy\_obs ( ctl\_t \* *ctl*, obs\_t \* *obs\_dest*, obs\_t \* *obs\_src*, int *init* )

Copy and initialize observation data.

Definition at line 2957 of file [jurassic.c](#).

```

02961         {
02962
02963     int id, ir;
02964
02965     size_t s;
02966
02967     /* Data size... */
02968     s = (size_t) obs_src->nr * sizeof(double);
02969
02970     /* Copy data... */
02971     obs_dest->nr = obs_src->nr;
02972     memcpy(obs_dest->time, obs_src->time, s);
02973     memcpy(obs_dest->obsz, obs_src->obsz, s);
02974     memcpy(obs_dest->obslon, obs_src->obslon, s);
02975     memcpy(obs_dest->obslat, obs_src->obslat, s);
02976     memcpy(obs_dest->vpz, obs_src->vpz, s);
02977     memcpy(obs_dest->vplon, obs_src->vplon, s);
02978     memcpy(obs_dest->vplat, obs_src->vplat, s);
02979     memcpy(obs_dest->tpz, obs_src->tpz, s);
02980     memcpy(obs_dest->tplon, obs_src->tplon, s);
02981     memcpy(obs_dest->tplat, obs_src->tplat, s);
02982     for (id = 0; id < ctl->nd; id++)
02983         memcpy(obs_dest->rad[id], obs_src->rad[id], s);
02984     for (id = 0; id < ctl->nd; id++)
02985         memcpy(obs_dest->tau[id], obs_src->tau[id], s);
02986
02987     /* Initialize... */
02988     if (init)
02989         for (id = 0; id < ctl->nd; id++)
02990             for (ir = 0; ir < obs_dest->nr; ir++)
02991                 if (gsl_finite(obs_dest->rad[id][ir])) {
02992                     obs_dest->rad[id][ir] = 0;
02993                     obs_dest->tau[id][ir] = 0;
02994                 }
02995 }
```

### 5.21.2.12 int find\_emitter ( ctl\_t \* *ctl*, const char \* *emitter* )

Find index of an emitter.

Definition at line 2999 of file [jurassic.c](#).

```

03001         {
03002
03003     int ig;
03004
03005     for (ig = 0; ig < ctl->ng; ig++)
03006         if (strcasecmp(ctl->emitter[ig], emitter) == 0)
03007             return ig;
03008
03009     return -1;
03010 }
```

### 5.21.2.13 void formod ( ctl\_t \* *ctl*, atm\_t \* *atm*, obs\_t \* *obs* )

Determine ray paths and compute radiative transfer.

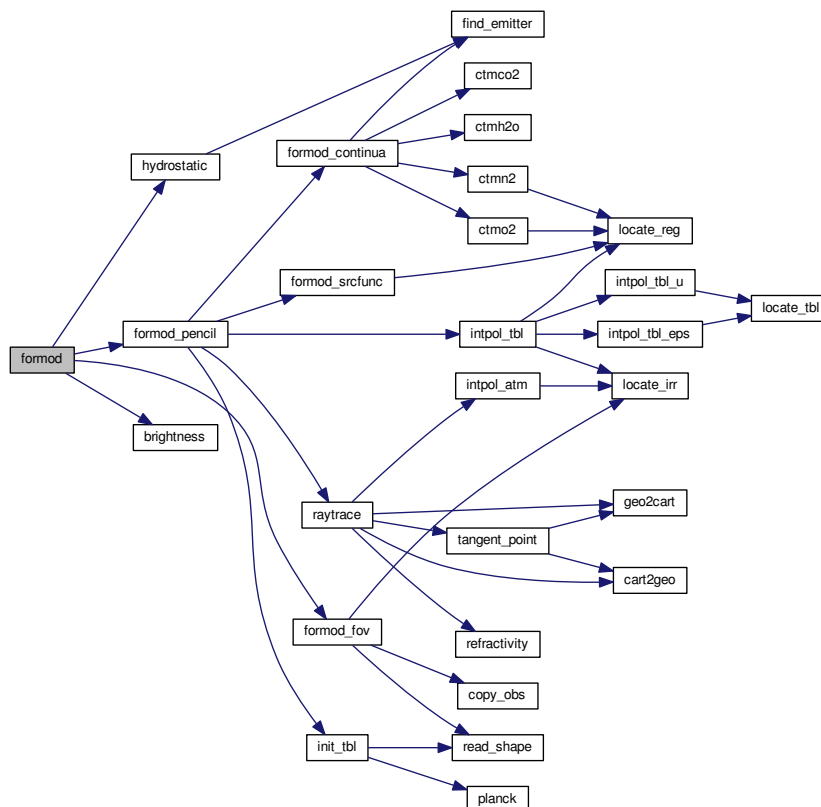
Definition at line 3014 of file [jurassic.c](#).

```

03017      {
03018
03019      int id, ir, *mask;
03020
03021      /* Allocate... */
03022      ALLOC(mask, int,
03023            ND * NR);
03024
03025      /* Save observation mask... */
03026      for (id = 0; id < ctl->nd; id++)
03027          for (ir = 0; ir < obs->nr; ir++)
03028              mask[id * NR + ir] = !gsl_finite(obs->rad[id][ir]);
03029
03030      /* Hydrostatic equilibrium... */
03031      hydrostatic(ctl, atm);
03032
03033      /* Calculate pencil beams... */
03034      for (ir = 0; ir < obs->nr; ir++)
03035          formod_pencil(ctl, atm, obs, ir);
03036
03037      /* Apply field-of-view convolution... */
03038      formod_fov(ctl, obs);
03039
03040      /* Convert radiance to brightness temperature... */
03041      if (ctl->write_bbt)
03042          for (id = 0; id < ctl->nd; id++)
03043              for (ir = 0; ir < obs->nr; ir++)
03044                  obs->rad[id][ir] = brightness(obs->rad[id][ir], ctl->nu[id]);
03045
03046      /* Apply observation mask... */
03047      for (id = 0; id < ctl->nd; id++)
03048          for (ir = 0; ir < obs->nr; ir++)
03049              if (mask[id * NR + ir])
03050                  obs->rad[id][ir] = GSL_NAN;
03051
03052      /* Free... */
03053      free(mask);
03054 }

```

Here is the call graph for this function:



#### 5.21.2.14 void formod\_continua ( ctl\_t \* *ctl*, los\_t \* *los*, int *ip*, double \* *beta* )

Compute absorption coefficient of continua.

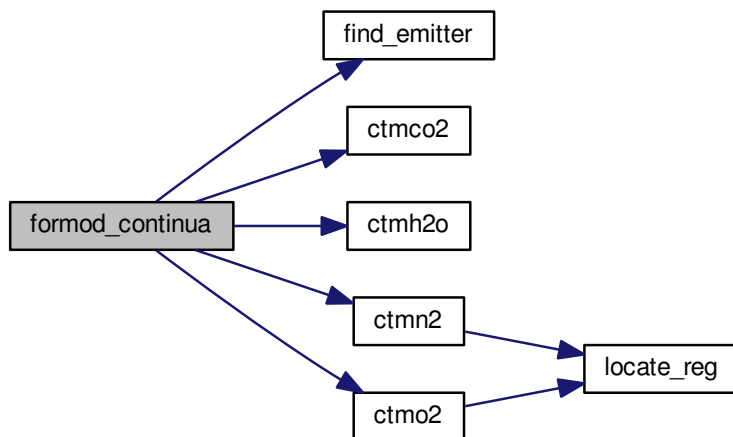
Definition at line 3058 of file [jurassic.c](#).

```

03062         {
03063
03064     static int ig_co2 = -999, ig_h2o = -999;
03065
03066     int id;
03067
03068     /* Extinction... */
03069     for (id = 0; id < ctl->nd; id++)
03070         beta[id] = los->k[ctl->window[id]][ip];
03071
03072     /* CO2 continuum... */
03073     if (ctl->ctm_co2) {
03074         if (ig_co2 == -999)
03075             ig_co2 = find_emitter(ctl, "CO2");
03076         if (ig_co2 >= 0)
03077             for (id = 0; id < ctl->nd; id++)
03078                 beta[id] += ctmco2(ctl->nu[id], los->p[ip], los->t[ip],
03079                                     los->u[ig_co2][ip]) / los->ds[ip];
03080     }
03081
03082     /* H2O continuum... */
03083     if (ctl->ctm_h2o) {
03084         if (ig_h2o == -999)
03085             ig_h2o = find_emitter(ctl, "H2O");
03086         if (ig_h2o >= 0)
03087             for (id = 0; id < ctl->nd; id++)
03088                 beta[id] += ctmh2o(ctl->nu[id], los->p[ip], los->t[ip],
03089                                     los->q[ig_h2o][ip],
03090                                     los->u[ig_h2o][ip]) / los->ds[ip];
03091     }
03092
03093     /* N2 continuum... */
03094     if (ctl->ctm_n2)
03095         for (id = 0; id < ctl->nd; id++)
03096             beta[id] += ctmn2(ctl->nu[id], los->p[ip], los->t[ip]);
03097
03098     /* O2 continuum... */
03099     if (ctl->ctm_o2)
03100         for (id = 0; id < ctl->nd; id++)
03101             beta[id] += ctmo2(ctl->nu[id], los->p[ip], los->t[ip]);
03102 }

```

Here is the call graph for this function:



5.21.2.15 void formod\_fov ( ctl\_t \* *ctl*, obs\_t \* *obs* )

Apply field of view convolution.

Definition at line 3106 of file [jurassic.c](#).

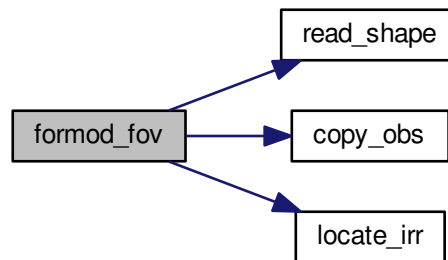
```

03108         {
03109
03110     static double dz[NSHAPE], w[NSHAPE];
03111
03112     static int init = 0, n;
03113
03114     obs_t *obs2;
03115
03116     double rad[ND][NR], tau[ND][NR], wsum, z[NR], zfov;
03117
03118     int i, id, idx, ir, ir2, nz;
03119
03120     /* Do not take into account FOV... */
03121     if (ctl->fov[0] == '-')
03122         return;
03123
03124     /* Initialize FOV data... */
03125     if (!init) {
03126         init = 1;
03127         read_shape(ctl->fov, dz, w, &n);
03128     }
03129
03130     /* Allocate... */
03131     ALLOC(obs2, obs_t, 1);
03132
03133     /* Copy observation data... */
03134     copy_obs(ctl, obs2, obs, 0);
03135
03136     /* Loop over ray paths... */
03137     for (ir = 0; ir < obs->nr; ir++) {
03138
03139         /* Get radiance and transmittance profiles... */
03140         nz = 0;
03141         for (ir2 = GSL_MAX(ir - NFOV, 0); ir2 < GSL_MIN(ir + 1 + NFOV, obs->nr);
03142             ir2++)
03143             if (obs->time[ir2] == obs->time[ir]) {
03144                 z[nz] = obs2->vpz[ir2];
03145                 for (id = 0; id < ctl->nd; id++) {
03146                     rad[id][nz] = obs2->rad[id][ir2];
03147                     tau[id][nz] = obs2->tau[id][ir2];
03148                 }
03149                 nz++;
03150             }
03151         if (nz < 2)
03152             ERRMSG("Cannot apply FOV convolution!");
03153
03154         /* Convolute profiles with FOV... */
03155         wsum = 0;
03156         for (id = 0; id < ctl->nd; id++) {
03157             obs->rad[id][ir] = 0;
03158             obs->tau[id][ir] = 0;
03159         }
03160         for (i = 0; i < n; i++) {
03161             zfov = obs->vpz[ir] + dz[i];
03162             idx = locate_irr(z, nz, zfov);
03163             for (id = 0; id < ctl->nd; id++) {
03164                 obs->rad[id][ir] += w[i]
03165                     * LIN(z[idx], rad[id][idx], z[idx + 1], rad[id][idx + 1], zfov);
03166                 obs->tau[id][ir] += w[i]
03167                     * LIN(z[idx], tau[id][idx], z[idx + 1], tau[id][idx + 1], zfov);
03168             }
03169             wsum += w[i];
03170         }
03171         for (id = 0; id < ctl->nd; id++) {
03172             obs->rad[id][ir] /= wsum;
03173             obs->tau[id][ir] /= wsum;
03174         }
03175     }
03176
03177     /* Free... */
03178     free(obs2);
03179 }

```



Here is the call graph for this function:



**5.21.2.16** void formod\_pencil ( ctl\_t \* *ctl*, atm\_t \* *atm*, obs\_t \* *obs*, int *ir* )

Compute radiative transfer for a pencil beam.

Definition at line 3183 of file [jurassic.c](#).

```

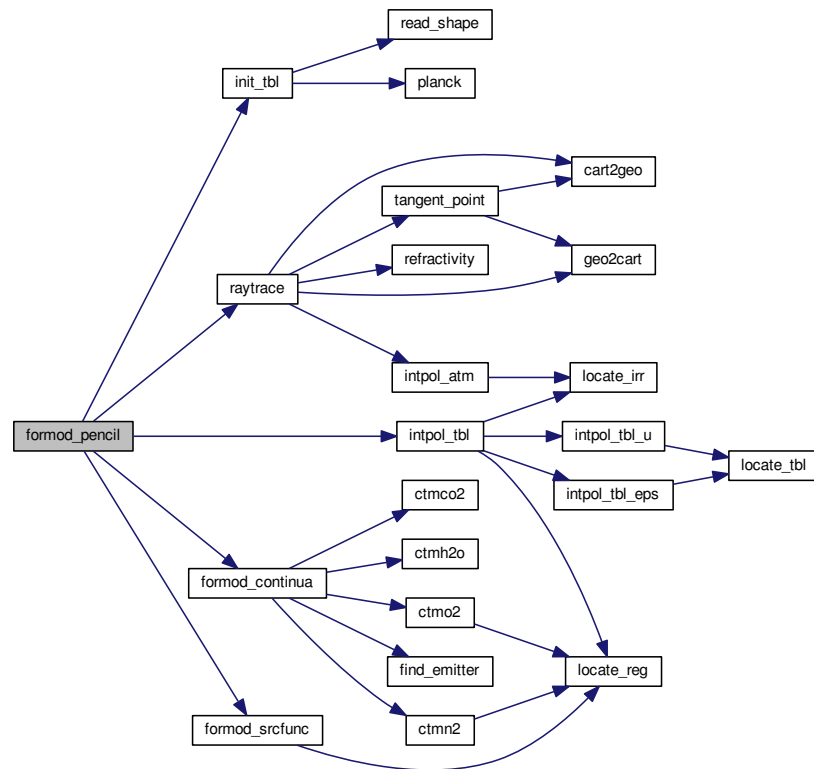
03187     {
03188
03189     static tbl_t *tbl;
03190
03191     static int init = 0;
03192
03193     los_t *los;
03194
03195     double beta_ctm[ND], eps, src_planck[ND], tau_path[NG][ND], tau_gas[ND];
03196
03197     int id, ip;
03198
03199     /* Initialize look-up tables... */
03200     if (!init) {
03201         init = 1;
03202         ALLOC(tbl, tbl_t, 1);
03203         init_tbl(ctl, tbl);
03204     }
03205
03206     /* Allocate... */
03207     ALLOC(los, los_t, 1);
03208
03209     /* Initialize... */
03210     for (id = 0; id < ctl->nd; id++) {
03211         obs->rad[id][ir] = 0;
03212         obs->tau[id][ir] = 1;
03213     }
03214
03215     /* Raytracing... */
03216     raytrace(ctl, atm, obs, los, ir);
03217
03218     /* Loop over LOS points... */
03219     for (ip = 0; ip < los->np; ip++) {
03220
03221         /* Get trace gas transmittance... */
03222         intpol_tbl(ctl, tbl, los, ip, tau_path, tau_gas);
03223
03224         /* Get continuum absorption... */
03225         formod_continua(ctl, los, ip, beta_ctm);
03226
03227         /* Compute Planck function... */
03228         formod_srcfunc(ctl, tbl, los->t[ip], src_planck);
03229
03230         /* Loop over channels... */
03231         for (id = 0; id < ctl->nd; id++)
03232             if (tau_gas[id] > 0) {
03233
  
```

```

03234      /* Get segment emissivity... */
03235      eps = 1 - tau_gas[id] * exp(-beta_ctm[id] * los->ds[ip]);
03236
03237      /* Compute radiance... */
03238      obs->rad[id][ir] += src_planck[id] * eps * obs->tau[id][ir];
03239
03240      /* Compute path transmittance... */
03241      obs->tau[id][ir] *= (1 - eps);
03242  }
03243  }
03244
03245  /* Add surface... */
03246  if (los->tsurf > 0) {
03247      formod_srcfunc(ctl, tbl, los->tsurf, src_planck);
03248      for (id = 0; id < ctl->nd; id++)
03249          obs->rad[id][ir] += src_planck[id] * obs->tau[id][ir];
03250  }
03251
03252  /* Free... */
03253  free(los);
03254  }

```

Here is the call graph for this function:



#### 5.21.2.17 void formod\_srcfunc ( ctl\_t \* ctl, tbl\_t \* tbl, double t, double \* src )

Compute Planck source function.

Definition at line 3258 of file [jurassic.c](#).

```

03262      {
03263
03264      int id, it;

```

```

03265
03266  /* Determine index in temperature array... */
03267  it = locate_reg(tbl->st, TBLNS, t);
03268
03269  /* Interpolate Planck function value... */
03270  for (id = 0; id < ctl->nd; id++)
03271      src[id] = LIN(tbl->st[it], tbl->sr[id][it],
03272                  tbl->st[it + 1], tbl->sr[id][it + 1], t);
03273 }

```

Here is the call graph for this function:



#### 5.21.2.18 void geo2cart ( double z, double lon, double lat, double \* x )

Convert geolocation to Cartesian coordinates.

Definition at line 3277 of file [jurassic.c](#).

```

03281      {
03282
03283      double radius;
03284
03285      radius = z + RE;
03286      x[0] = radius * cos(lat / 180 * M_PI) * cos(lon / 180 * M_PI);
03287      x[1] = radius * cos(lat / 180 * M_PI) * sin(lon / 180 * M_PI);
03288      x[2] = radius * sin(lat / 180 * M_PI);
03289 }

```

#### 5.21.2.19 void hydrostatic ( ctl\_t \* ctl, atm\_t \* atm )

Set hydrostatic equilibrium.

Definition at line 3293 of file [jurassic.c](#).

```

03295      {
03296
03297      static int ig_h2o = -999;
03298
03299      double dzmin = 1e99, e = 0, mean, mmair = 28.96456e-3, mmh2o = 18.0153e-3;
03300
03301      int i, ip, ipref = 0, ipt = 20;
03302
03303      /* Check reference height... */
03304      if (ctl->hydz < 0)
03305          return;
03306
03307      /* Determine emitter index of H2O... */
03308      if (ig_h2o == -999)
03309          ig_h2o = find_emitter(ctl, "H2O");
03310
03311      /* Find air parcel next to reference height... */
03312      for (ip = 0; ip < atm->np; ip++)
03313          if (fabs(atm->z[ip] - ctl->hydz) < dzmin) {
03314              dzmin = fabs(atm->z[ip] - ctl->hydz);
03315              ipref = ip;
03316          }

```

```

03317
03318 /* Upper part of profile... */
03319 for (ip = ipref + 1; ip < atm->np; ip++) {
03320     mean = 0;
03321     for (i = 0; i < ipt; i++) {
03322         if (ig_h2o >= 0)
03323             e = LIN(0.0, atm->q[ig_h2o][ip - 1],
03324                     ipt - 1.0, atm->q[ig_h2o][ip], (double) i);
03325         mean += (e * mmh2o + (1 - e) * mmair)
03326             * G0 / RI
03327             / LIN(0.0, atm->t[ip - 1], ipt - 1.0, atm->t[ip], (double) i) / ipt;
03328     }
03329
03330 /* Compute p(z,T)... */
03331 atm->p[ip] =
03332     exp(log(atm->p[ip - 1]) - mean * 1000 * (atm->z[ip] - atm->z[ip - 1]));
03333 }
03334
03335 /* Lower part of profile... */
03336 for (ip = ipref - 1; ip >= 0; ip--) {
03337     mean = 0;
03338     for (i = 0; i < ipt; i++) {
03339         if (ig_h2o >= 0)
03340             e = LIN(0.0, atm->q[ig_h2o][ip + 1],
03341                     ipt - 1.0, atm->q[ig_h2o][ip], (double) i);
03342         mean += (e * mmh2o + (1 - e) * mmair)
03343             * G0 / RI
03344             / LIN(0.0, atm->t[ip + 1], ipt - 1.0, atm->t[ip], (double) i) / ipt;
03345     }
03346
03347 /* Compute p(z,T)... */
03348 atm->p[ip] =
03349     exp(log(atm->p[ip + 1]) - mean * 1000 * (atm->z[ip] - atm->z[ip + 1]));
03350 }
03351 }

```

Here is the call graph for this function:



#### 5.21.2.20 void idx2name( ctl\_t \*ctl, int idx, char \*quantity )

Determine name of state vector quantity for given index.

Definition at line 3355 of file [jurassic.c](#).

```

03358     {
03359
03360     int ig, iw;
03361
03362     if (idx == IDXP)
03363         sprintf(quantity, "PRESSURE");
03364
03365     if (idx == IDXT)
03366         sprintf(quantity, "TEMPERATURE");
03367
03368     for (ig = 0; ig < ctl->ng; ig++)
03369         if (idx == IDXQ(ig))
03370             sprintf(quantity, "%s", ctl->emitter[ig]);
03371
03372     for (iw = 0; iw < ctl->nw; iw++)
03373         if (idx == IDXK(iw))
03374             sprintf(quantity, "EXTINCT_WINDOW%d", iw);
03375 }

```

### 5.21.2.21 void init\_tbl ( ctl\_t \* ctl, tbl\_t \* tbl )

Initialize look-up tables.

Definition at line 3379 of file [jurassic.c](#).

```

03381         {
03382
03383     FILE *in;
03384
03385     char filename[2 * LEN], line[LEN];
03386
03387     double eps, eps_old, press, press_old, temp, temp_old, u, u_old,
03388            f[NSHAPE], fsum, nu[NSHAPE];
03389
03390     int i, id, ig, ip, it, n;
03391
03392     /* Loop over trace gases and channels... */
03393     for (ig = 0; ig < ctl->ng; ig++)
03394 #pragma omp parallel for default(none) shared(ctl,tbl,ig) private(in,filename,line,eps,eps_old,press,
03395            press_old,temp,temp_old,u,u_old,id,ip,it)
03396     for (id = 0; id < ctl->nd; id++) {
03397
03398         /* Initialize... */
03399         tbl->np[ig][id] = -1;
03400         eps_old = -999;
03401         press_old = -999;
03402         temp_old = -999;
03403         u_old = -999;
03404
03405         /* Try to open file... */
03406         sprintf(filename, "%s%.4f%s.tab",
03407                ctl->tblbase, ctl->nu[id], ctl->emitter[ig]);
03408         if (!(in = fopen(filename, "r"))) {
03409             printf("Missing emissivity table: %s\n", filename);
03410             continue;
03411         }
03412         printf("Read emissivity table: %s\n", filename);
03413
03414         /* Read data... */
03415         while (fgets(line, LEN, in)) {
03416
03417             /* Parse line... */
03418             if (sscanf(line, "%lg %lg %lg %lg", &press, &temp, &u, &eps) != 4)
03419                 continue;
03420
03421             /* Determine pressure index... */
03422             if (press != press_old) {
03423                 press_old = press;
03424                 if ((++tbl->np[ig][id]) >= TBLNP)
03425                     ERRMSG("Too many pressure levels!");
03426                 tbl->nt[ig][id][tbl->np[ig][id]] = -1;
03427             }
03428
03429             /* Determine temperature index... */
03430             if (temp != temp_old) {
03431                 temp_old = temp;
03432                 if ((++tbl->nt[ig][id][tbl->np[ig][id]]) >= TBLNT)
03433                     ERRMSG("Too many temperatures!");
03434                 tbl->nu[ig][id][tbl->np[ig][id]]
03435                     [tbl->nt[ig][id][tbl->np[ig][id]]] = -1;
03436             }
03437
03438             /* Determine column density index... */
03439             if ((eps > eps_old && u > u_old) || tbl->nu[ig][id][tbl->np[ig][id]]
03440                 [tbl->nt[ig][id][tbl->np[ig][id]]] < 0) {
03441                 eps_old = eps;
03442                 u_old = u;
03443                 if ((++tbl->nu[ig][id][tbl->np[ig][id]]
03444                     [tbl->nt[ig][id][tbl->np[ig][id]]]) >= TBLNU) {
03445                     tbl->nu[ig][id][tbl->np[ig][id]]
03446                         [tbl->nt[ig][id][tbl->np[ig][id]]]--;
03447                     continue;
03448                 }
03449             }
03450
03451             /* Store data... */
03452             tbl->p[ig][id][tbl->np[ig][id]] = press;
03453             tbl->t[ig][id][tbl->np[ig][id]][tbl->nt[ig][id][tbl->np[ig][id]]]
03454                 = temp;
03455             tbl->u[ig][id][tbl->np[ig][id]][tbl->nt[ig][id][tbl->np[ig][id]]]
03456                 [tbl->nu[ig][id][tbl->np[ig][id]]
03457                  [tbl->nt[ig][id][tbl->np[ig][id]]]] = (float) u;

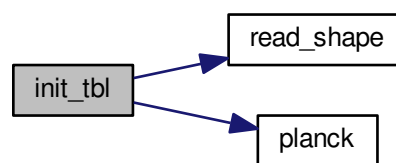
```

```

03457         tbl->eps[ig][id][tbl->np[ig][id]][tbl->nt[ig][id][tbl->np[ig][id]]]
03458         [tbl->nu[ig][id][tbl->np[ig][id]]]
03459         [tbl->nt[ig][id][tbl->np[ig][id]]] = (float) eps;
03460     }
03461
03462     /* Increment counters... */
03463     tbl->np[ig][id]++;
03464     for (ip = 0; ip < tbl->np[ig][id]; ip++) {
03465         tbl->nt[ig][id][ip]++;
03466         for (it = 0; it < tbl->nt[ig][id][ip]; it++)
03467             tbl->nu[ig][id][ip][it]++;
03468     }
03469
03470     /* Close file... */
03471     fclose(in);
03472 }
03473
03474 /* Write info... */
03475 printf("Initialize source function table...\n");
03476
03477 /* Loop over channels... */
03478 #pragma omp parallel for default(none) shared(ctl,tbl,ig) private(filename,it,i,n,f,fsum,nu)
03479 for (id = 0; id < ctl->nd; id++) {
03480
03481     /* Read filter function... */
03482     sprintf(filename, "%s_%.4f.filt", ctl->tbase, ctl->nu[id]);
03483     read_shape(filename, nu, f, &n);
03484
03485     /* Compute source function table... */
03486     for (it = 0; it < TBLNS; it++) {
03487
03488         /* Set temperature... */
03489         tbl->st[it] = LIN(0.0, TMIN, TBLNS - 1.0, TMAX, (double) it);
03490
03491         /* Integrate Planck function... */
03492         fsum = 0;
03493         tbl->sr[id][it] = 0;
03494         for (i = 0; i < n; i++) {
03495             fsum += f[i];
03496             tbl->sr[id][it] += f[i] * planck(tbl->st[it], nu[i]);
03497         }
03498         tbl->sr[id][it] /= fsum;
03499     }
03500 }
03501 }

```

Here is the call graph for this function:



**5.21.2.22** 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 3505 of file `jurassic.c`.

```

03512     {
03513
03514     int ig, ip, iw;
03515

```

```

03516  /* Get array index... */
03517  ip = locate_irr(atm->z, atm->np, z);
03518
03519  /* Interpolate... */
03520  *p = EXP(atm->z[ip], atm->p[ip], atm->z[ip + 1], atm->p[ip + 1], z);
03521  *t = LIN(atm->z[ip], atm->t[ip], atm->z[ip + 1], atm->t[ip + 1], z);
03522  for (ig = 0; ig < ctl->ng; ig++)
03523      q[ig] =
03524          LIN(atm->z[ip], atm->q[ig][ip], atm->z[ip + 1], atm->q[ig][ip + 1], z);
03525  for (iw = 0; iw < ctl->nw; iw++)
03526      k[iw] =
03527          LIN(atm->z[ip], atm->k[iw][ip], atm->z[ip + 1], atm->k[iw][ip + 1], z);
03528  }

```

Here is the call graph for this function:



**5.21.2.23** 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 3532 of file [jurassic.c](#).

```

03538      {
03539
03540      double eps, eps00, eps01, eps10, eps11, u;
03541
03542      int id, ig, ipr, it0, it1;
03543
03544      /* Initialize... */
03545      if (ip <= 0)
03546          for (ig = 0; ig < ctl->ng; ig++)
03547              for (id = 0; id < ctl->nd; id++)
03548                  tau_path[ig][id] = 1;
03549
03550      /* Loop over channels... */
03551      for (id = 0; id < ctl->nd; id++) {
03552
03553          /* Initialize... */
03554          tau_seg[id] = 1;
03555
03556          /* Loop over emitters... */
03557          for (ig = 0; ig < ctl->ng; ig++) {
03558
03559              /* Check size of table (pressure)... */
03560              if (tbl->np[ig][id] < 2)
03561                  eps = 0;
03562
03563              /* Check transmittance... */
03564              else if (tau_path[ig][id] < 1e-9)
03565                  eps = 1;
03566
03567              /* Interpolate... */
03568              else {
03569
03570                  /* Determine pressure and temperature indices... */
03571                  ipr = locate_irr(tbl->p[ig][id], tbl->np[ig][id], los->p[ip]);
03572                  it0 =
03573                      locate_irr(tbl->t[ig][id][ipr], tbl->nt[ig][id][ipr], los->
03574                          t[ip]);
03575                  it1 =
03576                      locate_reg(tbl->t[ig][id][ipr + 1], tbl->nt[ig][id][ipr + 1],
03577                          los->t[ip]);

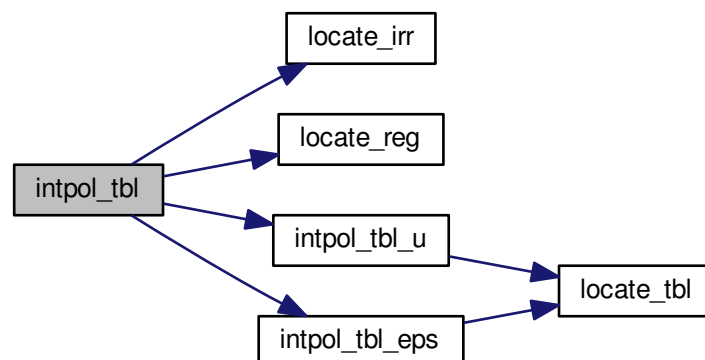
```

```

03577
03578 /* Check size of table (temperature and column density)... */
03579 if (tbl->nt[ig][id][ipr] < 2 || tbl->nt[ig][id][ipr + 1] < 2
03580     || tbl->nu[ig][id][ipr][it0] < 2
03581     || tbl->nu[ig][id][ipr][it0 + 1] < 2
03582     || tbl->nu[ig][id][ipr + 1][it1] < 2
03583     || tbl->nu[ig][id][ipr + 1][it1 + 1] < 2)
03584     eps = 0;
03585
03586 else {
03587
03588     /* Get emissivities of extended path... */
03589     u = intpol_tbl_u(tbl, ig, id, ipr, it0, 1 - tau_path[ig][id]);
03590     eps00 = intpol_tbl_eps(tbl, ig, id, ipr, it0, u + los->u[ig][ip]);
03591
03592     u = intpol_tbl_u(tbl, ig, id, ipr, it0 + 1, 1 - tau_path[ig][id]);
03593     eps01 =
03594         intpol_tbl_eps(tbl, ig, id, ipr, it0 + 1, u + los->u[ig][ip]);
03595
03596     u = intpol_tbl_u(tbl, ig, id, ipr + 1, it1, 1 - tau_path[ig][id]);
03597     eps10 =
03598         intpol_tbl_eps(tbl, ig, id, ipr + 1, it1, u + los->u[ig][ip]);
03599
03600     u =
03601         intpol_tbl_u(tbl, ig, id, ipr + 1, it1 + 1, 1 - tau_path[ig][id]);
03602     eps11 =
03603         intpol_tbl_eps(tbl, ig, id, ipr + 1, it1 + 1, u + los->
03604         u[ig][ip]);
03605
03606     /* Interpolate with respect to temperature... */
03607     eps00 = LIN(tbl->t[ig][id][ipr][it0], eps00,
03608         tbl->t[ig][id][ipr][it0 + 1], eps01, los->t[ip]);
03609     eps11 = LIN(tbl->t[ig][id][ipr + 1][it1], eps10,
03610         tbl->t[ig][id][ipr + 1][it1 + 1], eps11, los->t[ip]);
03611
03612     /* Interpolate with respect to pressure... */
03613     eps00 = LIN(tbl->p[ig][id][ipr], eps00,
03614         tbl->p[ig][id][ipr + 1], eps11, los->p[ip]);
03615
03616     /* Check emssivity range... */
03617     eps00 = GSL_MAX(GSL_MIN(eps00, 1), 0);
03618
03619     /* Determine segment emissivity... */
03620     eps = 1 - (1 - eps00) / tau_path[ig][id];
03621 }
03622
03623 /* Get transmittance of extended path... */
03624 tau_path[ig][id] *= (1 - eps);
03625
03626 /* Get segment transmittance... */
03627 tau_seg[id] *= (1 - eps);
03628 }
03629 }
03630 }

```

Here is the call graph for this function:





#### 5.21.2.24 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 3634 of file [jurassic.c](#).

```

03640         {
03641
03642     int idx;
03643
03644     /* Lower boundary... */
03645     if (u < tbl->u[ig][id][ip][it][0])
03646         return LIN(0, 0, tbl->u[ig][id][ip][it][0], tbl->eps[ig][id][ip][it][0],
03647             u);
03648
03649     /* Upper boundary... */
03650     else if (u > tbl->u[ig][id][ip][it][tbl->nu[ig][id][ip][it] - 1])
03651         return LIN(tbl->u[ig][id][ip][it][tbl->nu[ig][id][ip][it] - 1],
03652             tbl->eps[ig][id][ip][it][tbl->nu[ig][id][ip][it] - 1],
03653             1e30, 1, u);
03654
03655     /* Interpolation... */
03656     else {
03657
03658         /* Get index... */
03659         idx = locate_tbl(tbl->u[ig][id][ip][it], tbl->nu[ig][id][ip][it], u);
03660
03661         /* Interpolate... */
03662         return
03663             LIN(tbl->u[ig][id][ip][it][idx], tbl->eps[ig][id][ip][it][idx],
03664                 tbl->u[ig][id][ip][it][idx + 1], tbl->eps[ig][id][ip][it][idx + 1],
03665                 u);
03666     }
03667 }

```

Here is the call graph for this function:



#### 5.21.2.25 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 3671 of file [jurassic.c](#).

```

03677         {
03678
03679     int idx;
03680
03681     /* Lower boundary... */
03682     if (eps < tbl->eps[ig][id][ip][it][0])
03683         return LIN(0, 0, tbl->eps[ig][id][ip][it][0], tbl->u[ig][id][ip][it][0],
03684             eps);
03685
03686     /* Upper boundary... */
03687     else if (eps > tbl->eps[ig][id][ip][it][tbl->nu[ig][id][ip][it] - 1])
03688         return LIN(tbl->eps[ig][id][ip][it][tbl->nu[ig][id][ip][it] - 1],
03689             tbl->u[ig][id][ip][it][tbl->nu[ig][id][ip][it] - 1],
03690             1, 1e30, eps);
03691

```

```

03692  /* Interpolation... */
03693  else {
03694
03695      /* Get index... */
03696      idx = locate_tbl(tbl->eps[ig][id][ip][it], tbl->nu[ig][id][ip][it], eps);
03697
03698      /* Interpolate... */
03699      return
03700      LIN(tbl->eps[ig][id][ip][it][idx], tbl->u[ig][id][ip][it][idx],
03701         tbl->eps[ig][id][ip][it][idx + 1], tbl->u[ig][id][ip][it][idx + 1],
03702         eps);
03703  }
03704 }

```

Here is the call graph for this function:



#### 5.21.2.26 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 3708 of file [jurassic.c](#).

```

03716      {
03717
03718      struct tm t0, *t1;
03719
03720      time_t jsec0;
03721
03722      t0.tm_year = 100;
03723      t0.tm_mon = 0;
03724      t0.tm_mday = 1;
03725      t0.tm_hour = 0;
03726      t0.tm_min = 0;
03727      t0.tm_sec = 0;
03728
03729      jsec0 = (time_t) jsec + timegm(&t0);
03730      t1 = gmtime(&jsec0);
03731
03732      *year = t1->tm_year + 1900;
03733      *mon = t1->tm_mon + 1;
03734      *day = t1->tm_mday;
03735      *hour = t1->tm_hour;
03736      *min = t1->tm_min;
03737      *sec = t1->tm_sec;
03738      *remain = jsec - floor(jsec);
03739  }

```

#### 5.21.2.27 void kernel ( ctl\_t \* ctl, atm\_t \* atm, obs\_t \* obs, gsl\_matrix \* k )

Compute Jacobians.

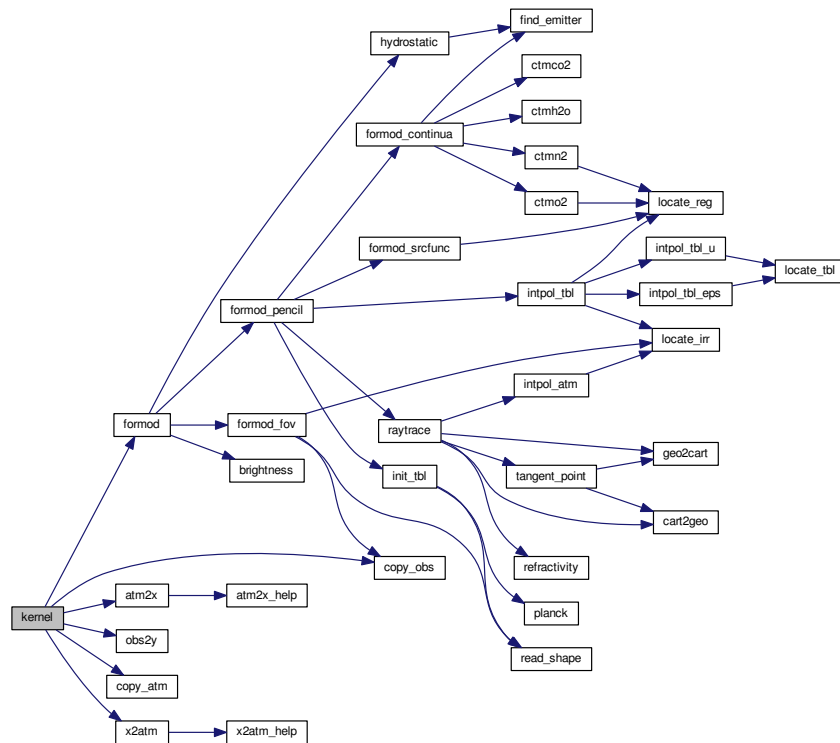
Definition at line 3743 of file [jurassic.c](#).

```

03747         {
03748
03749     atm_t *atml;
03750     obs_t *obs1;
03751
03752     gsl_vector *x0, *x1, *yy0, *yy1;
03753
03754     int *iqa, j;
03755
03756     double h;
03757
03758     size_t i, n, m;
03759
03760     /* Get sizes... */
03761     m = k->size1;
03762     n = k->size2;
03763
03764     /* Allocate... */
03765     x0 = gsl_vector_alloc(n);
03766     yy0 = gsl_vector_alloc(m);
03767     ALLOC(iqa, int,
03768           N);
03769
03770     /* Compute radiance for undisturbed atmospheric data... */
03771     formod(ctl, atm, obs);
03772
03773     /* Compose vectors... */
03774     atm2x(ctl, atm, x0, iqa, NULL);
03775     obs2y(ctl, obs, yy0, NULL, NULL);
03776
03777     /* Initialize kernel matrix... */
03778     gsl_matrix_set_zero(k);
03779
03780     /* Loop over state vector elements... */
03781 #pragma omp parallel for default(none) shared(ctl,atm,obs,k,x0,yy0,n,m,iqa) private(i, j, h, x1, yy1, atml,
03782     obs1)
03783     for (j = 0; j < (int) n; j++) {
03784
03785         /* Allocate... */
03786         x1 = gsl_vector_alloc(n);
03787         yy1 = gsl_vector_alloc(m);
03788         ALLOC(atml, atm_t, 1);
03789         ALLOC(obs1, obs_t, 1);
03790
03791         /* Set perturbation size... */
03792         if (iqa[j] == IDXP)
03793             h = GSL_MAX(fabs(0.01 * gsl_vector_get(x0, (size_t) j)), 1e-7);
03794         else if (iqa[j] == IDXT)
03795             h = 1;
03796         else if (iqa[j] >= IDXQ(0) && iqa[j] < IDXQ(ctl->ng))
03797             h = GSL_MAX(fabs(0.01 * gsl_vector_get(x0, (size_t) j)), 1e-15);
03798         else if (iqa[j] >= IDXK(0) && iqa[j] < IDXK(ctl->nw))
03799             h = 1e-4;
03800         else
03801             ERRMSG("Cannot set perturbation size!");
03802
03803         /* Disturb state vector element... */
03804         gsl_vector_memcpy(x1, x0);
03805         gsl_vector_set(x1, (size_t) j, gsl_vector_get(x1, (size_t) j) + h);
03806         copy_atm(ctl, atml, atm, 0);
03807         copy_obs(ctl, obs1, obs, 0);
03808         x2atm(ctl, x1, atml);
03809
03810         /* Compute radiance for disturbed atmospheric data... */
03811         formod(ctl, atml, obs1);
03812
03813         /* Compose measurement vector for disturbed radiance data... */
03814         obs2y(ctl, obs1, yy1, NULL, NULL);
03815
03816         /* Compute derivatives... */
03817         for (i = 0; i < m; i++)
03818             gsl_matrix_set(k, i, (size_t) j,
03819                           (gsl_vector_get(yy1, i) - gsl_vector_get(yy0, i)) / h);
03820
03821         /* Free... */
03822         gsl_vector_free(x1);
03823         gsl_vector_free(yy1);
03824         free(atml);
03825         free(obs1);
03826     }
03827
03828     /* Free... */
03829     gsl_vector_free(x0);
03830     gsl_vector_free(yy0);
03831     free(iqa);
03832 }

```

Here is the call graph for this function:



#### 5.21.2.28 int locate\_irr ( double \* xx, int n, double x )

Find array index for irregular grid.

Definition at line 3835 of file [jurassic.c](#).

```

03838     {
03839
03840     int i, ilo, ihi;
03841
03842     ilo = 0;
03843     ihi = n - 1;
03844     i = (ihi + ilo) >> 1;
03845
03846     if (xx[i] < xx[i + 1])
03847         while (ihi > ilo + 1) {
03848             i = (ihi + ilo) >> 1;
03849             if (xx[i] > x)
03850                 ihi = i;
03851             else
03852                 ilo = i;
03853         } else
03854             while (ihi > ilo + 1) {
03855                 i = (ihi + ilo) >> 1;
03856                 if (xx[i] <= x)
03857                     ihi = i;
03858                 else
03859                     ilo = i;
03860             }
03861     return ilo;
03862 }
03863 }
```

### 5.21.2.29 int locate\_reg ( double \* xx, int n, double x )

Find array index for regular grid.

Definition at line 3867 of file [jurassic.c](#).

```
03870     {
03871
03872     int i;
03873
03874     /* Calculate index... */
03875     i = (int) ((x - xx[0]) / (xx[1] - xx[0]));
03876
03877     /* Check range... */
03878     if (i < 0)
03879         i = 0;
03880     else if (i >= n - 2)
03881         i = n - 2;
03882
03883     return i;
03884 }
```

### 5.21.2.30 int locate\_tbl ( float \* xx, int n, double x )

Find array index in float array.

Definition at line 3888 of file [jurassic.c](#).

```
03891     {
03892
03893     int i, ilo, ihi;
03894
03895     ilo = 0;
03896     ihi = n - 1;
03897     i = (ihi + ilo) >> 1;
03898
03899     while (ihi > ilo + 1) {
03900         i = (ihi + ilo) >> 1;
03901         if (xx[i] > x)
03902             ihi = i;
03903         else
03904             ilo = i;
03905     }
03906
03907     return ilo;
03908 }
```

### 5.21.2.31 size\_t obs2y ( ctl\_t \* ctl, obs\_t \* obs, gsl\_vector \* y, int \* ida, int \* ira )

Compose measurement vector.

Definition at line 3912 of file [jurassic.c](#).

```
03917     {
03918
03919     int id, ir;
03920
03921     size_t m = 0;
03922
03923     /* Determine measurement vector... */
03924     for (ir = 0; ir < obs->nr; ir++)
03925         for (id = 0; id < ctl->nd; id++)
03926             if (gsl_finite(obs->rad[id][ir])) {
03927                 if (y != NULL)
03928                     gsl_vector_set(y, m, obs->rad[id][ir]);
03929                 if (ida != NULL)
03930                     ida[m] = id;
03931                 if (ira != NULL)
03932                     ira[m] = ir;
03933                 m++;
03934             }
03935
03936     return m;
03937 }
```

5.21.2.32 double planck ( double *t*, double *nu* )

Compute Planck function.

Definition at line 3941 of file [jurassic.c](#).

```

03943     {
03944
03945     return C1 * POW3(nu) / gsl_expml(C2 * nu / t);
03946 }
```

5.21.2.33 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 3950 of file [jurassic.c](#).

```

03955     {
03956
03957     double cosa, d, dmax, dmin = 0, ds, ex0[3], ex1[3], frac, h = 0.02, k[NW],
03958     lat, lon, n, naux, ng[3], norm, p, q[NG], t, x[3], xh[3],
03959     xobs[3], xvp[3], z = 1e99, zmax, zmin, zrefrac = 60;
03960
03961     int i, ig, ip, iw, stop = 0;
03962
03963     /* Initialize... */
03964     los->np = 0;
03965     los->tsurf = -999;
03966     obs->tpz[ir] = obs->vpz[ir];
03967     obs->tplon[ir] = obs->vplon[ir];
03968     obs->tplat[ir] = obs->vplat[ir];
03969
03970     /* Get altitude range of atmospheric data... */
03971     gsl_stats_minmax(&zmin, &zmax, atm->z, 1, (size_t) atm->np);
03972
03973     /* Check observer altitude... */
03974     if (obs->obsz[ir] < zmin)
03975         ERRMSG("Observer below surface!");
03976
03977     /* Check view point altitude... */
03978     if (obs->vpz[ir] > zmax)
03979         return;
03980
03981     /* Determine Cartesian coordinates for observer and view point... */
03982     geo2cart(obs->obsz[ir], obs->obslon[ir], obs->obslat[ir], xobs);
03983     geo2cart(obs->vpz[ir], obs->vplon[ir], obs->vplat[ir], xvp);
03984
03985     /* Determine initial tangent vector... */
03986     for (i = 0; i < 3; i++)
03987         ex0[i] = xvp[i] - xobs[i];
03988     norm = NORM(ex0);
03989     for (i = 0; i < 3; i++)
03990         ex0[i] /= norm;
03991
03992     /* Observer within atmosphere... */
03993     for (i = 0; i < 3; i++)
03994         x[i] = xobs[i];
03995
03996     /* Observer above atmosphere (search entry point)... */
03997     if (obs->obsz[ir] > zmax) {
03998         dmax = norm;
03999         while (fabs(dmin - dmax) > 0.001) {
04000             d = (dmax + dmin) / 2;
04001             for (i = 0; i < 3; i++)
04002                 x[i] = xobs[i] + d * ex0[i];
04003             cart2geo(x, &z, &lon, &lat);
04004             if (z <= zmax && z > zmax - 0.001)
04005                 break;
04006             if (z < zmax - 0.0005)
04007                 dmax = d;
04008             else
04009                 dmin = d;
04010         }
04011     }
04012
04013     /* Ray-tracing... */
```

```

04014 while (1) {
04015
04016     /* Set step length... */
04017     ds = ctl->rayds;
04018     if (ctl->raydz > 0) {
04019         norm = NORM(x);
04020         for (i = 0; i < 3; i++)
04021             xh[i] = x[i] / norm;
04022         cosa = fabs(DOTP(ex0, xh));
04023         if (cosa != 0)
04024             ds = GSL_MIN(ctl->rayds, ctl->raydz / cosa);
04025     }
04026
04027     /* Determine geolocation... */
04028     cart2geo(x, &z, &lon, &lat);
04029
04030     /* Check if LOS hits the ground or has left atmosphere... */
04031     if (z < zmin || z > zmax) {
04032         stop = (z < zmin ? 2 : 1);
04033         frac =
04034             ((z <
04035              zmin ? zmin : zmax) - los->z[los->np - 1]) / (z - los->z[los->np -
04036                                                           1]);
04037         geo2cart(los->z[los->np - 1], los->lon[los->np - 1],
04038                 los->lat[los->np - 1], xh);
04039         for (i = 0; i < 3; i++)
04040             x[i] = xh[i] + frac * (x[i] - xh[i]);
04041         cart2geo(x, &z, &lon, &lat);
04042         los->ds[los->np - 1] = ds * frac;
04043         ds = 0;
04044     }
04045
04046     /* Interpolate atmospheric data... */
04047     intpol_atm(ctl, atm, z, &p, &t, q, k);
04048
04049     /* Save data... */
04050     los->lon[los->np] = lon;
04051     los->lat[los->np] = lat;
04052     los->z[los->np] = z;
04053     los->p[los->np] = p;
04054     los->t[los->np] = t;
04055     for (ig = 0; ig < ctl->ng; ig++)
04056         los->q[ig][los->np] = q[ig];
04057     for (iw = 0; iw < ctl->nw; iw++)
04058         los->k[iw][los->np] = k[iw];
04059     los->ds[los->np] = ds;
04060
04061     /* Increment and check number of LOS points... */
04062     if ((++los->np) > NLOS)
04063         ERRMSG("Too many LOS points!");
04064
04065     /* Check stop flag... */
04066     if (stop) {
04067         los->tsurf = (stop == 2 ? t : -999);
04068         break;
04069     }
04070
04071     /* Determine refractivity... */
04072     if (ctl->refrac && z <= zrefrac)
04073         n = 1 + refractivity(p, t);
04074     else
04075         n = 1;
04076
04077     /* Construct new tangent vector (first term)... */
04078     for (i = 0; i < 3; i++)
04079         exl[i] = ex0[i] * n;
04080
04081     /* Compute gradient of refractivity... */
04082     if (ctl->refrac && z <= zrefrac) {
04083         for (i = 0; i < 3; i++)
04084             xh[i] = x[i] + 0.5 * ds * ex0[i];
04085         cart2geo(xh, &z, &lon, &lat);
04086         intpol_atm(ctl, atm, z, &p, &t, q, k);
04087         n = refractivity(p, t);
04088         for (i = 0; i < 3; i++) {
04089             xh[i] += h;
04090             cart2geo(xh, &z, &lon, &lat);
04091             intpol_atm(ctl, atm, z, &p, &t, q, k);
04092             naux = refractivity(p, t);
04093             ng[i] = (naux - n) / h;
04094             xh[i] -= h;
04095         }
04096     } else
04097         for (i = 0; i < 3; i++)
04098             ng[i] = 0;
04099
04100     /* Construct new tangent vector (second term)... */

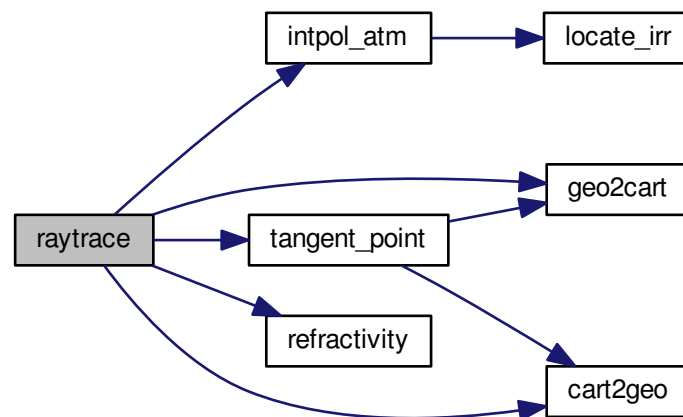
```

```

04101     for (i = 0; i < 3; i++)
04102         exl[i] += ds * ng[i];
04103
04104     /* Normalize new tangent vector... */
04105     norm = NORM(exl);
04106     for (i = 0; i < 3; i++)
04107         exl[i] /= norm;
04108
04109     /* Determine next point of LOS... */
04110     for (i = 0; i < 3; i++)
04111         x[i] += 0.5 * ds * (ex0[i] + exl[i]);
04112
04113     /* Copy tangent vector... */
04114     for (i = 0; i < 3; i++)
04115         ex0[i] = exl[i];
04116 }
04117
04118 /* Get tangent point (to be done before changing segment lengths!)... */
04119 tangent_point(los, &obs->tpz[ir], &obs->tplon[ir], &obs->
04120 tpplat[ir]);
04121
04122 /* Change segment lengths according to trapezoid rule... */
04123 for (ip = los->np - 1; ip >= 1; ip--)
04124     los->ds[ip] = 0.5 * (los->ds[ip - 1] + los->ds[ip]);
04125 los->ds[0] *= 0.5;
04126
04127 /* Compute column density... */
04128 for (ip = 0; ip < los->np; ip++)
04129     for (ig = 0; ig < ctl->ng; ig++)
04130         los->u[ig][ip] = 10 * los->q[ig][ip] * los->p[ip]
04131         / (KB * los->t[ip]) * los->ds[ip];
04132 }

```

Here is the call graph for this function:



#### 5.21.2.34 void read\_atm ( const char \* dirname, const char \* filename, ctl\_t \* ctl, atm\_t \* atm )

Read atmospheric data.

Definition at line 4135 of file [jurassic.c](#).

```

04139     {
04140
04141     FILE *in;
04142
04143     char file[LEN], line[LEN], *tok;

```



```

04144
04145     int ig, iw;
04146
04147     /* Init... */
04148     atm->np = 0;
04149
04150     /* Set filename... */
04151     if (dirname != NULL)
04152         sprintf(file, "%s/%s", dirname, filename);
04153     else
04154         sprintf(file, "%s", filename);
04155
04156     /* Write info... */
04157     printf("Read atmospheric data: %s\n", file);
04158
04159     /* Open file... */
04160     if (!(in = fopen(file, "r")))
04161         ERRMSG("Cannot open file!");
04162
04163     /* Read line... */
04164     while (fgets(line, LEN, in)) {
04165
04166         /* Read data... */
04167         TOK(line, tok, "%lg", atm->time[atm->np]);
04168         TOK(NULL, tok, "%lg", atm->z[atm->np]);
04169         TOK(NULL, tok, "%lg", atm->lon[atm->np]);
04170         TOK(NULL, tok, "%lg", atm->lat[atm->np]);
04171         TOK(NULL, tok, "%lg", atm->p[atm->np]);
04172         TOK(NULL, tok, "%lg", atm->t[atm->np]);
04173         for (ig = 0; ig < ctl->ng; ig++)
04174             TOK(NULL, tok, "%lg", atm->q[ig][atm->np]);
04175         for (iw = 0; iw < ctl->nw; iw++)
04176             TOK(NULL, tok, "%lg", atm->k[iw][atm->np]);
04177
04178         /* Increment data point counter... */
04179         if ((++atm->np) > NP)
04180             ERRMSG("Too many data points!");
04181     }
04182
04183     /* Close file... */
04184     fclose(in);
04185
04186     /* Check number of points... */
04187     if (atm->np < 1)
04188         ERRMSG("Could not read any data!");
04189 }

```

#### 5.21.2.35 void read\_ctl ( int argc, char \* argv[], ctl\_t \* ctl )

Read forward model control parameters.

Definition at line 4193 of file [jurassic.c](#).

```

04196     {
04197
04198     int id, ig, iw;
04199
04200     /* Write info... */
04201     printf("\nJuelich Rapid Spectral Simulation Code (JURASSIC)\n"
04202           "(executable: %s | compiled: %s, %s)\n\n",
04203           argv[0], __DATE__, __TIME__);
04204
04205     /* Emitters... */
04206     ctl->ng = (int) scan_ctl(argc, argv, "NG", -1, "0", NULL);
04207     if (ctl->ng < 0 || ctl->ng > NG)
04208         ERRMSG("Set 0 <= NG <= MAX!");
04209     for (ig = 0; ig < ctl->ng; ig++)
04210         scan_ctl(argc, argv, "EMITTER", ig, "", ctl->emitter[ig]);
04211
04212     /* Radiance channels... */
04213     ctl->nd = (int) scan_ctl(argc, argv, "ND", -1, "0", NULL);
04214     if (ctl->nd < 0 || ctl->nd > ND)
04215         ERRMSG("Set 0 <= ND <= MAX!");
04216     for (id = 0; id < ctl->nd; id++)
04217         ctl->nu[id] = scan_ctl(argc, argv, "NU", id, "", NULL);
04218
04219     /* Spectral windows... */
04220     ctl->nw = (int) scan_ctl(argc, argv, "NW", -1, "1", NULL);
04221     if (ctl->nw < 0 || ctl->nw > NW)
04222         ERRMSG("Set 0 <= NW <= MAX!");

```

```

04223     for (id = 0; id < ctl->nd; id++)
04224         ctl->window[id] = (int) scan_ctl(argc, argv, "WINDOW", id, "0", NULL);
04225
04226     /* Emissivity look-up tables... */
04227     scan_ctl(argc, argv, "TBLBASE", -1, "-", ctl->tblbase);
04228
04229     /* Hydrostatic equilibrium... */
04230     ctl->hydZ = scan_ctl(argc, argv, "HYDZ", -1, "-999", NULL);
04231
04232     /* Continua... */
04233     ctl->ctm_co2 = (int) scan_ctl(argc, argv, "CTM_CO2", -1, "1", NULL);
04234     ctl->ctm_h2o = (int) scan_ctl(argc, argv, "CTM_H2O", -1, "1", NULL);
04235     ctl->ctm_n2 = (int) scan_ctl(argc, argv, "CTM_N2", -1, "1", NULL);
04236     ctl->ctm_o2 = (int) scan_ctl(argc, argv, "CTM_O2", -1, "1", NULL);
04237
04238     /* Ray-tracing... */
04239     ctl->refrac = (int) scan_ctl(argc, argv, "REFRAC", -1, "1", NULL);
04240     ctl->rayds = scan_ctl(argc, argv, "RAYDS", -1, "10", NULL);
04241     ctl->raydz = scan_ctl(argc, argv, "RAYDZ", -1, "0.5", NULL);
04242
04243     /* Field of view... */
04244     scan_ctl(argc, argv, "FOV", -1, "-", ctl->fov);
04245
04246     /* Retrieval interface... */
04247     ctl->retp_zmin = scan_ctl(argc, argv, "RETP_ZMIN", -1, "-999", NULL);
04248     ctl->retp_zmax = scan_ctl(argc, argv, "RETP_ZMAX", -1, "-999", NULL);
04249     ctl->rett_zmin = scan_ctl(argc, argv, "RETT_ZMIN", -1, "-999", NULL);
04250     ctl->rett_zmax = scan_ctl(argc, argv, "RETT_ZMAX", -1, "-999", NULL);
04251     for (ig = 0; ig < ctl->ng; ig++) {
04252         ctl->retq_zmin[ig] = scan_ctl(argc, argv, "RETQ_ZMIN", ig, "-999", NULL);
04253         ctl->retq_zmax[ig] = scan_ctl(argc, argv, "RETQ_ZMAX", ig, "-999", NULL);
04254     }
04255     for (iw = 0; iw < ctl->nw; iw++) {
04256         ctl->retk_zmin[iw] = scan_ctl(argc, argv, "RETK_ZMIN", iw, "-999", NULL);
04257         ctl->retk_zmax[iw] = scan_ctl(argc, argv, "RETK_ZMAX", iw, "-999", NULL);
04258     }
04259
04260     /* Output flags... */
04261     ctl->write_bbt = (int) scan_ctl(argc, argv, "WRITE_BBT", -1, "0", NULL);
04262     ctl->write_matrix =
04263         (int) scan_ctl(argc, argv, "WRITE_MATRIX", -1, "0", NULL);
04264 }

```

Here is the call graph for this function:



#### 5.21.2.36 void read\_matrix ( const char \* *dirname*, const char \* *filename*, gsl\_matrix \* *matrix* )

Read matrix.

Definition at line 4268 of file [jurassic.c](#).

```

04271     {
04272
04273     FILE *in;
04274
04275     char dum[LEN], file[LEN], line[LEN];
04276
04277     double value;
04278
04279     int i, j;
04280
04281     /* Set filename... */

```

```

04282     if (dirname != NULL)
04283         sprintf(file, "%s/%s", dirname, filename);
04284     else
04285         sprintf(file, "%s", filename);
04286
04287     /* Write info... */
04288     printf("Read matrix: %s\n", file);
04289
04290     /* Open file... */
04291     if (!(in = fopen(file, "r")))
04292         ERRMSG("Cannot open file!");
04293
04294     /* Read data... */
04295     gsl_matrix_set_zero(matrix);
04296     while (fgets(line, LEN, in))
04297         if (sscanf(line, "%d %s %s %s %s %s %d %s %s %s %s %s %lg",
04298             &i, dum, dum, dum, dum, dum,
04299             &j, dum, dum, dum, dum, dum, &value) == 13)
04300         gsl_matrix_set(matrix, (size_t) i, (size_t) j, value);
04301
04302     /* Close file... */
04303     fclose(in);
04304 }

```

**5.21.2.37** void read\_obs ( const char \* *dirname*, const char \* *filename*, ctl\_t \* *ctl*, obs\_t \* *obs* )

Read observation data.

Definition at line 4308 of file [jurassic.c](#).

```

04312         {
04313
04314         FILE *in;
04315
04316         char file[LEN], line[LEN], *tok;
04317
04318         int id;
04319
04320         /* Init... */
04321         obs->nr = 0;
04322
04323         /* Set filename... */
04324         if (dirname != NULL)
04325             sprintf(file, "%s/%s", dirname, filename);
04326         else
04327             sprintf(file, "%s", filename);
04328
04329         /* Write info... */
04330         printf("Read observation data: %s\n", file);
04331
04332         /* Open file... */
04333         if (!(in = fopen(file, "r")))
04334             ERRMSG("Cannot open file!");
04335
04336         /* Read line... */
04337         while (fgets(line, LEN, in)) {
04338
04339             /* Read data... */
04340             TOK(line, tok, "%lg", obs->time[obs->nr]);
04341             TOK(NULL, tok, "%lg", obs->obsz[obs->nr]);
04342             TOK(NULL, tok, "%lg", obs->obslon[obs->nr]);
04343             TOK(NULL, tok, "%lg", obs->obslat[obs->nr]);
04344             TOK(NULL, tok, "%lg", obs->vpz[obs->nr]);
04345             TOK(NULL, tok, "%lg", obs->vplon[obs->nr]);
04346             TOK(NULL, tok, "%lg", obs->vplat[obs->nr]);
04347             TOK(NULL, tok, "%lg", obs->tpz[obs->nr]);
04348             TOK(NULL, tok, "%lg", obs->tplon[obs->nr]);
04349             TOK(NULL, tok, "%lg", obs->tplat[obs->nr]);
04350             for (id = 0; id < ctl->nd; id++)
04351                 TOK(NULL, tok, "%lg", obs->rad[id][obs->nr]);
04352             for (id = 0; id < ctl->nd; id++)
04353                 TOK(NULL, tok, "%lg", obs->tau[id][obs->nr]);
04354
04355             /* Increment counter... */
04356             if ((++obs->nr) > NR)
04357                 ERRMSG("Too many rays!");
04358         }
04359
04360         /* Close file... */
04361         fclose(in);

```

```

04362
04363  /* Check number of points... */
04364  if (obs->nr < 1)
04365      ERRMSG("Could not read any data!");
04366  }

```

#### 5.21.2.38 void read\_shape ( const char \* filename, double \* x, double \* y, int \* n )

Read shape function.

Definition at line 4370 of file [jurassic.c](#).

```

04374      {
04375
04376      FILE *in;
04377
04378      char line[LEN];
04379
04380      /* Write info... */
04381      printf("Read shape function: %s\n", filename);
04382
04383      /* Open file... */
04384      if (!(in = fopen(filename, "r")))
04385          ERRMSG("Cannot open file!");
04386
04387      /* Read data... */
04388      *n = 0;
04389      while (fgets(line, LEN, in))
04390          if (sscanf(line, "%lg %lg", &x[*n], &y[*n]) == 2)
04391              if (++(*n) > NSHAPE)
04392                  ERRMSG("Too many data points!");
04393
04394      /* Check number of points... */
04395      if (*n < 1)
04396          ERRMSG("Could not read any data!");
04397
04398      /* Close file... */
04399      fclose(in);
04400  }

```

#### 5.21.2.39 double refractivity ( double p, double t )

Compute refractivity (return value is n - 1).

Definition at line 4404 of file [jurassic.c](#).

```

04406      {
04407
04408      /* Refractivity of air at 4 to 15 micron... */
04409      return 7.753e-05 * p / t;
04410  }

```

#### 5.21.2.40 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 4414 of file [jurassic.c](#).

```

04420         {
04421
04422     FILE *in = NULL;
04423
04424     char dummy[LEN], fullname1[LEN], fullname2[LEN], line[LEN],
04425           msg[2 * LEN], rvarname[LEN], rval[LEN];
04426
04427     int contain = 0, i;
04428
04429     /* Open file... */
04430     if (argv[1][0] != '-')
04431         if (!(in = fopen(argv[1], "r")))
04432             ERRMSG("Cannot open file!");
04433
04434     /* Set full variable name... */
04435     if (arridx >= 0) {
04436         sprintf(fullname1, "%s[%d]", varname, arridx);
04437         sprintf(fullname2, "%s[*]", varname);
04438     } else {
04439         sprintf(fullname1, "%s", varname);
04440         sprintf(fullname2, "%s", varname);
04441     }
04442
04443     /* Read data... */
04444     if (in != NULL)
04445         while (fgets(line, LEN, in))
04446             if (sscanf(line, "%s %s %s", rvarname, dummy, rval) == 3)
04447                 if (strcasemp(rvarname, fullname1) == 0 ||
04448                     strcasemp(rvarname, fullname2) == 0) {
04449                     contain = 1;
04450                     break;
04451                 }
04452     for (i = 1; i < argc - 1; i++)
04453         if (strcasemp(argv[i], fullname1) == 0 ||
04454             strcasemp(argv[i], fullname2) == 0) {
04455             sprintf(rval, "%s", argv[i + 1]);
04456             contain = 1;
04457             break;
04458         }
04459
04460     /* Close file... */
04461     if (in != NULL)
04462         fclose(in);
04463
04464     /* Check for missing variables... */
04465     if (!contain) {
04466         if (strlen(defvalue) > 0)
04467             sprintf(rval, "%s", defvalue);
04468         else {
04469             sprintf(msg, "Missing variable %s!\n", fullname1);
04470             ERRMSG(msg);
04471         }
04472     }
04473
04474     /* Write info... */
04475     printf("%s = %s\n", fullname1, rval);
04476
04477     /* Return values... */
04478     if (value != NULL)
04479         sprintf(value, "%s", rval);
04480     return atof(rval);
04481 }

```

#### 5.21.2.41 void tangent\_point ( los\_t \* los, double \* tpz, double \* tplon, double \* tplat )

Find tangent point of a given LOS.

Definition at line 4485 of file [jurassic.c](#).

```

04489         {
04490
04491     double a, b, c, dummy, v[3], v0[3], v2[3], x, x1, x2, yy0, yy1, yy2;
04492
04493     size_t i, ip;
04494
04495     /* Find minimum altitude... */
04496     ip = gsl_stats_min_index(los->z, 1, (size_t) los->np);
04497
04498     /* Nadir or zenith... */
04499     if (ip <= 0 || ip >= (size_t) los->np - 1) {

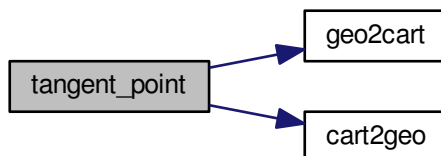
```

```

04500     *tpz = los->z[los->np - 1];
04501     *tplon = los->lon[los->np - 1];
04502     *tplat = los->lat[los->np - 1];
04503 }
04504
04505 /* Limb... */
04506 else {
04507
04508     /* Determine interpolating polynomial y=a*x^2+b*x+c... */
04509     yy0 = los->z[ip - 1];
04510     yy1 = los->z[ip];
04511     yy2 = los->z[ip + 1];
04512     x1 = sqrt(POW2(los->ds[ip]) - POW2(yy1 - yy0));
04513     x2 = x1 + sqrt(POW2(los->ds[ip + 1]) - POW2(yy2 - yy1));
04514     a = 1 / (x1 - x2) * (-(yy0 - yy1) / x1 + (yy0 - yy2) / x2);
04515     b = -(yy0 - yy1) / x1 - a * x1;
04516     c = yy0;
04517
04518     /* Get tangent point location... */
04519     x = -b / (2 * a);
04520     *tpz = a * x * x + b * x + c;
04521     geo2cart(los->z[ip - 1], los->lon[ip - 1], los->lat[ip - 1], v0);
04522     geo2cart(los->z[ip + 1], los->lon[ip + 1], los->lat[ip + 1], v2);
04523     for (i = 0; i < 3; i++)
04524         v[i] = LIN(0.0, v0[i], x2, v2[i], x);
04525     cart2geo(v, &dummy, tplon, tplat);
04526 }
04527 }

```

Here is the call graph for this function:



#### 5.21.2.42 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 4531 of file [jurassic.c](#).

```

04539     {
04540
04541     struct tm t0, t1;
04542
04543     t0.tm_year = 100;
04544     t0.tm_mon = 0;
04545     t0.tm_mday = 1;
04546     t0.tm_hour = 0;
04547     t0.tm_min = 0;
04548     t0.tm_sec = 0;
04549
04550     t1.tm_year = year - 1900;
04551     t1.tm_mon = mon - 1;
04552     t1.tm_mday = day;
04553     t1.tm_hour = hour;
04554     t1.tm_min = min;
04555     t1.tm_sec = sec;
04556
04557     *jsec = (double) timegm(&t1) - (double) timegm(&t0) + remain;
04558 }

```

#### 5.21.2.43 void timer ( const char \* name, const char \* file, const char \* func, int line, int mode )

Measure wall-clock time.

Definition at line 4562 of file [jurassic.c](#).

```

04567         {
04568
04569     static double w0[10];
04570
04571     static int l0[10], nt;
04572
04573     /* Start new timer... */
04574     if (mode == 1) {
04575         w0[nt] = omp_get_wtime();
04576         l0[nt] = line;
04577         if ((++nt) >= 10)
04578             ERRMSG("Too many timers!");
04579     }
04580
04581     /* Write elapsed time... */
04582     else {
04583
04584         /* Check timer index... */
04585         if (nt - 1 < 0)
04586             ERRMSG("Coding error!");
04587
04588         /* Write elapsed time... */
04589         printf("Timer '%s' (%s, %s, l%d-%d): %.3f sec\n",
04590             name, file, func, l0[nt - 1], line, omp_get_wtime() - w0[nt - 1]);
04591     }
04592
04593     /* Stop timer... */
04594     if (mode == 3)
04595         nt--;
04596 }

```

#### 5.21.2.44 void write\_atm ( const char \* dirname, const char \* filename, ctl\_t \* ctl, atm\_t \* atm )

Write atmospheric data.

Definition at line 4600 of file [jurassic.c](#).

```

04604         {
04605
04606     FILE *out;
04607
04608     char file[LEN];
04609
04610     int ig, ip, iw, n = 6;
04611
04612     /* Set filename... */
04613     if (dirname != NULL)
04614         sprintf(file, "%s/%s", dirname, filename);
04615     else
04616         sprintf(file, "%s", filename);
04617
04618     /* Write info... */
04619     printf("Write atmospheric data: %s\n", file);
04620
04621     /* Create file... */
04622     if (!(out = fopen(file, "w")))
04623         ERRMSG("Cannot create file!");
04624
04625     /* Write header... */
04626     fprintf(out,
04627         "# $1 = time (seconds since 2000-01-01T00:00Z)\n"
04628         "# $2 = altitude [km]\n"
04629         "# $3 = longitude [deg]\n"
04630         "# $4 = latitude [deg]\n"
04631         "# $5 = pressure [hPa]\n" "# $6 = temperature [K]\n");
04632     for (ig = 0; ig < ctl->ng; ig++)
04633         fprintf(out, "# $%d = %s volume mixing ratio\n", ++n, ctl->emitter[ig]);
04634     for (iw = 0; iw < ctl->nw; iw++)
04635         fprintf(out, "# $%d = window %d: extinction [1/km]\n", ++n, iw);
04636 }

```

```

04637  /* Write data... */
04638  for (ip = 0; ip < atm->np; ip++) {
04639      if (ip == 0 || atm->lat[ip] != atm->lat[ip - 1]
04640          || atm->lon[ip] != atm->lon[ip - 1])
04641          fprintf(out, "\n");
04642      fprintf(out, "%.2f %g %g %g %g", atm->time[ip], atm->z[ip],
04643          atm->lon[ip], atm->lat[ip], atm->p[ip], atm->t[ip]);
04644      for (ig = 0; ig < ctl->ng; ig++)
04645          fprintf(out, " %g", atm->q[ig][ip]);
04646      for (iw = 0; iw < ctl->nw; iw++)
04647          fprintf(out, " %g", atm->k[iw][ip]);
04648      fprintf(out, "\n");
04649  }
04650
04651  /* Close file... */
04652  fclose(out);
04653 }

```

**5.21.2.45** void write\_matrix ( const char \* dirname, const char \* filename, ctl\_t \* ctl, gsl\_matrix \* matrix, atm\_t \* atm, obs\_t \* obs, const char \* row\_space, const char \* col\_space, const char \* sort )

Write matrix.

Definition at line 4657 of file [jurassic.c](#).

```

04666      {
04667
04668      FILE *out;
04669
04670      char file[LEN], quantity[LEN];
04671
04672      int *cida, *ciqa, *cipa, *cira, *rida, *riqa, *ripa, *rira;
04673
04674      size_t i, j, nc, nr;
04675
04676      /* Check output flag... */
04677      if (!ctl->write_matrix)
04678          return;
04679
04680      /* Allocate... */
04681      ALLOC(cida, int, M);
04682      ALLOC(ciqa, int,
04683          N);
04684      ALLOC(cipa, int,
04685          N);
04686      ALLOC(cira, int,
04687          M);
04688      ALLOC(rida, int,
04689          M);
04690      ALLOC(riqa, int,
04691          N);
04692      ALLOC(ripa, int,
04693          N);
04694      ALLOC(rira, int,
04695          M);
04696
04697      /* Set filename... */
04698      if (dirname != NULL)
04699          sprintf(file, "%s/%s", dirname, filename);
04700      else
04701          sprintf(file, "%s", filename);
04702
04703      /* Write info... */
04704      printf("Write matrix: %s\n", file);
04705
04706      /* Create file... */
04707      if (!(out = fopen(file, "w")))
04708          ERRMSG("Cannot create file!");
04709
04710      /* Write header (row space)... */
04711      if (row_space[0] == 'y') {
04712          fprintf(out,
04713              "# $1 = Row: index (measurement space)\n"
04714              "# $2 = Row: channel wavenumber [cm^-1]\n"
04715              "# $3 = Row: time (seconds since 2000-01-01T00:00Z)\n"
04716              "# $4 = Row: view point altitude [km]\n"
04717              "# $5 = Row: view point longitude [deg]\n"
04718              "# $6 = Row: view point latitude [deg]\n");
04719
04720

```



```

04721      /* Get number of rows... */
04722      nr = obs2y(ctl, obs, NULL, rida, rira);
04723
04724  } else {
04725
04726      fprintf(out,
04727              "# $1 = Row: index (state space)\n"
04728              "# $2 = Row: name of quantity\n"
04729              "# $3 = Row: time (seconds since 2000-01-01T00:00Z)\n"
04730              "# $4 = Row: altitude [km]\n"
04731              "# $5 = Row: longitude [deg]\n" "# $6 = Row: latitude [deg]\n");
04732
04733      /* Get number of rows... */
04734      nr = atm2x(ctl, atm, NULL, riq, ripa);
04735  }
04736
04737  /* Write header (column space)... */
04738  if (colspace[0] == 'y') {
04739
04740      fprintf(out,
04741              "# $7 = Col: index (measurement space)\n"
04742              "# $8 = Col: channel wavenumber [cm^-1]\n"
04743              "# $9 = Col: time (seconds since 2000-01-01T00:00Z)\n"
04744              "# $10 = Col: view point altitude [km]\n"
04745              "# $11 = Col: view point longitude [deg]\n"
04746              "# $12 = Col: view point latitude [deg]\n");
04747
04748      /* Get number of columns... */
04749      nc = obs2y(ctl, obs, NULL, cida, cira);
04750
04751  } else {
04752
04753      fprintf(out,
04754              "# $7 = Col: index (state space)\n"
04755              "# $8 = Col: name of quantity\n"
04756              "# $9 = Col: time (seconds since 2000-01-01T00:00Z)\n"
04757              "# $10 = Col: altitude [km]\n"
04758              "# $11 = Col: longitude [deg]\n" "# $12 = Col: latitude [deg]\n");
04759
04760      /* Get number of columns... */
04761      nc = atm2x(ctl, atm, NULL, cira, cipa);
04762  }
04763
04764  /* Write header entry... */
04765  fprintf(out, "# $13 = Matrix element\n\n");
04766
04767  /* Write matrix data... */
04768  i = j = 0;
04769  while (i < nr && j < nc) {
04770
04771      /* Write info about the row... */
04772      if (rowspan[0] == 'y')
04773          fprintf(out, "%d %g %.2f %g %g %g",
04774                  (int) i, ctl->nu[rida[i]],
04775                  obs->time[rira[i]], obs->vpz[rira[i]],
04776                  obs->vplon[rira[i]], obs->vplat[rira[i]]);
04777      else {
04778          idx2name(ctl, riq[i], quantity);
04779          fprintf(out, "%d %s %.2f %g %g %g", (int) i, quantity,
04780                  atm->time[ripa[i]], atm->z[ripa[i]],
04781                  atm->lon[ripa[i]], atm->lat[ripa[i]]);
04782      }
04783
04784      /* Write info about the column... */
04785      if (colspace[0] == 'y')
04786          fprintf(out, " %d %g %.2f %g %g %g",
04787                  (int) j, ctl->nu[cida[j]],
04788                  obs->time[cira[j]], obs->vpz[cira[j]],
04789                  obs->vplon[cira[j]], obs->vplat[cira[j]]);
04790      else {
04791          idx2name(ctl, cira[j], quantity);
04792          fprintf(out, " %d %s %.2f %g %g %g", (int) j, quantity,
04793                  atm->time[cipa[j]], atm->z[cipa[j]],
04794                  atm->lon[cipa[j]], atm->lat[cipa[j]]);
04795      }
04796
04797      /* Write matrix entry... */
04798      fprintf(out, " %g\n", gsl_matrix_get(matrix, i, j));
04799
04800      /* Set matrix indices... */
04801      if (sort[0] == 'r') {
04802          j++;
04803          if (j >= nc) {
04804              j = 0;
04805              i++;
04806              fprintf(out, "\n");
04807          }
04808      }

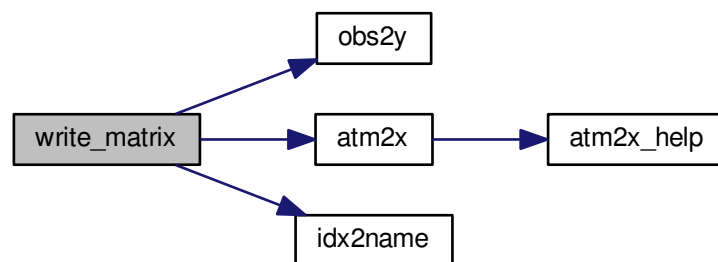
```

```

04808     } else {
04809         i++;
04810         if (i >= nr) {
04811             i = 0;
04812             j++;
04813             fprintf(out, "\n");
04814         }
04815     }
04816 }
04817
04818 /* Close file... */
04819 fclose(out);
04820
04821 /* Free... */
04822 free(cida);
04823 free(ciga);
04824 free(cipa);
04825 free(cira);
04826 free(rida);
04827 free(riqa);
04828 free(ripa);
04829 free(rira);
04830 }

```

Here is the call graph for this function:



#### 5.21.2.46 void write\_obs ( const char \* *dirname*, const char \* *filename*, *ctl\_t* \* *ctl*, *obs\_t* \* *obs* )

Write observation data.

Definition at line 4834 of file [jurassic.c](#).

```

04838     {
04839
04840     FILE *out;
04841
04842     char file[LEN];
04843
04844     int id, ir, n = 10;
04845
04846     /* Set filename... */
04847     if (dirname != NULL)
04848         sprintf(file, "%s/%s", dirname, filename);
04849     else
04850         sprintf(file, "%s", filename);
04851
04852     /* Write info... */
04853     printf("Write observation data: %s\n", file);
04854
04855     /* Create file... */
04856     if (!(out = fopen(file, "w")))
04857         ERRMSG("Cannot create file!");
04858

```

```

04859  /* Write header... */
04860  fprintf(out,
04861          "# $1 = time (seconds since 2000-01-01T00:00Z)\n"
04862          "# $2 = observer altitude [km]\n"
04863          "# $3 = observer longitude [deg]\n"
04864          "# $4 = observer latitude [deg]\n"
04865          "# $5 = view point altitude [km]\n"
04866          "# $6 = view point longitude [deg]\n"
04867          "# $7 = view point latitude [deg]\n"
04868          "# $8 = tangent point altitude [km]\n"
04869          "# $9 = tangent point longitude [deg]\n"
04870          "# $10 = tangent point latitude [deg]\n");
04871  for (id = 0; id < ctl->nd; id++)
04872      fprintf(out, "# $d = channel %g: radiance [W/(m^2 sr cm^-1)]\n",
04873              ++n, ctl->nu[id]);
04874  for (id = 0; id < ctl->nd; id++)
04875      fprintf(out, "# $d = channel %g: transmittance\n", ++n, ctl->nu[id]);
04876
04877  /* Write data... */
04878  for (ir = 0; ir < obs->nr; ir++) {
04879      if (ir == 0 || obs->time[ir] != obs->time[ir - 1])
04880          fprintf(out, "\n");
04881      fprintf(out, "%.2f %g %g %g %g %g %g %g %g", obs->time[ir],
04882              obs->obsz[ir], obs->obslon[ir], obs->obslat[ir],
04883              obs->vpz[ir], obs->vplon[ir], obs->vplat[ir],
04884              obs->tpz[ir], obs->tplon[ir], obs->tplat[ir]);
04885      for (id = 0; id < ctl->nd; id++)
04886          fprintf(out, " %g", obs->rad[id][ir]);
04887      for (id = 0; id < ctl->nd; id++)
04888          fprintf(out, " %g", obs->tau[id][ir]);
04889      fprintf(out, "\n");
04890  }
04891
04892  /* Close file... */
04893  fclose(out);
04894 }

```

#### 5.21.2.47 void x2atm ( ctl\_t \* ctl, gsl\_vector \* x, atm\_t \* atm )

Decompose parameter vector or state vector.

Definition at line 4898 of file [jurassic.c](#).

```

04901      {
04902
04903      int ig, iw;
04904
04905      size_t n = 0;
04906
04907      /* Set pressure... */
04908      x2atm_help(atm, ctl->retp_zmin, ctl->retp_zmax, atm->
04909                  p, x, &n);
04909
04910      /* Set temperature... */
04911      x2atm_help(atm, ctl->rett_zmin, ctl->rett_zmax, atm->
04912                  t, x, &n);
04912
04913      /* Set volume mixing ratio... */
04914      for (ig = 0; ig < ctl->ng; ig++)
04915          x2atm_help(atm, ctl->retq_zmin[ig], ctl->retq_zmax[ig],
04916                      atm->q[ig], x, &n);
04917
04918      /* Set extinction... */
04919      for (iw = 0; iw < ctl->nw; iw++)
04920          x2atm_help(atm, ctl->retk_zmin[iw], ctl->retk_zmax[iw],
04921                      atm->k[iw], x, &n);
04922 }

```

Here is the call graph for this function:



5.21.2.48 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 4926 of file jurassic.c.

```

04932         {
04933
04934     int ip;
04935
04936     /* Extract state vector elements... */
04937     for (ip = 0; ip < atm->np; ip++)
04938         if (atm->z[ip] >= zmin && atm->z[ip] <= zmax) {
04939             value[ip] = gsl_vector_get(x, *n);
04940             (*n)++;
04941         }
04942 }
```

5.21.2.49 void y2obs ( ctl\_t \* ctl, gsl\_vector \* y, obs\_t \* obs )

Decompose measurement vector.

Definition at line 4946 of file jurassic.c.

```

04949         {
04950
04951     int id, ir;
04952
04953     size_t m = 0;
04954
04955     /* Decompose measurement vector... */
04956     for (ir = 0; ir < obs->nr; ir++)
04957         for (id = 0; id < ctl->nd; id++)
04958             if (gsl_finite(obs->rad[id][ir])) {
04959                 obs->rad[id][ir] = gsl_vector_get(y, m);
04960                 m++;
04961             }
04962 }
```

## 5.22 jurassic.c

```

00001 /*
00002  This file is part of JURASSIC.
00003
00004  JURASSIC is free software: you can redistribute it and/or modify
00005  it under the terms of the GNU General Public License as published by
00006  the Free Software Foundation, either version 3 of the License, or
00007  (at your option) any later version.
00008
00009  JURASSIC is distributed in the hope that it will be useful,
00010  but WITHOUT ANY WARRANTY; without even the implied warranty of
00011  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012  GNU General Public License for more details.
00013
00014  You should have received a copy of the GNU General Public License
00015  along with JURASSIC. If not, see <http://www.gnu.org/licenses/>.
00016
00017  Copright (C) 2003-2015 Forschungszentrum Juelich GmbH
00018 */
00019
00025 #include "jurassic.h"
00026
00027 /*****
00028
00029 size_t atm2x(
00030     ctl_t * ctl,
00031     atm_t * atm,
00032     gsl_vector * x,
00033     int *iga,
00034     int *ipa) {
00035
00036     int ig, iw;
```

```

00037
00038     size_t n = 0;
00039
00040     /* Add pressure... */
00041     atm2x_help(atm, ctl->retp_zmin, ctl->retp_zmax,
00042               atm->p, IDXP, x, iqa, ipa, &n);
00043
00044     /* Add temperature... */
00045     atm2x_help(atm, ctl->rett_zmin, ctl->rett_zmax,
00046               atm->t, IDXT, x, iqa, ipa, &n);
00047
00048     /* Add volume mixing ratios... */
00049     for (ig = 0; ig < ctl->ng; ig++)
00050         atm2x_help(atm, ctl->retq_zmin[ig], ctl->retq_zmax[ig],
00051                   atm->q[ig], IDXQ(ig), x, iqa, ipa, &n);
00052
00053     /* Add extinction... */
00054     for (iw = 0; iw < ctl->nw; iw++)
00055         atm2x_help(atm, ctl->retk_zmin[iw], ctl->retk_zmax[iw],
00056                   atm->k[iw], IDXK(iw), x, iqa, ipa, &n);
00057
00058     return n;
00059 }
00060
00061 /*****
00062
00063 void atm2x_help(
00064     atm_t * atm,
00065     double zmin,
00066     double zmax,
00067     double *value,
00068     int val_iqa,
00069     gsl_vector * x,
00070     int *iqa,
00071     int *ipa,
00072     size_t * n) {
00073
00074     int ip;
00075
00076     /* Add elements to state vector... */
00077     for (ip = 0; ip < atm->np; ip++)
00078         if (atm->z[ip] >= zmin && atm->z[ip] <= zmax) {
00079             if (x != NULL)
00080                 gsl_vector_set(x, *n, value[ip]);
00081             if (iqa != NULL)
00082                 iqa[*n] = val_iqa;
00083             if (ipa != NULL)
00084                 ipa[*n] = ip;
00085             (*n)++;
00086         }
00087 }
00088
00089 /*****
00090
00091 double brightness(
00092     double rad,
00093     double nu) {
00094
00095     return C2 * nu / gsl_loglp(C1 * POW3(nu) / rad);
00096 }
00097
00098 /*****
00099
00100 void cart2geo(
00101     double *x,
00102     double *z,
00103     double *lon,
00104     double *lat) {
00105
00106     double radius;
00107
00108     radius = NORM(x);
00109     *lat = asin(x[2] / radius) * 180 / M_PI;
00110     *lon = atan2(x[1], x[0]) * 180 / M_PI;
00111     *z = radius - RE;
00112 }
00113
00114 /*****
00115
00116 void climatology(
00117     ctl_t * ctl,
00118     atm_t * atm) {
00119
00120     static double z[121] = {
00121         0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19,
00122         20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37,

```

```

00124     38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55,
00125     56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73,
00126     74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91,
00127     92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107,
00128     108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 120
00129 };
00130
00131 static double pre[121] = {
00132     1017, 901.083, 796.45, 702.227, 617.614, 541.644, 473.437, 412.288,
00133     357.603, 308.96, 265.994, 228.348, 195.619, 167.351, 143.039, 122.198,
00134     104.369, 89.141, 76.1528, 65.0804, 55.641, 47.591, 40.7233, 34.8637,
00135     29.8633, 25.5956, 21.9534, 18.8445, 16.1909, 13.9258, 11.9913,
00136     10.34, 8.92988, 7.72454, 6.6924, 5.80701, 5.04654, 4.39238, 3.82902,
00137     3.34337, 2.92413, 2.56128, 2.2464, 1.97258, 1.73384, 1.52519, 1.34242,
00138     1.18197, 1.04086, 0.916546, 0.806832, 0.709875, 0.624101, 0.548176,
00139     0.480974, 0.421507, 0.368904, 0.322408, 0.281386, 0.245249, 0.213465,
00140     0.185549, 0.161072, 0.139644, 0.120913, 0.104568, 0.0903249, 0.0779269,
00141     0.0671493, 0.0577962, 0.0496902, 0.0426736, 0.0366093, 0.0313743,
00142     0.0268598, 0.0229699, 0.0196206, 0.0167399, 0.0142646, 0.0121397,
00143     0.0103181, 0.00875775, 0.00742226, 0.00628076, 0.00530519, 0.00447183,
00144     0.00376124, 0.00315632, 0.00264248, 0.00220738, 0.00184003, 0.00153095,
00145     0.00127204, 0.00105608, 0.000876652, 0.00072798, 0.00060492,
00146     0.000503201, 0.000419226, 0.000349896, 0.000292659, 0.000245421,
00147     0.000206394, 0.000174125, 0.000147441, 0.000125333, 0.000106985,
00148     9.173e-05, 7.90172e-05, 6.84172e-05, 5.95574e-05, 5.21183e-05,
00149     4.58348e-05, 4.05127e-05, 3.59987e-05, 3.21583e-05, 2.88718e-05,
00150     2.60322e-05, 2.35687e-05, 2.14263e-05, 1.95489e-05
00151 };
00152
00153 static double tem[121] = {
00154     285.14, 279.34, 273.91, 268.3, 263.24, 256.55, 250.2, 242.82, 236.17,
00155     229.87, 225.04, 221.19, 218.85, 217.19, 216.2, 215.68, 215.42, 215.55,
00156     215.92, 216.4, 216.93, 217.45, 218, 218.68, 219.39, 220.25, 221.3,
00157     222.41, 223.88, 225.42, 227.2, 229.52, 231.89, 234.51, 236.85, 239.42,
00158     241.94, 244.57, 247.36, 250.32, 253.34, 255.82, 258.27, 260.39,
00159     262.03, 263.45, 264.2, 264.78, 264.67, 264.38, 263.24, 262.03, 260.02,
00160     258.09, 255.63, 253.28, 250.43, 247.81, 245.26, 242.77, 240.38,
00161     237.94, 235.79, 233.53, 231.5, 229.53, 227.6, 225.62, 223.77, 222.06,
00162     220.33, 218.69, 217.18, 215.64, 214.13, 212.52, 210.86, 209.25,
00163     207.49, 205.81, 204.11, 202.22, 200.32, 198.39, 195.92, 193.46,
00164     190.94, 188.31, 185.82, 183.57, 181.43, 179.74, 178.64, 178.1, 178.25,
00165     178.7, 179.41, 180.67, 182.31, 184.18, 186.6, 189.53, 192.66, 196.54,
00166     201.13, 205.93, 211.73, 217.86, 225, 233.53, 242.57, 252.14, 261.48,
00167     272.97, 285.26, 299.12, 312.2, 324.17, 338.34, 352.56, 365.28
00168 };
00169
00170 static double c2h2[121] = {
00171     1.352e-09, 2.83e-10, 1.269e-10, 6.926e-11, 4.346e-11, 2.909e-11,
00172     2.014e-11, 1.363e-11, 8.71e-12, 5.237e-12, 2.718e-12, 1.375e-12,
00173     5.786e-13, 2.16e-13, 7.317e-14, 2.551e-14, 1.055e-14, 4.758e-15,
00174     2.056e-15, 7.703e-16, 2.82e-16, 1.035e-16, 4.382e-17, 1.946e-17,
00175     9.638e-18, 5.2e-18, 2.811e-18, 1.494e-18, 7.925e-19, 4.213e-19,
00176     1.998e-19, 8.78e-20, 3.877e-20, 1.728e-20, 7.743e-21, 3.536e-21,
00177     1.623e-21, 7.508e-22, 3.508e-22, 1.65e-22, 7.837e-23, 3.733e-23,
00178     1.808e-23, 8.77e-24, 4.285e-24, 2.095e-24, 1.032e-24, 5.082e-25,
00179     2.506e-25, 1.236e-25, 6.088e-26, 2.996e-26, 1.465e-26, 0, 0, 0,
00180     0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
00181     0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
00182     0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
00183 };
00184
00185 static double c2h6[121] = {
00186     2.667e-09, 2.02e-09, 1.658e-09, 1.404e-09, 1.234e-09, 1.109e-09,
00187     1.012e-09, 9.262e-10, 8.472e-10, 7.71e-10, 6.932e-10, 6.216e-10,
00188     5.503e-10, 4.872e-10, 4.342e-10, 3.861e-10, 3.347e-10, 2.772e-10,
00189     2.209e-10, 1.672e-10, 1.197e-10, 8.536e-11, 5.783e-11, 3.846e-11,
00190     2.495e-11, 1.592e-11, 1.017e-11, 6.327e-12, 3.895e-12, 2.403e-12,
00191     1.416e-12, 8.101e-13, 4.649e-13, 2.686e-13, 1.557e-13, 9.14e-14,
00192     5.386e-14, 3.19e-14, 1.903e-14, 1.14e-14, 6.875e-15, 4.154e-15,
00193     2.538e-15, 1.553e-15, 9.548e-16, 5.872e-16, 3.63e-16, 2.244e-16,
00194     1.388e-16, 8.587e-17, 5.308e-17, 3.279e-17, 2.017e-17, 1.238e-17,
00195     7.542e-18, 4.585e-18, 2.776e-18, 1.671e-18, 9.985e-19, 5.937e-19,
00196     3.518e-19, 2.07e-19, 1.215e-19, 7.06e-20, 4.097e-20, 2.37e-20,
00197     1.363e-20, 7.802e-21, 4.441e-21, 2.523e-21, 1.424e-21, 8.015e-22,
00198     4.497e-22, 2.505e-22, 1.391e-22, 7.691e-23, 4.238e-23, 2.331e-23,
00199     1.274e-23, 6.929e-24, 3.752e-24, 2.02e-24, 1.083e-24, 5.774e-25,
00200     3.041e-25, 1.593e-25, 8.308e-26, 4.299e-26, 2.195e-26, 1.112e-26,
00201     0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
00202     0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
00203 };
00204
00205 static double ccl4[121] = {
00206     1.075e-10, 1.075e-10, 1.075e-10, 1.075e-10, 1.075e-10, 1.075e-10,
00207     1.075e-10, 1.075e-10, 1.075e-10, 1.06e-10, 1.024e-10, 9.69e-11,
00208     8.93e-11, 8.078e-11, 7.213e-11, 6.307e-11, 5.383e-11, 4.49e-11,
00209     3.609e-11, 2.705e-11, 1.935e-11, 1.385e-11, 8.35e-12, 5.485e-12,
00210     3.853e-12, 2.22e-12, 5.875e-13, 3.445e-13, 1.015e-13, 6.075e-14,

```

```
00211      4.383e-14, 2.692e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14,
00212      1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14,
00213      1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14,
00214      1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14,
00215      1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14,
00216      1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14,
00217      1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14,
00218      1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14,
00219      1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14,
00220      1e-14, 1e-14, 1e-14
00221  };
00222
00223  static double ch4[121] = {
00224      1.864e-06, 1.835e-06, 1.819e-06, 1.805e-06, 1.796e-06, 1.788e-06,
00225      1.782e-06, 1.776e-06, 1.769e-06, 1.761e-06, 1.749e-06, 1.734e-06,
00226      1.716e-06, 1.692e-06, 1.654e-06, 1.61e-06, 1.567e-06, 1.502e-06,
00227      1.433e-06, 1.371e-06, 1.323e-06, 1.277e-06, 1.232e-06, 1.188e-06,
00228      1.147e-06, 1.108e-06, 1.07e-06, 1.027e-06, 9.854e-07, 9.416e-07,
00229      8.933e-07, 8.478e-07, 7.988e-07, 7.515e-07, 7.07e-07, 6.64e-07,
00230      6.239e-07, 5.864e-07, 5.512e-07, 5.184e-07, 4.87e-07, 4.571e-07,
00231      4.296e-07, 4.04e-07, 3.802e-07, 3.578e-07, 3.383e-07, 3.203e-07,
00232      3.032e-07, 2.889e-07, 2.76e-07, 2.635e-07, 2.519e-07, 2.409e-07,
00233      2.302e-07, 2.219e-07, 2.144e-07, 2.071e-07, 1.999e-07, 1.93e-07,
00234      1.862e-07, 1.795e-07, 1.731e-07, 1.668e-07, 1.607e-07, 1.548e-07,
00235      1.49e-07, 1.434e-07, 1.38e-07, 1.328e-07, 1.277e-07, 1.227e-07,
00236      1.18e-07, 1.134e-07, 1.089e-07, 1.046e-07, 1.004e-07, 9.635e-08,
00237      9.245e-08, 8.867e-08, 8.502e-08, 8.15e-08, 7.809e-08, 7.48e-08,
00238      7.159e-08, 6.849e-08, 6.55e-08, 6.262e-08, 5.98e-08, 5.708e-08,
00239      5.448e-08, 5.194e-08, 4.951e-08, 4.72e-08, 4.5e-08, 4.291e-08,
00240      4.093e-08, 3.905e-08, 3.729e-08, 3.563e-08, 3.408e-08, 3.265e-08,
00241      3.128e-08, 2.996e-08, 2.87e-08, 2.76e-08, 2.657e-08, 2.558e-08,
00242      2.467e-08, 2.385e-08, 2.307e-08, 2.234e-08, 2.168e-08, 2.108e-08,
00243      2.05e-08, 1.998e-08, 1.947e-08, 1.902e-08, 1.86e-08, 1.819e-08,
00244      1.782e-08
00245  };
00246
00247  static double clo[121] = {
00248      7.419e-15, 1.061e-14, 1.518e-14, 2.195e-14, 3.175e-14, 4.666e-14,
00249      6.872e-14, 1.03e-13, 1.553e-13, 2.375e-13, 3.664e-13, 5.684e-13,
00250      8.915e-13, 1.402e-12, 2.269e-12, 4.125e-12, 7.501e-12, 1.257e-11,
00251      2.048e-11, 3.338e-11, 5.44e-11, 8.846e-11, 1.008e-10, 1.082e-10,
00252      1.157e-10, 1.232e-10, 1.312e-10, 1.539e-10, 1.822e-10, 2.118e-10,
00253      2.387e-10, 2.687e-10, 2.875e-10, 3.031e-10, 3.23e-10, 3.648e-10,
00254      4.117e-10, 4.477e-10, 4.633e-10, 4.794e-10, 4.95e-10, 5.104e-10,
00255      5.259e-10, 5.062e-10, 4.742e-10, 4.443e-10, 4.051e-10, 3.659e-10,
00256      3.305e-10, 2.911e-10, 2.54e-10, 2.215e-10, 1.927e-10, 1.675e-10,
00257      1.452e-10, 1.259e-10, 1.09e-10, 9.416e-11, 8.119e-11, 6.991e-11,
00258      6.015e-11, 5.163e-11, 4.43e-11, 3.789e-11, 3.24e-11, 2.769e-11,
00259      2.361e-11, 2.011e-11, 1.71e-11, 1.453e-11, 1.233e-11, 1.045e-11,
00260      8.851e-12, 7.48e-12, 6.316e-12, 5.326e-12, 4.487e-12, 3.778e-12,
00261      3.176e-12, 2.665e-12, 2.234e-12, 1.87e-12, 1.563e-12, 1.304e-12,
00262      1.085e-12, 9.007e-13, 7.468e-13, 6.179e-13, 5.092e-13, 4.188e-13,
00263      3.442e-13, 2.816e-13, 2.304e-13, 1.885e-13, 1.542e-13, 1.263e-13,
00264      1.035e-13, 8.5e-14, 7.004e-14, 5.783e-14, 4.795e-14, 4.007e-14,
00265      3.345e-14, 2.792e-14, 2.33e-14, 1.978e-14, 1.686e-14, 1.438e-14,
00266      1.234e-14, 1.07e-14, 9.312e-15, 8.131e-15, 7.164e-15, 6.367e-15,
00267      5.67e-15, 5.088e-15, 4.565e-15, 4.138e-15, 3.769e-15, 3.432e-15,
00268      3.148e-15
00269  };
00270
00271  static double clono2[121] = {
00272      1.011e-13, 1.515e-13, 2.272e-13, 3.446e-13, 5.231e-13, 8.085e-13,
00273      1.253e-12, 1.979e-12, 3.149e-12, 5.092e-12, 8.312e-12, 1.366e-11,
00274      2.272e-11, 3.791e-11, 6.209e-11, 9.101e-11, 1.334e-10, 1.951e-10,
00275      2.853e-10, 3.94e-10, 4.771e-10, 5.771e-10, 6.675e-10, 7.665e-10,
00276      8.504e-10, 8.924e-10, 9.363e-10, 8.923e-10, 8.411e-10, 7.646e-10,
00277      6.525e-10, 5.576e-10, 4.398e-10, 3.403e-10, 2.612e-10, 1.915e-10,
00278      1.407e-10, 1.028e-10, 7.455e-11, 5.42e-11, 3.708e-11, 2.438e-11,
00279      1.618e-11, 1.075e-11, 7.17e-12, 4.784e-12, 3.205e-12, 2.147e-12,
00280      1.44e-12, 9.654e-13, 6.469e-13, 4.332e-13, 2.891e-13, 1.926e-13,
00281      1.274e-13, 8.422e-14, 5.547e-14, 3.636e-14, 2.368e-14, 1.536e-14,
00282      9.937e-15, 6.39e-15, 4.101e-15, 2.61e-15, 1.659e-15, 1.052e-15,
00283      6.638e-16, 4.172e-16, 2.61e-16, 1.63e-16, 1.013e-16, 6.275e-17,
00284      3.879e-17, 2.383e-17, 1.461e-17, 8.918e-18, 5.43e-18, 3.301e-18,
00285      1.997e-18, 1.203e-18, 7.216e-19, 4.311e-19, 2.564e-19, 1.519e-19,
00286      8.911e-20, 5.203e-20, 3.026e-20, 1.748e-20, 9.99e-21, 5.673e-21,
00287      3.215e-21, 1.799e-21, 1.006e-21, 5.628e-22, 3.146e-22, 1.766e-22,
00288      9.94e-23, 5.614e-23, 3.206e-23, 1.841e-23, 1.071e-23, 6.366e-24,
00289      3.776e-24, 2.238e-24, 1.326e-24, 8.253e-25, 5.201e-25, 3.279e-25,
00290      2.108e-25, 1.395e-25, 9.326e-26, 6.299e-26, 4.365e-26, 3.104e-26,
00291      2.219e-26, 1.621e-26, 1.185e-26, 8.92e-27, 6.804e-27, 5.191e-27,
00292      4.041e-27
00293  };
00294
00295  static double co[121] = {
00296      1.907e-07, 1.553e-07, 1.362e-07, 1.216e-07, 1.114e-07, 1.036e-07,
00297      9.737e-08, 9.152e-08, 8.559e-08, 7.966e-08, 7.277e-08, 6.615e-08,
```

```
00298 5.884e-08, 5.22e-08, 4.699e-08, 4.284e-08, 3.776e-08, 3.274e-08,
00299 2.845e-08, 2.479e-08, 2.246e-08, 2.054e-08, 1.991e-08, 1.951e-08,
00300 1.94e-08, 2.009e-08, 2.1e-08, 2.201e-08, 2.322e-08, 2.45e-08,
00301 2.602e-08, 2.73e-08, 2.867e-08, 2.998e-08, 3.135e-08, 3.255e-08,
00302 3.352e-08, 3.426e-08, 3.484e-08, 3.53e-08, 3.593e-08, 3.671e-08,
00303 3.759e-08, 3.945e-08, 4.192e-08, 4.49e-08, 5.03e-08, 5.703e-08,
00304 6.538e-08, 7.878e-08, 9.644e-08, 1.196e-07, 1.498e-07, 1.904e-07,
00305 2.422e-07, 3.055e-07, 3.804e-07, 4.747e-07, 5.899e-07, 7.272e-07,
00306 8.91e-07, 1.071e-06, 1.296e-06, 1.546e-06, 1.823e-06, 2.135e-06,
00307 2.44e-06, 2.714e-06, 2.967e-06, 3.189e-06, 3.391e-06, 3.58e-06,
00308 3.773e-06, 4.022e-06, 4.346e-06, 4.749e-06, 5.199e-06, 5.668e-06,
00309 6.157e-06, 6.688e-06, 7.254e-06, 7.867e-06, 8.539e-06, 9.26e-06,
00310 1.009e-05, 1.119e-05, 1.228e-05, 1.365e-05, 1.506e-05, 1.641e-05,
00311 1.784e-05, 1.952e-05, 2.132e-05, 2.323e-05, 2.531e-05, 2.754e-05,
00312 3.047e-05, 3.459e-05, 3.922e-05, 4.439e-05, 4.825e-05, 5.077e-05,
00313 5.34e-05, 5.618e-05, 5.909e-05, 6.207e-05, 6.519e-05, 6.845e-05,
00314 6.819e-05, 6.726e-05, 6.622e-05, 6.512e-05, 6.671e-05, 6.862e-05,
00315 7.048e-05, 7.264e-05, 7.3e-05, 7.3e-05, 7.3e-05, 7.3e-05, 7.3e-05,
00316 };
00317
00318 static double cof2[121] = {
00319 7.5e-14, 1.055e-13, 1.485e-13, 2.111e-13, 3.001e-13, 4.333e-13,
00320 6.269e-13, 9.221e-13, 1.364e-12, 2.046e-12, 3.093e-12, 4.703e-12,
00321 7.225e-12, 1.113e-11, 1.66e-11, 2.088e-11, 2.626e-11, 3.433e-11,
00322 4.549e-11, 5.886e-11, 7.21e-11, 8.824e-11, 1.015e-10, 1.155e-10,
00323 1.288e-10, 1.388e-10, 1.497e-10, 1.554e-10, 1.606e-10, 1.639e-10,
00324 1.64e-10, 1.64e-10, 1.596e-10, 1.542e-10, 1.482e-10, 1.382e-10,
00325 1.289e-10, 1.198e-10, 1.109e-10, 1.026e-10, 9.484e-11, 8.75e-11,
00326 8.086e-11, 7.49e-11, 6.948e-11, 6.446e-11, 5.961e-11, 5.505e-11,
00327 5.085e-11, 4.586e-11, 4.1e-11, 3.665e-11, 3.235e-11, 2.842e-11,
00328 2.491e-11, 2.11e-11, 1.769e-11, 1.479e-11, 1.197e-11, 9.631e-12,
00329 7.74e-12, 6.201e-12, 4.963e-12, 3.956e-12, 3.151e-12, 2.507e-12,
00330 1.99e-12, 1.576e-12, 1.245e-12, 9.83e-13, 7.742e-13, 6.088e-13,
00331 4.782e-13, 3.745e-13, 2.929e-13, 2.286e-13, 1.782e-13, 1.388e-13,
00332 1.079e-13, 8.362e-14, 6.471e-14, 4.996e-14, 3.85e-14, 2.96e-14,
00333 2.265e-14, 1.729e-14, 1.317e-14, 9.998e-15, 7.549e-15, 5.683e-15,
00334 4.273e-15, 3.193e-15, 2.385e-15, 1.782e-15, 1.331e-15, 9.957e-16,
00335 7.461e-16, 5.601e-16, 4.228e-16, 3.201e-16, 2.438e-16, 1.878e-16,
00336 1.445e-16, 1.111e-16, 8.544e-17, 6.734e-17, 5.341e-17, 4.237e-17,
00337 3.394e-17, 2.759e-17, 2.254e-17, 1.851e-17, 1.54e-17, 1.297e-17,
00338 1.096e-17, 9.365e-18, 8e-18, 6.938e-18, 6.056e-18, 5.287e-18,
00339 4.662e-18
00340 };
00341
00342 static double f11[121] = {
00343 2.65e-10, 2.65e-10, 2.65e-10, 2.65e-10, 2.65e-10, 2.65e-10, 2.65e-10,
00344 2.65e-10, 2.65e-10, 2.65e-10, 2.65e-10, 2.65e-10, 2.635e-10, 2.536e-10,
00345 2.44e-10, 2.348e-10, 2.258e-10, 2.153e-10, 2.046e-10, 1.929e-10,
00346 1.782e-10, 1.648e-10, 1.463e-10, 1.291e-10, 1.1e-10, 8.874e-11,
00347 7.165e-11, 5.201e-11, 3.744e-11, 2.577e-11, 1.64e-11, 1.048e-11,
00348 5.993e-12, 3.345e-12, 1.839e-12, 9.264e-13, 4.688e-13, 2.329e-13,
00349 1.129e-13, 5.505e-14, 2.825e-14, 1.492e-14, 7.997e-15, 5.384e-15,
00350 3.988e-15, 2.955e-15, 2.196e-15, 1.632e-15, 1.214e-15, 9.025e-16,
00351 6.708e-16, 4.984e-16, 3.693e-16, 2.733e-16, 2.013e-16, 1.481e-16,
00352 1.087e-16, 7.945e-17, 5.782e-17, 4.195e-17, 3.038e-17, 2.19e-17,
00353 1.577e-17, 1.128e-17, 8.063e-18, 5.753e-18, 4.09e-18, 2.899e-18,
00354 2.048e-18, 1.444e-18, 1.015e-18, 7.12e-19, 4.985e-19, 3.474e-19,
00355 2.417e-19, 1.677e-19, 1.161e-19, 8.029e-20, 5.533e-20, 3.799e-20,
00356 2.602e-20, 1.776e-20, 1.209e-20, 8.202e-21, 5.522e-21, 3.707e-21,
00357 2.48e-21, 1.652e-21, 1.091e-21, 7.174e-22, 4.709e-22, 3.063e-22,
00358 1.991e-22, 1.294e-22, 8.412e-23, 5.483e-23, 3.581e-23, 2.345e-23,
00359 1.548e-23, 1.027e-23, 6.869e-24, 4.673e-24, 3.173e-24, 2.153e-24,
00360 1.461e-24, 1.028e-24, 7.302e-25, 5.188e-25, 3.739e-25, 2.753e-25,
00361 2.043e-25, 1.528e-25, 1.164e-25, 9.041e-26, 7.051e-26, 5.587e-26,
00362 4.428e-26, 3.588e-26, 2.936e-26, 2.402e-26, 1.995e-26
00363 };
00364
00365 static double f12[121] = {
00366 5.45e-10, 5.45e-10, 5.45e-10, 5.45e-10, 5.45e-10, 5.45e-10, 5.45e-10,
00367 5.45e-10, 5.45e-10, 5.45e-10, 5.45e-10, 5.45e-10, 5.429e-10, 5.291e-10,
00368 5.155e-10, 5.022e-10, 4.893e-10, 4.772e-10, 4.655e-10, 4.497e-10,
00369 4.249e-10, 4.015e-10, 3.632e-10, 3.261e-10, 2.858e-10, 2.408e-10,
00370 2.03e-10, 1.685e-10, 1.4e-10, 1.163e-10, 9.65e-11, 8.02e-11, 6.705e-11,
00371 5.624e-11, 4.764e-11, 4.249e-11, 3.792e-11, 3.315e-11, 2.819e-11,
00372 2.4e-11, 1.999e-11, 1.64e-11, 1.352e-11, 1.14e-11, 9.714e-12,
00373 8.28e-12, 7.176e-12, 6.251e-12, 5.446e-12, 4.72e-12, 4.081e-12,
00374 3.528e-12, 3.08e-12, 2.699e-12, 2.359e-12, 2.111e-12, 1.901e-12,
00375 1.709e-12, 1.534e-12, 1.376e-12, 1.233e-12, 1.103e-12, 9.869e-13,
00376 8.808e-13, 7.859e-13, 7.008e-13, 6.241e-13, 5.553e-13, 4.935e-13,
00377 4.383e-13, 3.889e-13, 3.447e-13, 3.054e-13, 2.702e-13, 2.389e-13,
00378 2.11e-13, 1.862e-13, 1.643e-13, 1.448e-13, 1.274e-13, 1.121e-13,
00379 9.844e-14, 8.638e-14, 7.572e-14, 6.62e-14, 5.782e-14, 5.045e-14,
00380 4.394e-14, 3.817e-14, 3.311e-14, 2.87e-14, 2.48e-14, 2.142e-14,
00381 1.851e-14, 1.599e-14, 1.383e-14, 1.196e-14, 1.036e-14, 9e-15,
00382 7.828e-15, 6.829e-15, 5.992e-15, 5.254e-15, 4.606e-15, 4.037e-15,
00383 3.583e-15, 3.19e-15, 2.841e-15, 2.542e-15, 2.291e-15, 2.07e-15,
00384 1.875e-15, 1.71e-15, 1.57e-15, 1.442e-15, 1.333e-15, 1.232e-15,
```



```
00385     1.147e-15, 1.071e-15, 1.001e-15, 9.396e-16
00386 };
00387
00388 static double f14[121] = {
00389     9e-11, 9e-11, 9e-11, 9e-11, 9e-11, 9e-11, 9e-11, 9e-11, 9e-11, 9e-11,
00390     9e-11, 9e-11, 9e-11, 9e-11, 9e-11, 8.91e-11, 8.73e-11, 8.46e-11,
00391     8.19e-11, 7.92e-11, 7.74e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11,
00392     7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11,
00393     7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11,
00394     7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11,
00395     7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11,
00396     7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11,
00397     7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11,
00398     7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11,
00399     7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11,
00400     7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11,
00401     7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11,
00402     7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11,
00403     7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11,
00404     7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11,
00405     7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11,
00406 };
00407
00408 static double f22[121] = {
00409     1.4e-10, 1.4e-10, 1.4e-10, 1.4e-10, 1.4e-10, 1.4e-10, 1.4e-10, 1.4e-10,
00410     1.4e-10, 1.4e-10, 1.4e-10, 1.372e-10, 1.317e-10, 1.235e-10, 1.153e-10,
00411     1.075e-10, 1.002e-10, 9.332e-11, 8.738e-11, 8.194e-11, 7.7e-11,
00412     7.165e-11, 6.753e-11, 6.341e-11, 5.971e-11, 5.6e-11, 5.229e-11,
00413     4.859e-11, 4.488e-11, 4.118e-11, 3.83e-11, 3.568e-11, 3.308e-11,
00414     3.047e-11, 2.82e-11, 2.594e-11, 2.409e-11, 2.237e-11, 2.065e-11,
00415     1.894e-11, 1.771e-11, 1.647e-11, 1.532e-11, 1.416e-11, 1.332e-11,
00416     1.246e-11, 1.161e-11, 1.087e-11, 1.017e-11, 9.471e-12, 8.853e-12,
00417     8.235e-12, 7.741e-12, 7.247e-12, 6.836e-12, 6.506e-12, 6.176e-12,
00418     5.913e-12, 5.65e-12, 5.419e-12, 5.221e-12, 5.024e-12, 4.859e-12,
00419     4.694e-12, 4.546e-12, 4.414e-12, 4.282e-12, 4.15e-12, 4.019e-12,
00420     3.903e-12, 3.805e-12, 3.706e-12, 3.607e-12, 3.508e-12, 3.41e-12,
00421     3.31e-12, 3.212e-12, 3.129e-12, 3.047e-12, 2.964e-12, 2.882e-12,
00422     2.8e-12, 2.734e-12, 2.668e-12, 2.602e-12, 2.537e-12, 2.471e-12,
00423     2.421e-12, 2.372e-12, 2.322e-12, 2.273e-12, 2.224e-12, 2.182e-12,
00424     2.141e-12, 2.1e-12, 2.059e-12, 2.018e-12, 1.977e-12, 1.935e-12,
00425     1.894e-12, 1.853e-12, 1.812e-12, 1.77e-12, 1.73e-12, 1.688e-12,
00426     1.647e-12, 1.606e-12, 1.565e-12, 1.524e-12, 1.483e-12, 1.441e-12,
00427     1.4e-12, 1.359e-12, 1.317e-12, 1.276e-12, 1.235e-12, 1.194e-12,
00428     1.153e-12, 1.112e-12, 1.071e-12, 1.029e-12, 9.883e-13
00429 };
00430
00431 static double h2o[121] = {
00432     0.01166, 0.008269, 0.005742, 0.003845, 0.00277, 0.001897, 0.001272,
00433     0.000827, 0.000539, 0.0003469, 0.0001579, 3.134e-05, 1.341e-05,
00434     6.764e-06, 4.498e-06, 3.703e-06, 3.724e-06, 3.899e-06, 4.002e-06,
00435     4.122e-06, 4.277e-06, 4.438e-06, 4.558e-06, 4.673e-06, 4.763e-06,
00436     4.809e-06, 4.856e-06, 4.936e-06, 5.021e-06, 5.114e-06, 5.222e-06,
00437     5.331e-06, 5.414e-06, 5.488e-06, 5.563e-06, 5.633e-06, 5.704e-06,
00438     5.767e-06, 5.819e-06, 5.872e-06, 5.914e-06, 5.949e-06, 5.984e-06,
00439     6.015e-06, 6.044e-06, 6.073e-06, 6.104e-06, 6.136e-06, 6.167e-06,
00440     6.189e-06, 6.208e-06, 6.226e-06, 6.212e-06, 6.185e-06, 6.158e-06,
00441     6.114e-06, 6.066e-06, 6.018e-06, 5.877e-06, 5.728e-06, 5.582e-06,
00442     5.437e-06, 5.296e-06, 5.156e-06, 5.02e-06, 4.886e-06, 4.754e-06,
00443     4.625e-06, 4.498e-06, 4.374e-06, 4.242e-06, 4.096e-06, 3.955e-06,
00444     3.817e-06, 3.683e-06, 3.491e-06, 3.204e-06, 2.94e-06, 2.696e-06,
00445     2.47e-06, 2.252e-06, 2.019e-06, 1.808e-06, 1.618e-06, 1.445e-06,
00446     1.285e-06, 1.105e-06, 9.489e-07, 8.121e-07, 6.938e-07, 5.924e-07,
00447     5.04e-07, 4.288e-07, 3.648e-07, 3.103e-07, 2.642e-07, 2.252e-07,
00448     1.921e-07, 1.643e-07, 1.408e-07, 1.211e-07, 1.048e-07, 9.063e-08,
00449     7.835e-08, 6.774e-08, 5.936e-08, 5.221e-08, 4.592e-08, 4.061e-08,
00450     3.62e-08, 3.236e-08, 2.902e-08, 2.62e-08, 2.383e-08, 2.171e-08,
00451     1.989e-08, 1.823e-08, 1.684e-08, 1.562e-08, 1.449e-08, 1.351e-08
00452 };
00453
00454 static double h2o2[121] = {
00455     1.779e-10, 7.938e-10, 8.953e-10, 8.032e-10, 6.564e-10, 5.159e-10,
00456     4.003e-10, 3.026e-10, 2.222e-10, 1.58e-10, 1.044e-10, 6.605e-11,
00457     3.413e-11, 1.453e-11, 1.062e-11, 1.009e-11, 9.597e-12, 1.175e-11,
00458     1.572e-11, 2.091e-11, 2.746e-11, 3.603e-11, 4.791e-11, 6.387e-11,
00459     8.239e-11, 1.007e-10, 1.23e-10, 1.363e-10, 1.489e-10, 1.585e-10,
00460     1.608e-10, 1.632e-10, 1.576e-10, 1.502e-10, 1.423e-10, 1.302e-10,
00461     1.192e-10, 1.085e-10, 9.795e-11, 8.854e-11, 8.057e-11, 7.36e-11,
00462     6.736e-11, 6.362e-11, 6.087e-11, 5.825e-11, 5.623e-11, 5.443e-11,
00463     5.27e-11, 5.098e-11, 4.931e-11, 4.769e-11, 4.611e-11, 4.458e-11,
00464     4.308e-11, 4.102e-11, 3.887e-11, 3.682e-11, 3.521e-11, 3.369e-11,
00465     3.224e-11, 3.082e-11, 2.946e-11, 2.814e-11, 2.687e-11, 2.566e-11,
00466     2.449e-11, 2.336e-11, 2.227e-11, 2.123e-11, 2.023e-11, 1.927e-11,
00467     1.835e-11, 1.746e-11, 1.661e-11, 1.58e-11, 1.502e-11, 1.428e-11,
00468     1.357e-11, 1.289e-11, 1.224e-11, 1.161e-11, 1.102e-11, 1.045e-11,
00469     9.895e-12, 9.369e-12, 8.866e-12, 8.386e-12, 7.922e-12, 7.479e-12,
00470     7.06e-12, 6.656e-12, 6.274e-12, 5.914e-12, 5.575e-12, 5.257e-12,
00471     4.959e-12, 4.679e-12, 4.42e-12, 4.178e-12, 3.954e-12, 3.75e-12,
```

```
00472     3.557e-12, 3.372e-12, 3.198e-12, 3.047e-12, 2.908e-12, 2.775e-12,
00473     2.653e-12, 2.544e-12, 2.442e-12, 2.346e-12, 2.26e-12, 2.183e-12,
00474     2.11e-12, 2.044e-12, 1.98e-12, 1.924e-12, 1.871e-12, 1.821e-12,
00475     1.775e-12
00476 };
00477
00478 static double hcn[121] = {
00479     5.5e-10, 5.5e-10, 5.5e-10, 5.5e-10, 5.5e-10, 5.5e-10, 5.5e-10,
00480     5.5e-10, 5.5e-10, 5.5e-10, 5.5e-10, 5.498e-10, 5.495e-10, 5.493e-10,
00481     5.49e-10, 5.488e-10, 4.717e-10, 3.946e-10, 3.174e-10, 2.4e-10,
00482     1.626e-10, 1.619e-10, 1.612e-10, 1.602e-10, 1.593e-10, 1.582e-10,
00483     1.572e-10, 1.56e-10, 1.549e-10, 1.539e-10, 1.53e-10, 1.519e-10,
00484     1.506e-10, 1.487e-10, 1.467e-10, 1.449e-10, 1.43e-10, 1.413e-10,
00485     1.397e-10, 1.382e-10, 1.368e-10, 1.354e-10, 1.337e-10, 1.315e-10,
00486     1.292e-10, 1.267e-10, 1.241e-10, 1.215e-10, 1.19e-10, 1.165e-10,
00487     1.141e-10, 1.118e-10, 1.096e-10, 1.072e-10, 1.047e-10, 1.021e-10,
00488     9.968e-11, 9.739e-11, 9.539e-11, 9.339e-11, 9.135e-11, 8.898e-11,
00489     8.664e-11, 8.439e-11, 8.249e-11, 8.075e-11, 7.904e-11, 7.735e-11,
00490     7.565e-11, 7.399e-11, 7.245e-11, 7.109e-11, 6.982e-11, 6.863e-11,
00491     6.755e-11, 6.657e-11, 6.587e-11, 6.527e-11, 6.476e-11, 6.428e-11,
00492     6.382e-11, 6.343e-11, 6.307e-11, 6.272e-11, 6.238e-11, 6.205e-11,
00493     6.17e-11, 6.137e-11, 6.102e-11, 6.072e-11, 6.046e-11, 6.03e-11,
00494     6.018e-11, 6.01e-11, 6.001e-11, 5.992e-11, 5.984e-11, 5.975e-11,
00495     5.967e-11, 5.958e-11, 5.95e-11, 5.941e-11, 5.933e-11, 5.925e-11,
00496     5.916e-11, 5.908e-11, 5.899e-11, 5.891e-11, 5.883e-11, 5.874e-11,
00497     5.866e-11, 5.858e-11, 5.85e-11, 5.841e-11, 5.833e-11, 5.825e-11,
00498     5.817e-11, 5.808e-11, 5.8e-11, 5.792e-11, 5.784e-11
00499 };
00500
00501 static double hno3[121] = {
00502     1.809e-10, 7.234e-10, 5.899e-10, 4.342e-10, 3.277e-10, 2.661e-10,
00503     2.35e-10, 2.267e-10, 2.389e-10, 2.651e-10, 3.255e-10, 4.099e-10,
00504     5.42e-10, 6.978e-10, 8.807e-10, 1.112e-09, 1.405e-09, 2.04e-09,
00505     3.111e-09, 4.5e-09, 5.762e-09, 7.37e-09, 7.852e-09, 8.109e-09,
00506     8.067e-09, 7.554e-09, 7.076e-09, 6.268e-09, 5.524e-09, 4.749e-09,
00507     3.909e-09, 3.223e-09, 2.517e-09, 1.942e-09, 1.493e-09, 1.122e-09,
00508     8.449e-10, 6.361e-10, 4.787e-10, 3.611e-10, 2.804e-10, 2.215e-10,
00509     1.758e-10, 1.441e-10, 1.197e-10, 9.953e-11, 8.505e-11, 7.334e-11,
00510     6.325e-11, 5.625e-11, 5.058e-11, 4.548e-11, 4.122e-11, 3.748e-11,
00511     3.402e-11, 3.088e-11, 2.8e-11, 2.536e-11, 2.293e-11, 2.072e-11,
00512     1.871e-11, 1.687e-11, 1.52e-11, 1.368e-11, 1.23e-11, 1.105e-11,
00513     9.922e-12, 8.898e-12, 7.972e-12, 7.139e-12, 6.385e-12, 5.708e-12,
00514     5.099e-12, 4.549e-12, 4.056e-12, 3.613e-12, 3.216e-12, 2.862e-12,
00515     2.544e-12, 2.259e-12, 2.004e-12, 1.776e-12, 1.572e-12, 1.391e-12,
00516     1.227e-12, 1.082e-12, 9.528e-13, 8.379e-13, 7.349e-13, 6.436e-13,
00517     5.634e-13, 4.917e-13, 4.291e-13, 3.745e-13, 3.267e-13, 2.854e-13,
00518     2.494e-13, 2.181e-13, 1.913e-13, 1.68e-13, 1.479e-13, 1.31e-13,
00519     1.159e-13, 1.025e-13, 9.067e-14, 8.113e-14, 7.281e-14, 6.535e-14,
00520     5.892e-14, 5.348e-14, 4.867e-14, 4.439e-14, 4.073e-14, 3.76e-14,
00521     3.476e-14, 3.229e-14, 3e-14, 2.807e-14, 2.635e-14, 2.473e-14,
00522     2.332e-14
00523 };
00524
00525 static double hno4[121] = {
00526     6.118e-12, 3.594e-12, 2.807e-12, 3.04e-12, 4.458e-12, 7.986e-12,
00527     1.509e-11, 2.661e-11, 3.738e-11, 4.652e-11, 4.429e-11, 3.992e-11,
00528     3.347e-11, 3.005e-11, 3.173e-11, 4.055e-11, 5.812e-11, 8.489e-11,
00529     1.19e-10, 1.482e-10, 1.766e-10, 2.103e-10, 2.35e-10, 2.598e-10,
00530     2.801e-10, 2.899e-10, 3e-10, 2.817e-10, 2.617e-10, 2.332e-10,
00531     1.933e-10, 1.605e-10, 1.232e-10, 9.285e-11, 6.941e-11, 4.951e-11,
00532     3.539e-11, 2.402e-11, 1.522e-11, 9.676e-12, 6.056e-12, 3.745e-12,
00533     2.34e-12, 1.463e-12, 9.186e-13, 5.769e-13, 3.322e-13, 1.853e-13,
00534     1.035e-13, 7.173e-14, 5.382e-14, 4.036e-14, 3.401e-14, 2.997e-14,
00535     2.635e-14, 2.316e-14, 2.034e-14, 1.783e-14, 1.56e-14, 1.363e-14,
00536     1.19e-14, 1.037e-14, 9.032e-15, 7.846e-15, 6.813e-15, 5.912e-15,
00537     5.121e-15, 4.431e-15, 3.829e-15, 3.306e-15, 2.851e-15, 2.456e-15,
00538     2.114e-15, 1.816e-15, 1.559e-15, 1.337e-15, 1.146e-15, 9.811e-16,
00539     8.389e-16, 7.162e-16, 6.109e-16, 5.203e-16, 4.425e-16, 3.76e-16,
00540     3.184e-16, 2.692e-16, 2.274e-16, 1.917e-16, 1.61e-16, 1.35e-16,
00541     1.131e-16, 9.437e-17, 7.874e-17, 6.57e-17, 5.481e-17, 4.579e-17,
00542     3.828e-17, 3.204e-17, 2.691e-17, 2.264e-17, 1.912e-17, 1.626e-17,
00543     1.382e-17, 1.174e-17, 9.972e-18, 8.603e-18, 7.45e-18, 6.453e-18,
00544     5.623e-18, 4.944e-18, 4.361e-18, 3.859e-18, 3.443e-18, 3.096e-18,
00545     2.788e-18, 2.528e-18, 2.293e-18, 2.099e-18, 1.929e-18, 1.773e-18,
00546     1.64e-18
00547 };
00548
00549 static double hocl[121] = {
00550     1.056e-12, 1.194e-12, 1.35e-12, 1.531e-12, 1.737e-12, 1.982e-12,
00551     2.263e-12, 2.599e-12, 2.991e-12, 3.459e-12, 4.012e-12, 4.662e-12,
00552     5.438e-12, 6.35e-12, 7.425e-12, 8.686e-12, 1.016e-11, 1.188e-11,
00553     1.389e-11, 1.659e-11, 2.087e-11, 2.621e-11, 3.265e-11, 4.064e-11,
00554     4.859e-11, 5.441e-11, 6.09e-11, 6.373e-11, 6.611e-11, 6.94e-11,
00555     7.44e-11, 7.97e-11, 8.775e-11, 9.722e-11, 1.064e-10, 1.089e-10,
00556     1.114e-10, 1.106e-10, 1.053e-10, 1.004e-10, 9.006e-11, 7.778e-11,
00557     6.739e-11, 5.636e-11, 4.655e-11, 3.845e-11, 3.042e-11, 2.368e-11,
00558     1.845e-11, 1.442e-11, 1.127e-11, 8.814e-12, 6.544e-12, 4.763e-12,
```

```
00559    3.449e-12, 2.612e-12, 1.999e-12, 1.526e-12, 1.16e-12, 8.793e-13,
00560    6.655e-13, 5.017e-13, 3.778e-13, 2.829e-13, 2.117e-13, 1.582e-13,
00561    1.178e-13, 8.755e-14, 6.486e-14, 4.799e-14, 3.54e-14, 2.606e-14,
00562    1.916e-14, 1.403e-14, 1.026e-14, 7.48e-15, 5.446e-15, 3.961e-15,
00563    2.872e-15, 2.076e-15, 1.498e-15, 1.077e-15, 7.726e-16, 5.528e-16,
00564    3.929e-16, 2.785e-16, 1.969e-16, 1.386e-16, 9.69e-17, 6.747e-17,
00565    4.692e-17, 3.236e-17, 2.232e-17, 1.539e-17, 1.061e-17, 7.332e-18,
00566    5.076e-18, 3.522e-18, 2.461e-18, 1.726e-18, 1.22e-18, 8.75e-19,
00567    6.264e-19, 4.482e-19, 3.207e-19, 2.368e-19, 1.762e-19, 1.312e-19,
00568    9.891e-20, 7.595e-20, 5.87e-20, 4.567e-20, 3.612e-20, 2.904e-20,
00569    2.343e-20, 1.917e-20, 1.568e-20, 1.308e-20, 1.1e-20, 9.25e-21,
00570    7.881e-21
00571    };
00572
00573    static double n2o[121] = {
00574        3.17e-07, 3.17e-07, 3.17e-07, 3.17e-07, 3.17e-07, 3.17e-07, 3.17e-07,
00575        3.17e-07, 3.17e-07, 3.17e-07, 3.124e-07, 3.077e-07, 3.03e-07,
00576        2.984e-07, 2.938e-07, 2.892e-07, 2.847e-07, 2.779e-07, 2.705e-07,
00577        2.631e-07, 2.557e-07, 2.484e-07, 2.345e-07, 2.201e-07, 2.01e-07,
00578        1.754e-07, 1.532e-07, 1.329e-07, 1.154e-07, 1.003e-07, 8.735e-08,
00579        7.617e-08, 6.512e-08, 5.547e-08, 4.709e-08, 3.915e-08, 3.259e-08,
00580        2.738e-08, 2.327e-08, 1.98e-08, 1.711e-08, 1.493e-08, 1.306e-08,
00581        1.165e-08, 1.049e-08, 9.439e-09, 8.375e-09, 7.391e-09, 6.525e-09,
00582        5.759e-09, 5.083e-09, 4.485e-09, 3.953e-09, 3.601e-09, 3.27e-09,
00583        2.975e-09, 2.757e-09, 2.556e-09, 2.37e-09, 2.195e-09, 2.032e-09,
00584        1.912e-09, 1.79e-09, 1.679e-09, 1.572e-09, 1.482e-09, 1.402e-09,
00585        1.326e-09, 1.254e-09, 1.187e-09, 1.127e-09, 1.071e-09, 1.02e-09,
00586        9.673e-10, 9.193e-10, 8.752e-10, 8.379e-10, 8.017e-10, 7.66e-10,
00587        7.319e-10, 7.004e-10, 6.721e-10, 6.459e-10, 6.199e-10, 5.942e-10,
00588        5.703e-10, 5.488e-10, 5.283e-10, 5.082e-10, 4.877e-10, 4.696e-10,
00589        4.52e-10, 4.355e-10, 4.198e-10, 4.039e-10, 3.888e-10, 3.754e-10,
00590        3.624e-10, 3.499e-10, 3.381e-10, 3.267e-10, 3.163e-10, 3.058e-10,
00591        2.959e-10, 2.864e-10, 2.77e-10, 2.686e-10, 2.604e-10, 2.534e-10,
00592        2.462e-10, 2.386e-10, 2.318e-10, 2.247e-10, 2.189e-10, 2.133e-10,
00593        2.071e-10, 2.014e-10, 1.955e-10, 1.908e-10, 1.86e-10, 1.817e-10
00594    };
00595
00596    static double n2o5[121] = {
00597        1.231e-11, 3.035e-12, 1.702e-12, 9.877e-13, 8.081e-13, 9.039e-13,
00598        1.169e-12, 1.474e-12, 1.651e-12, 1.795e-12, 1.998e-12, 2.543e-12,
00599        4.398e-12, 7.698e-12, 1.28e-11, 2.131e-11, 3.548e-11, 5.894e-11,
00600        7.645e-11, 1.089e-10, 1.391e-10, 1.886e-10, 2.386e-10, 2.986e-10,
00601        3.487e-10, 3.994e-10, 4.5e-10, 4.6e-10, 4.591e-10, 4.1e-10, 3.488e-10,
00602        2.846e-10, 2.287e-10, 1.696e-10, 1.011e-10, 6.428e-11, 4.324e-11,
00603        2.225e-11, 6.214e-12, 3.608e-12, 8.793e-13, 4.491e-13, 1.04e-13,
00604        6.1e-14, 3.436e-14, 6.671e-15, 1.171e-15, 5.848e-16, 1.212e-16,
00605        1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16,
00606        1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16,
00607        1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16,
00608        1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16,
00609        1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16,
00610        1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16,
00611        1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16,
00612        1e-16, 1e-16
00613    };
00614
00615    static double nh3[121] = {
00616        1e-10, 1e-10, 1e-10, 1e-10, 1e-10, 1e-10, 1e-10, 1e-10, 1e-10, 1e-10,
00617        1e-10, 1e-10, 1e-10, 1e-10, 9.444e-11, 8.488e-11, 7.241e-11, 5.785e-11,
00618        4.178e-11, 3.018e-11, 2.18e-11, 1.574e-11, 1.137e-11, 8.211e-12,
00619        5.973e-12, 4.327e-12, 3.118e-12, 2.234e-12, 1.573e-12, 1.04e-12,
00620        6.762e-13, 4.202e-13, 2.406e-13, 1.335e-13, 6.938e-14, 3.105e-14,
00621        1.609e-14, 1.033e-14, 6.432e-15, 4.031e-15, 2.555e-15, 1.656e-15,
00622        1.115e-15, 7.904e-16, 5.63e-16, 4.048e-16, 2.876e-16, 2.004e-16,
00623        1.356e-16, 9.237e-17, 6.235e-17, 4.223e-17, 3.009e-17, 2.328e-17,
00624        2.002e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17,
00625        1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17,
00626        1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17,
00627        1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17,
00628        1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17,
00629        1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17,
00630        1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17,
00631        1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17,
00632        1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17,
00633        1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17,
00634        1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17,
00635        1.914e-17
00636    };
00637
00638    static double no[121] = {
00639        2.586e-10, 4.143e-11, 1.566e-11, 9.591e-12, 8.088e-12, 8.462e-12,
00640        1.013e-11, 1.328e-11, 1.855e-11, 2.678e-11, 3.926e-11, 5.464e-11,
00641        7.012e-11, 8.912e-11, 1.127e-10, 1.347e-10, 1.498e-10, 1.544e-10,
00642        1.602e-10, 1.824e-10, 2.078e-10, 2.366e-10, 2.691e-10, 5.141e-10,
00643        8.259e-10, 1.254e-09, 1.849e-09, 2.473e-09, 3.294e-09, 4.16e-09,
00644        5.095e-09, 6.11e-09, 6.93e-09, 7.888e-09, 8.903e-09, 9.713e-09,
00645        1.052e-08, 1.115e-08, 1.173e-08, 1.21e-08, 1.228e-08, 1.239e-08,
```

```
00646     1.231e-08, 1.213e-08, 1.192e-08, 1.138e-08, 1.085e-08, 1.008e-08,
00647     9.224e-09, 8.389e-09, 7.262e-09, 6.278e-09, 5.335e-09, 4.388e-09,
00648     3.589e-09, 2.761e-09, 2.129e-09, 1.633e-09, 1.243e-09, 9.681e-10,
00649     8.355e-10, 7.665e-10, 7.442e-10, 8.584e-10, 9.732e-10, 1.063e-09,
00650     1.163e-09, 1.286e-09, 1.472e-09, 1.707e-09, 2.032e-09, 2.474e-09,
00651     2.977e-09, 3.506e-09, 4.102e-09, 5.013e-09, 6.493e-09, 8.414e-09,
00652     1.077e-08, 1.367e-08, 1.777e-08, 2.625e-08, 3.926e-08, 5.545e-08,
00653     7.195e-08, 9.464e-08, 1.404e-07, 2.183e-07, 3.329e-07, 4.535e-07,
00654     6.158e-07, 8.187e-07, 1.075e-06, 1.422e-06, 1.979e-06, 2.71e-06,
00655     3.58e-06, 4.573e-06, 5.951e-06, 7.999e-06, 1.072e-05, 1.372e-05,
00656     1.697e-05, 2.112e-05, 2.643e-05, 3.288e-05, 3.994e-05, 4.794e-05,
00657     5.606e-05, 6.383e-05, 7.286e-05, 8.156e-05, 8.883e-05, 9.469e-05,
00658     9.848e-05, 0.0001023, 0.0001066, 0.0001115, 0.0001145, 0.0001142,
00659     0.0001133
00660 };
00661
00662 static double no2[121] = {
00663     3.036e-09, 2.945e-10, 9.982e-11, 5.069e-11, 3.485e-11, 2.982e-11,
00664     2.947e-11, 3.164e-11, 3.714e-11, 4.586e-11, 6.164e-11, 8.041e-11,
00665     9.982e-11, 1.283e-10, 1.73e-10, 2.56e-10, 3.909e-10, 5.959e-10,
00666     9.081e-10, 1.384e-09, 1.788e-09, 2.189e-09, 2.686e-09, 3.091e-09,
00667     3.49e-09, 3.796e-09, 4.2e-09, 5.103e-09, 6.005e-09, 6.3e-09, 6.706e-09,
00668     7.07e-09, 7.434e-09, 7.663e-09, 7.788e-09, 7.8e-09, 7.597e-09,
00669     7.482e-09, 7.227e-09, 6.403e-09, 5.585e-09, 4.606e-09, 3.703e-09,
00670     2.984e-09, 2.183e-09, 1.48e-09, 8.441e-10, 5.994e-10, 3.799e-10,
00671     2.751e-10, 1.927e-10, 1.507e-10, 1.102e-10, 6.971e-11, 5.839e-11,
00672     3.904e-11, 3.087e-11, 2.176e-11, 1.464e-11, 1.209e-11, 8.497e-12,
00673     6.477e-12, 4.371e-12, 2.914e-12, 2.424e-12, 1.753e-12, 1.35e-12,
00674     9.417e-13, 6.622e-13, 5.148e-13, 3.841e-13, 3.446e-13, 3.01e-13,
00675     2.551e-13, 2.151e-13, 1.829e-13, 1.64e-13, 1.475e-13, 1.352e-13,
00676     1.155e-13, 9.963e-14, 9.771e-14, 9.577e-14, 9.384e-14, 9.186e-14,
00677     9e-14, 9e-14, 9e-14, 9e-14, 9e-14, 9e-14, 9e-14, 9e-14, 9e-14, 9e-14,
00678     9e-14, 9e-14, 9e-14, 9e-14, 9e-14, 9e-14, 9e-14, 9e-14, 9e-14, 9e-14,
00679     9e-14, 9e-14, 9e-14, 9e-14, 9e-14, 9e-14, 9e-14, 9e-14, 9e-14, 9e-14,
00680     9e-14, 9e-14, 9e-14, 9e-14, 9e-14, 9e-14
00681 };
00682
00683 static double o3[121] = {
00684     2.218e-08, 3.394e-08, 3.869e-08, 4.219e-08, 4.501e-08, 4.778e-08,
00685     5.067e-08, 5.402e-08, 5.872e-08, 6.521e-08, 7.709e-08, 9.461e-08,
00686     1.269e-07, 1.853e-07, 2.723e-07, 3.964e-07, 5.773e-07, 8.2e-07,
00687     1.155e-06, 1.59e-06, 2.076e-06, 2.706e-06, 3.249e-06, 3.848e-06,
00688     4.459e-06, 4.986e-06, 5.573e-06, 5.958e-06, 6.328e-06, 6.661e-06,
00689     6.9e-06, 7.146e-06, 7.276e-06, 7.374e-06, 7.447e-06, 7.383e-06,
00690     7.321e-06, 7.161e-06, 6.879e-06, 6.611e-06, 6.216e-06, 5.765e-06,
00691     5.355e-06, 4.905e-06, 4.471e-06, 4.075e-06, 3.728e-06, 3.413e-06,
00692     3.125e-06, 2.856e-06, 2.607e-06, 2.379e-06, 2.17e-06, 1.978e-06,
00693     1.8e-06, 1.646e-06, 1.506e-06, 1.376e-06, 1.233e-06, 1.102e-06,
00694     9.839e-07, 8.771e-07, 7.814e-07, 6.947e-07, 6.102e-07, 5.228e-07,
00695     4.509e-07, 3.922e-07, 3.501e-07, 3.183e-07, 2.909e-07, 2.686e-07,
00696     2.476e-07, 2.284e-07, 2.109e-07, 2.003e-07, 2.013e-07, 2.022e-07,
00697     2.032e-07, 2.042e-07, 2.097e-07, 2.361e-07, 2.656e-07, 2.989e-07,
00698     3.37e-07, 3.826e-07, 4.489e-07, 5.26e-07, 6.189e-07, 7.312e-07,
00699     8.496e-07, 8.444e-07, 8.392e-07, 8.339e-07, 8.286e-07, 8.234e-07,
00700     8.181e-07, 8.129e-07, 8.077e-07, 8.026e-07, 6.918e-07, 5.176e-07,
00701     3.865e-07, 2.885e-07, 2.156e-07, 1.619e-07, 1.219e-07, 9.161e-08,
00702     6.972e-08, 5.399e-08, 3.498e-08, 2.111e-08, 1.322e-08, 8.482e-09,
00703     5.527e-09, 3.423e-09, 2.071e-09, 1.314e-09, 8.529e-10, 5.503e-10,
00704     3.665e-10
00705 };
00706
00707 static double ocs[121] = {
00708     6e-10, 6e-10, 6e-10, 6e-10, 6e-10, 6e-10, 6e-10, 6e-10, 5.997e-10,
00709     5.989e-10, 5.881e-10, 5.765e-10, 5.433e-10, 5.074e-10, 4.567e-10,
00710     4.067e-10, 3.601e-10, 3.093e-10, 2.619e-10, 2.232e-10, 1.805e-10,
00711     1.46e-10, 1.187e-10, 8.03e-11, 5.435e-11, 3.686e-11, 2.217e-11,
00712     1.341e-11, 8.756e-12, 4.511e-12, 2.37e-12, 1.264e-12, 8.28e-13,
00713     5.263e-13, 3.209e-13, 1.717e-13, 9.068e-14, 4.709e-14, 2.389e-14,
00714     1.236e-14, 1.127e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14,
00715     1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14,
00716     1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14,
00717     1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14,
00718     1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14,
00719     1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14,
00720     1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14,
00721     1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14,
00722     1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14,
00723     1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14,
00724     1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14,
00725     1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14,
00726     1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14,
00727     1.091e-14, 1.091e-14, 1.091e-14
00728 };
00729
00730 static double sf6[121] = {
00731     4.103e-12, 4.103e-12, 4.103e-12, 4.103e-12, 4.103e-12, 4.103e-12,
00732     4.103e-12, 4.103e-12, 4.103e-12, 4.087e-12, 4.064e-12, 4.023e-12,
```

```

00733      3.988e-12, 3.941e-12, 3.884e-12, 3.755e-12, 3.622e-12, 3.484e-12,
00734      3.32e-12, 3.144e-12, 2.978e-12, 2.811e-12, 2.653e-12, 2.489e-12,
00735      2.332e-12, 2.199e-12, 2.089e-12, 2.013e-12, 1.953e-12, 1.898e-12,
00736      1.859e-12, 1.826e-12, 1.798e-12, 1.776e-12, 1.757e-12, 1.742e-12,
00737      1.728e-12, 1.717e-12, 1.707e-12, 1.698e-12, 1.691e-12, 1.685e-12,
00738      1.679e-12, 1.675e-12, 1.671e-12, 1.668e-12, 1.665e-12, 1.663e-12,
00739      1.661e-12, 1.659e-12, 1.658e-12, 1.657e-12, 1.656e-12, 1.655e-12,
00740      1.654e-12, 1.653e-12, 1.653e-12, 1.652e-12, 1.652e-12, 1.652e-12,
00741      1.651e-12, 1.651e-12, 1.651e-12, 1.651e-12, 1.651e-12, 1.651e-12,
00742      1.651e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12,
00743      1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12,
00744      1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12,
00745      1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12,
00746      1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12,
00747      1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12,
00748      1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12,
00749      1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12
00750  };
00751
00752  static double so2[121] = {
00753      1e-10, 1e-10, 1e-10, 1e-10, 1e-10, 1e-10, 1e-10, 1e-10, 1e-10, 1e-10,
00754      1e-10, 1e-10, 9.867e-11, 9.537e-11, 9e-11, 8.404e-11, 7.799e-11,
00755      7.205e-11, 6.616e-11, 6.036e-11, 5.475e-11, 5.007e-11, 4.638e-11,
00756      4.346e-11, 4.055e-11, 3.763e-11, 3.471e-11, 3.186e-11, 2.905e-11,
00757      2.631e-11, 2.358e-11, 2.415e-11, 2.949e-11, 3.952e-11, 5.155e-11,
00758      6.76e-11, 8.741e-11, 1.099e-10, 1.278e-10, 1.414e-10, 1.512e-10,
00759      1.607e-10, 1.699e-10, 1.774e-10, 1.832e-10, 1.871e-10, 1.907e-10,
00760      1.943e-10, 1.974e-10, 1.993e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10,
00761      2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10,
00762      2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10,
00763      2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10,
00764      2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10,
00765      2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10,
00766      2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10,
00767      2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10,
00768  };
00769
00770  static int ig_co2 = -999;
00771
00772  double co2, *q[NG] = { NULL };
00773
00774  int ig, ip, iw, iz;
00775
00776  /* Find emitter index of CO2... */
00777  if (ig_co2 == -999)
00778      ig_co2 = find_emitter(ctl, "CO2");
00779
00780  /* Identify variable... */
00781  for (ig = 0; ig < ctl->ng; ig++) {
00782      q[ig] = NULL;
00783      if (strcasecmp(ctl->emitter[ig], "C2H2") == 0)
00784          q[ig] = c2h2;
00785      if (strcasecmp(ctl->emitter[ig], "C2H6") == 0)
00786          q[ig] = c2h6;
00787      if (strcasecmp(ctl->emitter[ig], "CCl4") == 0)
00788          q[ig] = ccl4;
00789      if (strcasecmp(ctl->emitter[ig], "CH4") == 0)
00790          q[ig] = ch4;
00791      if (strcasecmp(ctl->emitter[ig], "ClO") == 0)
00792          q[ig] = clo;
00793      if (strcasecmp(ctl->emitter[ig], "ClONO2") == 0)
00794          q[ig] = clono2;
00795      if (strcasecmp(ctl->emitter[ig], "CO") == 0)
00796          q[ig] = co;
00797      if (strcasecmp(ctl->emitter[ig], "COF2") == 0)
00798          q[ig] = cof2;
00799      if (strcasecmp(ctl->emitter[ig], "F11") == 0)
00800          q[ig] = f11;
00801      if (strcasecmp(ctl->emitter[ig], "F12") == 0)
00802          q[ig] = f12;
00803      if (strcasecmp(ctl->emitter[ig], "F14") == 0)
00804          q[ig] = f14;
00805      if (strcasecmp(ctl->emitter[ig], "F22") == 0)
00806          q[ig] = f22;
00807      if (strcasecmp(ctl->emitter[ig], "H2O") == 0)
00808          q[ig] = h2o;
00809      if (strcasecmp(ctl->emitter[ig], "H2O2") == 0)
00810          q[ig] = h2o2;
00811      if (strcasecmp(ctl->emitter[ig], "HCN") == 0)
00812          q[ig] = hcn;
00813      if (strcasecmp(ctl->emitter[ig], "HNO3") == 0)
00814          q[ig] = hno3;
00815      if (strcasecmp(ctl->emitter[ig], "HNO4") == 0)
00816          q[ig] = hno4;
00817      if (strcasecmp(ctl->emitter[ig], "HOC1") == 0)
00818          q[ig] = hocl;
00819      if (strcasecmp(ctl->emitter[ig], "N2O") == 0)

```

```

00820     q[ig] = n2o;
00821     if (strcasecmp(ctl->emitter[ig], "N2O5") == 0)
00822         q[ig] = n2o5;
00823     if (strcasecmp(ctl->emitter[ig], "NH3") == 0)
00824         q[ig] = nh3;
00825     if (strcasecmp(ctl->emitter[ig], "NO") == 0)
00826         q[ig] = no;
00827     if (strcasecmp(ctl->emitter[ig], "NO2") == 0)
00828         q[ig] = no2;
00829     if (strcasecmp(ctl->emitter[ig], "O3") == 0)
00830         q[ig] = o3;
00831     if (strcasecmp(ctl->emitter[ig], "OCS") == 0)
00832         q[ig] = ocs;
00833     if (strcasecmp(ctl->emitter[ig], "SF6") == 0)
00834         q[ig] = sf6;
00835     if (strcasecmp(ctl->emitter[ig], "SO2") == 0)
00836         q[ig] = so2;
00837 }
00838
00839 /* Loop over atmospheric data points... */
00840 for (ip = 0; ip < atm->np; ip++) {
00841
00842     /* Get altitude index... */
00843     iz = locate_reg(z, 121, atm->z[ip]);
00844
00845     /* Interpolate pressure... */
00846     atm->p[ip] = EXP(z[iz], pre[iz], z[iz + 1], pre[iz + 1], atm->z[ip]);
00847
00848     /* Interpolate temperature... */
00849     atm->t[ip] = LIN(z[iz], tem[iz], z[iz + 1], tem[iz + 1], atm->z[ip]);
00850
00851     /* Interpolate trace gases... */
00852     for (ig = 0; ig < ctl->ng; ig++)
00853         if (q[ig] != NULL)
00854             atm->q[ig][ip] =
00855                 LIN(z[iz], q[ig][iz], z[iz + 1], q[ig][iz + 1], atm->z[ip]);
00856         else
00857             atm->q[ig][ip] = 0;
00858
00859     /* Set CO2... */
00860     if (ig_co2 >= 0) {
00861         co2 =
00862             371.789948e-6 + 2.026214e-6 * (atm->time[ip] - 63158400.) / 31557600.;
00863         atm->q[ig_co2][ip] = co2;
00864     }
00865
00866     /* Set extinction to zero... */
00867     for (iw = 0; iw < ctl->nw; iw++)
00868         atm->k[iw][ip] = 0;
00869 }
00870 }
00871
00872 /*****
00873
00874 double ctmc02(
00875     double nu,
00876     double p,
00877     double t,
00878     double u) {
00879
00880     static double co2296[2001] = { 9.3388e-5, 9.7711e-5, 1.0224e-4, 1.0697e-4,
00881         1.1193e-4, 1.1712e-4, 1.2255e-4, 1.2824e-4, 1.3419e-4, 1.4043e-4,
00882         1.4695e-4, 1.5378e-4, 1.6094e-4, 1.6842e-4, 1.7626e-4, 1.8447e-4,
00883         1.9307e-4, 2.0207e-4, 2.1149e-4, 2.2136e-4, 2.3169e-4, 2.4251e-4,
00884         2.5384e-4, 2.657e-4, 2.7813e-4, 2.9114e-4, 3.0477e-4, 3.1904e-4,
00885         3.3399e-4, 3.4965e-4, 3.6604e-4, 3.8322e-4, 4.0121e-4, 4.2006e-4,
00886         4.398e-4, 4.6047e-4, 4.8214e-4, 5.0483e-4, 5.286e-4, 5.535e-4,
00887         5.7959e-4, 6.0693e-4, 6.3557e-4, 6.6558e-4, 6.9702e-4, 7.2996e-4,
00888         7.6449e-4, 8.0066e-4, 8.3856e-4, 8.7829e-4, 9.1991e-4, 9.6354e-4,
00889         .0010093, .0010572, .0011074, .00116, .0012152, .001273,
00890         .0013336, .0013972, .0014638, .0015336, .0016068, .0016835,
00891         .001764, .0018483, .0019367, .0020295, .0021267, .0022286,
00892         .0023355, .0024476, .0025652, .0026885, .0028178, .0029534,
00893         .0030956, .0032448, .0034012, .0035654, .0037375, .0039181,
00894         .0041076, .0043063, .0045148, .0047336, .0049632, .005204,
00895         .0054567, .0057219, .0060002, .0062923, .0065988, .0069204,
00896         .007258, .0076123, .0079842, .0083746, .0087844, .0092146,
00897         .0096663, .01014, .010638, .011161, .01171, .012286, .012891,
00898         .013527, .014194, .014895, .015631, .016404, .017217, .01807,
00899         .018966, .019908, .020897, .021936, .023028, .024176, .025382,
00900         .026649, .027981, .02938, .030851, .032397, .034023, .035732,
00901         .037528, .039416, .041402, .04349, .045685, .047994, .050422,
00902         .052975, .055661, .058486, .061458, .064584, .067873, .071334,
00903         .074975, .078807, .082839, .087082, .091549, .096249, .1012,
00904         .10641, .11189, .11767, .12375, .13015, .13689, .14399, .15147,
00905         .15935, .16765, .17639, .18561, .19531, .20554, .21632, .22769,
00906         .23967, .25229, .2656, .27964, .29443, .31004, .3265, .34386,

```

00907 .36218, .3815, .40188, .42339, .44609, .47004, .49533, .52202,  
 00908 .5502, .57995, .61137, .64455, .6796, .71663, .75574, .79707,  
 00909 .84075, .88691, .9357, .98728, 1.0418, 1.0995, 1.1605, 1.225,  
 00910 1.2932, 1.3654, 1.4418, 1.5227, 1.6083, 1.6989, 1.7948, 1.8964,  
 00911 2.004, 2.118, 2.2388, 2.3668, 2.5025, 2.6463, 2.7988, 2.9606,  
 00912 3.1321, 3.314, 3.5071, 3.712, 3.9296, 4.1605, 4.4058, 4.6663,  
 00913 4.9431, 5.2374, 5.5501, 5.8818, 6.2353, 6.6114, 7.0115, 7.4372,  
 00914 7.8905, 8.3731, 8.8871, 9.4349, 10.019, 10.641, 11.305, 12.013,  
 00915 12.769, 13.576, 14.437, 15.358, 16.342, 17.39, 18.513, 19.716,  
 00916 21.003, 22.379, 23.854, 25.436, 27.126, 28.942, 30.89, 32.973,  
 00917 35.219, 37.634, 40.224, 43.021, 46.037, 49.29, 52.803, 56.447,  
 00918 60.418, 64.792, 69.526, 74.637, 80.182, 86.193, 92.713, 99.786,  
 00919 107.47, 115.84, 124.94, 134.86, 145.69, 157.49, 170.3, 184.39,  
 00920 199.83, 216.4, 234.55, 254.72, 276.82, 299.85, 326.16, 354.99,  
 00921 386.51, 416.68, 449.89, 490.12, 534.35, 578.25, 632.26, 692.61,  
 00922 756.43, 834.75, 924.11, 1016.9, 996.96, 1102.7, 1219.2, 1351.9,  
 00923 1494.3, 1654.1, 1826.5, 2027.9, 2249., 2453.8, 2714.4, 2999.4,  
 00924 3209.5, 3509., 3840.4, 3907.5, 4190.7, 4533.5, 4648.3, 5059.1,  
 00925 5561.6, 6191.4, 6820.8, 7905.9, 9362.2, 2431.3, 2211.3, 2046.8,  
 00926 2023.8, 1985.9, 1905.9, 1491.1, 1369.8, 1262.2, 1200.7, 887.74,  
 00927 820.25, 885.23, 887.21, 816.73, 1126.9, 1216.2, 1272.4, 1579.5,  
 00928 1634.2, 1656.3, 1657.9, 1789.5, 1670.8, 1509.5, 8474.6, 7489.2,  
 00929 6793.6, 6117., 5574.1, 5141.2, 5084.6, 4745.1, 4413.2, 4102.8,  
 00930 4024.7, 3715., 3398.6, 3100.8, 2900.4, 2629.2, 2374., 2144.7,  
 00931 1955.8, 1760.8, 1591.2, 1435.2, 1296.2, 1174., 1065.1, 967.76,  
 00932 999.48, 897.45, 809.23, 732.77, 670.26, 611.93, 560.11, 518.77,  
 00933 476.84, 438.8, 408.48, 380.21, 349.24, 322.71, 296.65, 272.85,  
 00934 251.96, 232.04, 213.88, 197.69, 182.41, 168.41, 155.79, 144.05,  
 00935 133.31, 123.48, 114.5, 106.21, 98.591, 91.612, 85.156, 79.204,  
 00936 73.719, 68.666, 63.975, 59.637, 56.35, 52.545, 49.042, 45.788,  
 00937 42.78, 39.992, 37.441, 35.037, 32.8, 30.744, 28.801, 26.986,  
 00938 25.297, 23.731, 22.258, 20.883, 19.603, 18.403, 17.295, 16.249,  
 00939 15.271, 14.356, 13.501, 12.701, 11.954, 11.254, 10.6, 9.9864,  
 00940 9.4118, 8.8745, 8.3714, 7.8997, 7.4578, 7.0446, 6.6573, 6.2949,  
 00941 5.9577, 5.6395, 5.3419, 5.063, 4.8037, 4.5608, 4.3452, 4.1364,  
 00942 3.9413, 3.7394, 3.562, 3.3932, 3.2325, 3.0789, 2.9318, 2.7898,  
 00943 2.6537, 2.5225, 2.3958, 2.2305, 2.1215, 2.0245, 1.9427, 1.8795,  
 00944 1.8336, 1.7604, 1.7016, 1.6419, 1.5282, 1.4611, 1.3443, 1.27,  
 00945 1.1675, 1.0824, 1.0534, .99833, .95854, .92981, .90887, .89346,  
 00946 .88113, .87068, .86102, .85096, .88262, .86151, .83565, .80518,  
 00947 .77045, .73736, .74744, .74954, .75773, .82267, .83493, .89402,  
 00948 .89725, .93426, .95564, .94045, .94174, .93404, .92035, .90456,  
 00949 .88621, .86673, .78117, .7515, .72056, .68822, .65658, .62764,  
 00950 .55984, .55598, .57407, .60963, .63763, .66198, .61132, .60972,  
 00951 .52496, .50649, .41872, .3964, .32422, .27276, .24048, .23772,  
 00952 .2286, .22711, .23999, .32038, .34371, .36621, .38561, .39953,  
 00953 .40636, .44913, .42716, .3919, .35477, .33935, .3351, .39746,  
 00954 .40993, .49398, .49956, .56157, .54742, .57295, .57386, .55417,  
 00955 .50745, .471, .43446, .39102, .34993, .31269, .27888, .24912,  
 00956 .22291, .19994, .17972, .16197, .14633, .13252, .12029, .10942,  
 00957 .099745, .091118, .083404, .076494, .070292, .064716, .059697,  
 00958 .055173, .051093, .047411, .044089, .041092, .038392, .035965,  
 00959 .033789, .031846, .030122, .028607, .02729, .026169, .025209,  
 00960 .024405, .023766, .023288, .022925, .022716, .022681, .022685,  
 00961 .022768, .023133, .023325, .023486, .024004, .024126, .024083,  
 00962 .023785, .024023, .023029, .021649, .021108, .019454, .017809,  
 00963 .017292, .016635, .017037, .018068, .018977, .018756, .017847,  
 00964 .016557, .016142, .014459, .012869, .012381, .010875, .0098701,  
 00965 .009285, .0091698, .0091701, .0096145, .010553, .01106, .012613,  
 00966 .014362, .015017, .016507, .017741, .01768, .017784, .0171,  
 00967 .016357, .016172, .017257, .018978, .020935, .021741, .023567,  
 00968 .025183, .025589, .026732, .027648, .028278, .028215, .02856,  
 00969 .029015, .029062, .028851, .028497, .027825, .027801, .026523,  
 00970 .02487, .022967, .022168, .020194, .018605, .017903, .018439,  
 00971 .019697, .020311, .020855, .020057, .018608, .016738, .015963,  
 00972 .013844, .011801, .011134, .0097573, .0086007, .0086226,  
 00973 .0083721, .0090978, .0097616, .0098426, .011317, .012853, .01447,  
 00974 .014657, .015771, .016351, .016079, .014829, .013431, .013185,  
 00975 .013207, .01448, .016176, .017971, .018265, .019526, .020455,  
 00976 .019797, .019802, .0194, .018176, .017505, .016197, .015339,  
 00977 .014401, .013213, .012203, .011186, .010236, .0093288, .0084854,  
 00978 .0076837, .0069375, .0062614, .0056628, .0051153, .0046015,  
 00979 .0041501, .003752, .0033996, .0030865, .0028077, .0025586,  
 00980 .0023355, .0021353, .0019553, .0017931, .0016466, .0015141,  
 00981 .0013941, .0012852, .0011862, .0010962, .0010142, 9.3935e-4,  
 00982 8.71e-4, 8.0851e-4, 7.5132e-4, 6.9894e-4, 6.5093e-4, 6.0689e-4,  
 00983 5.6647e-4, 5.2935e-4, 4.9525e-4, 4.6391e-4, 4.3509e-4, 4.086e-4,  
 00984 3.8424e-4, 3.6185e-4, 3.4126e-4, 3.2235e-4, 3.0498e-4, 2.8904e-4,  
 00985 2.7444e-4, 2.6106e-4, 2.4883e-4, 2.3766e-4, 2.275e-4, 2.1827e-4,  
 00986 2.0992e-4, 2.0239e-4, 1.9563e-4, 1.896e-4, 1.8427e-4, 1.796e-4,  
 00987 1.7555e-4, 1.7209e-4, 1.692e-4, 1.6687e-4, 1.6505e-4, 1.6375e-4,  
 00988 1.6294e-4, 1.6261e-4, 1.6274e-4, 1.6334e-4, 1.6438e-4, 1.6587e-4,  
 00989 1.678e-4, 1.7017e-4, 1.7297e-4, 1.762e-4, 1.7988e-4, 1.8399e-4,  
 00990 1.8855e-4, 1.9355e-4, 1.9902e-4, 2.0494e-4, 2.1134e-4, 2.1823e-4,  
 00991 2.2561e-4, 2.335e-4, 2.4192e-4, 2.5088e-4, 2.604e-4, 2.705e-4,  
 00992 2.8119e-4, 2.9251e-4, 3.0447e-4, 3.171e-4, 3.3042e-4, 3.4447e-4,  
 00993 3.5927e-4, 3.7486e-4, 3.9127e-4, 4.0854e-4, 4.267e-4, 4.4579e-4,

```

00994 4.6586e-4, 4.8696e-4, 5.0912e-4, 5.324e-4, 5.5685e-4, 5.8253e-4,
00995 6.0949e-4, 6.378e-4, 6.6753e-4, 6.9873e-4, 7.3149e-4, 7.6588e-4,
00996 8.0198e-4, 8.3987e-4, 8.7964e-4, 9.2139e-4, 9.6522e-4, .0010112,
00997 .0010595, .0011102, .0011634, .0012193, .001278, .0013396,
00998 .0014043, .0014722, .0015436, .0016185, .0016972, .0017799,
00999 .0018668, .001958, .0020539, .0021547, .0022606, .0023719,
01000 .002489, .002612, .0027414, .0028775, .0030206, .0031712,
01001 .0033295, .0034962, .0036716, .0038563, .0040506, .0042553,
01002 .0044709, .004698, .0049373, .0051894, .0054552, .0057354,
01003 .006031, .0063427, .0066717, .0070188, .0073854, .0077726,
01004 .0081816, .0086138, .0090709, .0095543, .010066, .010607,
01005 .011181, .011789, .012433, .013116, .013842, .014613, .015432,
01006 .016304, .017233, .018224, .019281, .020394, .021574, .022836,
01007 .024181, .025594, .027088, .028707, .030401, .032245, .034219,
01008 .036262, .038539, .040987, .043578, .04641, .04949, .052726,
01009 .056326, .0602, .064093, .068521, .073278, .077734, .083064,
01010 .088731, .093885, .1003, .1072, .11365, .12187, .13078, .13989,
01011 .15095, .16299, .17634, .19116, .20628, .22419, .24386, .26587,
01012 .28811, .31399, .34321, .36606, .39675, .42742, .44243, .47197,
01013 .49993, .49027, .51147, .52803, .48931, .49729, .5026, .43854,
01014 .441, .44766, .43414, .46151, .50029, .55247, .43855, .32115,
01015 .32607, .3431, .36119, .38029, .41179, .43996, .47144, .51853,
01016 .55362, .59122, .66338, .69877, .74001, .82923, .86907, .90361,
01017 1.0025, 1.031, 1.0559, 1.104, 1.1178, 1.1341, 1.1547, 1.351,
01018 1.4772, 1.4812, 1.4907, 1.512, 1.5442, 1.5853, 1.6358, 1.6963,
01019 1.7674, 1.8474, 1.9353, 2.0335, 2.143, 2.2592, 2.3853, 2.5217,
01020 2.6686, 2.8273, 2.9998, 3.183, 3.3868, 3.6109, 3.8564, 4.1159,
01021 4.4079, 4.7278, 5.0497, 5.3695, 5.758, 6.0834, 6.4976, 6.9312,
01022 7.38, 7.5746, 7.9833, 8.3791, 8.3956, 8.7501, 9.1067, 9.072,
01023 9.4649, 9.9112, 10.402, 10.829, 11.605, 12.54, 12.713, 10.443,
01024 10.825, 11.375, 11.955, 12.623, 13.326, 14.101, 15.041, 15.547,
01025 16.461, 17.439, 18.716, 19.84, 21.036, 22.642, 23.901, 25.244,
01026 27.03, 28.411, 29.871, 31.403, 33.147, 34.744, 36.456, 39.239,
01027 43.605, 45.162, 47.004, 49.093, 51.391, 53.946, 56.673, 59.629,
01028 63.167, 66.576, 70.254, 74.222, 78.477, 83.034, 87.914, 93.18,
01029 98.77, 104.74, 111.15, 117.95, 125.23, 133.01, 141.33, 150.21,
01030 159.71, 169.89, 180.93, 192.54, 204.99, 218.34, 232.65, 248.,
01031 264.47, 282.14, 301.13, 321.53, 343.48, 367.08, 392.5, 419.88,
01032 449.4, 481.26, 515.64, 552.79, 592.99, 636.48, 683.61, 734.65,
01033 789.99, 850.02, 915.14, 985.81, 1062.5, 1147.1, 1237.8, 1336.4,
01034 1443.2, 1558.9, 1684.2, 1819.2, 1965.2, 2122.6, 2291.7, 2470.8,
01035 2665.7, 2874.9, 3099.4, 3337.9, 3541., 3813.3, 4111.9, 4439.3,
01036 4798.9, 5196., 5639.2, 6087.5, 6657.7, 7306.7, 8040.7, 8845.5,
01037 9702.2, 10670., 11739., 12842., 14141., 15498., 17068., 18729.,
01038 20557., 22559., 25248., 27664., 30207., 32915., 35611., 38081.,
01039 40715., 43191., 41651., 42750., 43785., 44353., 44366., 44189.,
01040 43618., 42862., 41878., 35133., 35215., 36383., 39420., 44055.,
01041 44155., 45850., 46853., 39197., 38274., 29942., 28553., 21792.,
01042 21228., 17106., 14955., 18181., 19557., 21427., 23728., 26301.,
01043 28584., 30775., 32536., 33867., 40089., 39204., 37329., 34452.,
01044 31373., 33921., 34800., 36043., 44415., 45162., 52181., 50895.,
01045 54140., 50840., 50468., 48302., 44915., 40910., 36754., 32755.,
01046 29093., 25860., 22962., 20448., 18247., 16326., 14645., 13165.,
01047 11861., 10708., 9686.9, 8779.7, 7971.9, 7250.8, 6605.7, 6027.2,
01048 5507.3, 5039.1, 4616.6, 4234.8, 3889., 3575.4, 3290.5, 3031.3,
01049 2795.2, 2579.9, 2383.1, 2203.3, 2038.6, 1887.6, 1749.1, 1621.9,
01050 1505., 1397.4, 1298.3, 1207., 1122.8, 1045., 973.1, 906.64,
01051 845.16, 788.22, 735.48, 686.57, 641.21, 599.1, 559.99, 523.64,
01052 489.85, 458.42, 429.16, 401.92, 376.54, 352.88, 330.82, 310.24,
01053 291.03, 273.09, 256.34, 240.69, 226.05, 212.37, 199.57, 187.59,
01054 176.37, 165.87, 156.03, 146.82, 138.17, 130.07, 122.47, 115.34,
01055 108.65, 102.37, 96.473, 90.934, 85.73, 80.84, 76.243, 71.922,
01056 67.858, 64.034, 60.438, 57.052, 53.866, 50.866, 48.04, 45.379,
01057 42.872, 40.51, 38.285, 36.188, 34.211, 32.347, 30.588, 28.929,
01058 27.362, 25.884, 24.489, 23.171, 21.929, 20.755, 19.646, 18.599,
01059 17.61, 16.677, 15.795, 14.961, 14.174, 13.43, 12.725, 12.06,
01060 11.431, 10.834, 10.27, 9.7361, 9.2302, 8.7518, 8.2997, 7.8724,
01061 7.4674, 7.0848, 6.7226, 6.3794, 6.054, 5.745, 5.4525, 5.1752,
01062 4.9121, 4.6625, 4.4259, 4.2015, 3.9888, 3.7872, 3.5961, 3.4149,
01063 3.2431, 3.0802, 2.9257, 2.7792, 2.6402, 2.5084, 2.3834, 2.2648,
01064 2.1522, 2.0455, 1.9441, 1.848, 1.7567, 1.6701, 1.5878, 1.5097,
01065 1.4356, 1.3651, 1.2981, 1.2345, 1.174, 1.1167, 1.062, 1.0101,
01066 .96087, .91414, .86986, .82781, .78777, .74971, .71339, .67882,
01067 .64604, .61473, .58507, .55676, .52987, .5044, .48014, .45715,
01068 .43527, .41453, .3948, .37609, .35831, .34142, .32524, .30995,
01069 .29536, .28142, .26807, .25527, .24311, .23166, .22077, .21053,
01070 .20081, .19143, .18261, .17407, .16603, .15833, .15089, .14385,
01071 .13707, .13065, .12449, .11865, .11306, .10774, .10266, .097818,
01072 .093203, .088815, .084641, .080671, .076892, .073296, .069873,
01073 .066613, .06351, .060555, .05774, .055058, .052504, .050071,
01074 .047752, .045543, .043438, .041432, .039521, .037699, .035962,
01075 .034307, .032729, .031225, .029791, .028423, .02712, .025877,
01076 .024692, .023563, .022485, .021458, .020478, .019543, .018652,
01077 .017802, .016992, .016219, .015481, .014778, .014107, .013467,
01078 .012856, .012274, .011718, .011188, .010682, .0102, .0097393,
01079 .0093001, .008881, .0084812, .0080997, .0077358, .0073885,
01080 .0070571, .0067409, .0064393, .0061514, .0058768, .0056147,

```



```

01081 .0053647, .0051262, .0048987, .0046816, .0044745, .0042769,
01082 .0040884, .0039088, .0037373, .0035739, .003418, .0032693,
01083 .0031277, .0029926, .0028639, .0027413, .0026245, .0025133,
01084 .0024074, .0023066, .0022108, .0021196, .002033, .0019507,
01085 .0018726, .0017985, .0017282, .0016617, .0015988, .0015394,
01086 .0014834, .0014306, .0013811, .0013346, .0012911, .0012506,
01087 .0012131, .0011784, .0011465, .0011175, .0010912, .0010678,
01088 .0010472, .0010295, .0010147, .001003, 9.9428e-4, 9.8883e-4,
01089 9.8673e-4, 9.8821e-4, 9.9343e-4, .0010027, .0010164, .0010348,
01090 .0010586, .0010882, .0011245, .0011685, .0012145, .0012666,
01091 .0013095, .0013688, .0014048, .0014663, .0015309, .0015499,
01092 .0016144, .0016312, .001705, .0017892, .0018499, .0019715,
01093 .0021102, .0022442, .0024284, .0025893, .0027703, .0029445,
01094 .0031193, .003346, .0034552, .0036906, .0037584, .0040084,
01095 .0041934, .0044587, .0047093, .0049759, .0053421, .0055134,
01096 .0059048, .0058663, .0061036, .0063259, .0059657, .0060653,
01097 .0060972, .0055539, .0055653, .0055772, .005331, .0054953,
01098 .0055919, .0058684, .006183, .0066675, .0069808, .0075142,
01099 .0078536, .0084282, .0089454, .0094625, .0093703, .0095857,
01100 .0099283, .010063, .010521, .0097778, .0098175, .010379, .010447,
01101 .0105, .010617, .010706, .01078, .011177, .011212, .011304,
01102 .011446, .011603, .011816, .012165, .012545, .013069, .013539,
01103 .01411, .014776, .016103, .017016, .017994, .018978, .01998,
01104 .021799, .022745, .023681, .024627, .025562, .026992, .027958,
01105 .029013, .030154, .031402, .03228, .033651, .035272, .037088,
01106 .039021, .041213, .043597, .045977, .04877, .051809, .054943,
01107 .058064, .061528, .06537, .069309, .071928, .075752, .079589,
01108 .083352, .084096, .087497, .090817, .091198, .094966, .099045,
01109 .10429, .10867, .11518, .12269, .13126, .14087, .15161, .16388,
01110 .16423, .1759, .18721, .19994, .21275, .22513, .23041, .24231,
01111 .25299, .25396, .26396, .27696, .27929, .2908, .30595, .31433,
01112 .3282, .3429, .35944, .37467, .39277, .41245, .43326, .45649,
01113 .48152, .51897, .54686, .57877, .61263, .64962, .68983, .73945,
01114 .78619, .83537, .89622, .95002, 1.0067, 1.0742, 1.1355, 1.2007,
01115 1.2738, 1.347, 1.4254, 1.5094, 1.6009, 1.6976, 1.8019, 1.9148,
01116 2.0357, 2.166, 2.3066, 2.4579, 2.6208, 2.7966, 2.986, 3.188,
01117 3.4081, 3.6456, 3.9, 4.1747, 4.4712, 4.7931, 5.1359, 5.5097,
01118 5.9117, 6.3435, 6.8003, 7.3001, 7.8385, 8.3945, 9.011, 9.6869,
01119 10.392, 11.18, 12.036, 12.938, 13.944, 14.881, 16.029, 17.255,
01120 18.574, 19.945, 21.38, 22.9, 24.477, 26.128, 27.87, 29.037,
01121 30.988, 33.145, 35.506, 37.76, 40.885, 44.487, 48.505, 52.911,
01122 57.56, 61.964, 67.217, 72.26, 78.343, 85.08, 91.867, 99.435,
01123 107.68, 116.97, 127.12, 138.32, 150.26, 163.04, 174.81, 189.26,
01124 205.61, 224.68, 240.98, 261.88, 285.1, 307.58, 334.35, 363.53,
01125 394.68, 427.85, 458.85, 489.25, 472.87, 486.93, 496.27, 501.52,
01126 501.57, 497.14, 488.09, 476.32, 393.76, 388.51, 393.42, 414.45,
01127 455.12, 514.62, 520.38, 547.42, 562.6, 487.47, 480.83, 391.06,
01128 376.92, 303.7, 295.91, 256.03, 236.73, 280.38, 310.71, 335.53,
01129 367.88, 401.94, 435.52, 469.13, 497.94, 588.82, 597.94, 597.2,
01130 588.28, 571.2, 555.75, 603.56, 638.15, 680.75, 801.72, 848.01,
01131 962.15, 990.06, 1068.1, 1076.2, 1115.3, 1134.2, 1136.6, 1119.1,
01132 1108.9, 1090.6, 1068.7, 1041.9, 1005.4, 967.98, 927.08, 780.1,
01133 751.41, 733.12, 742.65, 785.56, 855.16, 852.45, 878.1, 784.59,
01134 777.81, 765.13, 622.93, 498.09, 474.89, 386.9, 378.48, 336.17,
01135 322.04, 329.57, 350.5, 383.38, 420.02, 462.39, 499.71, 531.98,
01136 654.99, 653.43, 639.99, 605.16, 554.16, 504.42, 540.64, 552.33,
01137 679.46, 699.51, 713.91, 832.17, 919.91, 884.96, 907.57, 846.56,
01138 818.56, 768.93, 706.71, 642.17, 575.95, 515.38, 459.07, 409.02,
01139 364.61, 325.46, 291.1, 260.89, 234.39, 211.01, 190.38, 172.11,
01140 155.91, 141.49, 128.63, 117.13, 106.84, 97.584, 89.262, 81.756,
01141 74.975, 68.842, 63.28, 58.232, 53.641, 49.46, 45.649, 42.168,
01142 38.991, 36.078, 33.409, 30.96, 28.71, 26.642, 24.737, 22.985,
01143 21.37, 19.882, 18.512, 17.242, 16.073, 14.987, 13.984, 13.05,
01144 12.186, 11.384, 10.637, 9.9436, 9.2988, 8.6991, 8.141, 7.6215,
01145 7.1378, 6.6872, 6.2671, 5.8754, 5.51, 5.1691, 4.851, 4.5539,
01146 4.2764, 4.0169, 3.7742, 3.5472, 3.3348, 3.1359, 2.9495, 2.7749,
01147 2.6113, 2.4578, 2.3139, 2.1789, 2.0523, 1.9334, 1.8219, 1.7171,
01148 1.6188, 1.5263, 1.4395, 1.3579, 1.2812, 1.209, 1.1411, 1.0773,
01149 1.0171, .96048, .90713, .85684, .80959, .76495, .72282, .68309,
01150 .64563, .61035, .57707, .54573, .51622, .48834, .46199, .43709,
01151 .41359, .39129, .37034, .35064, .33198, .31442, .29784, .28218,
01152 .26732, .25337, .24017, .22774, .21601, .20479, .19426
01153 };
01154
01155 static double co2260[2001] = { 5.7971e-5, 6.0733e-5, 6.3628e-5, 6.6662e-5,
01156 6.9843e-5, 7.3176e-5, 7.6671e-5, 8.0334e-5, 8.4175e-5, 8.8201e-5,
01157 9.2421e-5, 9.6846e-5, 1.0149e-4, 1.0635e-4, 1.1145e-4, 1.1679e-4,
01158 1.224e-4, 1.2828e-4, 1.3444e-4, 1.409e-4, 1.4768e-4, 1.5479e-4,
01159 1.6224e-4, 1.7006e-4, 1.7826e-4, 1.8685e-4, 1.9587e-4, 2.0532e-4,
01160 2.1524e-4, 2.2565e-4, 2.3656e-4, 2.48e-4, 2.6001e-4, 2.7261e-4,
01161 2.8582e-4, 2.9968e-4, 3.1422e-4, 3.2948e-4, 3.4548e-4, 3.6228e-4,
01162 3.799e-4, 3.9838e-4, 4.1778e-4, 4.3814e-4, 4.595e-4, 4.8191e-4,
01163 5.0543e-4, 5.3012e-4, 5.5603e-4, 5.8321e-4, 6.1175e-4, 6.417e-4,
01164 6.7314e-4, 7.0614e-4, 7.4078e-4, 7.7714e-4, 8.1531e-4, 8.5538e-4,
01165 8.9745e-4, 9.4162e-4, 9.8798e-4, .0010367, .0010878, .0011415,
01166 .0011978, .001257, .0013191, .0013844, .001453, .0015249,
01167 .0016006, .00168, .0017634, .001851, .001943, .0020397, .0021412,

```

```
01168 .0022479, .00236, .0024778, .0026015, .0027316, .0028682,
01169 .0030117, .0031626, .0033211, .0034877, .0036628, .0038469,
01170 .0040403, .0042436, .0044574, .004682, .0049182, .0051665,
01171 .0054276, .0057021, .0059907, .0062942, .0066133, .0069489,
01172 .0073018, .0076729, .0080632, .0084738, .0089056, .0093599,
01173 .0098377, .01034, .010869, .011426, .012011, .012627, .013276,
01174 .013958, .014676, .015431, .016226, .017063, .017944, .018872,
01175 .019848, .020876, .021958, .023098, .024298, .025561, .026892,
01176 .028293, .029769, .031323, .032961, .034686, .036503, .038418,
01177 .040435, .042561, .044801, .047161, .049649, .052271, .055035,
01178 .057948, .061019, .064256, .06767, .07127, .075066, .079069,
01179 .083291, .087744, .092441, .097396, .10262, .10814, .11396,
01180 .1201, .12658, .13342, .14064, .14826, .1563, .1648, .17376,
01181 .18323, .19324, .2038, .21496, .22674, .23919, .25234, .26624,
01182 .28093, .29646, .31287, .33021, .34855, .36794, .38844, .41012,
01183 .43305, .45731, .48297, .51011, .53884, .56924, .60141, .63547,
01184 .67152, .70969, .75012, .79292, .83826, .8863, .93718, .99111,
01185 1.0482, 1.1088, 1.173, 1.2411, 1.3133, 1.3898, 1.471, 1.5571,
01186 1.6485, 1.7455, 1.8485, 1.9577, 2.0737, 2.197, 2.3278, 2.4668,
01187 2.6145, 2.7715, 2.9383, 3.1156, 3.3042, 3.5047, 3.7181, 3.9451,
01188 4.1866, 4.4437, 4.7174, 5.0089, 5.3192, 5.65, 6.0025, 6.3782,
01189 6.7787, 7.206, 7.6617, 8.1479, 8.6669, 9.221, 9.8128, 10.445,
01190 11.12, 11.843, 12.615, 13.441, 14.325, 15.271, 16.283, 17.367,
01191 18.529, 19.776, 21.111, 22.544, 24.082, 25.731, 27.504, 29.409,
01192 31.452, 33.654, 36.024, 38.573, 41.323, 44.29, 47.492, 50.951,
01193 54.608, 58.588, 62.929, 67.629, 72.712, 78.226, 84.207, 90.699,
01194 97.749, 105.42, 113.77, 122.86, 132.78, 143.61, 155.44, 168.33,
01195 182.48, 198.01, 214.87, 233.39, 253.86, 276.34, 300.3, 327.28,
01196 356.89, 389.48, 422.29, 458.99, 501.39, 548.13, 595.62, 652.74,
01197 716.54, 784.57, 866.78, 960.59, 1062.8, 1072.5, 1189.5, 1319.4,
01198 1467.6, 1630.2, 1813.7, 2016.9, 2253., 2515.3, 2773.5, 3092.8,
01199 3444.4, 3720.4, 4104.3, 4527.5, 4645.9, 5021.7, 5462.2, 5597.,
01200 6110.6, 6732.5, 7513.8, 8270.6, 9640.6, 11487., 2796.1, 2680.1,
01201 2441.6, 2404.2, 2334.8, 2215.2, 1642.5, 1477.9, 1328.1, 1223.5,
01202 843.34, 766.96, 831.65, 834.84, 774.85, 1156.3, 1275.6, 1366.1,
01203 1795.6, 1885., 1936.5, 1953.4, 2154.4, 2002.7, 1789.8, 10381.,
01204 9040., 8216.5, 7384.7, 6721.9, 6187.7, 6143.8, 5703.9, 5276.6,
01205 4873.1, 4736., 4325.3, 3927., 3554.1, 3286.1, 2950.1, 2642.4,
01206 2368.7, 2138.9, 1914., 1719.6, 1543.9, 1388.6, 1252.1, 1132.2,
01207 1024.1, 1025.4, 920.58, 829.59, 750.54, 685.01, 624.25, 570.14,
01208 525.81, 481.85, 441.95, 408.71, 377.23, 345.86, 318.51, 292.26,
01209 268.34, 247.04, 227.14, 209.02, 192.69, 177.59, 163.78, 151.26,
01210 139.73, 129.19, 119.53, 110.7, 102.57, 95.109, 88.264, 81.948,
01211 76.13, 70.768, 65.827, 61.251, 57.022, 53.495, 49.824, 46.443,
01212 43.307, 40.405, 37.716, 35.241, 32.923, 30.77, 28.78, 26.915,
01213 25.177, 23.56, 22.059, 20.654, 19.345, 18.126, 16.988, 15.93,
01214 14.939, 14.014, 13.149, 12.343, 11.589, 10.884, 10.225, 9.6093,
01215 9.0327, 8.4934, 7.9889, 7.5166, 7.0744, 6.6604, 6.2727, 5.9098,
01216 5.5701, 5.2529, 4.955, 4.676, 4.4148, 4.171, 3.9426, 3.7332,
01217 3.5347, 3.3493, 3.1677, 3.0025, 2.8466, 2.6994, 2.5601, 2.4277,
01218 2.3016, 2.1814, 2.0664, 1.9564, 1.8279, 1.7311, 1.6427, 1.5645,
01219 1.4982, 1.443, 1.374, 1.3146, 1.2562, 1.17, 1.1105, 1.0272,
01220 .96863, .89718, .83654, .80226, .75908, .72431, .69573, .67174,
01221 .65126, .63315, .61693, .60182, .58715, .59554, .57649, .55526,
01222 .53177, .50622, .48176, .4813, .47642, .47492, .50273, .50293,
01223 .52687, .52239, .53419, .53814, .52626, .52211, .51492, .50622,
01224 .49746, .48841, .4792, .43534, .41999, .40349, .38586, .36799,
01225 .35108, .31089, .30803, .3171, .33599, .35041, .36149, .32924,
01226 .32462, .27309, .25961, .20922, .19504, .15683, .13098, .11588,
01227 .11478, .11204, .11363, .12135, .16423, .17785, .19094, .20236,
01228 .21084, .2154, .24108, .22848, .20871, .18797, .17963, .17834,
01229 .21552, .22284, .26945, .27052, .30108, .28977, .29772, .29224,
01230 .27658, .24956, .22777, .20654, .18392, .16338, .1452, .12916,
01231 .1152, .10304, .092437, .083163, .075031, .067878, .061564,
01232 .055976, .051018, .046609, .042679, .03917, .036032, .033223,
01233 .030706, .02845, .026428, .024617, .022998, .021554, .02027,
01234 .019136, .018141, .017278, .016541, .015926, .015432, .015058,
01235 .014807, .014666, .014635, .014728, .014947, .01527, .015728,
01236 .016345, .017026, .017798, .018839, .019752, .020636, .021886,
01237 .022695, .02327, .023478, .024292, .023544, .022222, .021932,
01238 .020052, .018143, .017722, .017031, .017782, .01938, .020734,
01239 .020476, .019255, .017477, .016878, .014617, .012489, .011765,
01240 .0099077, .0086446, .0079446, .0078644, .0079763, .008671,
01241 .01001, .0108, .012933, .015349, .016341, .018484, .020254,
01242 .020254, .020478, .019591, .018595, .018385, .019913, .022254,
01243 .024847, .025809, .028053, .029924, .030212, .031367, .03222,
01244 .032739, .032537, .03286, .033344, .033507, .033499, .033339,
01245 .032809, .033041, .031723, .029837, .027511, .026603, .024032,
01246 .021914, .020948, .021701, .023425, .024259, .024987, .023818,
01247 .021768, .019223, .018144, .015282, .012604, .01163, .0097907,
01248 .008336, .0082473, .0079582, .0088077, .009779, .010129, .012145,
01249 .014378, .016761, .01726, .018997, .019998, .019809, .01819,
01250 .016358, .016099, .01617, .017939, .020223, .022521, .02277,
01251 .024279, .025247, .024222, .023989, .023224, .021493, .020362,
01252 .018596, .017309, .015975, .014466, .013171, .011921, .01078,
01253 .0097229, .0087612, .0078729, .0070682, .0063494, .0057156,
01254 .0051459, .0046273, .0041712, .0037686, .0034119, .003095,
```

01255 .0028126, .0025603, .0023342, .0021314, .0019489, .0017845,  
 01256 .001636, .0015017, .00138, .0012697, .0011694, .0010782,  
 01257 9.9507e-4, 9.1931e-4, 8.5013e-4, 7.869e-4, 7.2907e-4, 6.7611e-4,  
 01258 6.2758e-4, 5.8308e-4, 5.4223e-4, 5.0473e-4, 4.7027e-4, 4.3859e-4,  
 01259 4.0946e-4, 3.8265e-4, 3.5798e-4, 3.3526e-4, 3.1436e-4, 2.9511e-4,  
 01260 2.7739e-4, 2.6109e-4, 2.4609e-4, 2.3229e-4, 2.1961e-4, 2.0797e-4,  
 01261 1.9729e-4, 1.875e-4, 1.7855e-4, 1.7038e-4, 1.6294e-4, 1.5619e-4,  
 01262 1.5007e-4, 1.4456e-4, 1.3961e-4, 1.3521e-4, 1.3131e-4, 1.2789e-4,  
 01263 1.2494e-4, 1.2242e-4, 1.2032e-4, 1.1863e-4, 1.1733e-4, 1.1641e-4,  
 01264 1.1585e-4, 1.1565e-4, 1.158e-4, 1.1629e-4, 1.1712e-4, 1.1827e-4,  
 01265 1.1976e-4, 1.2158e-4, 1.2373e-4, 1.262e-4, 1.2901e-4, 1.3214e-4,  
 01266 1.3562e-4, 1.3944e-4, 1.4361e-4, 1.4814e-4, 1.5303e-4, 1.5829e-4,  
 01267 1.6394e-4, 1.6999e-4, 1.7644e-4, 1.8332e-4, 1.9063e-4, 1.984e-4,  
 01268 2.0663e-4, 2.1536e-4, 2.246e-4, 2.3436e-4, 2.4468e-4, 2.5558e-4,  
 01269 2.6708e-4, 2.7921e-4, 2.92e-4, 3.0548e-4, 3.1968e-4, 3.3464e-4,  
 01270 3.5039e-4, 3.6698e-4, 3.8443e-4, 4.0281e-4, 4.2214e-4, 4.4248e-4,  
 01271 4.6389e-4, 4.8464e-4, 5.1009e-4, 5.3501e-4, 5.6123e-4, 5.888e-4,  
 01272 6.1781e-4, 6.4833e-4, 6.8043e-4, 7.142e-4, 7.4973e-4, 7.8711e-4,  
 01273 8.2644e-4, 8.6783e-4, 9.1137e-4, 9.5721e-4, .0010054, .0010562,  
 01274 .0011096, .0011659, .0012251, .0012875, .0013532, .0014224,  
 01275 .0014953, .001572, .0016529, .0017381, .0018279, .0019226,  
 01276 .0020224, .0021277, .0022386, .0023557, .0024792, .0026095,  
 01277 .002747, .0028921, .0030453, .0032071, .003378, .0035586,  
 01278 .0037494, .003951, .0041642, .0043897, .0046282, .0048805,  
 01279 .0051476, .0054304, .00573, .0060473, .0063837, .0067404,  
 01280 .0071188, .0075203, .0079466, .0083994, .0088806, .0093922,  
 01281 .0099366, .010516, .011134, .011792, .012494, .013244, .014046,  
 01282 .014898, .015808, .016781, .017822, .018929, .020108, .02138,  
 01283 .022729, .02419, .02576, .027412, .029233, .031198, .033301,  
 01284 .035594, .038092, .040767, .04372, .046918, .050246, .053974,  
 01285 .058009, .061976, .066586, .071537, .076209, .081856, .087998,  
 01286 .093821, .10113, .10913, .11731, .12724, .13821, .15025, .1639,  
 01287 .17807, .19472, .21356, .23496, .25758, .28387, .31389, .34104,  
 01288 .37469, .40989, .43309, .46845, .5042, .5023, .52981, .55275,  
 01289 .51075, .51976, .52457, .44779, .44721, .4503, .4243, .45244,  
 01290 .49491, .55399, .39021, .24802, .2501, .2618, .27475, .28879,  
 01291 .31317, .33643, .36257, .4018, .43275, .46525, .53333, .56599,  
 01292 .60557, .70142, .74194, .77736, .88567, .91182, .93294, .98407,  
 01293 .98772, .99176, .9995, 1.2405, 1.3602, 1.338, 1.3255, 1.3267,  
 01294 1.3404, 1.3634, 1.3967, 1.4407, 1.4961, 1.5603, 1.6328, 1.7153,  
 01295 1.8094, 1.9091, 2.018, 2.1367, 2.264, 2.4035, 2.5562, 2.7179,  
 01296 2.9017, 3.1052, 3.3304, 3.5731, 3.8488, 4.1553, 4.4769, 4.7818,  
 01297 5.1711, 5.5204, 5.9516, 6.4097, 6.8899, 7.1118, 7.5469, 7.9735,  
 01298 7.9511, 8.3014, 8.6418, 8.4757, 8.8256, 9.2294, 9.6923, 10.033,  
 01299 10.842, 11.851, 11.78, 8.8435, 9.1381, 9.5956, 10.076, 10.629,  
 01300 11.22, 11.883, 12.69, 13.163, 13.974, 14.846, 16.027, 17.053,  
 01301 18.148, 19.715, 20.907, 22.163, 23.956, 25.235, 26.566, 27.94,  
 01302 29.576, 30.956, 32.432, 35.337, 39.911, 41.128, 42.625, 44.386,  
 01303 46.369, 48.619, 51.031, 53.674, 56.825, 59.921, 63.286, 66.929,  
 01304 70.859, 75.081, 79.618, 84.513, 89.739, 95.335, 101.35, 107.76,  
 01305 114.63, 121.98, 129.87, 138.3, 147.34, 157.04, 167.56, 178.67,  
 01306 190.61, 203.43, 217.19, 231.99, 247.88, 264.98, 283.37, 303.17,  
 01307 324.49, 347.47, 372.25, 398.98, 427.85, 459.06, 492.8, 529.31,  
 01308 568.89, 611.79, 658.35, 708.91, 763.87, 823.65, 888.72, 959.58,  
 01309 1036.8, 1121.8, 1213.9, 1314.3, 1423.8, 1543., 1672.8, 1813.4,  
 01310 1966.1, 2131.4, 2309.5, 2499.3, 2705., 2925.7, 3161.6, 3411.3,  
 01311 3611.5, 3889.2, 4191.1, 4519.3, 4877.9, 5272.9, 5712.9, 6142.7,  
 01312 6719.6, 7385., 8145., 8977.7, 9831.9, 10827., 11934., 13063.,  
 01313 14434., 15878., 17591., 19435., 21510., 23835., 26835., 29740.,  
 01314 32878., 36305., 39830., 43273., 46931., 50499., 49586., 51598.,  
 01315 53429., 54619., 55081., 55102., 54485., 53487., 52042., 42689.,  
 01316 42607., 44020., 47994., 54169., 53916., 55808., 56642., 46049.,  
 01317 44243., 32929., 30658., 21963., 20835., 15962., 13679., 17652.,  
 01318 19680., 22388., 25625., 29184., 32520., 35720., 38414., 40523.,  
 01319 49228., 48173., 45678., 41768., 37600., 41313., 42654., 44465.,  
 01320 55736., 56630., 65409., 63308., 66572., 61845., 60379., 56777.,  
 01321 51920., 46601., 41367., 36529., 32219., 28470., 25192., 22362.,  
 01322 19907., 17772., 15907., 14273., 12835., 11567., 10445., 9450.2,  
 01323 8565.1, 7776., 7070.8, 6439.2, 5872.3, 5362.4, 4903., 4488.3,  
 01324 4113.4, 3773.8, 3465.8, 3186.1, 2931.7, 2700.1, 2488.8, 2296.,  
 01325 2119.8, 1958.6, 1810.9, 1675.6, 1551.4, 1437.3, 1332.4, 1236.,  
 01326 1147.2, 1065.3, 989.86, 920.22, 855.91, 796.48, 741.53, 690.69,  
 01327 643.62, 600.02, 559.6, 522.13, 487.35, 455.06, 425.08, 397.21,  
 01328 371.3, 347.2, 324.78, 303.9, 284.46, 266.34, 249.45, 233.7,  
 01329 219.01, 205.3, 192.5, 180.55, 169.38, 158.95, 149.2, 140.07,  
 01330 131.54, 123.56, 116.09, 109.09, 102.54, 96.405, 90.655, 85.266,  
 01331 80.213, 75.475, 71.031, 66.861, 62.948, 59.275, 55.827, 52.587,  
 01332 49.544, 46.686, 43.998, 41.473, 39.099, 36.867, 34.768, 32.795,  
 01333 30.939, 29.192, 27.546, 25.998, 24.539, 23.164, 21.869, 20.65,  
 01334 19.501, 18.419, 17.399, 16.438, 15.532, 14.678, 13.874, 13.115,  
 01335 12.4, 11.726, 11.088, 10.488, 9.921, 9.3846, 8.8784, 8.3996,  
 01336 7.9469, 7.5197, 7.1174, 6.738, 6.379, 6.0409, 5.7213, 5.419,  
 01337 5.1327, 4.8611, 4.6046, 4.3617, 4.1316, 3.9138, 3.7077, 3.5125,  
 01338 3.3281, 3.1536, 2.9885, 2.8323, 2.6846, 2.5447, 2.4124, 2.2871,  
 01339 2.1686, 2.0564, 1.9501, 1.8495, 1.7543, 1.6641, 1.5787, 1.4978,  
 01340 1.4212, 1.3486, 1.2799, 1.2147, 1.1529, 1.0943, 1.0388, .98602,  
 01341 .93596, .8886, .84352, .80078, .76029, .722, .68585, .65161,

```

01342 .61901, .58808, .55854, .53044, .5039, .47853, .45459, .43173,
01343 .41008, .38965, .37021, .35186, .33444, .31797, .30234, .28758,
01344 .2736, .26036, .24764, .2357, .22431, .21342, .20295, .19288,
01345 .18334, .17444, .166, .15815, .15072, .14348, .13674, .13015,
01346 .12399, .11807, .11231, .10689, .10164, .096696, .091955,
01347 .087476, .083183, .079113, .075229, .071536, .068026, .064698,
01348 .06154, .058544, .055699, .052997, .050431, .047993, .045676,
01349 .043475, .041382, .039392, .037501, .035702, .033991, .032364,
01350 .030817, .029345, .027945, .026613, .025345, .024139, .022991,
01351 .021899, .02086, .019871, .018929, .018033, .01718, .016368,
01352 .015595, .014859, .014158, .013491, .012856, .012251, .011675,
01353 .011126, .010604, .010107, .0096331, .009182, .0087523, .0083431,
01354 .0079533, .0075821, .0072284, .0068915, .0065706, .0062649,
01355 .0059737, .0056963, .005432, .0051802, .0049404, .0047118,
01356 .0044941, .0042867, .0040891, .0039009, .0037216, .0035507,
01357 .003388, .0032329, .0030852, .0029445, .0028105, .0026829,
01358 .0025613, .0024455, .0023353, .0022303, .0021304, .0020353,
01359 .0019448, .0018587, .0017767, .0016988, .0016247, .0015543,
01360 .0014874, .0014238, .0013635, .0013062, .0012519, .0012005,
01361 .0011517, .0011057, .0010621, .001021, 9.8233e-4, 9.4589e-4,
01362 9.1167e-4, 8.7961e-4, 8.4964e-4, 8.2173e-4, 7.9582e-4, 7.7189e-4,
01363 7.499e-4, 7.2983e-4, 7.1167e-4, 6.9542e-4, 6.8108e-4, 6.6866e-4,
01364 6.5819e-4, 6.4971e-4, 6.4328e-4, 6.3895e-4, 6.3681e-4, 6.3697e-4,
01365 6.3956e-4, 6.4472e-4, 6.5266e-4, 6.6359e-4, 6.778e-4, 6.9563e-4,
01366 7.1749e-4, 7.4392e-4, 7.7556e-4, 8.1028e-4, 8.4994e-4, 8.8709e-4,
01367 9.3413e-4, 9.6953e-4, .0010202, .0010738, .0010976, .0011507,
01368 .0011686, .0012264, .001291, .0013346, .0014246, .0015293,
01369 .0016359, .0017824, .0019255, .0020854, .002247, .0024148,
01370 .0026199, .0027523, .0029704, .0030702, .0033047, .0035013,
01371 .0037576, .0040275, .0043089, .0046927, .0049307, .0053486,
01372 .0053809, .0056699, .0059325, .0055488, .005634, .0056392,
01373 .004946, .0048855, .0048208, .0044386, .0045498, .0046377,
01374 .0048939, .0052396, .0057324, .0060859, .0066906, .0071148,
01375 .0077224, .0082687, .008769, .0084471, .008572, .0087729,
01376 .008775, .0090742, .0080704, .0080288, .0085747, .0086087,
01377 .0086408, .0088752, .0089381, .0089757, .0093532, .0092824,
01378 .0092566, .0092645, .0092735, .009342, .0095806, .0097991,
01379 .010213, .010611, .011129, .011756, .013237, .01412, .015034,
01380 .015936, .01682, .018597, .019315, .019995, .020658, .021289,
01381 .022363, .022996, .023716, .024512, .025434, .026067, .027118,
01382 .028396, .029865, .031442, .033253, .03525, .037296, .039701,
01383 .042356, .045154, .048059, .051294, .054893, .058636, .061407,
01384 .065172, .068974, .072676, .073379, .076547, .079556, .079134,
01385 .082308, .085739, .090192, .09359, .099599, .10669, .11496,
01386 .1244, .13512, .14752, .14494, .15647, .1668, .17863, .19029,
01387 .20124, .20254, .21179, .21982, .21625, .22364, .23405, .23382,
01388 .2434, .25708, .26406, .27621, .28909, .30395, .31717, .33271,
01389 .3496, .36765, .38774, .40949, .446, .46985, .49846, .5287, .562,
01390 .59841, .64598, .68834, .7327, .78978, .8373, .88708, .94744,
01391 1.0006, 1.0574, 1.1215, 1.1856, 1.2546, 1.3292, 1.4107, 1.4974,
01392 1.5913, 1.6931, 1.8028, 1.9212, 2.0492, 2.1874, 2.3365, 2.4978,
01393 2.6718, 2.8588, 3.062, 3.2818, 3.5188, 3.7752, 4.0527, 4.3542,
01394 4.6782, 5.0312, 5.4123, 5.8246, 6.2639, 6.7435, 7.2636, 7.8064,
01395 8.4091, 9.0696, 9.7677, 10.548, 11.4, 12.309, 13.324, 14.284,
01396 15.445, 16.687, 18.019, 19.403, 20.847, 22.366, 23.925, 25.537,
01397 27.213, 28.069, 29.864, 31.829, 33.988, 35.856, 38.829, 42.321,
01398 46.319, 50.606, 55.126, 59.126, 64.162, 68.708, 74.615, 81.176,
01399 87.739, 95.494, 103.83, 113.38, 123.99, 135.8, 148.7, 162.58,
01400 176.32, 192.6, 211.47, 232.7, 252.64, 277.41, 305.38, 333.44,
01401 366.42, 402.66, 442.14, 484.53, 526.42, 568.15, 558.78, 582.6,
01402 600.98, 613.94, 619.44, 618.24, 609.84, 595.96, 484.86, 475.59,
01403 478.49, 501.56, 552.19, 628.44, 630.39, 658.92, 671.96, 562.7,
01404 545.88, 423.43, 400.14, 306.59, 294.13, 246.8, 226.51, 278.21,
01405 314.39, 347.22, 389.13, 433.16, 477.48, 521.67, 560.54, 683.6,
01406 696.37, 695.91, 683.1, 658.24, 634.89, 698.85, 742.87, 796.66,
01407 954.49, 1009.5, 1150.5, 1179.1, 1267.9, 1272.4, 1312.7, 1330.4,
01408 1331.6, 1315.8, 1308.3, 1293.3, 1274.6, 1249.5, 1213.2, 1172.1,
01409 1124.4, 930.33, 893.36, 871.27, 883.54, 940.76, 1036., 1025.6,
01410 1053.1, 914.51, 894.15, 865.03, 670.63, 508.41, 475.15, 370.85,
01411 361.06, 319.38, 312.75, 331.87, 367.13, 415., 467.94, 525.49,
01412 578.41, 624.66, 794.82, 796.97, 780.29, 736.49, 670.18, 603.75,
01413 659.67, 679.8, 857.12, 884.05, 900.65, 1046.1, 1141.9, 1083.,
01414 1089.2, 1e3, 947.08, 872.31, 787.91, 704.75, 624.93, 553.68,
01415 489.91, 434.21, 385.64, 343.3, 306.42, 274.18, 245.94, 221.11,
01416 199.23, 179.88, 162.73, 147.48, 133.88, 121.73, 110.86, 101.1,
01417 92.323, 84.417, 77.281, 70.831, 64.991, 59.694, 54.884, 50.509,
01418 46.526, 42.893, 39.58, 36.549, 33.776, 31.236, 28.907, 26.77,
01419 24.805, 23., 21.339, 19.81, 18.404, 17.105, 15.909, 14.801,
01420 13.778, 12.83, 11.954, 11.142, 10.389, 9.691, 9.0434, 8.4423,
01421 7.8842, 7.3657, 6.8838, 6.4357, 6.0189, 5.6308, 5.2696, 4.9332,
01422 4.6198, 4.3277, 4.0553, 3.8012, 3.5639, 3.3424, 3.1355, 2.9422,
01423 2.7614, 2.5924, 2.4343, 2.2864, 2.148, 2.0184, 1.8971, 1.7835,
01424 1.677, 1.5773, 1.4838, 1.3961, 1.3139, 1.2369, 1.1645, 1.0966,
01425 1.0329, .97309, .91686, .86406, .81439, .76767, .72381, .68252,
01426 .64359, .60695, .57247, .54008, .50957, .48092, .45401, .42862,
01427 .40465, .38202, .36072, .34052, .3216, .30386, .28711, .27135,
01428 .25651, .24252, .2293, .21689, .20517, .19416, .18381, .17396,

```

```

01429     .16469
01430     };
01431
01432     static double co2230[2001] = { 2.743e-5, 2.8815e-5, 3.027e-5, 3.1798e-5,
01433     3.3405e-5, 3.5094e-5, 3.6869e-5, 3.8734e-5, 4.0694e-5, 4.2754e-5,
01434     4.492e-5, 4.7196e-5, 4.9588e-5, 5.2103e-5, 5.4747e-5, 5.7525e-5,
01435     6.0446e-5, 6.3516e-5, 6.6744e-5, 7.0137e-5, 7.3704e-5, 7.7455e-5,
01436     8.1397e-5, 8.5543e-5, 8.9901e-5, 9.4484e-5, 9.9302e-5, 1.0437e-4,
01437     1.097e-4, 1.153e-4, 1.2119e-4, 1.2738e-4, 1.3389e-4, 1.4074e-4,
01438     1.4795e-4, 1.5552e-4, 1.6349e-4, 1.7187e-4, 1.8068e-4, 1.8995e-4,
01439     1.997e-4, 2.0996e-4, 2.2075e-4, 2.321e-4, 2.4403e-4, 2.5659e-4,
01440     2.698e-4, 2.837e-4, 2.9832e-4, 3.137e-4, 3.2988e-4, 3.4691e-4,
01441     3.6483e-4, 3.8368e-4, 4.0351e-4, 4.2439e-4, 4.4635e-4, 4.6947e-4,
01442     4.9379e-4, 5.1939e-4, 5.4633e-4, 5.7468e-4, 6.0452e-4, 6.3593e-4,
01443     6.69e-4, 7.038e-4, 7.4043e-4, 7.79e-4, 8.1959e-4, 8.6233e-4,
01444     9.0732e-4, 9.5469e-4, .0010046, .0010571, .0011124, .0011706,
01445     .0012319, .0012964, .0013644, .001436, .0015114, .0015908,
01446     .0016745, .0017625, .0018553, .0019531, .002056, .0021645,
01447     .0022788, .0023992, .002526, .0026596, .0028004, .0029488,
01448     .0031052, .0032699, .0034436, .0036265, .0038194, .0040227,
01449     .0042369, .0044628, .0047008, .0049518, .0052164, .0054953,
01450     .0057894, .0060995, .0064265, .0067713, .007135, .0075184,
01451     .0079228, .0083494, .0087993, .0092738, .0097745, .010303,
01452     .01086, .011448, .012068, .012722, .013413, .014142, .014911,
01453     .015723, .01658, .017484, .018439, .019447, .020511, .021635,
01454     .022821, .024074, .025397, .026794, .02827, .029829, .031475,
01455     .033215, .035052, .036994, .039045, .041213, .043504, .045926,
01456     .048485, .05119, .05405, .057074, .060271, .063651, .067225,
01457     .071006, .075004, .079233, .083708, .088441, .093449, .098749,
01458     .10436, .11029, .11657, .12322, .13026, .13772, .14561, .15397,
01459     .16282, .1722, .18214, .19266, .20381, .21563, .22816, .24143,
01460     .2555, .27043, .28625, .30303, .32082, .3397, .35972, .38097,
01461     .40352, .42746, .45286, .47983, .50847, .53888, .57119, .6055,
01462     .64196, .6807, .72187, .76564, .81217, .86165, .91427, .97025,
01463     1.0298, 1.0932, 1.1606, 1.2324, 1.3088, 1.3902, 1.477, 1.5693,
01464     1.6678, 1.7727, 1.8845, 2.0038, 2.131, 2.2666, 2.4114, 2.5659,
01465     2.7309, 2.907, 3.0951, 3.2961, 3.5109, 3.7405, 3.986, 4.2485,
01466     4.5293, 4.8299, 5.1516, 5.4961, 5.8651, 6.2605, 6.6842, 7.1385,
01467     7.6256, 8.1481, 8.7089, 9.3109, 9.9573, 10.652, 11.398, 12.2,
01468     13.063, 13.992, 14.99, 16.064, 17.222, 18.469, 19.813, 21.263,
01469     22.828, 24.516, 26.34, 28.31, 30.437, 32.738, 35.226, 37.914,
01470     40.824, 43.974, 47.377, 51.061, 55.011, 59.299, 63.961, 69.013,
01471     74.492, 80.444, 86.919, 93.836, 101.23, 109.25, 117.98, 127.47,
01472     137.81, 149.07, 161.35, 174.75, 189.42, 205.49, 223.02, 242.26,
01473     263.45, 286.75, 311.94, 340.01, 370.86, 404.92, 440.44, 480.27,
01474     525.17, 574.71, 626.22, 686.8, 754.38, 827.07, 913.38, 1011.7,
01475     1121.5, 1161.6, 1289.5, 1432.2, 1595.4, 1777., 1983.3, 2216.1,
01476     2485.7, 2788.3, 3101.5, 3481., 3902.1, 4257.1, 4740., 5272.8,
01477     5457.9, 5946.2, 6505.3, 6668.4, 7302.4, 8061.6, 9015.8, 9908.3,
01478     11613., 13956., 3249.6, 3243., 2901.5, 2841.3, 2729.6, 2558.2,
01479     1797.8, 1583.2, 1386., 1233.5, 787.74, 701.46, 761.66, 767.21,
01480     722.83, 1180.6, 1332.1, 1461.6, 2032.9, 2166., 2255.9, 2294.7,
01481     2587.2, 2396.5, 2122.4, 12553., 10784., 9832.5, 8827.3, 8029.1,
01482     7377.9, 7347.1, 6783.8, 6239.1, 5721.1, 5503., 4975.1, 4477.8,
01483     4021.3, 3676.8, 3275.3, 2914.9, 2597.4, 2328.2, 2075.4, 1857.6,
01484     1663.6, 1493.3, 1343.8, 1213.3, 1095.6, 1066.5, 958.91, 865.15,
01485     783.31, 714.35, 650.77, 593.98, 546.2, 499.9, 457.87, 421.75,
01486     387.61, 355.25, 326.62, 299.7, 275.21, 253.17, 232.83, 214.31,
01487     197.5, 182.08, 167.98, 155.12, 143.32, 132.5, 122.58, 113.48,
01488     105.11, 97.415, 90.182, 83.463, 77.281, 71.587, 66.341, 61.493,
01489     57.014, 53.062, 49.21, 45.663, 42.38, 39.348, 36.547, 33.967,
01490     31.573, 29.357, 27.314, 25.415, 23.658, 22.03, 20.524, 19.125,
01491     17.829, 16.627, 15.511, 14.476, 13.514, 12.618, 11.786, 11.013,
01492     10.294, 9.6246, 9.0018, 8.4218, 7.8816, 7.3783, 6.9092, 6.4719,
01493     6.0641, 5.6838, 5.3289, 4.998, 4.6893, 4.4014, 4.1325, 3.8813,
01494     3.6469, 3.4283, 3.2241, 3.035, 2.8576, 2.6922, 2.5348, 2.3896,
01495     2.2535, 2.1258, 2.0059, 1.8929, 1.7862, 1.6854, 1.5898, 1.4992,
01496     1.4017, 1.3218, 1.2479, 1.1809, 1.1215, 1.0693, 1.0116, .96016,
01497     .9105, .84859, .80105, .74381, .69982, .65127, .60899, .57843,
01498     .54592, .51792, .49336, .47155, .45201, .43426, .41807, .40303,
01499     .38876, .3863, .37098, .35492, .33801, .32032, .30341, .29874,
01500     .29193, .28689, .29584, .29155, .29826, .29195, .29287, .2904,
01501     .28199, .27709, .27162, .26622, .26133, .25676, .25235, .23137,
01502     .22365, .21519, .20597, .19636, .18699, .16485, .16262, .16643,
01503     .17542, .18198, .18631, .16759, .16338, .13505, .1267, .10053,
01504     .092554, .074093, .062159, .055523, .054849, .05401, .05528,
01505     .058982, .07952, .08647, .093244, .099285, .10393, .10661,
01506     .12072, .11417, .10396, .093265, .089137, .088909, .10902,
01507     .11277, .13625, .13565, .14907, .14167, .1428, .13744, .12768,
01508     .11382, .10244, .091686, .08109, .071739, .063616, .056579,
01509     .050504, .045251, .040689, .036715, .033237, .030181, .027488,
01510     .025107, .022998, .021125, .01946, .017979, .016661, .015489,
01511     .014448, .013526, .012712, .011998, .011375, .010839, .010384,
01512     .010007, .0097053, .0094783, .0093257, .0092489, .0092504,
01513     .0093346, .0095077, .0097676, .01012, .01058, .011157, .011844,
01514     .012672, .013665, .014766, .015999, .017509, .018972, .020444,
01515     .022311, .023742, .0249, .025599, .026981, .026462, .025143,

```

```

01516 .025066, .022814, .020458, .020026, .019142, .020189, .022371,
01517 .024163, .023728, .02199, .019506, .018591, .015576, .012784,
01518 .011744, .0094777, .0079148, .0070652, .006986, .0071758,
01519 .008086, .0098025, .01087, .013609, .016764, .018137, .021061,
01520 .023498, .023576, .023965, .022828, .021519, .021283, .023364,
01521 .026457, .029782, .030856, .033486, .035515, .035543, .036558,
01522 .037198, .037472, .037045, .037284, .03777, .038085, .038366,
01523 .038526, .038282, .038915, .037697, .035667, .032941, .031959,
01524 .028692, .025918, .024596, .025592, .027873, .028935, .02984,
01525 .028148, .025305, .021912, .020454, .016732, .013357, .01205,
01526 .009731, .0079881, .0077704, .0074387, .0083895, .0096776,
01527 .010326, .01293, .015955, .019247, .020145, .02267, .024231,
01528 .024184, .022131, .019784, .01955, .01971, .022119, .025116,
01529 .027978, .028107, .029808, .030701, .029164, .028551, .027286,
01530 .024946, .023259, .020982, .019221, .017471, .015643, .014074,
01531 .01261, .011301, .010116, .0090582, .0081036, .0072542, .0065034,
01532 .0058436, .0052571, .0047321, .0042697, .0038607, .0034977,
01533 .0031747, .0028864, .0026284, .002397, .002189, .0020017,
01534 .0018326, .0016798, .0015414, .0014159, .0013019, .0011983,
01535 .0011039, .0010177, 9.391e-4, 8.6717e-4, 8.0131e-4, 7.4093e-4,
01536 6.8553e-4, 6.3464e-4, 5.8787e-4, 5.4487e-4, 5.0533e-4, 4.69e-4,
01537 4.3556e-4, 4.0474e-4, 3.7629e-4, 3.5e-4, 3.2569e-4, 3.032e-4,
01538 2.8239e-4, 2.6314e-4, 2.4535e-4, 2.2891e-4, 2.1374e-4, 1.9975e-4,
01539 1.8685e-4, 1.7498e-4, 1.6406e-4, 1.5401e-4, 1.4479e-4, 1.3633e-4,
01540 1.2858e-4, 1.2148e-4, 1.1499e-4, 1.0907e-4, 1.0369e-4, 9.8791e-5,
01541 9.4359e-5, 9.0359e-5, 8.6766e-5, 8.3555e-5, 8.0703e-5, 7.8192e-5,
01542 7.6003e-5, 7.4119e-5, 7.2528e-5, 7.1216e-5, 7.0171e-5, 6.9385e-5,
01543 6.8848e-5, 6.8554e-5, 6.8496e-5, 6.8669e-5, 6.9069e-5, 6.9694e-5,
01544 7.054e-5, 7.1608e-5, 7.2896e-5, 7.4406e-5, 7.6139e-5, 7.8097e-5,
01545 8.0283e-5, 8.2702e-5, 8.5357e-5, 8.8255e-5, 9.1402e-5, 9.4806e-5,
01546 9.8473e-5, 1.0241e-4, 1.0664e-4, 1.1115e-4, 1.1598e-4, 1.2112e-4,
01547 1.2659e-4, 1.3241e-4, 1.3859e-4, 1.4515e-4, 1.521e-4, 1.5947e-4,
01548 1.6728e-4, 1.7555e-4, 1.8429e-4, 1.9355e-4, 2.0334e-4, 2.1369e-4,
01549 2.2463e-4, 2.3619e-4, 2.4841e-4, 2.6132e-4, 2.7497e-4, 2.8938e-4,
01550 3.0462e-4, 3.2071e-4, 3.3771e-4, 3.5567e-4, 3.7465e-4, 3.947e-4,
01551 4.1588e-4, 4.3828e-4, 4.6194e-4, 4.8695e-4, 5.1338e-4, 5.4133e-4,
01552 5.7087e-4, 6.0211e-4, 6.3515e-4, 6.701e-4, 7.0706e-4, 7.4617e-4,
01553 7.8756e-4, 8.3136e-4, 8.7772e-4, 9.2681e-4, 9.788e-4, .0010339,
01554 .0010922, .001154, .0012195, .0012889, .0013626, .0014407,
01555 .0015235, .0016114, .0017048, .0018038, .001909, .0020207,
01556 .0021395, .0022657, .0023998, .0025426, .0026944, .002856,
01557 .0030281, .0032114, .0034068, .003615, .0038371, .004074,
01558 .004327, .0045971, .0048857, .0051942, .0055239, .0058766,
01559 .0062538, .0066573, .0070891, .007551, .0080455, .0085747,
01560 .0091412, .0097481, .010397, .011092, .011837, .012638, .013495,
01561 .014415, .01541, .016475, .017621, .018857, .020175, .02162,
01562 .023185, .024876, .02672, .028732, .030916, .033319, .035939,
01563 .038736, .041847, .04524, .048715, .052678, .056977, .061203,
01564 .066184, .07164, .076952, .083477, .090674, .098049, .10697,
01565 .1169, .1277, .14011, .15323, .1684, .18601, .20626, .22831,
01566 .25417, .28407, .31405, .34957, .38823, .41923, .46026, .50409,
01567 .51227, .54805, .57976, .53818, .55056, .557, .46741, .46403,
01568 .4636, .42265, .45166, .49852, .56663, .34306, .17779, .17697,
01569 .18346, .19129, .20014, .21778, .23604, .25649, .28676, .31238,
01570 .33856, .39998, .4288, .46568, .56654, .60786, .64473, .76466,
01571 .7897, .80778, .86443, .85736, .84798, .84157, 1.1385, 1.2446,
01572 1.1923, 1.1552, 1.1338, 1.1266, 1.1292, 1.1431, 1.1683, 1.2059,
01573 1.2521, 1.3069, 1.3712, 1.4471, 1.5275, 1.6165, 1.7145, 1.8189,
01574 1.9359, 2.065, 2.2007, 2.3591, 2.5362, 2.7346, 2.9515, 3.2021,
01575 3.4851, 3.7935, 4.0694, 4.4463, 4.807, 5.2443, 5.7178, 6.2231,
01576 6.4796, 6.9461, 7.4099, 7.3652, 7.7182, 8.048, 7.7373, 8.0363,
01577 8.3855, 8.8044, 9.0257, 9.8574, 10.948, 10.563, 6.8979, 7.0744,
01578 7.4121, 7.7663, 8.1768, 8.6243, 9.1437, 9.7847, 10.182, 10.849,
01579 11.572, 12.602, 13.482, 14.431, 15.907, 16.983, 18.11, 19.884,
01580 21.02, 22.18, 23.355, 24.848, 25.954, 27.13, 30.186, 34.893,
01581 35.682, 36.755, 38.111, 39.703, 41.58, 43.606, 45.868, 48.573,
01582 51.298, 54.291, 57.559, 61.116, 64.964, 69.124, 73.628, 78.471,
01583 83.683, 89.307, 95.341, 101.84, 108.83, 116.36, 124.46, 133.18,
01584 142.57, 152.79, 163.69, 175.43, 188.11, 201.79, 216.55, 232.51,
01585 249.74, 268.38, 288.54, 310.35, 333.97, 359.55, 387.26, 417.3,
01586 449.88, 485.2, 523.54, 565.14, 610.28, 659.31, 712.56, 770.43,
01587 833.36, 901.82, 976.36, 1057.6, 1146.8, 1243.8, 1350., 1466.3,
01588 1593.6, 1732.7, 1884.1, 2049.1, 2228.2, 2421.9, 2629.4, 2853.7,
01589 3094.4, 3351.1, 3622.3, 3829.8, 4123.1, 4438.3, 4777.2, 5144.1,
01590 5545.4, 5990.5, 6404.5, 6996.8, 7687.6, 8482.9, 9349.4, 10203.,
01591 11223., 12358., 13493., 14916., 16416., 18236., 20222., 22501.,
01592 25102., 28358., 31707., 35404., 39538., 43911., 48391., 53193.,
01593 58028., 58082., 61276., 64193., 66294., 67480., 67921., 67423.,
01594 66254., 64341., 51737., 51420., 53072., 58145., 66195., 65358.,
01595 67377., 67869., 53509., 50553., 35737., 32425., 21704., 19974.,
01596 14457., 12142., 16798., 19489., 23049., 27270., 31910., 36457.,
01597 40877., 44748., 47876., 59793., 58626., 55454., 50337., 44893.,
01598 50228., 52216., 54747., 69541., 70455., 81014., 77694., 80533.,
01599 73953., 70927., 65539., 59002., 52281., 45953., 40292., 35360.,
01600 31124., 27478., 24346., 21647., 19308., 17271., 15491., 13927.,
01601 12550., 11331., 10250., 9288.8, 8431.4, 7664.9, 6978.3, 6361.8,
01602 5807.4, 5307.7, 4856.8, 4449., 4079.8, 3744.9, 3440.8, 3164.2,

```

01603 2912.3, 2682.7, 2473., 2281.4, 2106., 1945.3, 1797.9, 1662.5,  
 01604 1538.1, 1423.6, 1318.1, 1221., 1131.5, 1049., 972.99, 902.87,  
 01605 838.01, 777.95, 722.2, 670.44, 622.35, 577.68, 536.21, 497.76,  
 01606 462.12, 429.13, 398.61, 370.39, 344.29, 320.16, 297.85, 277.2,  
 01607 258.08, 240.38, 223.97, 208.77, 194.66, 181.58, 169.43, 158.15,  
 01608 147.67, 137.92, 128.86, 120.44, 112.6, 105.3, 98.499, 92.166,  
 01609 86.264, 80.763, 75.632, 70.846, 66.381, 62.213, 58.321, 54.685,  
 01610 51.288, 48.114, 45.145, 42.368, 39.772, 37.341, 35.065, 32.937,  
 01611 30.943, 29.077, 27.33, 25.693, 24.158, 22.717, 21.367, 20.099,  
 01612 18.909, 17.792, 16.744, 15.761, 14.838, 13.971, 13.157, 12.393,  
 01613 11.676, 11.003, 10.369, 9.775, 9.2165, 8.6902, 8.1963, 7.7314,  
 01614 7.2923, 6.8794, 6.4898, 6.122, 5.7764, 5.4525, 5.1484, 4.8611,  
 01615 4.5918, 4.3379, 4.0982, 3.8716, 3.6567, 3.4545, 3.2634, 3.0828,  
 01616 2.9122, 2.7512, 2.5993, 2.4561, 2.3211, 2.1938, 2.0737, 1.9603,  
 01617 1.8534, 1.7525, 1.6572, 1.5673, 1.4824, 1.4022, 1.3265, 1.2551,  
 01618 1.1876, 1.1239, 1.0637, 1.0069, .9532, .90248, .85454, .80921,  
 01619 .76631, .72569, .6872, .65072, .61635, .5836, .55261, .52336,  
 01620 .49581, .46998, .44559, .42236, .40036, .37929, .35924, .34043,  
 01621 .32238, .30547, .28931, .27405, .25975, .24616, .23341, .22133,  
 01622 .20997, .19924, .18917, .17967, .17075, .16211, .15411, .14646,  
 01623 .13912, .13201, .12509, .11857, .11261, .10698, .10186, .097039,  
 01624 .092236, .087844, .083443, .07938, .075452, .071564, .067931,  
 01625 .064389, .061078, .057901, .054921, .052061, .049364, .046789,  
 01626 .04435, .042044, .039866, .037808, .035863, .034023, .032282,  
 01627 .030634, .029073, .027595, .026194, .024866, .023608, .022415,  
 01628 .021283, .02021, .019193, .018228, .017312, .016443, .015619,  
 01629 .014837, .014094, .01339, .012721, .012086, .011483, .010911,  
 01630 .010368, .009852, .0093623, .0088972, .0084556, .0080362,  
 01631 .0076379, .0072596, .0069003, .006559, .0062349, .0059269,  
 01632 .0056344, .0053565, .0050925, .0048417, .0046034, .004377,  
 01633 .0041618, .0039575, .0037633, .0035788, .0034034, .0032368,  
 01634 .0030785, .002928, .0027851, .0026492, .0025201, .0023975,  
 01635 .0022809, .0021701, .0020649, .0019649, .0018699, .0017796,  
 01636 .0016938, .0016122, .0015348, .0014612, .0013913, .001325,  
 01637 .0012619, .0012021, .0011452, .0010913, .0010401, 9.9149e-4,  
 01638 9.454e-4, 9.0169e-4, 8.6024e-4, 8.2097e-4, 7.8377e-4, 7.4854e-4,  
 01639 7.1522e-4, 6.8371e-4, 6.5393e-4, 6.2582e-4, 5.9932e-4, 5.7435e-4,  
 01640 5.5087e-4, 5.2882e-4, 5.0814e-4, 4.8881e-4, 4.7076e-4, 4.5398e-4,  
 01641 4.3843e-4, 4.2407e-4, 4.109e-4, 3.9888e-4, 3.88e-4, 3.7826e-4,  
 01642 3.6963e-4, 3.6213e-4, 3.5575e-4, 3.505e-4, 3.464e-4, 3.4346e-4,  
 01643 3.4173e-4, 3.4125e-4, 3.4206e-4, 3.4424e-4, 3.4787e-4, 3.5303e-4,  
 01644 3.5986e-4, 3.6847e-4, 3.7903e-4, 3.9174e-4, 4.0681e-4, 4.2455e-4,  
 01645 4.4527e-4, 4.6942e-4, 4.9637e-4, 5.2698e-4, 5.5808e-4, 5.9514e-4,  
 01646 6.2757e-4, 6.689e-4, 7.1298e-4, 7.3955e-4, 7.8403e-4, 8.0449e-4,  
 01647 8.5131e-4, 9.0256e-4, 9.3692e-4, .0010051, .0010846, .0011678,  
 01648 .001282, .0014016, .0015355, .0016764, .0018272, .0020055,  
 01649 .0021455, .0023421, .0024615, .0026786, .0028787, .0031259,  
 01650 .0034046, .0036985, .0040917, .0043902, .0048349, .0049531,  
 01651 .0052989, .0056148, .0052452, .0053357, .005333, .0045069,  
 01652 .0043851, .004253, .003738, .0038084, .0039013, .0041505,  
 01653 .0045372, .0050569, .0054507, .0061267, .0066122, .0072449,  
 01654 .0078012, .0082651, .0076538, .0076573, .0076806, .0075227,  
 01655 .0076269, .0063758, .006254, .0067749, .0067909, .0068231,  
 01656 .0072143, .0072762, .0072954, .007679, .0075107, .0073658,  
 01657 .0072441, .0071074, .0070378, .007176, .0072472, .0075844,  
 01658 .0079291, .008412, .0090165, .010688, .011535, .012375, .013166,  
 01659 .013895, .015567, .016011, .016392, .016737, .017043, .017731,  
 01660 .018031, .018419, .018877, .019474, .019868, .020604, .021538,  
 01661 .022653, .023869, .025288, .026879, .028547, .030524, .03274,  
 01662 .035132, .03769, .040567, .043793, .047188, .049962, .053542,  
 01663 .057205, .060776, .061489, .064419, .067124, .065945, .068487,  
 01664 .071209, .074783, .077039, .082444, .08902, .09692, .10617,  
 01665 .11687, .12952, .12362, .13498, .14412, .15492, .16519, .1744,  
 01666 .17096, .17714, .18208, .17363, .17813, .18564, .18295, .19045,  
 01667 .20252, .20815, .21844, .22929, .24229, .25321, .26588, .2797,  
 01668 .29465, .31136, .32961, .36529, .38486, .41027, .43694, .4667,  
 01669 .49943, .54542, .58348, .62303, .67633, .71755, .76054, .81371,  
 01670 .85934, .90841, .96438, 1.0207, 1.0821, 1.1491, 1.2226, 1.3018,  
 01671 1.388, 1.4818, 1.5835, 1.6939, 1.8137, 1.9435, 2.0843, 2.237,  
 01672 2.4026, 2.5818, 2.7767, 2.9885, 3.2182, 3.4679, 3.7391, 4.0349,  
 01673 4.3554, 4.7053, 5.0849, 5.4986, 5.9436, 6.4294, 6.9598, 7.5203,  
 01674 8.143, 8.8253, 9.5568, 10.371, 11.267, 12.233, 13.31, 14.357,  
 01675 15.598, 16.93, 18.358, 19.849, 21.408, 23.04, 24.706, 26.409,  
 01676 28.153, 28.795, 30.549, 32.43, 34.49, 36.027, 38.955, 42.465,  
 01677 46.565, 50.875, 55.378, 59.002, 63.882, 67.949, 73.693, 80.095,  
 01678 86.403, 94.264, 102.65, 112.37, 123.3, 135.54, 149.14, 163.83,  
 01679 179.17, 196.89, 217.91, 240.94, 264.13, 292.39, 324.83, 358.21,  
 01680 397.16, 440.5, 488.6, 541.04, 595.3, 650.43, 652.03, 688.74,  
 01681 719.47, 743.54, 757.68, 762.35, 756.43, 741.42, 595.43, 580.97,  
 01682 580.83, 605.68, 667.88, 764.49, 759.93, 789.12, 798.17, 645.66,  
 01683 615.65, 455.05, 421.09, 306.45, 289.14, 235.7, 215.52, 274.57,  
 01684 316.53, 357.73, 409.89, 465.06, 521.84, 579.02, 630.64, 794.46,  
 01685 813., 813.56, 796.25, 761.57, 727.97, 812.14, 866.75, 932.5,  
 01686 1132.8, 1194.8, 1362.2, 1387.2, 1482.3, 1479.7, 1517.9, 1533.1,  
 01687 1534.2, 1523.3, 1522.5, 1515.5, 1505.2, 1486.5, 1454., 1412.,  
 01688 1358.8, 1107.8, 1060.9, 1033.5, 1048.2, 1122.4, 1248.9, 1227.1,  
 01689 1255.4, 1058.9, 1020.7, 970.59, 715.24, 512.56, 468.47, 349.3,

```

01690    338.26, 299.22, 301.26, 332.38, 382.08, 445.49, 515.87, 590.85,
01691    662.3, 726.05, 955.59, 964.11, 945.17, 891.48, 807.11, 720.9,
01692    803.36, 834.46, 1073.9, 1107.1, 1123.6, 1296., 1393.7, 1303.1,
01693    1284.3, 1161.8, 1078.8, 976.13, 868.72, 767.4, 674.72, 593.73,
01694    523.12, 462.24, 409.75, 364.34, 325., 290.73, 260.76, 234.46,
01695    211.28, 190.78, 172.61, 156.44, 142.01, 129.12, 117.57, 107.2,
01696    97.877, 89.47, 81.882, 75.021, 68.807, 63.171, 58.052, 53.396,
01697    49.155, 45.288, 41.759, 38.531, 35.576, 32.868, 30.384, 28.102,
01698    26.003, 24.071, 22.293, 20.655, 19.147, 17.756, 16.476, 15.292,
01699    14.198, 13.183, 12.241, 11.367, 10.554, 9.7989, 9.0978, 8.4475,
01700    7.845, 7.2868, 6.7704, 6.2927, 5.8508, 5.4421, 5.064, 4.714,
01701    4.3902, 4.0902, 3.8121, 3.5543, 3.315, 3.093, 2.8869, 2.6953,
01702    2.5172, 2.3517, 2.1977, 2.0544, 1.9211, 1.7969, 1.6812, 1.5735,
01703    1.4731, 1.3794, 1.2921, 1.2107, 1.1346, 1.0637, .99744, .93554,
01704    .87771, .82368, .77313, .72587, .6816, .64014, .60134, .565,
01705    .53086, .49883, .46881, .44074, .4144, .38979, .36679, .34513,
01706    .32474, .30552, .28751, .27045, .25458, .23976, .22584, .21278,
01707    .20051, .18899, .17815, .16801, .15846, .14954, .14117, .13328,
01708    .12584
01709    };
01710
01711    double xw, dw, ew, cw296, cw260, cw230, dt230, dt260, dt296, ctw, ctmph;
01712
01713    int iw;
01714
01715    /* Get CO2 continuum absorption... */
01716    xw = nu / 2 + 1;
01717    if (xw >= 1 && xw < 2001) {
01718        iw = (int) xw;
01719        dw = xw - iw;
01720        ew = 1 - dw;
01721        cw296 = ew * co2296[iw - 1] + dw * co2296[iw];
01722        cw260 = ew * co2260[iw - 1] + dw * co2260[iw];
01723        cw230 = ew * co2230[iw - 1] + dw * co2230[iw];
01724        dt230 = t - 230;
01725        dt260 = t - 260;
01726        dt296 = t - 296;
01727        ctw = dt260 * 5.050505e-4 * dt296 * cw230 - dt230 * 9.259259e-4
01728            * dt296 * cw260 + dt230 * 4.208754e-4 * dt260 * cw296;
01729        ctmph = u / NA / 1000 * p / P0 * ctw;
01730    } else
01731        ctmph = 0;
01732    return ctmph;
01733 }
01734
01735 /*****
01736
01737 double ctmh2o(
01738     double nu,
01739     double p,
01740     double t,
01741     double q,
01742     double u) {
01743
01744     static double h2o296[2001] = { .17, .1695, .172, .168, .1687, .1624, .1606,
01745         .1508, .1447, .1344, .1214, .1133, .1009, .09217, .08297, .06989,
01746         .06513, .05469, .05056, .04417, .03779, .03484, .02994, .0272,
01747         .02325, .02063, .01818, .01592, .01405, .01251, .0108, .009647,
01748         .008424, .007519, .006555, .00588, .005136, .004511, .003989,
01749         .003509, .003114, .00274, .002446, .002144, .001895, .001676,
01750         .001486, .001312, .001164, .001031, 9.129e-4, 8.106e-4, 7.213e-4,
01751         6.4e-4, 5.687e-4, 5.063e-4, 4.511e-4, 4.029e-4, 3.596e-4,
01752         3.22e-4, 2.889e-4, 2.597e-4, 2.337e-4, 2.108e-4, 1.907e-4,
01753         1.728e-4, 1.57e-4, 1.43e-4, 1.305e-4, 1.195e-4, 1.097e-4,
01754         1.009e-4, 9.307e-5, 8.604e-5, 7.971e-5, 7.407e-5, 6.896e-5,
01755         6.433e-5, 6.013e-5, 5.631e-5, 5.283e-5, 4.963e-5, 4.669e-5,
01756         4.398e-5, 4.148e-5, 3.917e-5, 3.702e-5, 3.502e-5, 3.316e-5,
01757         3.142e-5, 2.978e-5, 2.825e-5, 2.681e-5, 2.546e-5, 2.419e-5,
01758         2.299e-5, 2.186e-5, 2.079e-5, 1.979e-5, 1.884e-5, 1.795e-5,
01759         1.711e-5, 1.633e-5, 1.559e-5, 1.49e-5, 1.426e-5, 1.367e-5,
01760         1.312e-5, 1.263e-5, 1.218e-5, 1.178e-5, 1.143e-5, 1.112e-5,
01761         1.088e-5, 1.07e-5, 1.057e-5, 1.05e-5, 1.051e-5, 1.059e-5,
01762         1.076e-5, 1.1e-5, 1.133e-5, 1.18e-5, 1.237e-5, 1.308e-5,
01763         1.393e-5, 1.483e-5, 1.614e-5, 1.758e-5, 1.93e-5, 2.123e-5,
01764         2.346e-5, 2.647e-5, 2.93e-5, 3.279e-5, 3.745e-5, 4.152e-5,
01765         4.813e-5, 5.477e-5, 6.203e-5, 7.331e-5, 8.056e-5, 9.882e-5,
01766         1.05e-4, 1.21e-4, 1.341e-4, 1.572e-4, 1.698e-4, 1.968e-4,
01767         2.175e-4, 2.431e-4, 2.735e-4, 2.867e-4, 3.19e-4, 3.371e-4,
01768         3.554e-4, 3.726e-4, 3.837e-4, 3.878e-4, 3.864e-4, 3.858e-4,
01769         3.841e-4, 3.852e-4, 3.815e-4, 3.762e-4, 3.618e-4, 3.579e-4,
01770         3.45e-4, 3.202e-4, 3.018e-4, 2.785e-4, 2.602e-4, 2.416e-4,
01771         2.097e-4, 1.939e-4, 1.689e-4, 1.498e-4, 1.308e-4, 1.17e-4,
01772         1.011e-4, 9.237e-5, 7.909e-5, 7.006e-5, 6.112e-5, 5.401e-5,
01773         4.914e-5, 4.266e-5, 3.963e-5, 3.316e-5, 3.037e-5, 2.598e-5,
01774         2.294e-5, 2.066e-5, 1.813e-5, 1.583e-5, 1.423e-5, 1.247e-5,
01775         1.116e-5, 9.76e-6, 8.596e-6, 7.72e-6, 6.825e-6, 6.108e-6,
01776         5.366e-6, 4.733e-6, 4.229e-6, 3.731e-6, 3.346e-6, 2.972e-6,

```



01777 2.628e-6, 2.356e-6, 2.102e-6, 1.878e-6, 1.678e-6, 1.507e-6,  
 01778 1.348e-6, 1.21e-6, 1.089e-6, 9.806e-7, 8.857e-7, 8.004e-7,  
 01779 7.261e-7, 6.599e-7, 6.005e-7, 5.479e-7, 5.011e-7, 4.595e-7,  
 01780 4.219e-7, 3.885e-7, 3.583e-7, 3.314e-7, 3.071e-7, 2.852e-7,  
 01781 2.654e-7, 2.474e-7, 2.311e-7, 2.162e-7, 2.026e-7, 1.902e-7,  
 01782 1.788e-7, 1.683e-7, 1.587e-7, 1.497e-7, 1.415e-7, 1.338e-7,  
 01783 1.266e-7, 1.2e-7, 1.138e-7, 1.08e-7, 1.027e-7, 9.764e-8,  
 01784 9.296e-8, 8.862e-8, 8.458e-8, 8.087e-8, 7.744e-8, 7.429e-8,  
 01785 7.145e-8, 6.893e-8, 6.664e-8, 6.468e-8, 6.322e-8, 6.162e-8,  
 01786 6.07e-8, 5.992e-8, 5.913e-8, 5.841e-8, 5.796e-8, 5.757e-8,  
 01787 5.746e-8, 5.731e-8, 5.679e-8, 5.577e-8, 5.671e-8, 5.656e-8,  
 01788 5.594e-8, 5.593e-8, 5.602e-8, 5.62e-8, 5.693e-8, 5.725e-8,  
 01789 5.858e-8, 6.037e-8, 6.249e-8, 6.535e-8, 6.899e-8, 7.356e-8,  
 01790 7.918e-8, 8.618e-8, 9.385e-8, 1.039e-7, 1.158e-7, 1.29e-7,  
 01791 1.437e-7, 1.65e-7, 1.871e-7, 2.121e-7, 2.427e-7, 2.773e-7,  
 01792 3.247e-7, 3.677e-7, 4.037e-7, 4.776e-7, 5.101e-7, 6.214e-7,  
 01793 6.936e-7, 7.581e-7, 8.486e-7, 9.355e-7, 9.942e-7, 1.063e-6,  
 01794 1.123e-6, 1.191e-6, 1.215e-6, 1.247e-6, 1.26e-6, 1.271e-6,  
 01795 1.284e-6, 1.317e-6, 1.323e-6, 1.349e-6, 1.353e-6, 1.362e-6,  
 01796 1.344e-6, 1.329e-6, 1.336e-6, 1.327e-6, 1.325e-6, 1.359e-6,  
 01797 1.374e-6, 1.415e-6, 1.462e-6, 1.526e-6, 1.619e-6, 1.735e-6,  
 01798 1.863e-6, 2.034e-6, 2.265e-6, 2.482e-6, 2.756e-6, 3.103e-6,  
 01799 3.466e-6, 3.832e-6, 4.378e-6, 4.913e-6, 5.651e-6, 6.311e-6,  
 01800 7.169e-6, 8.057e-6, 9.253e-6, 1.047e-5, 1.212e-5, 1.36e-5,  
 01801 1.569e-5, 1.776e-5, 2.02e-5, 2.281e-5, 2.683e-5, 2.994e-5,  
 01802 3.488e-5, 3.896e-5, 4.499e-5, 5.175e-5, 6.035e-5, 6.34e-5,  
 01803 7.281e-5, 7.923e-5, 8.348e-5, 9.631e-5, 1.044e-4, 1.102e-4,  
 01804 1.176e-4, 1.244e-4, 1.283e-4, 1.326e-4, 1.4e-4, 1.395e-4,  
 01805 1.387e-4, 1.363e-4, 1.314e-4, 1.241e-4, 1.228e-4, 1.148e-4,  
 01806 1.086e-4, 1.018e-4, 8.89e-5, 8.316e-5, 7.292e-5, 6.452e-5,  
 01807 5.625e-5, 5.045e-5, 4.38e-5, 3.762e-5, 3.29e-5, 2.836e-5,  
 01808 2.485e-5, 2.168e-5, 1.895e-5, 1.659e-5, 1.453e-5, 1.282e-5,  
 01809 1.132e-5, 1.001e-5, 8.836e-6, 7.804e-6, 6.922e-6, 6.116e-6,  
 01810 5.429e-6, 4.824e-6, 4.278e-6, 3.788e-6, 3.371e-6, 2.985e-6,  
 01811 2.649e-6, 2.357e-6, 2.09e-6, 1.858e-6, 1.647e-6, 1.462e-6,  
 01812 1.299e-6, 1.155e-6, 1.028e-6, 9.142e-7, 8.132e-7, 7.246e-7,  
 01813 6.451e-7, 5.764e-7, 5.151e-7, 4.603e-7, 4.121e-7, 3.694e-7,  
 01814 3.318e-7, 2.985e-7, 2.69e-7, 2.428e-7, 2.197e-7, 1.992e-7,  
 01815 1.81e-7, 1.649e-7, 1.506e-7, 1.378e-7, 1.265e-7, 1.163e-7,  
 01816 1.073e-7, 9.918e-8, 9.191e-8, 8.538e-8, 7.949e-8, 7.419e-8,  
 01817 6.94e-8, 6.508e-8, 6.114e-8, 5.761e-8, 5.437e-8, 5.146e-8,  
 01818 4.89e-8, 4.636e-8, 4.406e-8, 4.201e-8, 4.015e-8, 3.84e-8,  
 01819 3.661e-8, 3.51e-8, 3.377e-8, 3.242e-8, 3.13e-8, 3.015e-8,  
 01820 2.918e-8, 2.83e-8, 2.758e-8, 2.707e-8, 2.656e-8, 2.619e-8,  
 01821 2.609e-8, 2.615e-8, 2.63e-8, 2.675e-8, 2.745e-8, 2.842e-8,  
 01822 2.966e-8, 3.125e-8, 3.318e-8, 3.565e-8, 3.85e-8, 4.191e-8,  
 01823 4.59e-8, 5.059e-8, 5.607e-8, 6.239e-8, 6.958e-8, 7.796e-8,  
 01824 8.773e-8, 9.88e-8, 1.114e-7, 1.258e-7, 1.422e-7, 1.61e-7,  
 01825 1.822e-7, 2.06e-7, 2.337e-7, 2.645e-7, 2.996e-7, 3.393e-7,  
 01826 3.843e-7, 4.363e-7, 4.935e-7, 5.607e-7, 6.363e-7, 7.242e-7,  
 01827 8.23e-7, 9.411e-7, 1.071e-6, 1.232e-6, 1.402e-6, 1.6e-6, 1.82e-6,  
 01828 2.128e-6, 2.386e-6, 2.781e-6, 3.242e-6, 3.653e-6, 4.323e-6,  
 01829 4.747e-6, 5.321e-6, 5.919e-6, 6.681e-6, 7.101e-6, 7.983e-6,  
 01830 8.342e-6, 8.741e-6, 9.431e-6, 9.952e-6, 1.026e-5, 1.055e-5,  
 01831 1.095e-5, 1.095e-5, 1.087e-5, 1.056e-5, 1.026e-5, 9.715e-6,  
 01832 9.252e-6, 8.452e-6, 7.958e-6, 7.268e-6, 6.295e-6, 6.003e-6, 5e-6,  
 01833 4.591e-6, 3.983e-6, 3.479e-6, 3.058e-6, 2.667e-6, 2.293e-6,  
 01834 1.995e-6, 1.747e-6, 1.517e-6, 1.335e-6, 1.165e-6, 1.028e-6,  
 01835 9.007e-7, 7.956e-7, 7.015e-7, 6.192e-7, 5.491e-7, 4.859e-7,  
 01836 4.297e-7, 3.799e-7, 3.38e-7, 3.002e-7, 2.659e-7, 2.366e-7,  
 01837 2.103e-7, 1.861e-7, 1.655e-7, 1.469e-7, 1.309e-7, 1.162e-7,  
 01838 1.032e-7, 9.198e-8, 8.181e-8, 7.294e-8, 6.516e-8, 5.787e-8,  
 01839 5.163e-8, 4.612e-8, 4.119e-8, 3.695e-8, 3.308e-8, 2.976e-8,  
 01840 2.67e-8, 2.407e-8, 2.171e-8, 1.965e-8, 1.78e-8, 1.617e-8,  
 01841 1.47e-8, 1.341e-8, 1.227e-8, 1.125e-8, 1.033e-8, 9.524e-9,  
 01842 8.797e-9, 8.162e-9, 7.565e-9, 7.04e-9, 6.56e-9, 6.129e-9,  
 01843 5.733e-9, 5.376e-9, 5.043e-9, 4.75e-9, 4.466e-9, 4.211e-9,  
 01844 3.977e-9, 3.759e-9, 3.558e-9, 3.373e-9, 3.201e-9, 3.043e-9,  
 01845 2.895e-9, 2.76e-9, 2.635e-9, 2.518e-9, 2.411e-9, 2.314e-9,  
 01846 2.23e-9, 2.151e-9, 2.087e-9, 2.035e-9, 1.988e-9, 1.946e-9,  
 01847 1.927e-9, 1.916e-9, 1.916e-9, 1.933e-9, 1.966e-9, 2.018e-9,  
 01848 2.09e-9, 2.182e-9, 2.299e-9, 2.442e-9, 2.623e-9, 2.832e-9,  
 01849 3.079e-9, 3.368e-9, 3.714e-9, 4.104e-9, 4.567e-9, 5.091e-9,  
 01850 5.701e-9, 6.398e-9, 7.194e-9, 8.127e-9, 9.141e-9, 1.035e-8,  
 01851 1.177e-8, 1.338e-8, 1.508e-8, 1.711e-8, 1.955e-8, 2.216e-8,  
 01852 2.534e-8, 2.871e-8, 3.291e-8, 3.711e-8, 4.285e-8, 4.868e-8,  
 01853 5.509e-8, 6.276e-8, 7.262e-8, 8.252e-8, 9.4e-8, 1.064e-7,  
 01854 1.247e-7, 1.411e-7, 1.626e-7, 1.827e-7, 2.044e-7, 2.284e-7,  
 01855 2.452e-7, 2.854e-7, 3.026e-7, 3.278e-7, 3.474e-7, 3.693e-7,  
 01856 3.93e-7, 4.104e-7, 4.22e-7, 4.439e-7, 4.545e-7, 4.778e-7,  
 01857 4.812e-7, 5.018e-7, 4.899e-7, 5.075e-7, 5.073e-7, 5.171e-7,  
 01858 5.131e-7, 5.25e-7, 5.617e-7, 5.846e-7, 6.239e-7, 6.696e-7,  
 01859 7.398e-7, 8.073e-7, 9.15e-7, 1.009e-6, 1.116e-6, 1.264e-6,  
 01860 1.439e-6, 1.644e-6, 1.856e-6, 2.147e-6, 2.317e-6, 2.713e-6,  
 01861 2.882e-6, 2.99e-6, 3.489e-6, 3.581e-6, 4.033e-6, 4.26e-6,  
 01862 4.543e-6, 4.84e-6, 4.826e-6, 5.013e-6, 5.252e-6, 5.277e-6,  
 01863 5.306e-6, 5.236e-6, 5.123e-6, 5.171e-6, 4.843e-6, 4.615e-6,

01864 4.385e-6, 3.97e-6, 3.693e-6, 3.231e-6, 2.915e-6, 2.495e-6,  
01865 2.144e-6, 1.91e-6, 1.639e-6, 1.417e-6, 1.226e-6, 1.065e-6,  
01866 9.29e-7, 8.142e-7, 7.161e-7, 6.318e-7, 5.581e-7, 4.943e-7,  
01867 4.376e-7, 3.884e-7, 3.449e-7, 3.06e-7, 2.712e-7, 2.412e-7,  
01868 2.139e-7, 1.903e-7, 1.689e-7, 1.499e-7, 1.331e-7, 1.183e-7,  
01869 1.05e-7, 9.362e-8, 8.306e-8, 7.403e-8, 6.578e-8, 5.853e-8,  
01870 5.216e-8, 4.632e-8, 4.127e-8, 3.678e-8, 3.279e-8, 2.923e-8,  
01871 2.612e-8, 2.339e-8, 2.094e-8, 1.877e-8, 1.686e-8, 1.516e-8,  
01872 1.366e-8, 1.234e-8, 1.114e-8, 1.012e-8, 9.182e-9, 8.362e-9,  
01873 7.634e-9, 6.981e-9, 6.406e-9, 5.888e-9, 5.428e-9, 5.021e-9,  
01874 4.65e-9, 4.326e-9, 4.033e-9, 3.77e-9, 3.536e-9, 3.327e-9,  
01875 3.141e-9, 2.974e-9, 2.825e-9, 2.697e-9, 2.584e-9, 2.488e-9,  
01876 2.406e-9, 2.34e-9, 2.292e-9, 2.259e-9, 2.244e-9, 2.243e-9,  
01877 2.272e-9, 2.31e-9, 2.378e-9, 2.454e-9, 2.618e-9, 2.672e-9,  
01878 2.831e-9, 3.05e-9, 3.225e-9, 3.425e-9, 3.677e-9, 3.968e-9,  
01879 4.221e-9, 4.639e-9, 4.96e-9, 5.359e-9, 5.649e-9, 6.23e-9,  
01880 6.716e-9, 7.218e-9, 7.746e-9, 7.988e-9, 8.627e-9, 8.999e-9,  
01881 9.442e-9, 9.82e-9, 1.015e-8, 1.06e-8, 1.079e-8, 1.109e-8,  
01882 1.137e-8, 1.186e-8, 1.18e-8, 1.187e-8, 1.194e-8, 1.192e-8,  
01883 1.224e-8, 1.245e-8, 1.246e-8, 1.318e-8, 1.377e-8, 1.471e-8,  
01884 1.582e-8, 1.713e-8, 1.853e-8, 2.063e-8, 2.27e-8, 2.567e-8,  
01885 2.891e-8, 3.264e-8, 3.744e-8, 4.286e-8, 4.915e-8, 5.623e-8,  
01886 6.336e-8, 7.293e-8, 8.309e-8, 9.319e-8, 1.091e-7, 1.243e-7,  
01887 1.348e-7, 1.449e-7, 1.62e-7, 1.846e-7, 1.937e-7, 2.04e-7,  
01888 2.179e-7, 2.298e-7, 2.433e-7, 2.439e-7, 2.464e-7, 2.611e-7,  
01889 2.617e-7, 2.582e-7, 2.453e-7, 2.401e-7, 2.349e-7, 2.203e-7,  
01890 2.066e-7, 1.939e-7, 1.78e-7, 1.558e-7, 1.391e-7, 1.203e-7,  
01891 1.048e-7, 9.464e-8, 8.306e-8, 7.239e-8, 6.317e-8, 5.52e-8,  
01892 4.847e-8, 4.282e-8, 3.796e-8, 3.377e-8, 2.996e-8, 2.678e-8,  
01893 2.4e-8, 2.134e-8, 1.904e-8, 1.705e-8, 1.523e-8, 1.35e-8,  
01894 1.204e-8, 1.07e-8, 9.408e-9, 8.476e-9, 7.47e-9, 6.679e-9,  
01895 5.929e-9, 5.267e-9, 4.711e-9, 4.172e-9, 3.761e-9, 3.288e-9,  
01896 2.929e-9, 2.609e-9, 2.315e-9, 2.042e-9, 1.844e-9, 1.64e-9,  
01897 1.47e-9, 1.31e-9, 1.176e-9, 1.049e-9, 9.377e-10, 8.462e-10,  
01898 7.616e-10, 6.854e-10, 6.191e-10, 5.596e-10, 5.078e-10, 4.611e-10,  
01899 4.197e-10, 3.83e-10, 3.505e-10, 3.215e-10, 2.956e-10, 2.726e-10,  
01900 2.521e-10, 2.338e-10, 2.173e-10, 2.026e-10, 1.895e-10, 1.777e-10,  
01901 1.672e-10, 1.579e-10, 1.496e-10, 1.423e-10, 1.358e-10, 1.302e-10,  
01902 1.254e-10, 1.216e-10, 1.187e-10, 1.163e-10, 1.147e-10, 1.145e-10,  
01903 1.15e-10, 1.17e-10, 1.192e-10, 1.25e-10, 1.298e-10, 1.345e-10,  
01904 1.405e-10, 1.538e-10, 1.648e-10, 1.721e-10, 1.872e-10, 1.968e-10,  
01905 2.089e-10, 2.172e-10, 2.317e-10, 2.389e-10, 2.503e-10, 2.585e-10,  
01906 2.686e-10, 2.8e-10, 2.895e-10, 3.019e-10, 3.037e-10, 3.076e-10,  
01907 3.146e-10, 3.198e-10, 3.332e-10, 3.397e-10, 3.54e-10, 3.667e-10,  
01908 3.895e-10, 4.071e-10, 4.565e-10, 4.983e-10, 5.439e-10, 5.968e-10,  
01909 6.676e-10, 7.456e-10, 8.405e-10, 9.478e-10, 1.064e-9, 1.218e-9,  
01910 1.386e-9, 1.581e-9, 1.787e-9, 2.032e-9, 2.347e-9, 2.677e-9,  
01911 3.008e-9, 3.544e-9, 4.056e-9, 4.687e-9, 5.331e-9, 6.227e-9,  
01912 6.854e-9, 8.139e-9, 8.945e-9, 9.865e-9, 1.125e-8, 1.178e-8,  
01913 1.364e-8, 1.436e-8, 1.54e-8, 1.672e-8, 1.793e-8, 1.906e-8,  
01914 2.036e-8, 2.144e-8, 2.292e-8, 2.371e-8, 2.493e-8, 2.606e-8,  
01915 2.706e-8, 2.866e-8, 3.036e-8, 3.136e-8, 3.405e-8, 3.665e-8,  
01916 3.837e-8, 4.229e-8, 4.748e-8, 5.32e-8, 5.763e-8, 6.677e-8,  
01917 7.216e-8, 7.716e-8, 8.958e-8, 9.419e-8, 1.036e-7, 1.108e-7,  
01918 1.189e-7, 1.246e-7, 1.348e-7, 1.31e-7, 1.361e-7, 1.364e-7,  
01919 1.363e-7, 1.343e-7, 1.293e-7, 1.254e-7, 1.235e-7, 1.158e-7,  
01920 1.107e-7, 9.961e-8, 9.011e-8, 7.91e-8, 6.916e-8, 6.338e-8,  
01921 5.564e-8, 4.827e-8, 4.198e-8, 3.695e-8, 3.276e-8, 2.929e-8,  
01922 2.633e-8, 2.391e-8, 2.192e-8, 2.021e-8, 1.89e-8, 1.772e-8,  
01923 1.667e-8, 1.603e-8, 1.547e-8, 1.537e-8, 1.492e-8, 1.515e-8,  
01924 1.479e-8, 1.45e-8, 1.513e-8, 1.495e-8, 1.529e-8, 1.565e-8,  
01925 1.564e-8, 1.553e-8, 1.569e-8, 1.584e-8, 1.57e-8, 1.538e-8,  
01926 1.513e-8, 1.472e-8, 1.425e-8, 1.349e-8, 1.328e-8, 1.249e-8,  
01927 1.17e-8, 1.077e-8, 9.514e-9, 8.614e-9, 7.46e-9, 6.621e-9,  
01928 5.775e-9, 5.006e-9, 4.308e-9, 3.747e-9, 3.24e-9, 2.84e-9,  
01929 2.481e-9, 2.184e-9, 1.923e-9, 1.71e-9, 1.504e-9, 1.334e-9,  
01930 1.187e-9, 1.053e-9, 9.367e-10, 8.306e-10, 7.419e-10, 6.63e-10,  
01931 5.918e-10, 5.277e-10, 4.717e-10, 4.222e-10, 3.783e-10, 3.39e-10,  
01932 3.036e-10, 2.729e-10, 2.455e-10, 2.211e-10, 1.995e-10, 1.804e-10,  
01933 1.635e-10, 1.485e-10, 1.355e-10, 1.24e-10, 1.139e-10, 1.051e-10,  
01934 9.757e-11, 9.114e-11, 8.577e-11, 8.139e-11, 7.792e-11, 7.52e-11,  
01935 7.39e-11, 7.311e-11, 7.277e-11, 7.482e-11, 7.698e-11, 8.162e-11,  
01936 8.517e-11, 8.968e-11, 9.905e-11, 1.075e-10, 1.187e-10, 1.291e-10,  
01937 1.426e-10, 1.573e-10, 1.734e-10, 1.905e-10, 2.097e-10, 2.28e-10,  
01938 2.473e-10, 2.718e-10, 2.922e-10, 3.128e-10, 3.361e-10, 3.641e-10,  
01939 3.91e-10, 4.196e-10, 4.501e-10, 4.932e-10, 5.258e-10, 5.755e-10,  
01940 6.253e-10, 6.664e-10, 7.344e-10, 7.985e-10, 8.877e-10, 1.005e-9,  
01941 1.118e-9, 1.251e-9, 1.428e-9, 1.61e-9, 1.888e-9, 2.077e-9,  
01942 2.331e-9, 2.751e-9, 3.061e-9, 3.522e-9, 3.805e-9, 4.181e-9,  
01943 4.575e-9, 5.167e-9, 5.634e-9, 6.007e-9, 6.501e-9, 6.829e-9,  
01944 7.211e-9, 7.262e-9, 7.696e-9, 7.832e-9, 7.799e-9, 7.651e-9,  
01945 7.304e-9, 7.15e-9, 6.977e-9, 6.603e-9, 6.209e-9, 5.69e-9,  
01946 5.432e-9, 4.764e-9, 4.189e-9, 3.64e-9, 3.203e-9, 2.848e-9,  
01947 2.51e-9, 2.194e-9, 1.946e-9, 1.75e-9, 1.567e-9, 1.426e-9,  
01948 1.302e-9, 1.197e-9, 1.109e-9, 1.035e-9, 9.719e-10, 9.207e-10,  
01949 8.957e-10, 8.578e-10, 8.262e-10, 8.117e-10, 7.987e-10, 7.875e-10,  
01950 7.741e-10, 7.762e-10, 7.537e-10, 7.424e-10, 7.474e-10, 7.294e-10,

01951 7.216e-10, 7.233e-10, 7.075e-10, 6.892e-10, 6.618e-10, 6.314e-10,  
01952 6.208e-10, 5.689e-10, 5.55e-10, 4.984e-10, 4.6e-10, 4.078e-10,  
01953 3.879e-10, 3.459e-10, 2.982e-10, 2.626e-10, 2.329e-10, 1.988e-10,  
01954 1.735e-10, 1.487e-10, 1.297e-10, 1.133e-10, 9.943e-11, 8.736e-11,  
01955 7.726e-11, 6.836e-11, 6.053e-11, 5.384e-11, 4.789e-11, 4.267e-11,  
01956 3.804e-11, 3.398e-11, 3.034e-11, 2.71e-11, 2.425e-11, 2.173e-11,  
01957 1.95e-11, 1.752e-11, 1.574e-11, 1.418e-11, 1.278e-11, 1.154e-11,  
01958 1.044e-11, 9.463e-12, 8.602e-12, 7.841e-12, 7.171e-12, 6.584e-12,  
01959 6.073e-12, 5.631e-12, 5.254e-12, 4.937e-12, 4.679e-12, 4.476e-12,  
01960 4.328e-12, 4.233e-12, 4.194e-12, 4.211e-12, 4.286e-12, 4.424e-12,  
01961 4.628e-12, 4.906e-12, 5.262e-12, 5.708e-12, 6.254e-12, 6.914e-12,  
01962 7.714e-12, 8.677e-12, 9.747e-12, 1.101e-11, 1.256e-11, 1.409e-11,  
01963 1.597e-11, 1.807e-11, 2.034e-11, 2.316e-11, 2.622e-11, 2.962e-11,  
01964 3.369e-11, 3.819e-11, 4.329e-11, 4.932e-11, 5.589e-11, 6.364e-11,  
01965 7.284e-11, 8.236e-11, 9.447e-11, 1.078e-10, 1.229e-10, 1.417e-10,  
01966 1.614e-10, 1.843e-10, 2.107e-10, 2.406e-10, 2.728e-10, 3.195e-10,  
01967 3.595e-10, 4.153e-10, 4.736e-10, 5.41e-10, 6.088e-10, 6.769e-10,  
01968 7.691e-10, 8.545e-10, 9.621e-10, 1.047e-9, 1.161e-9, 1.296e-9,  
01969 1.424e-9, 1.576e-9, 1.739e-9, 1.893e-9, 2.08e-9, 2.336e-9,  
01970 2.604e-9, 2.76e-9, 3.001e-9, 3.365e-9, 3.55e-9, 3.895e-9,  
01971 4.183e-9, 4.614e-9, 4.846e-9, 5.068e-9, 5.427e-9, 5.541e-9,  
01972 5.864e-9, 5.997e-9, 5.997e-9, 6.061e-9, 5.944e-9, 5.855e-9,  
01973 5.661e-9, 5.523e-9, 5.374e-9, 4.94e-9, 4.688e-9, 4.17e-9,  
01974 3.913e-9, 3.423e-9, 2.997e-9, 2.598e-9, 2.253e-9, 1.946e-9,  
01975 1.71e-9, 1.507e-9, 1.336e-9, 1.19e-9, 1.068e-9, 9.623e-10,  
01976 8.772e-10, 8.007e-10, 7.42e-10, 6.884e-10, 6.483e-10, 6.162e-10,  
01977 5.922e-10, 5.628e-10, 5.654e-10, 5.637e-10, 5.701e-10, 5.781e-10,  
01978 5.874e-10, 6.268e-10, 6.357e-10, 6.525e-10, 7.137e-10, 7.441e-10,  
01979 8.024e-10, 8.485e-10, 9.143e-10, 9.536e-10, 9.717e-10, 1.018e-9,  
01980 1.042e-9, 1.054e-9, 1.092e-9, 1.079e-9, 1.064e-9, 1.043e-9,  
01981 1.02e-9, 9.687e-10, 9.273e-10, 9.208e-10, 9.068e-10, 7.687e-10,  
01982 7.385e-10, 6.595e-10, 5.87e-10, 5.144e-10, 4.417e-10, 3.804e-10,  
01983 3.301e-10, 2.866e-10, 2.509e-10, 2.202e-10, 1.947e-10, 1.719e-10,  
01984 1.525e-10, 1.361e-10, 1.21e-10, 1.084e-10, 9.8e-11, 8.801e-11,  
01985 7.954e-11, 7.124e-11, 6.335e-11, 5.76e-11, 5.132e-11, 4.601e-11,  
01986 4.096e-11, 3.657e-11, 3.25e-11, 2.909e-11, 2.587e-11, 2.297e-11,  
01987 2.05e-11, 1.828e-11, 1.632e-11, 1.462e-11, 1.314e-11, 1.185e-11,  
01988 1.073e-11, 9.76e-12, 8.922e-12, 8.206e-12, 7.602e-12, 7.1e-12,  
01989 6.694e-12, 6.378e-12, 6.149e-12, 6.004e-12, 5.941e-12, 5.962e-12,  
01990 6.069e-12, 6.265e-12, 6.551e-12, 6.935e-12, 7.457e-12, 8.074e-12,  
01991 8.811e-12, 9.852e-12, 1.086e-11, 1.207e-11, 1.361e-11, 1.553e-11,  
01992 1.737e-11, 1.93e-11, 2.175e-11, 2.41e-11, 2.706e-11, 3.023e-11,  
01993 3.313e-11, 3.657e-11, 4.118e-11, 4.569e-11, 5.025e-11, 5.66e-11,  
01994 6.231e-11, 6.881e-11, 7.996e-11, 8.526e-11, 9.694e-11, 1.106e-10,  
01995 1.222e-10, 1.355e-10, 1.525e-10, 1.775e-10, 1.924e-10, 2.181e-10,  
01996 2.379e-10, 2.662e-10, 2.907e-10, 3.154e-10, 3.366e-10, 3.579e-10,  
01997 3.858e-10, 4.046e-10, 4.196e-10, 4.166e-10, 4.457e-10, 4.466e-10,  
01998 4.404e-10, 4.337e-10, 4.15e-10, 4.083e-10, 3.91e-10, 3.723e-10,  
01999 3.514e-10, 3.303e-10, 2.847e-10, 2.546e-10, 2.23e-10, 1.994e-10,  
02000 1.733e-10, 1.488e-10, 1.297e-10, 1.144e-10, 1.004e-10, 8.741e-11,  
02001 7.928e-11, 7.034e-11, 6.323e-11, 5.754e-11, 5.25e-11, 4.85e-11,  
02002 4.502e-11, 4.286e-11, 4.028e-11, 3.899e-11, 3.824e-11, 3.761e-11,  
02003 3.804e-11, 3.839e-11, 3.845e-11, 4.244e-11, 4.382e-11, 4.582e-11,  
02004 4.847e-11, 5.209e-11, 5.384e-11, 5.887e-11, 6.371e-11, 6.737e-11,  
02005 7.168e-11, 7.415e-11, 7.827e-11, 8.037e-11, 8.12e-11, 8.071e-11,  
02006 8.008e-11, 7.851e-11, 7.544e-11, 7.377e-11, 7.173e-11, 6.801e-11,  
02007 6.267e-11, 5.727e-11, 5.288e-11, 4.853e-11, 4.082e-11, 3.645e-11,  
02008 3.136e-11, 2.672e-11, 2.304e-11, 1.986e-11, 1.725e-11, 1.503e-11,  
02009 1.315e-11, 1.153e-11, 1.014e-11, 8.942e-12, 7.901e-12, 6.993e-12,  
02010 6.199e-12, 5.502e-12, 4.89e-12, 4.351e-12, 3.878e-12, 3.461e-12,  
02011 3.094e-12, 2.771e-12, 2.488e-12, 2.241e-12, 2.025e-12, 1.838e-12,  
02012 1.677e-12, 1.541e-12, 1.427e-12, 1.335e-12, 1.262e-12, 1.209e-12,  
02013 1.176e-12, 1.161e-12, 1.165e-12, 1.189e-12, 1.234e-12, 1.3e-12,  
02014 1.389e-12, 1.503e-12, 1.644e-12, 1.814e-12, 2.017e-12, 2.255e-12,  
02015 2.534e-12, 2.858e-12, 3.231e-12, 3.661e-12, 4.153e-12, 4.717e-12,  
02016 5.36e-12, 6.094e-12, 6.93e-12, 7.882e-12, 8.966e-12, 1.02e-11,  
02017 1.162e-11, 1.324e-11, 1.51e-11, 1.72e-11, 1.965e-11, 2.237e-11,  
02018 2.56e-11, 2.927e-11, 3.371e-11, 3.842e-11, 4.429e-11, 5.139e-11,  
02019 5.798e-11, 6.697e-11, 7.626e-11, 8.647e-11, 1.022e-10, 1.136e-10,  
02020 1.3e-10, 1.481e-10, 1.672e-10, 1.871e-10, 2.126e-10, 2.357e-10,  
02021 2.583e-10, 2.997e-10, 3.289e-10, 3.702e-10, 4.012e-10, 4.319e-10,  
02022 4.527e-10, 5.001e-10, 5.448e-10, 5.611e-10, 5.76e-10, 5.965e-10,  
02023 6.079e-10, 6.207e-10, 6.276e-10, 6.222e-10, 6.137e-10, 6e-10,  
02024 5.814e-10, 5.393e-10, 5.35e-10, 4.947e-10, 4.629e-10, 4.117e-10,  
02025 3.712e-10, 3.372e-10, 2.923e-10, 2.55e-10, 2.232e-10, 1.929e-10,  
02026 1.679e-10, 1.46e-10, 1.289e-10, 1.13e-10, 9.953e-11, 8.763e-11,  
02027 7.76e-11, 6.9e-11, 6.16e-11, 5.525e-11, 4.958e-11, 4.489e-11,  
02028 4.072e-11, 3.728e-11, 3.438e-11, 3.205e-11, 3.006e-11, 2.848e-11,  
02029 2.766e-11, 2.688e-11, 2.664e-11, 2.67e-11, 2.696e-11, 2.786e-11,  
02030 2.861e-11, 3.009e-11, 3.178e-11, 3.389e-11, 3.587e-11, 3.819e-11,  
02031 4.054e-11, 4.417e-11, 4.703e-11, 5.137e-11, 5.46e-11, 6.055e-11,  
02032 6.333e-11, 6.773e-11, 7.219e-11, 7.717e-11, 8.131e-11, 8.491e-11,  
02033 8.574e-11, 9.01e-11, 9.017e-11, 8.999e-11, 8.959e-11, 8.838e-11,  
02034 8.579e-11, 8.162e-11, 8.098e-11, 7.472e-11, 7.108e-11, 6.559e-11,  
02035 5.994e-11, 5.172e-11, 4.424e-11, 3.951e-11, 3.34e-11, 2.902e-11,  
02036 2.541e-11, 2.215e-11, 1.945e-11, 1.716e-11, 1.503e-11, 1.339e-11,  
02037 1.185e-11, 1.05e-11, 9.336e-12, 8.307e-12, 7.312e-12, 6.55e-12,

```

02038     5.836e-12, 5.178e-12, 4.6e-12, 4.086e-12, 3.639e-12, 3.247e-12,
02039     2.904e-12, 2.604e-12, 2.341e-12, 2.112e-12, 1.914e-12, 1.744e-12,
02040     1.598e-12, 1.476e-12, 1.374e-12, 1.293e-12, 1.23e-12, 1.185e-12,
02041     1.158e-12, 1.147e-12, 1.154e-12, 1.177e-12, 1.219e-12, 1.28e-12,
02042     1.36e-12, 1.463e-12, 1.591e-12, 1.75e-12, 1.94e-12, 2.156e-12,
02043     2.43e-12, 2.748e-12, 3.052e-12, 3.533e-12, 3.967e-12, 4.471e-12,
02044     5.041e-12, 5.86e-12, 6.664e-12, 7.522e-12, 8.342e-12, 9.412e-12,
02045     1.072e-11, 1.213e-11, 1.343e-11, 1.496e-11, 1.664e-11, 1.822e-11,
02046     2.029e-11, 2.233e-11, 2.457e-11, 2.709e-11, 2.928e-11, 3.115e-11,
02047     3.356e-11, 3.592e-11, 3.818e-11, 3.936e-11, 4.061e-11, 4.149e-11,
02048     4.299e-11, 4.223e-11, 4.251e-11, 4.287e-11, 4.177e-11, 4.094e-11,
02049     3.942e-11, 3.772e-11, 3.614e-11, 3.394e-11, 3.222e-11, 2.791e-11,
02050     2.665e-11, 2.309e-11, 2.032e-11, 1.74e-11, 1.535e-11, 1.323e-11,
02051     1.151e-11, 9.803e-12, 8.65e-12, 7.54e-12, 6.619e-12, 5.832e-12,
02052     5.113e-12, 4.503e-12, 3.975e-12, 3.52e-12, 3.112e-12, 2.797e-12,
02053     2.5e-12, 2.24e-12, 2.013e-12, 1.819e-12, 1.653e-12, 1.513e-12,
02054     1.395e-12, 1.299e-12, 1.225e-12, 1.168e-12, 1.124e-12, 1.148e-12,
02055     1.107e-12, 1.128e-12, 1.169e-12, 1.233e-12, 1.307e-12, 1.359e-12,
02056     1.543e-12, 1.686e-12, 1.794e-12, 2.028e-12, 2.21e-12, 2.441e-12,
02057     2.653e-12, 2.828e-12, 3.093e-12, 3.28e-12, 3.551e-12, 3.677e-12,
02058     3.803e-12, 3.844e-12, 4.068e-12, 4.093e-12, 4.002e-12, 3.904e-12,
02059     3.624e-12, 3.633e-12, 3.622e-12, 3.443e-12, 3.184e-12, 2.934e-12,
02060     2.476e-12, 2.212e-12, 1.867e-12, 1.594e-12, 1.37e-12, 1.192e-12,
02061     1.045e-12, 9.211e-13, 8.17e-13, 7.29e-13, 6.55e-13, 5.929e-13,
02062     5.415e-13, 4.995e-13, 4.661e-13, 4.406e-13, 4.225e-13, 4.116e-13,
02063     4.075e-13, 4.102e-13, 4.198e-13, 4.365e-13, 4.606e-13, 4.925e-13,
02064     5.326e-13, 5.818e-13, 6.407e-13, 7.104e-13, 7.92e-13, 8.868e-13,
02065     9.964e-13, 1.123e-12, 1.268e-12, 1.434e-12, 1.626e-12, 1.848e-12,
02066     2.107e-12, 2.422e-12, 2.772e-12, 3.145e-12, 3.704e-12, 4.27e-12,
02067     4.721e-12, 5.361e-12, 6.083e-12, 7.095e-12, 7.968e-12, 9.228e-12,
02068     1.048e-11, 1.187e-11, 1.336e-11, 1.577e-11, 1.772e-11, 2.017e-11,
02069     2.25e-11, 2.63e-11, 2.911e-11, 3.356e-11, 3.82e-11, 4.173e-11,
02070     4.811e-11, 5.254e-11, 5.839e-11, 6.187e-11, 6.805e-11, 7.118e-11,
02071     7.369e-11, 7.664e-11, 7.794e-11, 7.947e-11, 8.036e-11, 7.954e-11,
02072     7.849e-11, 7.518e-11, 7.462e-11, 6.926e-11, 6.531e-11, 6.197e-11,
02073     5.421e-11, 4.777e-11, 4.111e-11, 3.679e-11, 3.166e-11, 2.786e-11,
02074     2.436e-11, 2.144e-11, 1.859e-11, 1.628e-11, 1.414e-11, 1.237e-11,
02075     1.093e-11, 9.558e-12
02076 };
02077
02078 static double h2o260[2001] = { .2752, .2732, .2749, .2676, .2667, .2545,
02079     .2497, .2327, .2218, .2036, .1825, .1694, .1497, .1353, .121,
02080     .1014, .09405, .07848, .07195, .06246, .05306, .04853, .04138,
02081     .03735, .03171, .02785, .02431, .02111, .01845, .0164, .01405,
02082     .01255, .01098, .009797, .008646, .007779, .006898, .006099,
02083     .005453, .004909, .004413, .003959, .003581, .003199, .002871,
02084     .002583, .00233, .002086, .001874, .001684, .001512, .001361,
02085     .001225, .0011, 9.89e-4, 8.916e-4, 8.039e-4, 7.256e-4, 6.545e-4,
02086     5.918e-4, 5.359e-4, 4.867e-4, 4.426e-4, 4.033e-4, 3.682e-4,
02087     3.366e-4, 3.085e-4, 2.833e-4, 2.605e-4, 2.403e-4, 2.221e-4,
02088     2.055e-4, 1.908e-4, 1.774e-4, 1.653e-4, 1.544e-4, 1.443e-4,
02089     1.351e-4, 1.267e-4, 1.19e-4, 1.119e-4, 1.053e-4, 9.922e-5,
02090     9.355e-5, 8.831e-5, 8.339e-5, 7.878e-5, 7.449e-5, 7.043e-5,
02091     6.664e-5, 6.307e-5, 5.969e-5, 5.654e-5, 5.357e-5, 5.075e-5,
02092     4.81e-5, 4.56e-5, 4.322e-5, 4.102e-5, 3.892e-5, 3.696e-5,
02093     3.511e-5, 3.339e-5, 3.177e-5, 3.026e-5, 2.886e-5, 2.756e-5,
02094     2.636e-5, 2.527e-5, 2.427e-5, 2.337e-5, 2.257e-5, 2.185e-5,
02095     2.127e-5, 2.08e-5, 2.041e-5, 2.013e-5, 2e-5, 1.997e-5, 2.009e-5,
02096     2.031e-5, 2.068e-5, 2.124e-5, 2.189e-5, 2.267e-5, 2.364e-5,
02097     2.463e-5, 2.618e-5, 2.774e-5, 2.937e-5, 3.144e-5, 3.359e-5,
02098     3.695e-5, 4.002e-5, 4.374e-5, 4.947e-5, 5.431e-5, 6.281e-5,
02099     7.169e-5, 8.157e-5, 9.728e-5, 1.079e-4, 1.337e-4, 1.442e-4,
02100     1.683e-4, 1.879e-4, 2.223e-4, 2.425e-4, 2.838e-4, 3.143e-4,
02101     3.527e-4, 4.012e-4, 4.237e-4, 4.747e-4, 5.057e-4, 5.409e-4,
02102     5.734e-4, 5.944e-4, 6.077e-4, 6.175e-4, 6.238e-4, 6.226e-4,
02103     6.248e-4, 6.192e-4, 6.098e-4, 5.818e-4, 5.709e-4, 5.465e-4,
02104     5.043e-4, 4.699e-4, 4.294e-4, 3.984e-4, 3.672e-4, 3.152e-4,
02105     2.883e-4, 2.503e-4, 2.211e-4, 1.92e-4, 1.714e-4, 1.485e-4,
02106     1.358e-4, 1.156e-4, 1.021e-4, 8.887e-5, 7.842e-5, 7.12e-5,
02107     6.186e-5, 5.73e-5, 4.792e-5, 4.364e-5, 3.72e-5, 3.28e-5,
02108     2.946e-5, 2.591e-5, 2.261e-5, 2.048e-5, 1.813e-5, 1.63e-5,
02109     1.447e-5, 1.282e-5, 1.167e-5, 1.041e-5, 9.449e-6, 8.51e-6,
02110     7.596e-6, 6.961e-6, 6.272e-6, 5.728e-6, 5.198e-6, 4.667e-6,
02111     4.288e-6, 3.897e-6, 3.551e-6, 3.235e-6, 2.952e-6, 2.688e-6,
02112     2.449e-6, 2.241e-6, 2.05e-6, 1.879e-6, 1.722e-6, 1.582e-6,
02113     1.456e-6, 1.339e-6, 1.236e-6, 1.144e-6, 1.06e-6, 9.83e-7,
02114     9.149e-7, 8.535e-7, 7.973e-7, 7.466e-7, 6.999e-7, 6.574e-7,
02115     6.18e-7, 5.821e-7, 5.487e-7, 5.18e-7, 4.896e-7, 4.631e-7,
02116     4.386e-7, 4.16e-7, 3.945e-7, 3.748e-7, 3.562e-7, 3.385e-7,
02117     3.222e-7, 3.068e-7, 2.922e-7, 2.788e-7, 2.659e-7, 2.539e-7,
02118     2.425e-7, 2.318e-7, 2.219e-7, 2.127e-7, 2.039e-7, 1.958e-7,
02119     1.885e-7, 1.818e-7, 1.758e-7, 1.711e-7, 1.662e-7, 1.63e-7,
02120     1.605e-7, 1.58e-7, 1.559e-7, 1.545e-7, 1.532e-7, 1.522e-7,
02121     1.51e-7, 1.495e-7, 1.465e-7, 1.483e-7, 1.469e-7, 1.448e-7,
02122     1.444e-7, 1.436e-7, 1.426e-7, 1.431e-7, 1.425e-7, 1.445e-7,
02123     1.477e-7, 1.515e-7, 1.567e-7, 1.634e-7, 1.712e-7, 1.802e-7,
02124     1.914e-7, 2.024e-7, 2.159e-7, 2.295e-7, 2.461e-7, 2.621e-7,

```

02125 2.868e-7, 3.102e-7, 3.394e-7, 3.784e-7, 4.223e-7, 4.864e-7,  
02126 5.501e-7, 6.039e-7, 7.193e-7, 7.728e-7, 9.514e-7, 1.073e-6,  
02127 1.18e-6, 1.333e-6, 1.472e-6, 1.566e-6, 1.677e-6, 1.784e-6,  
02128 1.904e-6, 1.953e-6, 2.02e-6, 2.074e-6, 2.128e-6, 2.162e-6,  
02129 2.219e-6, 2.221e-6, 2.249e-6, 2.239e-6, 2.235e-6, 2.185e-6,  
02130 2.141e-6, 2.124e-6, 2.09e-6, 2.068e-6, 2.1e-6, 2.104e-6,  
02131 2.142e-6, 2.181e-6, 2.257e-6, 2.362e-6, 2.5e-6, 2.664e-6,  
02132 2.884e-6, 3.189e-6, 3.48e-6, 3.847e-6, 4.313e-6, 4.79e-6,  
02133 5.25e-6, 5.989e-6, 6.692e-6, 7.668e-6, 8.52e-6, 9.606e-6,  
02134 1.073e-5, 1.225e-5, 1.377e-5, 1.582e-5, 1.761e-5, 2.029e-5,  
02135 2.284e-5, 2.602e-5, 2.94e-5, 3.483e-5, 3.928e-5, 4.618e-5,  
02136 5.24e-5, 6.132e-5, 7.183e-5, 8.521e-5, 9.111e-5, 1.07e-4,  
02137 1.184e-4, 1.264e-4, 1.475e-4, 1.612e-4, 1.704e-4, 1.818e-4,  
02138 1.924e-4, 1.994e-4, 2.061e-4, 2.18e-4, 2.187e-4, 2.2e-4,  
02139 2.196e-4, 2.131e-4, 2.015e-4, 1.988e-4, 1.847e-4, 1.729e-4,  
02140 1.597e-4, 1.373e-4, 1.262e-4, 1.087e-4, 9.439e-5, 8.061e-5,  
02141 7.093e-5, 6.049e-5, 5.12e-5, 4.435e-5, 3.817e-5, 3.34e-5,  
02142 2.927e-5, 2.573e-5, 2.291e-5, 2.04e-5, 1.827e-5, 1.636e-5,  
02143 1.463e-5, 1.309e-5, 1.17e-5, 1.047e-5, 9.315e-6, 8.328e-6,  
02144 7.458e-6, 6.665e-6, 5.94e-6, 5.316e-6, 4.752e-6, 4.252e-6,  
02145 3.825e-6, 3.421e-6, 3.064e-6, 2.746e-6, 2.465e-6, 2.216e-6,  
02146 1.99e-6, 1.79e-6, 1.609e-6, 1.449e-6, 1.306e-6, 1.177e-6,  
02147 1.063e-6, 9.607e-7, 8.672e-7, 7.855e-7, 7.118e-7, 6.46e-7,  
02148 5.871e-7, 5.34e-7, 4.868e-7, 4.447e-7, 4.068e-7, 3.729e-7,  
02149 3.423e-7, 3.151e-7, 2.905e-7, 2.686e-7, 2.484e-7, 2.306e-7,  
02150 2.142e-7, 1.995e-7, 1.86e-7, 1.738e-7, 1.626e-7, 1.522e-7,  
02151 1.427e-7, 1.338e-7, 1.258e-7, 1.183e-7, 1.116e-7, 1.056e-7,  
02152 9.972e-8, 9.46e-8, 9.007e-8, 8.592e-8, 8.195e-8, 7.816e-8,  
02153 7.483e-8, 7.193e-8, 6.892e-8, 6.642e-8, 6.386e-8, 6.154e-8,  
02154 5.949e-8, 5.764e-8, 5.622e-8, 5.479e-8, 5.364e-8, 5.301e-8,  
02155 5.267e-8, 5.263e-8, 5.313e-8, 5.41e-8, 5.55e-8, 5.745e-8,  
02156 6.003e-8, 6.311e-8, 6.713e-8, 7.173e-8, 7.724e-8, 8.368e-8,  
02157 9.121e-8, 9.986e-8, 1.097e-7, 1.209e-7, 1.338e-7, 1.486e-7,  
02158 1.651e-7, 1.837e-7, 2.048e-7, 2.289e-7, 2.557e-7, 2.857e-7,  
02159 3.195e-7, 3.587e-7, 4.015e-7, 4.497e-7, 5.049e-7, 5.665e-7,  
02160 6.366e-7, 7.121e-7, 7.996e-7, 8.946e-7, 1.002e-6, 1.117e-6,  
02161 1.262e-6, 1.416e-6, 1.611e-6, 1.807e-6, 2.056e-6, 2.351e-6,  
02162 2.769e-6, 3.138e-6, 3.699e-6, 4.386e-6, 5.041e-6, 6.074e-6,  
02163 6.812e-6, 7.79e-6, 8.855e-6, 1.014e-5, 1.095e-5, 1.245e-5,  
02164 1.316e-5, 1.39e-5, 1.504e-5, 1.583e-5, 1.617e-5, 1.652e-5,  
02165 1.713e-5, 1.724e-5, 1.715e-5, 1.668e-5, 1.629e-5, 1.552e-5,  
02166 1.478e-5, 1.34e-5, 1.245e-5, 1.121e-5, 9.575e-6, 8.956e-6,  
02167 7.345e-6, 6.597e-6, 5.612e-6, 4.818e-6, 4.165e-6, 3.579e-6,  
02168 3.041e-6, 2.623e-6, 2.29e-6, 1.984e-6, 1.748e-6, 1.534e-6,  
02169 1.369e-6, 1.219e-6, 1.092e-6, 9.8e-7, 8.762e-7, 7.896e-7,  
02170 7.104e-7, 6.364e-7, 5.691e-7, 5.107e-7, 4.575e-7, 4.09e-7,  
02171 3.667e-7, 3.287e-7, 2.931e-7, 2.633e-7, 2.356e-7, 2.111e-7,  
02172 1.895e-7, 1.697e-7, 1.525e-7, 1.369e-7, 1.233e-7, 1.114e-7,  
02173 9.988e-8, 9.004e-8, 8.149e-8, 7.352e-8, 6.662e-8, 6.03e-8,  
02174 5.479e-8, 4.974e-8, 4.532e-8, 4.129e-8, 3.781e-8, 3.462e-8,  
02175 3.176e-8, 2.919e-8, 2.687e-8, 2.481e-8, 2.292e-8, 2.119e-8,  
02176 1.967e-8, 1.828e-8, 1.706e-8, 1.589e-8, 1.487e-8, 1.393e-8,  
02177 1.307e-8, 1.228e-8, 1.156e-8, 1.089e-8, 1.028e-8, 9.696e-9,  
02178 9.159e-9, 8.658e-9, 8.187e-9, 7.746e-9, 7.34e-9, 6.953e-9,  
02179 6.594e-9, 6.259e-9, 5.948e-9, 5.66e-9, 5.386e-9, 5.135e-9,  
02180 4.903e-9, 4.703e-9, 4.515e-9, 4.362e-9, 4.233e-9, 4.117e-9,  
02181 4.017e-9, 3.962e-9, 3.924e-9, 3.905e-9, 3.922e-9, 3.967e-9,  
02182 4.046e-9, 4.165e-9, 4.32e-9, 4.522e-9, 4.769e-9, 5.083e-9,  
02183 5.443e-9, 5.872e-9, 6.366e-9, 6.949e-9, 7.601e-9, 8.371e-9,  
02184 9.22e-9, 1.02e-8, 1.129e-8, 1.251e-8, 1.393e-8, 1.542e-8,  
02185 1.72e-8, 1.926e-8, 2.152e-8, 2.392e-8, 2.678e-8, 3.028e-8,  
02186 3.39e-8, 3.836e-8, 4.309e-8, 4.9e-8, 5.481e-8, 6.252e-8,  
02187 7.039e-8, 7.883e-8, 8.849e-8, 1.012e-7, 1.142e-7, 1.3e-7,  
02188 1.475e-7, 1.732e-7, 1.978e-7, 2.304e-7, 2.631e-7, 2.988e-7,  
02189 3.392e-7, 3.69e-7, 4.355e-7, 4.672e-7, 5.11e-7, 5.461e-7,  
02190 5.828e-7, 6.233e-7, 6.509e-7, 6.672e-7, 6.969e-7, 7.104e-7,  
02191 7.439e-7, 7.463e-7, 7.708e-7, 7.466e-7, 7.668e-7, 7.549e-7,  
02192 7.586e-7, 7.384e-7, 7.439e-7, 7.785e-7, 7.915e-7, 8.31e-7,  
02193 8.745e-7, 9.558e-7, 1.038e-6, 1.173e-6, 1.304e-6, 1.452e-6,  
02194 1.671e-6, 1.931e-6, 2.239e-6, 2.578e-6, 3.032e-6, 3.334e-6,  
02195 3.98e-6, 4.3e-6, 4.518e-6, 5.321e-6, 5.508e-6, 6.211e-6, 6.59e-6,  
02196 7.046e-6, 7.555e-6, 7.558e-6, 7.875e-6, 8.319e-6, 8.433e-6,  
02197 8.59e-6, 8.503e-6, 8.304e-6, 8.336e-6, 7.739e-6, 7.301e-6,  
02198 6.827e-6, 6.078e-6, 5.551e-6, 4.762e-6, 4.224e-6, 3.538e-6,  
02199 2.984e-6, 2.619e-6, 2.227e-6, 1.923e-6, 1.669e-6, 1.462e-6,  
02200 1.294e-6, 1.155e-6, 1.033e-6, 9.231e-7, 8.238e-7, 7.36e-7,  
02201 6.564e-7, 5.869e-7, 5.236e-7, 4.673e-7, 4.174e-7, 3.736e-7,  
02202 3.33e-7, 2.976e-7, 2.657e-7, 2.367e-7, 2.106e-7, 1.877e-7,  
02203 1.671e-7, 1.494e-7, 1.332e-7, 1.192e-7, 1.065e-7, 9.558e-8,  
02204 8.586e-8, 7.717e-8, 6.958e-8, 6.278e-8, 5.666e-8, 5.121e-8,  
02205 4.647e-8, 4.213e-8, 3.815e-8, 3.459e-8, 3.146e-8, 2.862e-8,  
02206 2.604e-8, 2.375e-8, 2.162e-8, 1.981e-8, 1.817e-8, 1.67e-8,  
02207 1.537e-8, 1.417e-8, 1.31e-8, 1.215e-8, 1.128e-8, 1.05e-8,  
02208 9.793e-9, 9.158e-9, 8.586e-9, 8.068e-9, 7.595e-9, 7.166e-9,  
02209 6.778e-9, 6.427e-9, 6.108e-9, 5.826e-9, 5.571e-9, 5.347e-9,  
02210 5.144e-9, 4.968e-9, 4.822e-9, 4.692e-9, 4.589e-9, 4.506e-9,  
02211 4.467e-9, 4.44e-9, 4.466e-9, 4.515e-9, 4.718e-9, 4.729e-9,

02212 4.937e-9, 5.249e-9, 5.466e-9, 5.713e-9, 6.03e-9, 6.436e-9,  
02213 6.741e-9, 7.33e-9, 7.787e-9, 8.414e-9, 8.908e-9, 9.868e-9,  
02214 1.069e-8, 1.158e-8, 1.253e-8, 1.3e-8, 1.409e-8, 1.47e-8,  
02215 1.548e-8, 1.612e-8, 1.666e-8, 1.736e-8, 1.763e-8, 1.812e-8,  
02216 1.852e-8, 1.923e-8, 1.897e-8, 1.893e-8, 1.888e-8, 1.868e-8,  
02217 1.895e-8, 1.899e-8, 1.876e-8, 1.96e-8, 2.02e-8, 2.121e-8,  
02218 2.239e-8, 2.379e-8, 2.526e-8, 2.766e-8, 2.994e-8, 3.332e-8,  
02219 3.703e-8, 4.158e-8, 4.774e-8, 5.499e-8, 6.355e-8, 7.349e-8,  
02220 8.414e-8, 9.846e-8, 1.143e-7, 1.307e-7, 1.562e-7, 1.817e-7,  
02221 2.011e-7, 2.192e-7, 2.485e-7, 2.867e-7, 3.035e-7, 3.223e-7,  
02222 3.443e-7, 3.617e-7, 3.793e-7, 3.793e-7, 3.839e-7, 4.081e-7,  
02223 4.117e-7, 4.085e-7, 3.92e-7, 3.851e-7, 3.754e-7, 3.49e-7,  
02224 3.229e-7, 2.978e-7, 2.691e-7, 2.312e-7, 2.029e-7, 1.721e-7,  
02225 1.472e-7, 1.308e-7, 1.132e-7, 9.736e-8, 8.458e-8, 7.402e-8,  
02226 6.534e-8, 5.811e-8, 5.235e-8, 4.762e-8, 4.293e-8, 3.896e-8,  
02227 3.526e-8, 3.165e-8, 2.833e-8, 2.551e-8, 2.288e-8, 2.036e-8,  
02228 1.82e-8, 1.626e-8, 1.438e-8, 1.299e-8, 1.149e-8, 1.03e-8,  
02229 9.148e-9, 8.122e-9, 7.264e-9, 6.425e-9, 5.777e-9, 5.06e-9,  
02230 4.502e-9, 4.013e-9, 3.567e-9, 3.145e-9, 2.864e-9, 2.553e-9,  
02231 2.311e-9, 2.087e-9, 1.886e-9, 1.716e-9, 1.556e-9, 1.432e-9,  
02232 1.311e-9, 1.202e-9, 1.104e-9, 1.013e-9, 9.293e-10, 8.493e-10,  
02233 7.79e-10, 7.185e-10, 6.642e-10, 6.141e-10, 5.684e-10, 5.346e-10,  
02234 5.032e-10, 4.725e-10, 4.439e-10, 4.176e-10, 3.93e-10, 3.714e-10,  
02235 3.515e-10, 3.332e-10, 3.167e-10, 3.02e-10, 2.887e-10, 2.769e-10,  
02236 2.665e-10, 2.578e-10, 2.503e-10, 2.436e-10, 2.377e-10, 2.342e-10,  
02237 2.305e-10, 2.296e-10, 2.278e-10, 2.321e-10, 2.355e-10, 2.402e-10,  
02238 2.478e-10, 2.67e-10, 2.848e-10, 2.982e-10, 3.263e-10, 3.438e-10,  
02239 3.649e-10, 3.829e-10, 4.115e-10, 4.264e-10, 4.473e-10, 4.63e-10,  
02240 4.808e-10, 4.995e-10, 5.142e-10, 5.313e-10, 5.318e-10, 5.358e-10,  
02241 5.452e-10, 5.507e-10, 5.698e-10, 5.782e-10, 5.983e-10, 6.164e-10,  
02242 6.532e-10, 6.811e-10, 7.624e-10, 8.302e-10, 9.067e-10, 9.937e-10,  
02243 1.104e-9, 1.221e-9, 1.361e-9, 1.516e-9, 1.675e-9, 1.883e-9,  
02244 2.101e-9, 2.349e-9, 2.614e-9, 2.92e-9, 3.305e-9, 3.724e-9,  
02245 4.142e-9, 4.887e-9, 5.614e-9, 6.506e-9, 7.463e-9, 8.817e-9,  
02246 9.849e-9, 1.187e-8, 1.321e-8, 1.474e-8, 1.698e-8, 1.794e-8,  
02247 2.09e-8, 2.211e-8, 2.362e-8, 2.556e-8, 2.729e-8, 2.88e-8,  
02248 3.046e-8, 3.167e-8, 3.367e-8, 3.457e-8, 3.59e-8, 3.711e-8,  
02249 3.826e-8, 4.001e-8, 4.211e-8, 4.315e-8, 4.661e-8, 5.01e-8,  
02250 5.249e-8, 5.84e-8, 6.628e-8, 7.512e-8, 8.253e-8, 9.722e-8,  
02251 1.067e-7, 1.153e-7, 1.347e-7, 1.428e-7, 1.577e-7, 1.694e-7,  
02252 1.833e-7, 1.938e-7, 2.108e-7, 2.059e-7, 2.157e-7, 2.185e-7,  
02253 2.208e-7, 2.182e-7, 2.093e-7, 2.014e-7, 1.962e-7, 1.819e-7,  
02254 1.713e-7, 1.51e-7, 1.34e-7, 1.154e-7, 9.89e-8, 8.88e-8, 7.673e-8,  
02255 6.599e-8, 5.73e-8, 5.081e-8, 4.567e-8, 4.147e-8, 3.773e-8,  
02256 3.46e-8, 3.194e-8, 2.953e-8, 2.759e-8, 2.594e-8, 2.442e-8,  
02257 2.355e-8, 2.283e-8, 2.279e-8, 2.231e-8, 2.279e-8, 2.239e-8,  
02258 2.21e-8, 2.309e-8, 2.293e-8, 2.352e-8, 2.415e-8, 2.43e-8,  
02259 2.426e-8, 2.465e-8, 2.5e-8, 2.496e-8, 2.465e-8, 2.445e-8,  
02260 2.383e-8, 2.299e-8, 2.165e-8, 2.113e-8, 1.968e-8, 1.819e-8,  
02261 1.644e-8, 1.427e-8, 1.27e-8, 1.082e-8, 9.428e-9, 8.091e-9,  
02262 6.958e-9, 5.988e-9, 5.246e-9, 4.601e-9, 4.098e-9, 3.664e-9,  
02263 3.287e-9, 2.942e-9, 2.656e-9, 2.364e-9, 2.118e-9, 1.903e-9,  
02264 1.703e-9, 1.525e-9, 1.365e-9, 1.229e-9, 1.107e-9, 9.96e-10,  
02265 8.945e-10, 8.08e-10, 7.308e-10, 6.616e-10, 5.994e-10, 5.422e-10,  
02266 4.929e-10, 4.478e-10, 4.07e-10, 3.707e-10, 3.379e-10, 3.087e-10,  
02267 2.823e-10, 2.592e-10, 2.385e-10, 2.201e-10, 2.038e-10, 1.897e-10,  
02268 1.774e-10, 1.667e-10, 1.577e-10, 1.502e-10, 1.437e-10, 1.394e-10,  
02269 1.358e-10, 1.324e-10, 1.329e-10, 1.324e-10, 1.36e-10, 1.39e-10,  
02270 1.424e-10, 1.544e-10, 1.651e-10, 1.817e-10, 1.984e-10, 2.195e-10,  
02271 2.438e-10, 2.7e-10, 2.991e-10, 3.322e-10, 3.632e-10, 3.957e-10,  
02272 4.36e-10, 4.701e-10, 5.03e-10, 5.381e-10, 5.793e-10, 6.19e-10,  
02273 6.596e-10, 7.004e-10, 7.561e-10, 7.934e-10, 8.552e-10, 9.142e-10,  
02274 9.57e-10, 1.027e-9, 1.097e-9, 1.193e-9, 1.334e-9, 1.47e-9,  
02275 1.636e-9, 1.871e-9, 2.122e-9, 2.519e-9, 2.806e-9, 3.203e-9,  
02276 3.846e-9, 4.362e-9, 5.114e-9, 5.643e-9, 6.305e-9, 6.981e-9,  
02277 7.983e-9, 8.783e-9, 9.419e-9, 1.017e-8, 1.063e-8, 1.121e-8,  
02278 1.13e-8, 1.201e-8, 1.225e-8, 1.232e-8, 1.223e-8, 1.177e-8,  
02279 1.151e-8, 1.116e-8, 1.047e-8, 9.698e-9, 8.734e-9, 8.202e-9,  
02280 7.041e-9, 6.074e-9, 5.172e-9, 4.468e-9, 3.913e-9, 3.414e-9,  
02281 2.975e-9, 2.65e-9, 2.406e-9, 2.173e-9, 2.009e-9, 1.861e-9,  
02282 1.727e-9, 1.612e-9, 1.514e-9, 1.43e-9, 1.362e-9, 1.333e-9,  
02283 1.288e-9, 1.249e-9, 1.238e-9, 1.228e-9, 1.217e-9, 1.202e-9,  
02284 1.209e-9, 1.177e-9, 1.157e-9, 1.165e-9, 1.142e-9, 1.131e-9,  
02285 1.138e-9, 1.117e-9, 1.1e-9, 1.069e-9, 1.023e-9, 1.005e-9,  
02286 9.159e-10, 8.863e-10, 7.865e-10, 7.153e-10, 6.247e-10, 5.846e-10,  
02287 5.133e-10, 4.36e-10, 3.789e-10, 3.335e-10, 2.833e-10, 2.483e-10,  
02288 2.155e-10, 1.918e-10, 1.709e-10, 1.529e-10, 1.374e-10, 1.235e-10,  
02289 1.108e-10, 9.933e-11, 8.932e-11, 8.022e-11, 7.224e-11, 6.52e-11,  
02290 5.896e-11, 5.328e-11, 4.813e-11, 4.365e-11, 3.961e-11, 3.594e-11,  
02291 3.266e-11, 2.967e-11, 2.701e-11, 2.464e-11, 2.248e-11, 2.054e-11,  
02292 1.878e-11, 1.721e-11, 1.579e-11, 1.453e-11, 1.341e-11, 1.241e-11,  
02293 1.154e-11, 1.078e-11, 1.014e-11, 9.601e-12, 9.167e-12, 8.838e-12,  
02294 8.614e-12, 8.493e-12, 8.481e-12, 8.581e-12, 8.795e-12, 9.131e-12,  
02295 9.601e-12, 1.021e-11, 1.097e-11, 1.191e-11, 1.303e-11, 1.439e-11,  
02296 1.601e-11, 1.778e-11, 1.984e-11, 2.234e-11, 2.474e-11, 2.766e-11,  
02297 3.085e-11, 3.415e-11, 3.821e-11, 4.261e-11, 4.748e-11, 5.323e-11,  
02298 5.935e-11, 6.619e-11, 7.418e-11, 8.294e-11, 9.26e-11, 1.039e-10,

02299	1.156e-10, 1.297e-10, 1.46e-10, 1.641e-10, 1.858e-10, 2.1e-10,
02300	2.383e-10, 2.724e-10, 3.116e-10, 3.538e-10, 4.173e-10, 4.727e-10,
02301	5.503e-10, 6.337e-10, 7.32e-10, 8.298e-10, 9.328e-10, 1.059e-9,
02302	1.176e-9, 1.328e-9, 1.445e-9, 1.593e-9, 1.77e-9, 1.954e-9,
02303	2.175e-9, 2.405e-9, 2.622e-9, 2.906e-9, 3.294e-9, 3.713e-9,
02304	3.98e-9, 4.384e-9, 4.987e-9, 5.311e-9, 5.874e-9, 6.337e-9,
02305	7.027e-9, 7.39e-9, 7.769e-9, 8.374e-9, 8.605e-9, 9.165e-9,
02306	9.415e-9, 9.511e-9, 9.704e-9, 9.588e-9, 9.45e-9, 9.086e-9,
02307	8.798e-9, 8.469e-9, 7.697e-9, 7.168e-9, 6.255e-9, 5.772e-9,
02308	4.97e-9, 4.271e-9, 3.653e-9, 3.154e-9, 2.742e-9, 2.435e-9,
02309	2.166e-9, 1.936e-9, 1.731e-9, 1.556e-9, 1.399e-9, 1.272e-9,
02310	1.157e-9, 1.066e-9, 9.844e-10, 9.258e-10, 8.787e-10, 8.421e-10,
02311	8.083e-10, 8.046e-10, 8.067e-10, 8.181e-10, 8.325e-10, 8.517e-10,
02312	9.151e-10, 9.351e-10, 9.677e-10, 1.071e-9, 1.126e-9, 1.219e-9,
02313	1.297e-9, 1.408e-9, 1.476e-9, 1.517e-9, 1.6e-9, 1.649e-9,
02314	1.678e-9, 1.746e-9, 1.742e-9, 1.728e-9, 1.699e-9, 1.655e-9,
02315	1.561e-9, 1.48e-9, 1.451e-9, 1.411e-9, 1.171e-9, 1.106e-9,
02316	9.714e-10, 8.523e-10, 7.346e-10, 6.241e-10, 5.371e-10, 4.704e-10,
02317	4.144e-10, 3.683e-10, 3.292e-10, 2.942e-10, 2.62e-10, 2.341e-10,
02318	2.104e-10, 1.884e-10, 1.7e-10, 1.546e-10, 1.394e-10, 1.265e-10,
02319	1.14e-10, 1.019e-10, 9.279e-11, 8.283e-11, 7.458e-11, 6.668e-11,
02320	5.976e-11, 5.33e-11, 4.794e-11, 4.289e-11, 3.841e-11, 3.467e-11,
02321	3.13e-11, 2.832e-11, 2.582e-11, 2.356e-11, 2.152e-11, 1.97e-11,
02322	1.808e-11, 1.664e-11, 1.539e-11, 1.434e-11, 1.344e-11, 1.269e-11,
02323	1.209e-11, 1.162e-11, 1.129e-11, 1.108e-11, 1.099e-11, 1.103e-11,
02324	1.119e-11, 1.148e-11, 1.193e-11, 1.252e-11, 1.329e-11, 1.421e-11,
02325	1.555e-11, 1.685e-11, 1.839e-11, 2.054e-11, 2.317e-11, 2.571e-11,
02326	2.839e-11, 3.171e-11, 3.49e-11, 3.886e-11, 4.287e-11, 4.645e-11,
02327	5.047e-11, 5.592e-11, 6.109e-11, 6.628e-11, 7.381e-11, 8.088e-11,
02328	8.966e-11, 1.045e-10, 1.12e-10, 1.287e-10, 1.486e-10, 1.662e-10,
02329	1.866e-10, 2.133e-10, 2.524e-10, 2.776e-10, 3.204e-10, 3.559e-10,
02330	4.028e-10, 4.448e-10, 4.882e-10, 5.244e-10, 5.605e-10, 6.018e-10,
02331	6.328e-10, 6.579e-10, 6.541e-10, 7.024e-10, 7.074e-10, 7.068e-10,
02332	7.009e-10, 6.698e-10, 6.545e-10, 6.209e-10, 5.834e-10, 5.412e-10,
02333	5.001e-10, 4.231e-10, 3.727e-10, 3.211e-10, 2.833e-10, 2.447e-10,
02334	2.097e-10, 1.843e-10, 1.639e-10, 1.449e-10, 1.27e-10, 1.161e-10,
02335	1.033e-10, 9.282e-11, 8.407e-11, 7.639e-11, 7.023e-11, 6.474e-11,
02336	6.142e-11, 5.76e-11, 5.568e-11, 5.472e-11, 5.39e-11, 5.455e-11,
02337	5.54e-11, 5.587e-11, 6.23e-11, 6.49e-11, 6.868e-11, 7.382e-11,
02338	8.022e-11, 8.372e-11, 9.243e-11, 1.004e-10, 1.062e-10, 1.13e-10,
02339	1.176e-10, 1.244e-10, 1.279e-10, 1.298e-10, 1.302e-10, 1.312e-10,
02340	1.295e-10, 1.244e-10, 1.211e-10, 1.167e-10, 1.098e-10, 9.927e-11,
02341	8.854e-11, 8.011e-11, 7.182e-11, 5.923e-11, 5.212e-11, 4.453e-11,
02342	3.832e-11, 3.371e-11, 2.987e-11, 2.651e-11, 2.354e-11, 2.093e-11,
02343	1.863e-11, 1.662e-11, 1.486e-11, 1.331e-11, 1.193e-11, 1.071e-11,
02344	9.628e-12, 8.66e-12, 7.801e-12, 7.031e-12, 6.347e-12, 5.733e-12,
02345	5.182e-12, 4.695e-12, 4.26e-12, 3.874e-12, 3.533e-12, 3.235e-12,
02346	2.979e-12, 2.76e-12, 2.579e-12, 2.432e-12, 2.321e-12, 2.246e-12,
02347	2.205e-12, 2.196e-12, 2.223e-12, 2.288e-12, 2.387e-12, 2.525e-12,
02348	2.704e-12, 2.925e-12, 3.191e-12, 3.508e-12, 3.876e-12, 4.303e-12,
02349	4.793e-12, 5.347e-12, 5.978e-12, 6.682e-12, 7.467e-12, 8.34e-12,
02350	9.293e-12, 1.035e-11, 1.152e-11, 1.285e-11, 1.428e-11, 1.586e-11,
02351	1.764e-11, 1.972e-11, 2.214e-11, 2.478e-11, 2.776e-11, 3.151e-11,
02352	3.591e-11, 4.103e-11, 4.66e-11, 5.395e-11, 6.306e-11, 7.172e-11,
02353	8.358e-11, 9.67e-11, 1.11e-10, 1.325e-10, 1.494e-10, 1.736e-10,
02354	2.007e-10, 2.296e-10, 2.608e-10, 3.004e-10, 3.361e-10, 3.727e-10,
02355	4.373e-10, 4.838e-10, 5.483e-10, 6.006e-10, 6.535e-10, 6.899e-10,
02356	7.687e-10, 8.444e-10, 8.798e-10, 9.135e-10, 9.532e-10, 9.757e-10,
02357	9.968e-10, 1.006e-9, 9.949e-10, 9.789e-10, 9.564e-10, 9.215e-10,
02358	8.51e-10, 8.394e-10, 7.707e-10, 7.152e-10, 6.274e-10, 5.598e-10,
02359	5.028e-10, 4.3e-10, 3.71e-10, 3.245e-10, 2.809e-10, 2.461e-10,
02360	2.154e-10, 1.91e-10, 1.685e-10, 1.487e-10, 1.313e-10, 1.163e-10,
02361	1.031e-10, 9.172e-11, 8.221e-11, 7.382e-11, 6.693e-11, 6.079e-11,
02362	5.581e-11, 5.167e-11, 4.811e-11, 4.506e-11, 4.255e-11, 4.083e-11,
02363	3.949e-11, 3.881e-11, 3.861e-11, 3.858e-11, 3.951e-11, 4.045e-11,
02364	4.24e-11, 4.487e-11, 4.806e-11, 5.133e-11, 5.518e-11, 5.919e-11,
02365	6.533e-11, 7.031e-11, 7.762e-11, 8.305e-11, 9.252e-11, 9.727e-11,
02366	1.045e-10, 1.117e-10, 1.2e-10, 1.275e-10, 1.341e-10, 1.362e-10,
02367	1.438e-10, 1.45e-10, 1.455e-10, 1.455e-10, 1.434e-10, 1.381e-10,
02368	1.301e-10, 1.276e-10, 1.163e-10, 1.089e-10, 9.911e-11, 8.943e-11,
02369	7.618e-11, 6.424e-11, 5.717e-11, 4.866e-11, 4.257e-11, 3.773e-11,
02370	3.331e-11, 2.958e-11, 2.629e-11, 2.316e-11, 2.073e-11, 1.841e-11,
02371	1.635e-11, 1.464e-11, 1.31e-11, 1.16e-11, 1.047e-11, 9.408e-12,
02372	8.414e-12, 7.521e-12, 6.705e-12, 5.993e-12, 5.371e-12, 4.815e-12,
02373	4.338e-12, 3.921e-12, 3.567e-12, 3.265e-12, 3.01e-12, 2.795e-12,
02374	2.613e-12, 2.464e-12, 2.346e-12, 2.256e-12, 2.195e-12, 2.165e-12,
02375	2.166e-12, 2.198e-12, 2.262e-12, 2.364e-12, 2.502e-12, 2.682e-12,
02376	2.908e-12, 3.187e-12, 3.533e-12, 3.946e-12, 4.418e-12, 5.013e-12,
02377	5.708e-12, 6.379e-12, 7.43e-12, 8.39e-12, 9.51e-12, 1.078e-11,
02378	1.259e-11, 1.438e-11, 1.63e-11, 1.814e-11, 2.055e-11, 2.348e-11,
02379	2.664e-11, 2.956e-11, 3.3e-11, 3.677e-11, 4.032e-11, 4.494e-11,
02380	4.951e-11, 5.452e-11, 6.014e-11, 6.5e-11, 6.915e-11, 7.45e-11,
02381	7.971e-11, 8.468e-11, 8.726e-11, 8.995e-11, 9.182e-11, 9.509e-11,
02382	9.333e-11, 9.386e-11, 9.457e-11, 9.21e-11, 9.019e-11, 8.68e-11,
02383	8.298e-11, 7.947e-11, 7.46e-11, 7.082e-11, 6.132e-11, 5.855e-11,
02384	5.073e-11, 4.464e-11, 3.825e-11, 3.375e-11, 2.911e-11, 2.535e-11,
02385	2.16e-11, 1.907e-11, 1.665e-11, 1.463e-11, 1.291e-11, 1.133e-11,

```

02386     9.997e-12, 8.836e-12, 7.839e-12, 6.943e-12, 6.254e-12, 5.6e-12,
02387     5.029e-12, 4.529e-12, 4.102e-12, 3.737e-12, 3.428e-12, 3.169e-12,
02388     2.959e-12, 2.798e-12, 2.675e-12, 2.582e-12, 2.644e-12, 2.557e-12,
02389     2.614e-12, 2.717e-12, 2.874e-12, 3.056e-12, 3.187e-12, 3.631e-12,
02390     3.979e-12, 4.248e-12, 4.817e-12, 5.266e-12, 5.836e-12, 6.365e-12,
02391     6.807e-12, 7.47e-12, 7.951e-12, 8.636e-12, 8.972e-12, 9.314e-12,
02392     9.445e-12, 1.003e-11, 1.013e-11, 9.937e-12, 9.729e-12, 9.064e-12,
02393     9.119e-12, 9.124e-12, 8.704e-12, 8.078e-12, 7.47e-12, 6.329e-12,
02394     5.674e-12, 4.808e-12, 4.119e-12, 3.554e-12, 3.103e-12, 2.731e-12,
02395     2.415e-12, 2.15e-12, 1.926e-12, 1.737e-12, 1.578e-12, 1.447e-12,
02396     1.34e-12, 1.255e-12, 1.191e-12, 1.146e-12, 1.121e-12, 1.114e-12,
02397     1.126e-12, 1.156e-12, 1.207e-12, 1.278e-12, 1.372e-12, 1.49e-12,
02398     1.633e-12, 1.805e-12, 2.01e-12, 2.249e-12, 2.528e-12, 2.852e-12,
02399     3.228e-12, 3.658e-12, 4.153e-12, 4.728e-12, 5.394e-12, 6.176e-12,
02400     7.126e-12, 8.188e-12, 9.328e-12, 1.103e-11, 1.276e-11, 1.417e-11,
02401     1.615e-11, 1.84e-11, 2.155e-11, 2.429e-11, 2.826e-11, 3.222e-11,
02402     3.664e-11, 4.14e-11, 4.906e-11, 5.536e-11, 6.327e-11, 7.088e-11,
02403     8.316e-11, 9.242e-11, 1.07e-10, 1.223e-10, 1.341e-10, 1.553e-10,
02404     1.703e-10, 1.9e-10, 2.022e-10, 2.233e-10, 2.345e-10, 2.438e-10,
02405     2.546e-10, 2.599e-10, 2.661e-10, 2.703e-10, 2.686e-10, 2.662e-10,
02406     2.56e-10, 2.552e-10, 2.378e-10, 2.252e-10, 2.146e-10, 1.885e-10,
02407     1.668e-10, 1.441e-10, 1.295e-10, 1.119e-10, 9.893e-11, 8.687e-11,
02408     7.678e-11, 6.685e-11, 5.879e-11, 5.127e-11, 4.505e-11, 3.997e-11,
02409     3.511e-11
02410 };
02411
02412 static double h2ofrn[2001] = { .01095, .01126, .01205, .01322, .0143,
02413     .01506, .01548, .01534, .01486, .01373, .01262, .01134, .01001,
02414     .008702, .007475, .006481, .00548, .0046, .003833, .00311,
02415     .002543, .002049, .00168, .001374, .001046, 8.193e-4, 6.267e-4,
02416     4.968e-4, 3.924e-4, 2.983e-4, 2.477e-4, 1.997e-4, 1.596e-4,
02417     1.331e-4, 1.061e-4, 8.942e-5, 7.168e-5, 5.887e-5, 4.848e-5,
02418     3.817e-5, 3.17e-5, 2.579e-5, 2.162e-5, 1.768e-5, 1.49e-5,
02419     1.231e-5, 1.013e-5, 8.555e-6, 7.328e-6, 6.148e-6, 5.207e-6,
02420     4.387e-6, 3.741e-6, 3.22e-6, 2.753e-6, 2.346e-6, 1.985e-6,
02421     1.716e-6, 1.475e-6, 1.286e-6, 1.122e-6, 9.661e-7, 8.284e-7,
02422     7.057e-7, 6.119e-7, 5.29e-7, 4.571e-7, 3.948e-7, 3.432e-7,
02423     2.983e-7, 2.589e-7, 2.265e-7, 1.976e-7, 1.704e-7, 1.456e-7,
02424     1.26e-7, 1.101e-7, 9.648e-8, 8.415e-8, 7.34e-8, 6.441e-8,
02425     5.643e-8, 4.94e-8, 4.276e-8, 3.703e-8, 3.227e-8, 2.825e-8,
02426     2.478e-8, 2.174e-8, 1.898e-8, 1.664e-8, 1.458e-8, 1.278e-8,
02427     1.126e-8, 9.891e-9, 8.709e-9, 7.652e-9, 6.759e-9, 5.975e-9,
02428     5.31e-9, 4.728e-9, 4.214e-9, 3.792e-9, 3.463e-9, 3.226e-9,
02429     2.992e-9, 2.813e-9, 2.749e-9, 2.809e-9, 2.913e-9, 3.037e-9,
02430     3.413e-9, 3.773e-9, 4.189e-9, 4.808e-9, 5.978e-9, 7.088e-9,
02431     8.071e-9, 9.61e-9, 1.21e-8, 1.5e-8, 1.764e-8, 2.221e-8, 2.898e-8,
02432     3.948e-8, 5.068e-8, 6.227e-8, 7.898e-8, 1.033e-7, 1.437e-7,
02433     1.889e-7, 2.698e-7, 3.59e-7, 4.971e-7, 7.156e-7, 9.983e-7,
02434     1.381e-6, 1.929e-6, 2.591e-6, 3.453e-6, 4.57e-6, 5.93e-6,
02435     7.552e-6, 9.556e-6, 1.183e-5, 1.425e-5, 1.681e-5, 1.978e-5,
02436     2.335e-5, 2.668e-5, 3.022e-5, 3.371e-5, 3.715e-5, 3.967e-5,
02437     4.06e-5, 4.01e-5, 3.809e-5, 3.491e-5, 3.155e-5, 2.848e-5,
02438     2.678e-5, 2.66e-5, 2.811e-5, 3.071e-5, 3.294e-5, 3.459e-5,
02439     3.569e-5, 3.56e-5, 3.434e-5, 3.186e-5, 2.916e-5, 2.622e-5,
02440     2.275e-5, 1.918e-5, 1.62e-5, 1.373e-5, 1.182e-5, 1.006e-5,
02441     8.556e-6, 7.26e-6, 6.107e-6, 5.034e-6, 4.211e-6, 3.426e-6,
02442     2.865e-6, 2.446e-6, 1.998e-6, 1.628e-6, 1.242e-6, 1.005e-6,
02443     7.853e-7, 6.21e-7, 5.071e-7, 4.156e-7, 3.548e-7, 2.825e-7,
02444     2.261e-7, 1.916e-7, 1.51e-7, 1.279e-7, 1.059e-7, 9.14e-8,
02445     7.707e-8, 6.17e-8, 5.311e-8, 4.263e-8, 3.518e-8, 2.961e-8,
02446     2.457e-8, 2.119e-8, 1.712e-8, 1.439e-8, 1.201e-8, 1.003e-8,
02447     8.564e-9, 7.199e-9, 6.184e-9, 5.206e-9, 4.376e-9, 3.708e-9,
02448     3.175e-9, 2.725e-9, 2.361e-9, 2.074e-9, 1.797e-9, 1.562e-9,
02449     1.364e-9, 1.196e-9, 1.042e-9, 8.862e-10, 7.648e-10, 6.544e-10,
02450     5.609e-10, 4.791e-10, 4.108e-10, 3.531e-10, 3.038e-10, 2.618e-10,
02451     2.268e-10, 1.969e-10, 1.715e-10, 1.496e-10, 1.308e-10, 1.147e-10,
02452     1.008e-10, 8.894e-11, 7.885e-11, 7.031e-11, 6.355e-11, 5.854e-11,
02453     5.534e-11, 5.466e-11, 5.725e-11, 6.447e-11, 7.943e-11, 1.038e-10,
02454     1.437e-10, 2.04e-10, 2.901e-10, 4.051e-10, 5.556e-10, 7.314e-10,
02455     9.291e-10, 1.134e-9, 1.321e-9, 1.482e-9, 1.596e-9, 1.669e-9,
02456     1.715e-9, 1.762e-9, 1.817e-9, 1.828e-9, 1.848e-9, 1.873e-9,
02457     1.902e-9, 1.894e-9, 1.864e-9, 1.841e-9, 1.797e-9, 1.704e-9,
02458     1.559e-9, 1.382e-9, 1.187e-9, 1.001e-9, 8.468e-10, 7.265e-10,
02459     6.521e-10, 6.381e-10, 6.66e-10, 7.637e-10, 9.705e-10, 1.368e-9,
02460     1.856e-9, 2.656e-9, 3.954e-9, 5.96e-9, 8.72e-9, 1.247e-8,
02461     1.781e-8, 2.491e-8, 3.311e-8, 4.272e-8, 5.205e-8, 6.268e-8,
02462     7.337e-8, 8.277e-8, 9.185e-8, 1.004e-7, 1.091e-7, 1.159e-7,
02463     1.188e-7, 1.175e-7, 1.124e-7, 1.033e-7, 9.381e-8, 8.501e-8,
02464     7.956e-8, 7.894e-8, 8.331e-8, 9.102e-8, 9.836e-8, 1.035e-7,
02465     1.064e-7, 1.06e-7, 1.032e-7, 9.808e-8, 9.139e-8, 8.442e-8,
02466     7.641e-8, 6.881e-8, 6.161e-8, 5.404e-8, 4.804e-8, 4.446e-8,
02467     4.328e-8, 4.259e-8, 4.421e-8, 4.673e-8, 4.985e-8, 5.335e-8,
02468     5.796e-8, 6.542e-8, 7.714e-8, 8.827e-8, 1.04e-7, 1.238e-7,
02469     1.499e-7, 1.829e-7, 2.222e-7, 2.689e-7, 3.303e-7, 3.981e-7,
02470     4.84e-7, 5.91e-7, 7.363e-7, 9.087e-7, 1.139e-6, 1.455e-6,
02471     1.866e-6, 2.44e-6, 3.115e-6, 3.941e-6, 4.891e-6, 5.992e-6,
02472     7.111e-6, 8.296e-6, 9.21e-6, 9.987e-6, 1.044e-5, 1.073e-5,

```



02473 1.092e-5, 1.106e-5, 1.138e-5, 1.171e-5, 1.186e-5, 1.186e-5,  
02474 1.179e-5, 1.166e-5, 1.151e-5, 1.16e-5, 1.197e-5, 1.241e-5,  
02475 1.268e-5, 1.26e-5, 1.184e-5, 1.063e-5, 9.204e-6, 7.584e-6,  
02476 6.053e-6, 4.482e-6, 3.252e-6, 2.337e-6, 1.662e-6, 1.18e-6,  
02477 8.15e-7, 5.95e-7, 4.354e-7, 3.302e-7, 2.494e-7, 1.93e-7,  
02478 1.545e-7, 1.25e-7, 1.039e-7, 8.602e-8, 7.127e-8, 5.897e-8,  
02479 4.838e-8, 4.018e-8, 3.28e-8, 2.72e-8, 2.307e-8, 1.972e-8,  
02480 1.654e-8, 1.421e-8, 1.174e-8, 1.004e-8, 8.739e-9, 7.358e-9,  
02481 6.242e-9, 5.303e-9, 4.567e-9, 3.94e-9, 3.375e-9, 2.864e-9,  
02482 2.422e-9, 2.057e-9, 1.75e-9, 1.505e-9, 1.294e-9, 1.101e-9,  
02483 9.401e-10, 8.018e-10, 6.903e-10, 5.965e-10, 5.087e-10, 4.364e-10,  
02484 3.759e-10, 3.247e-10, 2.809e-10, 2.438e-10, 2.123e-10, 1.853e-10,  
02485 1.622e-10, 1.426e-10, 1.26e-10, 1.125e-10, 1.022e-10, 9.582e-11,  
02486 9.388e-11, 9.801e-11, 1.08e-10, 1.276e-10, 1.551e-10, 1.903e-10,  
02487 2.291e-10, 2.724e-10, 3.117e-10, 3.4e-10, 3.562e-10, 3.625e-10,  
02488 3.619e-10, 3.429e-10, 3.221e-10, 2.943e-10, 2.645e-10, 2.338e-10,  
02489 2.062e-10, 1.901e-10, 1.814e-10, 1.827e-10, 1.906e-10, 1.984e-10,  
02490 2.04e-10, 2.068e-10, 2.075e-10, 2.018e-10, 1.959e-10, 1.897e-10,  
02491 1.852e-10, 1.791e-10, 1.696e-10, 1.634e-10, 1.598e-10, 1.561e-10,  
02492 1.518e-10, 1.443e-10, 1.377e-10, 1.346e-10, 1.342e-10, 1.375e-10,  
02493 1.525e-10, 1.767e-10, 2.108e-10, 2.524e-10, 2.981e-10, 3.477e-10,  
02494 4.262e-10, 5.326e-10, 6.646e-10, 8.321e-10, 1.069e-9, 1.386e-9,  
02495 1.743e-9, 2.216e-9, 2.808e-9, 3.585e-9, 4.552e-9, 5.907e-9,  
02496 7.611e-9, 9.774e-9, 1.255e-8, 1.666e-8, 2.279e-8, 3.221e-8,  
02497 4.531e-8, 6.4e-8, 9.187e-8, 1.295e-7, 1.825e-7, 2.431e-7,  
02498 3.181e-7, 4.009e-7, 4.941e-7, 5.88e-7, 6.623e-7, 7.155e-7,  
02499 7.451e-7, 7.594e-7, 7.541e-7, 7.467e-7, 7.527e-7, 7.935e-7,  
02500 8.461e-7, 8.954e-7, 9.364e-7, 9.843e-7, 1.024e-6, 1.05e-6,  
02501 1.059e-6, 1.074e-6, 1.072e-6, 1.043e-6, 9.789e-7, 8.803e-7,  
02502 7.662e-7, 7.678e-7, 5.133e-7, 3.958e-7, 2.914e-7, 2.144e-7,  
02503 1.57e-7, 1.14e-7, 8.47e-8, 6.2e-8, 4.657e-8, 3.559e-8, 2.813e-8,  
02504 2.222e-8, 1.769e-8, 1.391e-8, 1.125e-8, 9.186e-9, 7.704e-9,  
02505 6.447e-9, 5.381e-9, 4.442e-9, 3.669e-9, 3.057e-9, 2.564e-9,  
02506 2.153e-9, 1.784e-9, 1.499e-9, 1.281e-9, 1.082e-9, 9.304e-10,  
02507 8.169e-10, 6.856e-10, 5.866e-10, 5.043e-10, 4.336e-10, 3.731e-10,  
02508 3.175e-10, 2.75e-10, 2.374e-10, 2.007e-10, 1.737e-10, 1.508e-10,  
02509 1.302e-10, 1.13e-10, 9.672e-11, 8.375e-11, 7.265e-11, 6.244e-11,  
02510 5.343e-11, 4.654e-11, 3.975e-11, 3.488e-11, 3.097e-11, 2.834e-11,  
02511 2.649e-11, 2.519e-11, 2.462e-11, 2.443e-11, 2.44e-11, 2.398e-11,  
02512 2.306e-11, 2.183e-11, 2.021e-11, 1.821e-11, 1.599e-11, 1.403e-11,  
02513 1.196e-11, 1.023e-11, 8.728e-12, 7.606e-12, 6.941e-12, 6.545e-12,  
02514 6.484e-12, 6.6e-12, 6.718e-12, 6.785e-12, 6.746e-12, 6.724e-12,  
02515 6.764e-12, 6.995e-12, 7.144e-12, 7.32e-12, 7.33e-12, 7.208e-12,  
02516 6.789e-12, 6.09e-12, 5.337e-12, 4.62e-12, 4.037e-12, 3.574e-12,  
02517 3.311e-12, 3.346e-12, 3.566e-12, 3.836e-12, 4.076e-12, 4.351e-12,  
02518 4.691e-12, 5.114e-12, 5.427e-12, 6.167e-12, 7.436e-12, 8.842e-12,  
02519 1.038e-11, 1.249e-11, 1.54e-11, 1.915e-11, 2.48e-11, 3.256e-11,  
02520 4.339e-11, 5.611e-11, 7.519e-11, 1.037e-10, 1.409e-10, 1.883e-10,  
02521 2.503e-10, 3.38e-10, 4.468e-10, 5.801e-10, 7.335e-10, 8.98e-10,  
02522 1.11e-9, 1.363e-9, 1.677e-9, 2.104e-9, 2.681e-9, 3.531e-9,  
02523 4.621e-9, 6.106e-9, 8.154e-9, 1.046e-8, 1.312e-8, 1.607e-8,  
02524 1.948e-8, 2.266e-8, 2.495e-8, 2.655e-8, 2.739e-8, 2.739e-8,  
02525 2.662e-8, 2.589e-8, 2.59e-8, 2.664e-8, 2.833e-8, 3.023e-8,  
02526 3.305e-8, 3.558e-8, 3.793e-8, 3.961e-8, 4.056e-8, 4.102e-8,  
02527 4.025e-8, 3.917e-8, 3.706e-8, 3.493e-8, 3.249e-8, 3.096e-8,  
02528 3.011e-8, 3.111e-8, 3.395e-8, 3.958e-8, 4.875e-8, 6.066e-8,  
02529 7.915e-8, 1.011e-7, 1.3e-7, 1.622e-7, 2.003e-7, 2.448e-7,  
02530 2.863e-7, 3.317e-7, 3.655e-7, 3.96e-7, 4.098e-7, 4.168e-7,  
02531 4.198e-7, 4.207e-7, 4.289e-7, 4.384e-7, 4.471e-7, 4.524e-7,  
02532 4.574e-7, 4.633e-7, 4.785e-7, 5.028e-7, 5.371e-7, 5.727e-7,  
02533 5.955e-7, 5.998e-7, 5.669e-7, 5.082e-7, 4.397e-7, 3.596e-7,  
02534 2.814e-7, 2.074e-7, 1.486e-7, 1.057e-7, 7.25e-8, 4.946e-8,  
02535 3.43e-8, 2.447e-8, 1.793e-8, 1.375e-8, 1.096e-8, 9.091e-9,  
02536 7.709e-9, 6.631e-9, 5.714e-9, 4.886e-9, 4.205e-9, 3.575e-9,  
02537 3.07e-9, 2.631e-9, 2.284e-9, 2.002e-9, 1.745e-9, 1.509e-9,  
02538 1.284e-9, 1.084e-9, 9.163e-10, 7.663e-10, 6.346e-10, 5.283e-10,  
02539 4.354e-10, 3.59e-10, 2.982e-10, 2.455e-10, 2.033e-10, 1.696e-10,  
02540 1.432e-10, 1.211e-10, 1.02e-10, 8.702e-11, 7.38e-11, 6.293e-11,  
02541 5.343e-11, 4.532e-11, 3.907e-11, 3.365e-11, 2.945e-11, 2.558e-11,  
02542 2.192e-11, 1.895e-11, 1.636e-11, 1.42e-11, 1.228e-11, 1.063e-11,  
02543 9.348e-12, 8.2e-12, 7.231e-12, 6.43e-12, 5.702e-12, 5.052e-12,  
02544 4.469e-12, 4e-12, 3.679e-12, 3.387e-12, 3.197e-12, 3.158e-12,  
02545 3.327e-12, 3.675e-12, 4.292e-12, 5.437e-12, 7.197e-12, 1.008e-11,  
02546 1.437e-11, 2.035e-11, 2.905e-11, 4.062e-11, 5.528e-11, 7.177e-11,  
02547 9.064e-11, 1.109e-10, 1.297e-10, 1.473e-10, 1.652e-10, 1.851e-10,  
02548 2.079e-10, 2.313e-10, 2.619e-10, 2.958e-10, 3.352e-10, 3.796e-10,  
02549 4.295e-10, 4.923e-10, 5.49e-10, 5.998e-10, 6.388e-10, 6.645e-10,  
02550 6.712e-10, 6.549e-10, 6.38e-10, 6.255e-10, 6.253e-10, 6.459e-10,  
02551 6.977e-10, 7.59e-10, 8.242e-10, 8.92e-10, 9.403e-10, 9.701e-10,  
02552 9.483e-10, 9.135e-10, 8.617e-10, 7.921e-10, 7.168e-10, 6.382e-10,  
02553 5.677e-10, 5.045e-10, 4.572e-10, 4.312e-10, 4.145e-10, 4.192e-10,  
02554 4.541e-10, 5.368e-10, 6.771e-10, 8.962e-10, 1.21e-9, 1.659e-9,  
02555 2.33e-9, 3.249e-9, 4.495e-9, 5.923e-9, 7.642e-9, 9.607e-9,  
02556 1.178e-8, 1.399e-8, 1.584e-8, 1.73e-8, 1.816e-8, 1.87e-8,  
02557 1.868e-8, 1.87e-8, 1.884e-8, 1.99e-8, 2.15e-8, 2.258e-8,  
02558 2.364e-8, 2.473e-8, 2.602e-8, 2.689e-8, 2.731e-8, 2.816e-8,  
02559 2.859e-8, 2.839e-8, 2.703e-8, 2.451e-8, 2.149e-8, 1.787e-8,

02560 1.449e-8, 1.111e-8, 8.282e-9, 6.121e-9, 4.494e-9, 3.367e-9,  
02561 2.487e-9, 1.885e-9, 1.503e-9, 1.249e-9, 1.074e-9, 9.427e-10,  
02562 8.439e-10, 7.563e-10, 6.772e-10, 6.002e-10, 5.254e-10, 4.588e-10,  
02563 3.977e-10, 3.449e-10, 3.003e-10, 2.624e-10, 2.335e-10, 2.04e-10,  
02564 1.771e-10, 1.534e-10, 1.296e-10, 1.097e-10, 9.173e-11, 7.73e-11,  
02565 6.547e-11, 5.191e-11, 4.198e-11, 3.361e-11, 2.732e-11, 2.244e-11,  
02566 1.791e-11, 1.509e-11, 1.243e-11, 1.035e-11, 8.969e-12, 7.394e-12,  
02567 6.323e-12, 5.282e-12, 4.543e-12, 3.752e-12, 3.14e-12, 2.6e-12,  
02568 2.194e-12, 1.825e-12, 1.511e-12, 1.245e-12, 1.024e-12, 8.539e-13,  
02569 7.227e-13, 6.102e-13, 5.189e-13, 4.43e-13, 3.774e-13, 3.236e-13,  
02570 2.8e-13, 2.444e-13, 2.156e-13, 1.932e-13, 1.775e-13, 1.695e-13,  
02571 1.672e-13, 1.704e-13, 1.825e-13, 2.087e-13, 2.614e-13, 3.377e-13,  
02572 4.817e-13, 6.989e-13, 1.062e-12, 1.562e-12, 2.288e-12, 3.295e-12,  
02573 4.55e-12, 5.965e-12, 7.546e-12, 9.395e-12, 1.103e-11, 1.228e-11,  
02574 1.318e-11, 1.38e-11, 1.421e-11, 1.39e-11, 1.358e-11, 1.336e-11,  
02575 1.342e-11, 1.356e-11, 1.424e-11, 1.552e-11, 1.73e-11, 1.951e-11,  
02576 2.128e-11, 2.249e-11, 2.277e-11, 2.226e-11, 2.111e-11, 1.922e-11,  
02577 1.775e-11, 1.661e-11, 1.547e-11, 1.446e-11, 1.323e-11, 1.21e-11,  
02578 1.054e-11, 9.283e-12, 8.671e-12, 8.67e-12, 9.429e-12, 1.062e-11,  
02579 1.255e-11, 1.506e-11, 1.818e-11, 2.26e-11, 2.831e-11, 3.723e-11,  
02580 5.092e-11, 6.968e-11, 9.826e-11, 1.349e-10, 1.87e-10, 2.58e-10,  
02581 3.43e-10, 4.424e-10, 5.521e-10, 6.812e-10, 8.064e-10, 9.109e-10,  
02582 9.839e-10, 1.028e-9, 1.044e-9, 1.029e-9, 1.005e-9, 1.002e-9,  
02583 1.038e-9, 1.122e-9, 1.233e-9, 1.372e-9, 1.524e-9, 1.665e-9,  
02584 1.804e-9, 1.908e-9, 2.015e-9, 2.117e-9, 2.219e-9, 2.336e-9,  
02585 2.531e-9, 2.805e-9, 3.189e-9, 3.617e-9, 4.208e-9, 4.911e-9,  
02586 5.619e-9, 6.469e-9, 7.188e-9, 7.957e-9, 8.503e-9, 9.028e-9,  
02587 9.571e-9, 9.99e-9, 1.055e-8, 1.102e-8, 1.132e-8, 1.141e-8,  
02588 1.145e-8, 1.145e-8, 1.176e-8, 1.224e-8, 1.304e-8, 1.388e-8,  
02589 1.445e-8, 1.453e-8, 1.368e-8, 1.22e-8, 1.042e-8, 8.404e-9,  
02590 6.403e-9, 4.643e-9, 3.325e-9, 2.335e-9, 1.638e-9, 1.19e-9,  
02591 9.161e-10, 7.412e-10, 6.226e-10, 5.516e-10, 5.068e-10, 4.831e-10,  
02592 4.856e-10, 5.162e-10, 5.785e-10, 6.539e-10, 7.485e-10, 8.565e-10,  
02593 9.534e-10, 1.052e-9, 1.115e-9, 1.173e-9, 1.203e-9, 1.224e-9,  
02594 1.243e-9, 1.248e-9, 1.261e-9, 1.265e-9, 1.25e-9, 1.217e-9,  
02595 1.176e-9, 1.145e-9, 1.153e-9, 1.199e-9, 1.278e-9, 1.366e-9,  
02596 1.426e-9, 1.444e-9, 1.365e-9, 1.224e-9, 1.051e-9, 8.539e-10,  
02597 6.564e-10, 4.751e-10, 3.404e-10, 2.377e-10, 1.631e-10, 1.114e-10,  
02598 7.87e-11, 5.793e-11, 4.284e-11, 3.3e-11, 2.62e-11, 2.152e-11,  
02599 1.777e-11, 1.496e-11, 1.242e-11, 1.037e-11, 8.725e-12, 7.004e-12,  
02600 5.718e-12, 4.769e-12, 3.952e-12, 3.336e-12, 2.712e-12, 2.213e-12,  
02601 1.803e-12, 1.492e-12, 1.236e-12, 1.006e-12, 8.384e-13, 7.063e-13,  
02602 5.879e-13, 4.93e-13, 4.171e-13, 3.569e-13, 3.083e-13, 2.688e-13,  
02603 2.333e-13, 2.035e-13, 1.82e-13, 1.682e-13, 1.635e-13, 1.674e-13,  
02604 1.769e-13, 2.022e-13, 2.485e-13, 3.127e-13, 4.25e-13, 5.928e-13,  
02605 8.514e-13, 1.236e-12, 1.701e-12, 2.392e-12, 3.231e-12, 4.35e-12,  
02606 5.559e-12, 6.915e-12, 8.519e-12, 1.013e-11, 1.146e-11, 1.24e-11,  
02607 1.305e-11, 1.333e-11, 1.318e-11, 1.263e-11, 1.238e-11, 1.244e-11,  
02608 1.305e-11, 1.432e-11, 1.623e-11, 1.846e-11, 2.09e-11, 2.328e-11,  
02609 2.526e-11, 2.637e-11, 2.702e-11, 2.794e-11, 2.889e-11, 2.989e-11,  
02610 3.231e-11, 3.68e-11, 4.375e-11, 5.504e-11, 7.159e-11, 9.502e-11,  
02611 1.279e-10, 1.645e-10, 2.098e-10, 2.618e-10, 3.189e-10, 3.79e-10,  
02612 4.303e-10, 4.753e-10, 5.027e-10, 5.221e-10, 5.293e-10, 5.346e-10,  
02613 5.467e-10, 5.796e-10, 6.2e-10, 6.454e-10, 6.705e-10, 6.925e-10,  
02614 7.233e-10, 7.35e-10, 7.538e-10, 7.861e-10, 8.077e-10, 8.132e-10,  
02615 7.749e-10, 7.036e-10, 6.143e-10, 5.093e-10, 4.089e-10, 3.092e-10,  
02616 2.299e-10, 1.705e-10, 1.277e-10, 9.723e-11, 7.533e-11, 6.126e-11,  
02617 5.154e-11, 4.428e-11, 3.913e-11, 3.521e-11, 3.297e-11, 3.275e-11,  
02618 3.46e-11, 3.798e-11, 4.251e-11, 4.745e-11, 5.232e-11, 5.606e-11,  
02619 5.82e-11, 5.88e-11, 5.79e-11, 5.661e-11, 5.491e-11, 5.366e-11,  
02620 5.341e-11, 5.353e-11, 5.336e-11, 5.293e-11, 5.248e-11, 5.235e-11,  
02621 5.208e-11, 5.322e-11, 5.521e-11, 5.725e-11, 5.827e-11, 5.685e-11,  
02622 5.245e-11, 4.612e-11, 3.884e-11, 3.129e-11, 2.404e-11, 1.732e-11,  
02623 1.223e-11, 8.574e-12, 5.888e-12, 3.986e-12, 2.732e-12, 1.948e-12,  
02624 1.414e-12, 1.061e-12, 8.298e-13, 6.612e-13, 5.413e-13, 4.472e-13,  
02625 3.772e-13, 3.181e-13, 2.645e-13, 2.171e-13, 1.778e-13, 1.464e-13,  
02626 1.183e-13, 9.637e-14, 7.991e-14, 6.668e-14, 5.57e-14, 4.663e-14,  
02627 3.848e-14, 3.233e-14, 2.706e-14, 2.284e-14, 1.944e-14, 1.664e-14,  
02628 1.43e-14, 1.233e-14, 1.066e-14, 9.234e-15, 8.023e-15, 6.993e-15,  
02629 6.119e-15, 5.384e-15, 4.774e-15, 4.283e-15, 3.916e-15, 3.695e-15,  
02630 3.682e-15, 4.004e-15, 4.912e-15, 6.853e-15, 1.056e-14, 1.712e-14,  
02631 2.804e-14, 4.516e-14, 7.113e-14, 1.084e-13, 1.426e-13, 1.734e-13,  
02632 1.978e-13, 2.194e-13, 2.388e-13, 2.489e-13, 2.626e-13, 2.865e-13,  
02633 3.105e-13, 3.387e-13, 3.652e-13, 3.984e-13, 4.398e-13, 4.906e-13,  
02634 5.55e-13, 6.517e-13, 7.813e-13, 9.272e-13, 1.164e-12, 1.434e-12,  
02635 1.849e-12, 2.524e-12, 3.328e-12, 4.523e-12, 6.108e-12, 8.207e-12,  
02636 1.122e-11, 1.477e-11, 1.9e-11, 2.412e-11, 2.984e-11, 3.68e-11,  
02637 4.353e-11, 4.963e-11, 5.478e-11, 5.903e-11, 6.233e-11, 6.483e-11,  
02638 6.904e-11, 7.569e-11, 8.719e-11, 1.048e-10, 1.278e-10, 1.557e-10,  
02639 1.869e-10, 2.218e-10, 2.61e-10, 2.975e-10, 3.371e-10, 3.746e-10,  
02640 4.065e-10, 4.336e-10, 4.503e-10, 4.701e-10, 4.8e-10, 4.917e-10,  
02641 5.038e-10, 5.128e-10, 5.143e-10, 5.071e-10, 5.019e-10, 5.025e-10,  
02642 5.183e-10, 5.496e-10, 5.877e-10, 6.235e-10, 6.42e-10, 6.234e-10,  
02643 5.698e-10, 4.916e-10, 4.022e-10, 3.126e-10, 2.282e-10, 1.639e-10,  
02644 1.142e-10, 7.919e-11, 5.69e-11, 4.313e-11, 3.413e-11, 2.807e-11,  
02645 2.41e-11, 2.166e-11, 2.024e-11, 1.946e-11, 1.929e-11, 1.963e-11,  
02646 2.035e-11, 2.162e-11, 2.305e-11, 2.493e-11, 2.748e-11, 3.048e-11,

02647 3.413e-11, 3.754e-11, 4.155e-11, 4.635e-11, 5.11e-11, 5.734e-11,  
02648 6.338e-11, 6.99e-11, 7.611e-11, 8.125e-11, 8.654e-11, 8.951e-11,  
02649 9.182e-11, 9.31e-11, 9.273e-11, 9.094e-11, 8.849e-11, 8.662e-11,  
02650 8.67e-11, 8.972e-11, 9.566e-11, 1.025e-10, 1.083e-10, 1.111e-10,  
02651 1.074e-10, 9.771e-11, 8.468e-11, 6.958e-11, 5.47e-11, 4.04e-11,  
02652 2.94e-11, 2.075e-11, 1.442e-11, 1.01e-11, 7.281e-12, 5.409e-12,  
02653 4.138e-12, 3.304e-12, 2.784e-12, 2.473e-12, 2.273e-12, 2.186e-12,  
02654 2.118e-12, 2.066e-12, 1.958e-12, 1.818e-12, 1.675e-12, 1.509e-12,  
02655 1.349e-12, 1.171e-12, 9.838e-13, 8.213e-13, 6.765e-13, 5.378e-13,  
02656 4.161e-13, 3.119e-13, 2.279e-13, 1.637e-13, 1.152e-13, 8.112e-14,  
02657 5.919e-14, 4.47e-14, 3.492e-14, 2.811e-14, 2.319e-14, 1.948e-14,  
02658 1.66e-14, 1.432e-14, 1.251e-14, 1.109e-14, 1.006e-14, 9.45e-15,  
02659 9.384e-15, 1.012e-14, 1.216e-14, 1.636e-14, 2.305e-14, 3.488e-14,  
02660 5.572e-14, 8.479e-14, 1.265e-13, 1.905e-13, 2.73e-13, 3.809e-13,  
02661 4.955e-13, 6.303e-13, 7.861e-13, 9.427e-13, 1.097e-12, 1.212e-12,  
02662 1.328e-12, 1.415e-12, 1.463e-12, 1.495e-12, 1.571e-12, 1.731e-12,  
02663 1.981e-12, 2.387e-12, 2.93e-12, 3.642e-12, 4.584e-12, 5.822e-12,  
02664 7.278e-12, 9.193e-12, 1.135e-11, 1.382e-11, 1.662e-11, 1.958e-11,  
02665 2.286e-11, 2.559e-11, 2.805e-11, 2.988e-11, 3.106e-11, 3.182e-11,  
02666 3.2e-11, 3.258e-11, 3.362e-11, 3.558e-11, 3.688e-11, 3.8e-11,  
02667 3.929e-11, 4.062e-11, 4.186e-11, 4.293e-11, 4.48e-11, 4.643e-11,  
02668 4.704e-11, 4.571e-11, 4.206e-11, 3.715e-11, 3.131e-11, 2.541e-11,  
02669 1.978e-11, 1.508e-11, 1.146e-11, 8.7e-12, 6.603e-12, 5.162e-12,  
02670 4.157e-12, 3.408e-12, 2.829e-12, 2.405e-12, 2.071e-12, 1.826e-12,  
02671 1.648e-12, 1.542e-12, 1.489e-12, 1.485e-12, 1.493e-12, 1.545e-12,  
02672 1.637e-12, 1.814e-12, 2.061e-12, 2.312e-12, 2.651e-12, 3.03e-12,  
02673 3.46e-12, 3.901e-12, 4.306e-12, 4.721e-12, 5.008e-12, 5.281e-12,  
02674 5.541e-12, 5.791e-12, 6.115e-12, 6.442e-12, 6.68e-12, 6.791e-12,  
02675 6.831e-12, 6.839e-12, 6.946e-12, 7.128e-12, 7.537e-12, 8.036e-12,  
02676 8.392e-12, 8.562e-12, 8.11e-12, 7.325e-12, 6.329e-12, 5.183e-12,  
02677 4.081e-12, 2.985e-12, 2.141e-12, 1.492e-12, 1.015e-12, 6.684e-13,  
02678 4.414e-13, 2.987e-13, 2.038e-13, 1.391e-13, 9.86e-14, 7.24e-14,  
02679 5.493e-14, 4.288e-14, 3.427e-14, 2.787e-14, 2.296e-14, 1.909e-14,  
02680 1.598e-14, 1.344e-14, 1.135e-14, 9.616e-15, 8.169e-15, 6.957e-15,  
02681 5.938e-15, 5.08e-15, 4.353e-15, 3.738e-15, 3.217e-15, 2.773e-15,  
02682 2.397e-15, 2.077e-15, 1.805e-15, 1.575e-15, 1.382e-15, 1.221e-15,  
02683 1.09e-15, 9.855e-16, 9.068e-16, 8.537e-16, 8.27e-16, 8.29e-16,  
02684 8.634e-16, 9.359e-16, 1.055e-15, 1.233e-15, 1.486e-15, 1.839e-15,  
02685 2.326e-15, 2.998e-15, 3.934e-15, 5.256e-15, 7.164e-15, 9.984e-15,  
02686 1.427e-14, 2.099e-14, 3.196e-14, 5.121e-14, 7.908e-14, 1.131e-13,  
02687 1.602e-13, 2.239e-13, 3.075e-13, 4.134e-13, 5.749e-13, 7.886e-13,  
02688 1.071e-12, 1.464e-12, 2.032e-12, 2.8e-12, 3.732e-12, 4.996e-12,  
02689 6.483e-12, 8.143e-12, 1.006e-11, 1.238e-11, 1.484e-11, 1.744e-11,  
02690 2.02e-11, 2.274e-11, 2.562e-11, 2.848e-11, 3.191e-11, 3.617e-11,  
02691 4.081e-11, 4.577e-11, 4.937e-11, 5.204e-11, 5.401e-11, 5.462e-11,  
02692 5.507e-11, 5.51e-11, 5.605e-11, 5.686e-11, 5.739e-11, 5.766e-11,  
02693 5.74e-11, 5.754e-11, 5.761e-11, 5.777e-11, 5.712e-11, 5.51e-11,  
02694 5.088e-11, 4.438e-11, 3.728e-11, 2.994e-11, 2.305e-11, 1.715e-11,  
02695 1.256e-11, 9.208e-12, 6.745e-12, 5.014e-12, 3.785e-12, 2.9e-12,  
02696 2.239e-12, 1.757e-12, 1.414e-12, 1.142e-12, 9.482e-13, 8.01e-13,  
02697 6.961e-13, 6.253e-13, 5.735e-13, 5.433e-13, 5.352e-13, 5.493e-13,  
02698 5.706e-13, 6.068e-13, 6.531e-13, 7.109e-13, 7.767e-13, 8.59e-13,  
02699 9.792e-13, 1.142e-12, 1.371e-12, 1.65e-12, 1.957e-12, 2.302e-12,  
02700 2.705e-12, 3.145e-12, 3.608e-12, 4.071e-12, 4.602e-12, 5.133e-12,  
02701 5.572e-12, 5.987e-12, 6.248e-12, 6.533e-12, 6.757e-12, 6.935e-12,  
02702 7.224e-12, 7.422e-12, 7.538e-12, 7.547e-12, 7.495e-12, 7.543e-12,  
02703 7.725e-12, 8.139e-12, 8.627e-12, 9.146e-12, 9.443e-12, 9.318e-12,  
02704 8.649e-12, 7.512e-12, 6.261e-12, 4.915e-12, 3.647e-12, 2.597e-12,  
02705 1.785e-12, 1.242e-12, 8.66e-13, 6.207e-13, 4.61e-13, 3.444e-13,  
02706 2.634e-13, 2.1e-13, 1.725e-13, 1.455e-13, 1.237e-13, 1.085e-13,  
02707 9.513e-14, 7.978e-14, 6.603e-14, 5.288e-14, 4.084e-14, 2.952e-14,  
02708 2.157e-14, 1.593e-14, 1.199e-14, 9.267e-15, 7.365e-15, 6.004e-15,  
02709 4.995e-15, 4.218e-15, 3.601e-15, 3.101e-15, 2.692e-15, 2.36e-15,  
02710 2.094e-15, 1.891e-15, 1.755e-15, 1.699e-15, 1.755e-15, 1.987e-15,  
02711 2.506e-15, 3.506e-15, 5.289e-15, 8.311e-15, 1.325e-14, 2.129e-14,  
02712 3.237e-14, 4.595e-14, 6.441e-14, 8.433e-14, 1.074e-13, 1.383e-13,  
02713 1.762e-13, 2.281e-13, 2.831e-13, 3.523e-13, 4.38e-13, 5.304e-13,  
02714 6.29e-13, 7.142e-13, 8.032e-13, 8.934e-13, 9.888e-13, 1.109e-12,  
02715 1.261e-12, 1.462e-12, 1.74e-12, 2.099e-12, 2.535e-12, 3.008e-12,  
02716 3.462e-12, 3.856e-12, 4.098e-12, 4.239e-12, 4.234e-12, 4.132e-12,  
02717 3.986e-12, 3.866e-12, 3.829e-12, 3.742e-12, 3.705e-12, 3.694e-12,  
02718 3.765e-12, 3.849e-12, 3.929e-12, 4.056e-12, 4.092e-12, 4.047e-12,  
02719 3.792e-12, 3.407e-12, 2.953e-12, 2.429e-12, 1.931e-12, 1.46e-12,  
02720 1.099e-12, 8.199e-13, 6.077e-13, 4.449e-13, 3.359e-13, 2.524e-13,  
02721 1.881e-13, 1.391e-13, 1.02e-13, 7.544e-14, 5.555e-14, 4.22e-14,  
02722 3.321e-14, 2.686e-14, 2.212e-14, 1.78e-14, 1.369e-14, 1.094e-14,  
02723 9.13e-15, 8.101e-15, 7.828e-15, 8.393e-15, 1.012e-14, 1.259e-14,  
02724 1.538e-14, 1.961e-14, 2.619e-14, 3.679e-14, 5.049e-14, 6.917e-14,  
02725 8.88e-14, 1.115e-13, 1.373e-13, 1.619e-13, 1.878e-13, 2.111e-13,  
02726 2.33e-13, 2.503e-13, 2.613e-13, 2.743e-13, 2.826e-13, 2.976e-13,  
02727 3.162e-13, 3.36e-13, 3.491e-13, 3.541e-13, 3.595e-13, 3.608e-13,  
02728 3.709e-13, 3.869e-13, 4.12e-13, 4.366e-13, 4.504e-13, 4.379e-13,  
02729 3.955e-13, 3.385e-13, 2.741e-13, 2.089e-13, 1.427e-13, 9.294e-14,  
02730 5.775e-14, 3.565e-14, 2.21e-14, 1.398e-14, 9.194e-15, 6.363e-15,  
02731 4.644e-15, 3.55e-15, 2.808e-15, 2.274e-15, 1.871e-15, 1.557e-15,  
02732 1.308e-15, 1.108e-15, 9.488e-16, 8.222e-16, 7.238e-16, 6.506e-16,  
02733 6.008e-16, 5.742e-16, 5.724e-16, 5.991e-16, 6.625e-16, 7.775e-16,

```

02734     9.734e-16, 1.306e-15, 1.88e-15, 2.879e-15, 4.616e-15, 7.579e-15,
02735     1.248e-14, 2.03e-14, 3.244e-14, 5.171e-14, 7.394e-14, 9.676e-14,
02736     1.199e-13, 1.467e-13, 1.737e-13, 2.02e-13, 2.425e-13, 3.016e-13,
02737     3.7e-13, 4.617e-13, 5.949e-13, 7.473e-13, 9.378e-13, 1.191e-12,
02738     1.481e-12, 1.813e-12, 2.232e-12, 2.722e-12, 3.254e-12, 3.845e-12,
02739     4.458e-12, 5.048e-12, 5.511e-12, 5.898e-12, 6.204e-12, 6.293e-12,
02740     6.386e-12, 6.467e-12, 6.507e-12, 6.466e-12, 6.443e-12, 6.598e-12,
02741     6.873e-12, 7.3e-12, 7.816e-12, 8.368e-12, 8.643e-12, 8.466e-12,
02742     7.871e-12, 6.853e-12, 5.714e-12, 4.482e-12, 3.392e-12, 2.613e-12,
02743     2.008e-12, 1.562e-12, 1.228e-12, 9.888e-13, 7.646e-13, 5.769e-13,
02744     4.368e-13, 3.324e-13, 2.508e-13, 1.916e-13
02745 };
02746
02747 static double xfcrev[15] =
02748 { 1.003, 1.009, 1.015, 1.023, 1.029, 1.033, 1.037,
02749   1.039, 1.04, 1.046, 1.036, 1.027, 1.01, 1.002, 1.
02750 };
02751
02752 double a1, a2, a3, dw, ew, dx, xw, xx, vf2, vf6, cw260, cw296,
02753 sfac, fscal, cwfrn, ctmph, ctwfrn, ctws1f;
02754
02755 int iw, ix;
02756
02757 /* Get H2O continuum absorption... */
02758 xw = nu / 10 + 1;
02759 if (xw >= 1 && xw < 2001) {
02760     iw = (int) xw;
02761     dw = xw - iw;
02762     ew = 1 - dw;
02763     cw296 = ew * h2o296[iw - 1] + dw * h2o296[iw];
02764     cw260 = ew * h2o260[iw - 1] + dw * h2o260[iw];
02765     cwfrn = ew * h2ofrn[iw - 1] + dw * h2ofrn[iw];
02766     if (nu <= 820 || nu >= 960) {
02767         sfac = 1;
02768     } else {
02769         xx = (nu - 820) / 10;
02770         ix = (int) xx;
02771         dx = xx - ix;
02772         sfac = (1 - dx) * xfcrev[ix] + dx * xfcrev[ix + 1];
02773     }
02774     ctws1f = sfac * cw296 * pow(cw260 / cw296, (296 - t) / (296 - 260));
02775     vf2 = POW2(nu - 370);
02776     vf6 = POW3(vf2);
02777     fscal = 36100 / (vf2 + vf6 * 1e-8 + 36100) * -.25 + 1;
02778     ctwfrn = cwfrn * fscal;
02779     a1 = nu * u * tanh(.7193876 / t * nu);
02780     a2 = 296 / t;
02781     a3 = p / P0 * (q * ctws1f + (1 - q) * ctwfrn) * 1e-20;
02782     ctmph = a1 * a2 * a3;
02783 } else
02784     ctmph = 0;
02785 return ctmph;
02786 }
02787
02788 /*****
02789
02790 double ctmn2(
02791     double nu,
02792     double p,
02793     double t) {
02794
02795     static double ba[98] = { 0., 4.45e-8, 5.22e-8, 6.46e-8, 7.75e-8, 9.03e-8,
02796     1.06e-7, 1.21e-7, 1.37e-7, 1.57e-7, 1.75e-7, 2.01e-7, 2.3e-7,
02797     2.59e-7, 2.95e-7, 3.26e-7, 3.66e-7, 4.05e-7, 4.47e-7, 4.92e-7,
02798     5.34e-7, 5.84e-7, 6.24e-7, 6.67e-7, 7.14e-7, 7.26e-7, 7.54e-7,
02799     7.84e-7, 8.09e-7, 8.42e-7, 8.62e-7, 8.87e-7, 9.11e-7, 9.36e-7,
02800     9.76e-7, 1.03e-6, 1.11e-6, 1.23e-6, 1.39e-6, 1.61e-6, 1.76e-6,
02801     1.94e-6, 1.97e-6, 1.87e-6, 1.75e-6, 1.56e-6, 1.42e-6, 1.35e-6,
02802     1.32e-6, 1.29e-6, 1.29e-6, 1.29e-6, 1.3e-6, 1.32e-6, 1.33e-6,
02803     1.34e-6, 1.35e-6, 1.33e-6, 1.31e-6, 1.29e-6, 1.24e-6, 1.2e-6,
02804     1.16e-6, 1.1e-6, 1.04e-6, 9.96e-7, 9.38e-7, 8.63e-7, 7.98e-7,
02805     7.26e-7, 6.55e-7, 5.94e-7, 5.35e-7, 4.74e-7, 4.24e-7, 3.77e-7,
02806     3.33e-7, 2.96e-7, 2.63e-7, 2.34e-7, 2.08e-7, 1.85e-7, 1.67e-7,
02807     1.47e-7, 1.32e-7, 1.2e-7, 1.09e-7, 9.85e-8, 9.08e-8, 8.18e-8,
02808     7.56e-8, 6.85e-8, 6.14e-8, 5.83e-8, 5.77e-8, 5e-8, 4.32e-8, 0.
02809 };
02810
02811     static double betaa[98] = { 802., 802., 761., 722., 679., 646., 609., 562.,
02812     511., 472., 436., 406., 377., 355., 338., 319., 299., 278., 255.,
02813     233., 208., 184., 149., 107., 66., 25., -13., -49., -82., -104.,
02814     -119., -130., -139., -144., -146., -146., -147., -148., -150.,
02815     -153., -160., -169., -181., -189., -195., -200., -205., -209.,
02816     -211., -210., -210., -209., -205., -199., -190., -180., -168.,
02817     -157., -143., -126., -108., -89., -63., -32., 1., 35., 65., 95.,
02818     121., 141., 152., 161., 164., 164., 161., 155., 148., 143., 137.,
02819     133., 131., 133., 139., 150., 165., 187., 213., 248., 284., 321.,
02820     372., 449., 514., 569., 609., 642., 673., 673.

```

```

02821     };
02822
02823     static double nua[98] = { 2120., 2125., 2130., 2135., 2140., 2145., 2150.,
02824         2155., 2160., 2165., 2170., 2175., 2180., 2185., 2190., 2195.,
02825         2200., 2205., 2210., 2215., 2220., 2225., 2230., 2235., 2240.,
02826         2245., 2250., 2255., 2260., 2265., 2270., 2275., 2280., 2285.,
02827         2290., 2295., 2300., 2305., 2310., 2315., 2320., 2325., 2330.,
02828         2335., 2340., 2345., 2350., 2355., 2360., 2365., 2370., 2375.,
02829         2380., 2385., 2390., 2395., 2400., 2405., 2410., 2415., 2420.,
02830         2425., 2430., 2435., 2440., 2445., 2450., 2455., 2460., 2465.,
02831         2470., 2475., 2480., 2485., 2490., 2495., 2500., 2505., 2510.,
02832         2515., 2520., 2525., 2530., 2535., 2540., 2545., 2550., 2555.,
02833         2560., 2565., 2570., 2575., 2580., 2585., 2590., 2595., 2600., 2605.
02834     };
02835
02836     double b, beta, q_n2 = 0.79, t0 = 273, tr = 296;
02837
02838     int idx;
02839
02840     /* Check wavenumber range... */
02841     if (nu < nua[0] || nu > nua[97])
02842         return 0;
02843
02844     /* Interpolate B and beta... */
02845     idx = locate_reg(nua, 98, nu);
02846     b = LIN(nua[idx], ba[idx], nua[idx + 1], ba[idx + 1], nu);
02847     beta = LIN(nua[idx], betaa[idx], nua[idx + 1], betaa[idx + 1], nu);
02848
02849     /* Compute absorption coefficient... */
02850     return 0.1 * POW2(p / P0 * t0 / t) * exp(beta * (1 / tr - 1 / t))
02851         * q_n2 * b * (q_n2 + (1 - q_n2) * (1.294 - 0.4545 * t / tr));
02852 }
02853
02854 /*****
02855
02856 double ctmo2(
02857     double nu,
02858     double p,
02859     double t) {
02860
02861     static double ba[90] = { 0., .061, .074, .084, .096, .12, .162, .208, .246,
02862         .285, .314, .38, .444, .5, .571, .673, .768, .853, .966, 1.097,
02863         1.214, 1.333, 1.466, 1.591, 1.693, 1.796, 1.922, 2.037, 2.154,
02864         2.264, 2.375, 2.508, 2.671, 2.847, 3.066, 3.417, 3.828, 4.204,
02865         4.453, 4.599, 4.528, 4.284, 3.955, 3.678, 3.477, 3.346, 3.29,
02866         3.251, 3.231, 3.226, 3.212, 3.192, 3.108, 3.033, 2.911, 2.798,
02867         2.646, 2.508, 2.322, 2.13, 1.928, 1.757, 1.588, 1.417, 1.253,
02868         1.109, .99, .888, .791, .678, .587, .524, .464, .403, .357, .32,
02869         .29, .267, .242, .215, .182, .16, .146, .128, .103, .087, .081,
02870         .071, .064, 0.
02871     };
02872
02873     static double betaa[90] = { 467., 467., 400., 315., 379., 368., 475., 521.,
02874         531., 512., 442., 444., 430., 381., 335., 324., 296., 248., 215.,
02875         193., 158., 127., 101., 71., 31., -6., -26., -47., -63., -79.,
02876         -88., -88., -87., -90., -98., -99., -109., -134., -160., -167.,
02877         -164., -158., -153., -151., -156., -166., -168., -173., -170.,
02878         -161., -145., -126., -108., -84., -59., -29., 4., 41., 73., 97.,
02879         123., 159., 198., 220., 242., 256., 281., 311., 334., 319., 313.,
02880         321., 323., 310., 315., 320., 335., 361., 378., 373., 338., 319.,
02881         346., 322., 291., 290., 350., 371., 504., 504.
02882     };
02883
02884     static double nua[90] = { 1360., 1365., 1370., 1375., 1380., 1385., 1390.,
02885         1395., 1400., 1405., 1410., 1415., 1420., 1425., 1430., 1435.,
02886         1440., 1445., 1450., 1455., 1460., 1465., 1470., 1475., 1480.,
02887         1485., 1490., 1495., 1500., 1505., 1510., 1515., 1520., 1525.,
02888         1530., 1535., 1540., 1545., 1550., 1555., 1560., 1565., 1570.,
02889         1575., 1580., 1585., 1590., 1595., 1600., 1605., 1610., 1615.,
02890         1620., 1625., 1630., 1635., 1640., 1645., 1650., 1655., 1660.,
02891         1665., 1670., 1675., 1680., 1685., 1690., 1695., 1700., 1705.,
02892         1710., 1715., 1720., 1725., 1730., 1735., 1740., 1745., 1750.,
02893         1755., 1760., 1765., 1770., 1775., 1780., 1785., 1790., 1795.,
02894         1800., 1805.
02895     };
02896
02897     double b, beta, q_o2 = 0.21, t0 = 273, tr = 296;
02898
02899     int idx;
02900
02901     /* Check wavenumber range... */
02902     if (nu < nua[0] || nu > nua[89])
02903         return 0;
02904
02905     /* Interpolate B and beta... */
02906     idx = locate_reg(nua, 90, nu);
02907     b = LIN(nua[idx], ba[idx], nua[idx + 1], ba[idx + 1], nu);

```

```

02908     beta = LIN(nua[idx], betaa[idx], nua[idx + 1], betaa[idx + 1], nu);
02909
02910     /* Compute absorption coefficient... */
02911     return 0.1 * POW2(p / P0 * t0 / t) * exp(beta * (1 / tr - 1 / t)) * q_o2 *
02912         b;
02913 }
02914
02915 /*****
02916
02917 void copy_atm(
02918     ctl_t * ctl,
02919     atm_t * atm_dest,
02920     atm_t * atm_src,
02921     int init) {
02922
02923     int ig, ip, iw;
02924
02925     size_t s;
02926
02927     /* Data size... */
02928     s = (size_t) atm_src->np * sizeof(double);
02929
02930     /* Copy data... */
02931     atm_dest->np = atm_src->np;
02932     memcpy(atm_dest->time, atm_src->time, s);
02933     memcpy(atm_dest->z, atm_src->z, s);
02934     memcpy(atm_dest->lon, atm_src->lon, s);
02935     memcpy(atm_dest->lat, atm_src->lat, s);
02936     memcpy(atm_dest->p, atm_src->p, s);
02937     memcpy(atm_dest->t, atm_src->t, s);
02938     for (ig = 0; ig < ctl->ng; ig++)
02939         memcpy(atm_dest->q[ig], atm_src->q[ig], s);
02940     for (iw = 0; iw < ctl->nw; iw++)
02941         memcpy(atm_dest->k[iw], atm_src->k[iw], s);
02942
02943     /* Initialize... */
02944     if (init)
02945         for (ip = 0; ip < atm_dest->np; ip++) {
02946             atm_dest->p[ip] = 0;
02947             atm_dest->t[ip] = 0;
02948             for (ig = 0; ig < ctl->ng; ig++)
02949                 atm_dest->q[ig][ip] = 0;
02950             for (iw = 0; iw < ctl->nw; iw++)
02951                 atm_dest->k[iw][ip] = 0;
02952         }
02953 }
02954
02955 /*****
02956
02957 void copy_obs(
02958     ctl_t * ctl,
02959     obs_t * obs_dest,
02960     obs_t * obs_src,
02961     int init) {
02962
02963     int id, ir;
02964
02965     size_t s;
02966
02967     /* Data size... */
02968     s = (size_t) obs_src->nr * sizeof(double);
02969
02970     /* Copy data... */
02971     obs_dest->nr = obs_src->nr;
02972     memcpy(obs_dest->time, obs_src->time, s);
02973     memcpy(obs_dest->obsz, obs_src->obsz, s);
02974     memcpy(obs_dest->obslon, obs_src->obslon, s);
02975     memcpy(obs_dest->obslat, obs_src->obslat, s);
02976     memcpy(obs_dest->vpz, obs_src->vpz, s);
02977     memcpy(obs_dest->vplon, obs_src->vplon, s);
02978     memcpy(obs_dest->vplat, obs_src->vplat, s);
02979     memcpy(obs_dest->tpz, obs_src->tpz, s);
02980     memcpy(obs_dest->tplon, obs_src->tplon, s);
02981     memcpy(obs_dest->tplat, obs_src->tplat, s);
02982     for (id = 0; id < ctl->nd; id++)
02983         memcpy(obs_dest->rad[id], obs_src->rad[id], s);
02984     for (id = 0; id < ctl->nd; id++)
02985         memcpy(obs_dest->tau[id], obs_src->tau[id], s);
02986
02987     /* Initialize... */
02988     if (init)
02989         for (id = 0; id < ctl->nd; id++)
02990             for (ir = 0; ir < obs_dest->nr; ir++)
02991                 if (gsl_finite(obs_dest->rad[id][ir])) {
02992                     obs_dest->rad[id][ir] = 0;
02993                     obs_dest->tau[id][ir] = 0;
02994                 }

```

```

02995 }
02996
02997 /*****
02998
02999 int find_emitter(
03000     ctl_t * ctl,
03001     const char *emitter) {
03002
03003     int ig;
03004
03005     for (ig = 0; ig < ctl->ng; ig++)
03006         if (strcasecmp(ctl->emitter[ig], emitter) == 0)
03007             return ig;
03008
03009     return -1;
03010 }
03011
03012 /*****
03013
03014 void formod(
03015     ctl_t * ctl,
03016     atm_t * atm,
03017     obs_t * obs) {
03018
03019     int id, ir, *mask;
03020
03021     /* Allocate... */
03022     ALLOC(mask, int,
03023           ND * NR);
03024
03025     /* Save observation mask... */
03026     for (id = 0; id < ctl->nd; id++)
03027         for (ir = 0; ir < obs->nr; ir++)
03028             mask[id * NR + ir] = !gsl_finite(obs->rad[id][ir]);
03029
03030     /* Hydrostatic equilibrium... */
03031     hydrostatic(ctl, atm);
03032
03033     /* Calculate pencil beams... */
03034     for (ir = 0; ir < obs->nr; ir++)
03035         formod_pencil(ctl, atm, obs, ir);
03036
03037     /* Apply field-of-view convolution... */
03038     formod_fov(ctl, obs);
03039
03040     /* Convert radiance to brightness temperature... */
03041     if (ctl->write_bbt)
03042         for (id = 0; id < ctl->nd; id++)
03043             for (ir = 0; ir < obs->nr; ir++)
03044                 obs->rad[id][ir] = brightness(obs->rad[id][ir], ctl->nu[id]);
03045
03046     /* Apply observation mask... */
03047     for (id = 0; id < ctl->nd; id++)
03048         for (ir = 0; ir < obs->nr; ir++)
03049             if (mask[id * NR + ir])
03050                 obs->rad[id][ir] = GSL_NAN;
03051
03052     /* Free... */
03053     free(mask);
03054 }
03055
03056 /*****
03057
03058 void formod_continua(
03059     ctl_t * ctl,
03060     los_t * los,
03061     int ip,
03062     double *beta) {
03063
03064     static int ig_co2 = -999, ig_h2o = -999;
03065
03066     int id;
03067
03068     /* Extinction... */
03069     for (id = 0; id < ctl->nd; id++)
03070         beta[id] = los->k[ctl->window[id]][ip];
03071
03072     /* CO2 continuum... */
03073     if (ctl->ctm_co2) {
03074         if (ig_co2 == -999)
03075             ig_co2 = find_emitter(ctl, "CO2");
03076         if (ig_co2 >= 0)
03077             for (id = 0; id < ctl->nd; id++)
03078                 beta[id] += ctmco2(ctl->nu[id], los->p[ip], los->t[ip],
03079                                   los->u[ig_co2][ip]) / los->ds[ip];
03080     }
03081

```

```

03082  /* H2O continuum... */
03083  if (ctl->ctm_h2o) {
03084      if (ig_h2o == -999)
03085          ig_h2o = find_emitter(ctl, "H2O");
03086      if (ig_h2o >= 0)
03087          for (id = 0; id < ctl->nd; id++)
03088              beta[id] += ctmh2o(ctl->nu[id], los->p[ip], los->t[ip],
03089                               los->q[ig_h2o][ip],
03090                               los->u[ig_h2o][ip]) / los->ds[ip];
03091  }
03092
03093  /* N2 continuum... */
03094  if (ctl->ctm_n2)
03095      for (id = 0; id < ctl->nd; id++)
03096          beta[id] += ctmn2(ctl->nu[id], los->p[ip], los->t[ip]);
03097
03098  /* O2 continuum... */
03099  if (ctl->ctm_o2)
03100      for (id = 0; id < ctl->nd; id++)
03101          beta[id] += ctmo2(ctl->nu[id], los->p[ip], los->t[ip]);
03102 }
03103
03104 /*****
03105 void formod_fov(
03106     ctl_t * ctl,
03107     obs_t * obs) {
03108
03109     static double dz[NSHAPE], w[NSHAPE];
03110
03111     static int init = 0, n;
03112
03113     obs_t *obs2;
03114
03115     double rad[ND][NR], tau[ND][NR], wsum, z[NR], zfov;
03116
03117     int i, id, idx, ir, ir2, nz;
03118
03119     /* Do not take into account FOV... */
03120     if (ctl->fov[0] == '-')
03121         return;
03122
03123     /* Initialize FOV data... */
03124     if (!init) {
03125         init = 1;
03126         read_shape(ctl->fov, dz, w, &n);
03127     }
03128
03129     /* Allocate... */
03130     ALLOC(obs2, obs_t, 1);
03131
03132     /* Copy observation data... */
03133     copy_obs(ctl, obs2, obs, 0);
03134
03135     /* Loop over ray paths... */
03136     for (ir = 0; ir < obs->nr; ir++) {
03137
03138         /* Get radiance and transmittance profiles... */
03139         nz = 0;
03140         for (ir2 = GSL_MAX(ir - NFOV, 0); ir2 < GSL_MIN(ir + 1 + NFOV, obs->nr);
03141              ir2++)
03142             if (obs->time[ir2] == obs->time[ir]) {
03143                 z[nz] = obs2->vpz[ir2];
03144                 for (id = 0; id < ctl->nd; id++) {
03145                     rad[id][nz] = obs2->rad[id][ir2];
03146                     tau[id][nz] = obs2->tau[id][ir2];
03147                 }
03148                 nz++;
03149             }
03150         if (nz < 2)
03151             ERRMSG("Cannot apply FOV convolution!");
03152
03153         /* Convolute profiles with FOV... */
03154         wsum = 0;
03155         for (id = 0; id < ctl->nd; id++) {
03156             obs->rad[id][ir] = 0;
03157             obs->tau[id][ir] = 0;
03158         }
03159         for (i = 0; i < n; i++) {
03160             zfov = obs->vpz[ir] + dz[i];
03161             idx = locate_irr(z, nz, zfov);
03162             for (id = 0; id < ctl->nd; id++) {
03163                 obs->rad[id][ir] += w[i]
03164                     * LIN(z[idx], rad[id][idx], z[idx + 1], rad[id][idx + 1], zfov);
03165                 obs->tau[id][ir] += w[i]
03166                     * LIN(z[idx], tau[id][idx], z[idx + 1], tau[id][idx + 1], zfov);
03167             }
03168         }
03169     }

```



```

03169     wsum += w[i];
03170 }
03171 for (id = 0; id < ctl->nd; id++) {
03172     obs->rad[id][ir] /= wsum;
03173     obs->tau[id][ir] /= wsum;
03174 }
03175 }
03176
03177 /* Free... */
03178 free(obs2);
03179 }
03180
03181 /*****
03182
03183 void formod_pencil(
03184     ctl_t * ctl,
03185     atm_t * atm,
03186     obs_t * obs,
03187     int ir) {
03188
03189     static tbl_t *tbl;
03190
03191     static int init = 0;
03192
03193     los_t *los;
03194
03195     double beta_ctm[ND], eps, src_planck[ND], tau_path[NG][ND], tau_gas[ND];
03196
03197     int id, ip;
03198
03199     /* Initialize look-up tables... */
03200     if (!init) {
03201         init = 1;
03202         ALLOC(tbl, tbl_t, 1);
03203         init_tbl(ctl, tbl);
03204     }
03205
03206     /* Allocate... */
03207     ALLOC(los, los_t, 1);
03208
03209     /* Initialize... */
03210     for (id = 0; id < ctl->nd; id++) {
03211         obs->rad[id][ir] = 0;
03212         obs->tau[id][ir] = 1;
03213     }
03214
03215     /* Raytracing... */
03216     raytrace(ctl, atm, obs, los, ir);
03217
03218     /* Loop over LOS points... */
03219     for (ip = 0; ip < los->np; ip++) {
03220
03221         /* Get trace gas transmittance... */
03222         intpol_tbl(ctl, tbl, los, ip, tau_path, tau_gas);
03223
03224         /* Get continuum absorption... */
03225         formod_continua(ctl, los, ip, beta_ctm);
03226
03227         /* Compute Planck function... */
03228         formod_srcfunc(ctl, tbl, los->t[ip], src_planck);
03229
03230         /* Loop over channels... */
03231         for (id = 0; id < ctl->nd; id++)
03232             if (tau_gas[id] > 0) {
03233
03234                 /* Get segment emissivity... */
03235                 eps = 1 - tau_gas[id] * exp(-beta_ctm[id] * los->ds[ip]);
03236
03237                 /* Compute radiance... */
03238                 obs->rad[id][ir] += src_planck[id] * eps * obs->tau[id][ir];
03239
03240                 /* Compute path transmittance... */
03241                 obs->tau[id][ir] *= (1 - eps);
03242             }
03243     }
03244
03245     /* Add surface... */
03246     if (los->tsurf > 0) {
03247         formod_srcfunc(ctl, tbl, los->tsurf, src_planck);
03248         for (id = 0; id < ctl->nd; id++)
03249             obs->rad[id][ir] += src_planck[id] * obs->tau[id][ir];
03250     }
03251
03252     /* Free... */
03253     free(los);
03254 }
03255

```

```

03256 /*****
03257
03258 void formod_srcfunc(
03259     ctl_t * ctl,
03260     tbl_t * tbl,
03261     double t,
03262     double *src) {
03263
03264     int id, it;
03265
03266     /* Determine index in temperature array... */
03267     it = locate_reg(tbl->st, TBLNS, t);
03268
03269     /* Interpolate Planck function value... */
03270     for (id = 0; id < ctl->nd; id++)
03271         src[id] = LIN(tbl->st[it], tbl->sr[id][it],
03272                     tbl->st[it + 1], tbl->sr[id][it + 1], t);
03273 }
03274
03275 /*****
03276
03277 void geo2cart(
03278     double z,
03279     double lon,
03280     double lat,
03281     double *x) {
03282
03283     double radius;
03284
03285     radius = z + RE;
03286     x[0] = radius * cos(lat / 180 * M_PI) * cos(lon / 180 * M_PI);
03287     x[1] = radius * cos(lat / 180 * M_PI) * sin(lon / 180 * M_PI);
03288     x[2] = radius * sin(lat / 180 * M_PI);
03289 }
03290
03291 /*****
03292
03293 void hydrostatic(
03294     ctl_t * ctl,
03295     atm_t * atm) {
03296
03297     static int ig_h2o = -999;
03298
03299     double dzmin = 1e99, e = 0, mean, mmair = 28.96456e-3, mmh2o = 18.0153e-3;
03300
03301     int i, ip, ipref = 0, ipt = 0;
03302
03303     /* Check reference height... */
03304     if (ctl->hyd < 0)
03305         return;
03306
03307     /* Determine emitter index of H2O... */
03308     if (ig_h2o == -999)
03309         ig_h2o = find_emitter(ctl, "H2O");
03310
03311     /* Find air parcel next to reference height... */
03312     for (ip = 0; ip < atm->np; ip++)
03313         if (fabs(atm->z[ip] - ctl->hyd) < dzmin) {
03314             dzmin = fabs(atm->z[ip] - ctl->hyd);
03315             ipref = ip;
03316         }
03317
03318     /* Upper part of profile... */
03319     for (ip = ipref + 1; ip < atm->np; ip++) {
03320         mean = 0;
03321         for (i = 0; i < ipt; i++) {
03322             if (ig_h2o >= 0)
03323                 e = LIN(0.0, atm->q[ig_h2o][ip - 1],
03324                     ipt - 1.0, atm->q[ig_h2o][ip], (double) i);
03325             mean += (e * mmh2o + (1 - e) * mmair)
03326                 * G0 / RI
03327                 / LIN(0.0, atm->t[ip - 1], ipt - 1.0, atm->t[ip], (double) i) / ipt;
03328         }
03329
03330         /* Compute p(z,T)... */
03331         atm->p[ip] =
03332             exp(log(atm->p[ip - 1]) - mean * 1000 * (atm->z[ip] - atm->z[ip - 1]));
03333     }
03334
03335     /* Lower part of profile... */
03336     for (ip = ipref - 1; ip >= 0; ip--) {
03337         mean = 0;
03338         for (i = 0; i < ipt; i++) {
03339             if (ig_h2o >= 0)
03340                 e = LIN(0.0, atm->q[ig_h2o][ip + 1],
03341                     ipt - 1.0, atm->q[ig_h2o][ip], (double) i);
03342             mean += (e * mmh2o + (1 - e) * mmair)

```

```

03343         * G0 / RI
03344         / LIN(0.0, atm->t[ip + 1], ipt - 1.0, atm->t[ip], (double) i) / ipt;
03345     }
03346
03347     /* Compute p(z,T)... */
03348     atm->p[ip] =
03349     exp(log(atm->p[ip + 1]) - mean * 1000 * (atm->z[ip] - atm->z[ip + 1]));
03350 }
03351 }
03352
03353 /*****
03354
03355 void idx2name(
03356     ctl_t * ctl,
03357     int idx,
03358     char *quantity) {
03359
03360     int ig, iw;
03361
03362     if (idx == IDXP)
03363         sprintf(quantity, "PRESSURE");
03364
03365     if (idx == IDXT)
03366         sprintf(quantity, "TEMPERATURE");
03367
03368     for (ig = 0; ig < ctl->ng; ig++)
03369         if (idx == IDXQ(ig))
03370             sprintf(quantity, "%s", ctl->emitter[ig]);
03371
03372     for (iw = 0; iw < ctl->nw; iw++)
03373         if (idx == IDXK(iw))
03374             sprintf(quantity, "EXTINCT_WINDOW%d", iw);
03375 }
03376
03377 /*****
03378
03379 void init_tbl(
03380     ctl_t * ctl,
03381     tbl_t * tbl) {
03382
03383     FILE *in;
03384
03385     char filename[2 * LEN], line[LEN];
03386
03387     double eps, eps_old, press, press_old, temp, temp_old, u, u_old,
03388         f[NSHAPE], fsum, nu[NSHAPE];
03389
03390     int i, id, ig, ip, it, n;
03391
03392     /* Loop over trace gases and channels... */
03393     for (ig = 0; ig < ctl->ng; ig++)
03394 #pragma omp parallel for default(none) shared(ctl,tbl,ig) private(in,filename,line,eps,eps_old,press,
03395     press_old,temp,temp_old,u,u_old,id,ip,it)
03396         for (id = 0; id < ctl->nd; id++) {
03397
03398         /* Initialize... */
03399         tbl->np[ig][id] = -1;
03400         eps_old = -999;
03401         press_old = -999;
03402         temp_old = -999;
03403         u_old = -999;
03404
03405         /* Try to open file... */
03406         sprintf(filename, "%s_%.4f_%.s.tab",
03407             ctl->tblbase, ctl->nu[id], ctl->emitter[ig]);
03408         if (!(in = fopen(filename, "r"))) {
03409             printf("Missing emissivity table: %s\n", filename);
03410             continue;
03411         }
03412         printf("Read emissivity table: %s\n", filename);
03413
03414         /* Read data... */
03415         while (fgets(line, LEN, in)) {
03416
03417             /* Parse line... */
03418             if (sscanf(line, "%lg %lg %lg %lg", &press, &temp, &u, &eps) != 4)
03419                 continue;
03420
03421             /* Determine pressure index... */
03422             if (press != press_old) {
03423                 press_old = press;
03424                 if ((tbl->np[ig][id]) >= TBLNP)
03425                     ERRMSG("Too many pressure levels!");
03426                 tbl->nt[ig][id][tbl->np[ig][id]] = -1;
03427             }
03428
03429             /* Determine temperature index... */

```

```

03429     if (temp != temp_old) {
03430         temp_old = temp;
03431         if ((++tbl->nt[ig][id][tbl->np[ig][id]] >= TBLNT)
03432             ERRMSG("Too many temperatures!");
03433         tbl->nu[ig][id][tbl->np[ig][id]]
03434             [tbl->nt[ig][id][tbl->np[ig][id]]] = -1;
03435     }
03436
03437     /* Determine column density index... */
03438     if ((eps > eps_old && u > u_old) || tbl->nu[ig][id][tbl->np[ig][id]]
03439         [tbl->nt[ig][id][tbl->np[ig][id]]] < 0) {
03440         eps_old = eps;
03441         u_old = u;
03442         if ((++tbl->nu[ig][id][tbl->np[ig][id]]
03443             [tbl->nt[ig][id][tbl->np[ig][id]]] >= TBLNU) {
03444             tbl->nu[ig][id][tbl->np[ig][id]]
03445                 [tbl->nt[ig][id][tbl->np[ig][id]]]--;
03446             continue;
03447         }
03448     }
03449
03450     /* Store data... */
03451     tbl->p[ig][id][tbl->np[ig][id]] = press;
03452     tbl->t[ig][id][tbl->np[ig][id]][tbl->nt[ig][id][tbl->np[ig][id]]]
03453         = temp;
03454     tbl->u[ig][id][tbl->np[ig][id]][tbl->nt[ig][id][tbl->np[ig][id]]]
03455         [tbl->nu[ig][id][tbl->np[ig][id]]]
03456         [tbl->nt[ig][id][tbl->np[ig][id]]] = (float) u;
03457     tbl->eps[ig][id][tbl->np[ig][id]][tbl->nt[ig][id][tbl->np[ig][id]]]
03458         [tbl->nu[ig][id][tbl->np[ig][id]]]
03459         [tbl->nt[ig][id][tbl->np[ig][id]]] = (float) eps;
03460 }
03461
03462 /* Increment counters... */
03463 tbl->np[ig][id]++;
03464 for (ip = 0; ip < tbl->np[ig][id]; ip++) {
03465     tbl->nt[ig][id][ip]++;
03466     for (it = 0; it < tbl->nt[ig][id][ip]; it++)
03467         tbl->nu[ig][id][ip][it]++;
03468 }
03469
03470 /* Close file... */
03471 fclose(in);
03472 }
03473
03474 /* Write info... */
03475 printf("Initialize source function table...\n");
03476
03477 /* Loop over channels... */
03478 #pragma omp parallel for default(none) shared(ctl,tbl,ig) private(filename,it,i,n,f,fsum,nu)
03479 for (id = 0; id < ctl->nd; id++) {
03480
03481     /* Read filter function... */
03482     sprintf(filename, "%s_%.4f.filt", ctl->tblbase, ctl->nu[id]);
03483     read_shape(filename, nu, f, &n);
03484
03485     /* Compute source function table... */
03486     for (it = 0; it < TBLNS; it++) {
03487
03488         /* Set temperature... */
03489         tbl->st[it] = LIN(0.0, TMIN, TBLNS - 1.0, TMAX, (double) it);
03490
03491         /* Integrate Planck function... */
03492         fsum = 0;
03493         tbl->sr[id][it] = 0;
03494         for (i = 0; i < n; i++) {
03495             fsum += f[i];
03496             tbl->sr[id][it] += f[i] * planck(tbl->st[it], nu[i]);
03497         }
03498         tbl->sr[id][it] /= fsum;
03499     }
03500 }
03501 }
03502
03503 /*****
03504
03505 void intpol_atm(
03506     ctl_t * ctl,
03507     atm_t * atm,
03508     double z,
03509     double *p,
03510     double *t,
03511     double *q,
03512     double *k) {
03513
03514     int ig, ip, iw;
03515

```

```

03516  /* Get array index... */
03517  ip = locate_irr(atm->z, atm->np, z);
03518
03519  /* Interpolate... */
03520  *p = EXP(atm->z[ip], atm->p[ip], atm->z[ip + 1], atm->p[ip + 1], z);
03521  *t = LIN(atm->z[ip], atm->t[ip], atm->z[ip + 1], atm->t[ip + 1], z);
03522  for (ig = 0; ig < ctl->ng; ig++)
03523      q[ig] =
03524          LIN(atm->z[ip], atm->q[ig][ip], atm->z[ip + 1], atm->q[ig][ip + 1], z);
03525  for (iw = 0; iw < ctl->nw; iw++)
03526      k[iw] =
03527          LIN(atm->z[ip], atm->k[iw][ip], atm->z[ip + 1], atm->k[iw][ip + 1], z);
03528 }
03529
03530 /*****
03531
03532 void intpol_tbl(
03533     ctl_t * ctl,
03534     tbl_t * tbl,
03535     los_t * los,
03536     int ip,
03537     double tau_path[NG][ND],
03538     double tau_seg[ND]) {
03539
03540     double eps, eps00, eps01, eps10, eps11, u;
03541
03542     int id, ig, ipr, it0, it1;
03543
03544     /* Initialize... */
03545     if (ip <= 0)
03546         for (ig = 0; ig < ctl->ng; ig++)
03547             for (id = 0; id < ctl->nd; id++)
03548                 tau_path[ig][id] = 1;
03549
03550     /* Loop over channels... */
03551     for (id = 0; id < ctl->nd; id++) {
03552
03553         /* Initialize... */
03554         tau_seg[id] = 1;
03555
03556         /* Loop over emitters... */
03557         for (ig = 0; ig < ctl->ng; ig++) {
03558
03559             /* Check size of table (pressure)... */
03560             if (tbl->np[ig][id] < 2)
03561                 eps = 0;
03562
03563             /* Check transmittance... */
03564             else if (tau_path[ig][id] < 1e-9)
03565                 eps = 1;
03566
03567             /* Interpolate... */
03568             else {
03569
03570                 /* Determine pressure and temperature indices... */
03571                 ipr = locate_irr(tbl->p[ig][id], tbl->np[ig][id], los->p[ip]);
03572                 it0 =
03573                     locate_irr(tbl->t[ig][id][ipr], tbl->nt[ig][id][ipr], los->
03574 t[ip]);
03575                 it1 =
03576                     locate_reg(tbl->t[ig][id][ipr + 1], tbl->nt[ig][id][ipr + 1],
03577                             los->t[ip]);
03578
03579                 /* Check size of table (temperature and column density)... */
03580                 if (tbl->nt[ig][id][ipr] < 2 || tbl->nt[ig][id][ipr + 1] < 2
03581                     || tbl->nu[ig][id][ipr][it0] < 2
03582                     || tbl->nu[ig][id][ipr][it0 + 1] < 2
03583                     || tbl->nu[ig][id][ipr + 1][it1] < 2
03584                     || tbl->nu[ig][id][ipr + 1][it1 + 1] < 2)
03585                     eps = 0;
03586                 else {
03587
03588                     /* Get emissivities of extended path... */
03589                     u = intpol_tbl_u(tbl, ig, id, ipr, it0, 1 - tau_path[ig][id]);
03590                     eps00 = intpol_tbl_eps(tbl, ig, id, ipr, it0, u + los->u[ip]);
03591
03592                     u = intpol_tbl_u(tbl, ig, id, ipr, it0 + 1, 1 - tau_path[ig][id]);
03593                     eps01 =
03594                         intpol_tbl_eps(tbl, ig, id, ipr, it0 + 1, u + los->u[ip]);
03595
03596                     u = intpol_tbl_u(tbl, ig, id, ipr + 1, it1, 1 - tau_path[ig][id]);
03597                     eps10 =
03598                         intpol_tbl_eps(tbl, ig, id, ipr + 1, it1, u + los->u[ip]);
03599
03600                     u =
03601                         intpol_tbl_u(tbl, ig, id, ipr + 1, it1 + 1, 1 - tau_path[ig][id]);

```

```

03602         eps11 =
03603         intpol_tbl_eps(tbl, ig, id, ipr + 1, it1 + 1, u + los->
u[ig][ip]);
03604
03605         /* Interpolate with respect to temperature... */
03606         eps00 = LIN(tbl->t[ig][id][ipr][it0], eps00,
03607         tbl->t[ig][id][ipr][it0 + 1], eps01, los->t[ip]);
03608         eps11 = LIN(tbl->t[ig][id][ipr + 1][it1], eps10,
03609         tbl->t[ig][id][ipr + 1][it1 + 1], eps11, los->t[ip]);
03610
03611         /* Interpolate with respect to pressure... */
03612         eps00 = LIN(tbl->p[ig][id][ipr], eps00,
03613         tbl->p[ig][id][ipr + 1], eps11, los->p[ip]);
03614
03615         /* Check emssivity range... */
03616         eps00 = GSL_MAX(GSL_MIN(eps00, 1), 0);
03617
03618         /* Determine segment emissivity... */
03619         eps = 1 - (1 - eps00) / tau_path[ig][id];
03620     }
03621 }
03622
03623 /* Get transmittance of extended path... */
03624 tau_path[ig][id] *= (1 - eps);
03625
03626 /* Get segment transmittance... */
03627 tau_seg[id] *= (1 - eps);
03628 }
03629 }
03630 }
03631
03632 /*****
03633
03634 double intpol_tbl_eps(
03635     tbl_t * tbl,
03636     int ig,
03637     int id,
03638     int ip,
03639     int it,
03640     double u) {
03641
03642     int idx;
03643
03644     /* Lower boundary... */
03645     if (u < tbl->u[ig][id][ip][it][0])
03646         return LIN(0, 0, tbl->u[ig][id][ip][it][0], tbl->eps[ig][id][ip][it][0],
03647         u);
03648
03649     /* Upper boundary... */
03650     else if (u > tbl->u[ig][id][ip][it][tbl->nu[ig][id][ip][it] - 1])
03651         return LIN(tbl->u[ig][id][ip][it][tbl->nu[ig][id][ip][it] - 1],
03652         tbl->eps[ig][id][ip][it][tbl->nu[ig][id][ip][it] - 1],
03653         1e30, 1, u);
03654
03655     /* Interpolation... */
03656     else {
03657
03658         /* Get index... */
03659         idx = locate_tbl(tbl->u[ig][id][ip][it], tbl->nu[ig][id][ip][it], u);
03660
03661         /* Interpolate... */
03662         return
03663         LIN(tbl->u[ig][id][ip][it][idx], tbl->eps[ig][id][ip][it][idx],
03664         tbl->u[ig][id][ip][it][idx + 1], tbl->eps[ig][id][ip][it][idx + 1],
03665         u);
03666     }
03667 }
03668
03669 /*****
03670
03671 double intpol_tbl_u(
03672     tbl_t * tbl,
03673     int ig,
03674     int id,
03675     int ip,
03676     int it,
03677     double eps) {
03678
03679     int idx;
03680
03681     /* Lower boundary... */
03682     if (eps < tbl->eps[ig][id][ip][it][0])
03683         return LIN(0, 0, tbl->eps[ig][id][ip][it][0], tbl->u[ig][id][ip][it][0],
03684         eps);
03685
03686     /* Upper boundary... */
03687     else if (eps > tbl->eps[ig][id][ip][it][tbl->nu[ig][id][ip][it] - 1])

```

```

03688     return LIN(tbl->eps[ig][id][ip][it][tbl->nu[ig][id][ip][it] - 1],
03689                tbl->u[ig][id][ip][it][tbl->nu[ig][id][ip][it] - 1],
03690                1, 1e30, eps);
03691
03692     /* Interpolation... */
03693     else {
03694
03695         /* Get index... */
03696         idx = locate_tbl(tbl->eps[ig][id][ip][it], tbl->nu[ig][id][ip][it], eps);
03697
03698         /* Interpolate... */
03699         return
03700             LIN(tbl->eps[ig][id][ip][it][idx], tbl->u[ig][id][ip][it][idx],
03701                tbl->eps[ig][id][ip][it][idx + 1], tbl->u[ig][id][ip][it][idx + 1],
03702                eps);
03703     }
03704 }
03705
03706 /*****
03707
03708 void jsec2time(
03709     double jsec,
03710     int *year,
03711     int *mon,
03712     int *day,
03713     int *hour,
03714     int *min,
03715     int *sec,
03716     double *remain) {
03717
03718     struct tm t0, *t1;
03719
03720     time_t jsec0;
03721
03722     t0.tm_year = 100;
03723     t0.tm_mon = 0;
03724     t0.tm_mday = 1;
03725     t0.tm_hour = 0;
03726     t0.tm_min = 0;
03727     t0.tm_sec = 0;
03728
03729     jsec0 = (time_t) jsec + timegm(&t0);
03730     t1 = gmtime(&jsec0);
03731
03732     *year = t1->tm_year + 1900;
03733     *mon = t1->tm_mon + 1;
03734     *day = t1->tm_mday;
03735     *hour = t1->tm_hour;
03736     *min = t1->tm_min;
03737     *sec = t1->tm_sec;
03738     *remain = jsec - floor(jsec);
03739 }
03740
03741 /*****
03742
03743 void kernel(
03744     ctl_t * ctl,
03745     atm_t * atm,
03746     obs_t * obs,
03747     gsl_matrix * k) {
03748
03749     atm_t *atm1;
03750     obs_t *obs1;
03751
03752     gsl_vector *x0, *x1, *yy0, *yy1;
03753
03754     int *iqa, j;
03755
03756     double h;
03757
03758     size_t i, n, m;
03759
03760     /* Get sizes... */
03761     m = k->size1;
03762     n = k->size2;
03763
03764     /* Allocate... */
03765     x0 = gsl_vector_alloc(n);
03766     yy0 = gsl_vector_alloc(m);
03767     ALLOC(iqa, int,
03768           N);
03769
03770     /* Compute radiance for undisturbed atmospheric data... */
03771     formod(ctl, atm, obs);
03772
03773     /* Compose vectors... */
03774     atm2x(ctl, atm, x0, iqa, NULL);

```

```

03775     obs2y(ctl, obs, yy0, NULL, NULL);
03776
03777     /* Initialize kernel matrix... */
03778     gsl_matrix_set_zero(k);
03779
03780     /* Loop over state vector elements... */
03781 #pragma omp parallel for default(none) shared(ctl,atm,obs,k,x0,yy0,n,m,iqa) private(i, j, h, x1, yy1, atm1,
03782     obs1)
03783     for (j = 0; j < (int) n; j++) {
03784
03785         /* Allocate... */
03786         x1 = gsl_vector_alloc(n);
03787         yy1 = gsl_vector_alloc(m);
03788         ALLOC(atm1, atm_t, 1);
03789         ALLOC(obs1, obs_t, 1);
03790
03791         /* Set perturbation size... */
03792         if (iqa[j] == IDXP)
03793             h = GSL_MAX(fabs(0.01 * gsl_vector_get(x0, (size_t) j)), 1e-7);
03794         else if (iqa[j] == IDXT)
03795             h = 1;
03796         else if (iqa[j] >= IDXQ(0) && iqa[j] < IDXQ(ctl->ng))
03797             h = GSL_MAX(fabs(0.01 * gsl_vector_get(x0, (size_t) j)), 1e-15);
03798         else if (iqa[j] >= IDXK(0) && iqa[j] < IDXK(ctl->nw))
03799             h = 1e-4;
03800         else
03801             ERRMSG("Cannot set perturbation size!");
03802
03803         /* Disturb state vector element... */
03804         gsl_vector_memcpy(x1, x0);
03805         gsl_vector_set(x1, (size_t) j, gsl_vector_get(x1, (size_t) j) + h);
03806         copy_atm(ctl, atm1, atm, 0);
03807         copy_obs(ctl, obs1, obs, 0);
03808         x2atm(ctl, x1, atm1);
03809
03810         /* Compute radiance for disturbed atmospheric data... */
03811         formod(ctl, atm1, obs1);
03812
03813         /* Compose measurement vector for disturbed radiance data... */
03814         obs2y(ctl, obs1, yy1, NULL, NULL);
03815
03816         /* Compute derivatives... */
03817         for (i = 0; i < m; i++)
03818             gsl_matrix_set(k, i, (size_t) j,
03819                 (gsl_vector_get(yy1, i) - gsl_vector_get(yy0, i)) / h);
03820
03821         /* Free... */
03822         gsl_vector_free(x1);
03823         gsl_vector_free(yy1);
03824         free(atm1);
03825         free(obs1);
03826     }
03827
03828     /* Free... */
03829     gsl_vector_free(x0);
03830     gsl_vector_free(yy0);
03831     free(iqa);
03832 }
03833
03834 /*****
03835 int locate_irr(
03836     double *xx,
03837     int n,
03838     double x) {
03839
03840     int i, ilo, ihi;
03841
03842     ilo = 0;
03843     ihi = n - 1;
03844     i = (ihi + ilo) >> 1;
03845
03846     if (xx[i] < xx[i + 1])
03847         while (ihi > ilo + 1) {
03848             i = (ihi + ilo) >> 1;
03849             if (xx[i] > x)
03850                 ihi = i;
03851             else
03852                 ilo = i;
03853         } else
03854         while (ihi > ilo + 1) {
03855             i = (ihi + ilo) >> 1;
03856             if (xx[i] <= x)
03857                 ihi = i;
03858             else
03859                 ilo = i;
03860         }

```



```

03861
03862     return ilo;
03863 }
03864
03865 /*****
03866
03867 int locate_reg(
03868     double *xx,
03869     int n,
03870     double x) {
03871
03872     int i;
03873
03874     /* Calculate index... */
03875     i = (int) ((x - xx[0]) / (xx[1] - xx[0]));
03876
03877     /* Check range... */
03878     if (i < 0)
03879         i = 0;
03880     else if (i >= n - 2)
03881         i = n - 2;
03882
03883     return i;
03884 }
03885
03886 /*****
03887
03888 int locate_tbl(
03889     float *xx,
03890     int n,
03891     double x) {
03892
03893     int i, ilo, ihi;
03894
03895     ilo = 0;
03896     ihi = n - 1;
03897     i = (ihi + ilo) >> 1;
03898
03899     while (ihi > ilo + 1) {
03900         i = (ihi + ilo) >> 1;
03901         if (xx[i] > x)
03902             ihi = i;
03903         else
03904             ilo = i;
03905     }
03906
03907     return ilo;
03908 }
03909
03910 /*****
03911
03912 size_t obs2y(
03913     ctl_t * ctl,
03914     obs_t * obs,
03915     gsl_vector * y,
03916     int *ida,
03917     int *ira) {
03918
03919     int id, ir;
03920
03921     size_t m = 0;
03922
03923     /* Determine measurement vector... */
03924     for (ir = 0; ir < obs->nr; ir++)
03925         for (id = 0; id < ctl->nd; id++)
03926             if (gsl_finite(obs->rad[id][ir])) {
03927                 if (y != NULL)
03928                     gsl_vector_set(y, m, obs->rad[id][ir]);
03929                 if (ida != NULL)
03930                     ida[m] = id;
03931                 if (ira != NULL)
03932                     ira[m] = ir;
03933                 m++;
03934             }
03935
03936     return m;
03937 }
03938
03939 /*****
03940
03941 double planck(
03942     double t,
03943     double nu) {
03944
03945     return C1 * POW3(nu) / gsl_expm1(C2 * nu / t);
03946 }
03947

```

```

03948 /*****
03949
03950 void raytrace(
03951     ctl_t * ctl,
03952     atm_t * atm,
03953     obs_t * obs,
03954     los_t * los,
03955     int ir) {
03956
03957     double cosa, d, dmax, dmin = 0, ds, ex0[3], ex1[3], frac, h = 0.02, k[NW],
03958         lat, lon, n, naux, ng[3], norm, p, q[NG], t, x[3], xh[3],
03959         xobs[3], xvp[3], z = 1e99, zmax, zmin, zrefrac = 60;
03960
03961     int i, ig, ip, iw, stop = 0;
03962
03963     /* Initialize... */
03964     los->np = 0;
03965     los->tsurf = -999;
03966     obs->tpz[ir] = obs->vpz[ir];
03967     obs->tplon[ir] = obs->vplon[ir];
03968     obs->tplat[ir] = obs->vplat[ir];
03969
03970     /* Get altitude range of atmospheric data... */
03971     gsl_stats_minmax(&zmin, &zmax, atm->z, 1, (size_t) atm->np);
03972
03973     /* Check observer altitude... */
03974     if (obs->obsz[ir] < zmin)
03975         ERRMSG("Observer below surface!");
03976
03977     /* Check view point altitude... */
03978     if (obs->vpz[ir] > zmax)
03979         return;
03980
03981     /* Determine Cartesian coordinates for observer and view point... */
03982     geo2cart(obs->obsz[ir], obs->obslon[ir], obs->obslat[ir], xobs);
03983     geo2cart(obs->vpz[ir], obs->vplon[ir], obs->vplat[ir], xvp);
03984
03985     /* Determine initial tangent vector... */
03986     for (i = 0; i < 3; i++)
03987         ex0[i] = xvp[i] - xobs[i];
03988     norm = NORM(ex0);
03989     for (i = 0; i < 3; i++)
03990         ex0[i] /= norm;
03991
03992     /* Observer within atmosphere... */
03993     for (i = 0; i < 3; i++)
03994         x[i] = xobs[i];
03995
03996     /* Observer above atmosphere (search entry point)... */
03997     if (obs->obsz[ir] > zmax) {
03998         dmax = norm;
03999         while (fabs(dmin - dmax) > 0.001) {
04000             d = (dmax + dmin) / 2;
04001             for (i = 0; i < 3; i++)
04002                 x[i] = xobs[i] + d * ex0[i];
04003             cart2geo(x, &z, &lon, &lat);
04004             if (z <= zmax && z > zmax - 0.001)
04005                 break;
04006             if (z < zmax - 0.0005)
04007                 dmax = d;
04008             else
04009                 dmin = d;
04010         }
04011     }
04012
04013     /* Ray-tracing... */
04014     while (1) {
04015
04016         /* Set step length... */
04017         ds = ctl->rayds;
04018         if (ctl->raydz > 0) {
04019             norm = NORM(x);
04020             for (i = 0; i < 3; i++)
04021                 xh[i] = x[i] / norm;
04022             cosa = fabs(DOTP(ex0, xh));
04023             if (cosa != 0)
04024                 ds = GSL_MIN(ctl->rayds, ctl->raydz / cosa);
04025         }
04026
04027         /* Determine geolocation... */
04028         cart2geo(x, &z, &lon, &lat);
04029
04030         /* Check if LOS hits the ground or has left atmosphere... */
04031         if (z < zmin || z > zmax) {
04032             stop = (z < zmin ? 2 : 1);
04033             frac =
04034                 ((z <

```

```

04035         zmin ? zmin : zmax) - los->z[los->np - 1]) / (z - los->z[los->np -
04036                                                     1]);
04037     geo2cart(los->z[los->np - 1], los->lon[los->np - 1],
04038             los->lat[los->np - 1], xh);
04039     for (i = 0; i < 3; i++)
04040         x[i] = xh[i] + frac * (x[i] - xh[i]);
04041     cart2geo(x, &z, &lon, &lat);
04042     los->ds[los->np - 1] = ds * frac;
04043     ds = 0;
04044 }
04045
04046 /* Interpolate atmospheric data... */
04047 intpol_atm(ctl, atm, z, &p, &t, q, k);
04048
04049 /* Save data... */
04050 los->lon[los->np] = lon;
04051 los->lat[los->np] = lat;
04052 los->z[los->np] = z;
04053 los->p[los->np] = p;
04054 los->t[los->np] = t;
04055 for (ig = 0; ig < ctl->ng; ig++)
04056     los->q[ig][los->np] = q[ig];
04057 for (iw = 0; iw < ctl->nw; iw++)
04058     los->k[iw][los->np] = k[iw];
04059 los->ds[los->np] = ds;
04060
04061 /* Increment and check number of LOS points... */
04062 if ((++los->np) > NLOS)
04063     ERRMSG("Too many LOS points!");
04064
04065 /* Check stop flag... */
04066 if (stop) {
04067     los->tsurf = (stop == 2 ? t : -999);
04068     break;
04069 }
04070
04071 /* Determine refractivity... */
04072 if (ctl->refrac && z <= zrefrac)
04073     n = 1 + refractivity(p, t);
04074 else
04075     n = 1;
04076
04077 /* Construct new tangent vector (first term)... */
04078 for (i = 0; i < 3; i++)
04079     ex1[i] = ex0[i] * n;
04080
04081 /* Compute gradient of refractivity... */
04082 if (ctl->refrac && z <= zrefrac) {
04083     for (i = 0; i < 3; i++)
04084         xh[i] = x[i] + 0.5 * ds * ex0[i];
04085     cart2geo(xh, &z, &lon, &lat);
04086     intpol_atm(ctl, atm, z, &p, &t, q, k);
04087     n = refractivity(p, t);
04088     for (i = 0; i < 3; i++) {
04089         xh[i] += h;
04090         cart2geo(xh, &z, &lon, &lat);
04091         intpol_atm(ctl, atm, z, &p, &t, q, k);
04092         naux = refractivity(p, t);
04093         ng[i] = (naux - n) / h;
04094         xh[i] -= h;
04095     }
04096 } else
04097     for (i = 0; i < 3; i++)
04098         ng[i] = 0;
04099
04100 /* Construct new tangent vector (second term)... */
04101 for (i = 0; i < 3; i++)
04102     ex1[i] += ds * ng[i];
04103
04104 /* Normalize new tangent vector... */
04105 norm = NORM(ex1);
04106 for (i = 0; i < 3; i++)
04107     ex1[i] /= norm;
04108
04109 /* Determine next point of LOS... */
04110 for (i = 0; i < 3; i++)
04111     x[i] += 0.5 * ds * (ex0[i] + ex1[i]);
04112
04113 /* Copy tangent vector... */
04114 for (i = 0; i < 3; i++)
04115     ex0[i] = ex1[i];
04116 }
04117
04118 /* Get tangent point (to be done before changing segment lengths!)... */
04119 tangent_point(los, &obs->tpz[ir], &obs->tplon[ir], &obs->
04120             tplat[ir]);

```

```

04121  /* Change segment lengths according to trapezoid rule... */
04122  for (ip = los->np - 1; ip >= 1; ip--)
04123      los->ds[ip] = 0.5 * (los->ds[ip - 1] + los->ds[ip]);
04124  los->ds[0] *= 0.5;
04125
04126  /* Compute column density... */
04127  for (ip = 0; ip < los->np; ip++)
04128      for (ig = 0; ig < ctl->ng; ig++)
04129          los->u[ig][ip] = 10 * los->q[ig][ip] * los->p[ip]
04130          / (KB * los->t[ip]) * los->ds[ip];
04131 }
04132
04133 /*****
04134
04135 void read_atm(
04136     const char *dirname,
04137     const char *filename,
04138     ctl_t *ctl,
04139     atm_t *atm) {
04140
04141     FILE *in;
04142
04143     char file[LEN], line[LEN], *tok;
04144
04145     int ig, iw;
04146
04147     /* Init... */
04148     atm->np = 0;
04149
04150     /* Set filename... */
04151     if (dirname != NULL)
04152         sprintf(file, "%s/%s", dirname, filename);
04153     else
04154         sprintf(file, "%s", filename);
04155
04156     /* Write info... */
04157     printf("Read atmospheric data: %s\n", file);
04158
04159     /* Open file... */
04160     if (!(in = fopen(file, "r")))
04161         ERRMSG("Cannot open file!");
04162
04163     /* Read line... */
04164     while (fgets(line, LEN, in)) {
04165
04166         /* Read data... */
04167         TOK(line, tok, "%lg", atm->time[atm->np]);
04168         TOK(NULL, tok, "%lg", atm->z[atm->np]);
04169         TOK(NULL, tok, "%lg", atm->lon[atm->np]);
04170         TOK(NULL, tok, "%lg", atm->lat[atm->np]);
04171         TOK(NULL, tok, "%lg", atm->p[atm->np]);
04172         TOK(NULL, tok, "%lg", atm->t[atm->np]);
04173         for (ig = 0; ig < ctl->ng; ig++)
04174             TOK(NULL, tok, "%lg", atm->q[ig][atm->np]);
04175         for (iw = 0; iw < ctl->nw; iw++)
04176             TOK(NULL, tok, "%lg", atm->k[iw][atm->np]);
04177
04178         /* Increment data point counter... */
04179         if ((++atm->np) > NP)
04180             ERRMSG("Too many data points!");
04181     }
04182
04183     /* Close file... */
04184     fclose(in);
04185
04186     /* Check number of points... */
04187     if (atm->np < 1)
04188         ERRMSG("Could not read any data!");
04189 }
04190
04191 /*****
04192
04193 void read_ctl(
04194     int argc,
04195     char *argv[],
04196     ctl_t *ctl) {
04197
04198     int id, ig, iw;
04199
04200     /* Write info... */
04201     printf("\nJuelich Rapid Spectral Simulation Code (JURASSIC)\n"
04202           "(executable: %s | compiled: %s, %s)\n\n",
04203           argv[0], __DATE__, __TIME__);
04204
04205     /* Emitters... */
04206     ctl->ng = (int) scan_ctl(argc, argv, "NG", -1, "0", NULL);
04207     if (ctl->ng < 0 || ctl->ng > NG)

```

```

04208     ERRMSG("Set 0 <= NG <= MAX!");
04209     for (ig = 0; ig < ctl->ng; ig++)
04210         scan_ctl(argc, argv, "EMITTER", ig, "", ctl->emitter[ig]);
04211
04212     /* Radiance channels... */
04213     ctl->nd = (int) scan_ctl(argc, argv, "ND", -1, "0", NULL);
04214     if (ctl->nd < 0 || ctl->nd > ND)
04215         ERRMSG("Set 0 <= ND <= MAX!");
04216     for (id = 0; id < ctl->nd; id++)
04217         ctl->nu[id] = scan_ctl(argc, argv, "NU", id, "", NULL);
04218
04219     /* Spectral windows... */
04220     ctl->nw = (int) scan_ctl(argc, argv, "NW", -1, "1", NULL);
04221     if (ctl->nw < 0 || ctl->nw > NW)
04222         ERRMSG("Set 0 <= NW <= MAX!");
04223     for (id = 0; id < ctl->nd; id++)
04224         ctl->window[id] = (int) scan_ctl(argc, argv, "WINDOW", id, "0", NULL);
04225
04226     /* Emissivity look-up tables... */
04227     scan_ctl(argc, argv, "TBLBASE", -1, "-", ctl->tblbase);
04228
04229     /* Hydrostatic equilibrium... */
04230     ctl->hydZ = scan_ctl(argc, argv, "HYDZ", -1, "-999", NULL);
04231
04232     /* Continua... */
04233     ctl->ctm_co2 = (int) scan_ctl(argc, argv, "CTM_CO2", -1, "1", NULL);
04234     ctl->ctm_h2o = (int) scan_ctl(argc, argv, "CTM_H2O", -1, "1", NULL);
04235     ctl->ctm_n2 = (int) scan_ctl(argc, argv, "CTM_N2", -1, "1", NULL);
04236     ctl->ctm_o2 = (int) scan_ctl(argc, argv, "CTM_O2", -1, "1", NULL);
04237
04238     /* Ray-tracing... */
04239     ctl->refrac = (int) scan_ctl(argc, argv, "REFRAC", -1, "1", NULL);
04240     ctl->rayds = scan_ctl(argc, argv, "RAYDS", -1, "10", NULL);
04241     ctl->raydz = scan_ctl(argc, argv, "RAYDZ", -1, "0.5", NULL);
04242
04243     /* Field of view... */
04244     scan_ctl(argc, argv, "FOV", -1, "-", ctl->fov);
04245
04246     /* Retrieval interface... */
04247     ctl->retp_zmin = scan_ctl(argc, argv, "RETP_ZMIN", -1, "-999", NULL);
04248     ctl->retp_zmax = scan_ctl(argc, argv, "RETP_ZMAX", -1, "-999", NULL);
04249     ctl->rett_zmin = scan_ctl(argc, argv, "RETT_ZMIN", -1, "-999", NULL);
04250     ctl->rett_zmax = scan_ctl(argc, argv, "RETT_ZMAX", -1, "-999", NULL);
04251     for (ig = 0; ig < ctl->ng; ig++) {
04252         ctl->retq_zmin[ig] = scan_ctl(argc, argv, "RETQ_ZMIN", ig, "-999", NULL);
04253         ctl->retq_zmax[ig] = scan_ctl(argc, argv, "RETQ_ZMAX", ig, "-999", NULL);
04254     }
04255     for (iw = 0; iw < ctl->nw; iw++) {
04256         ctl->retk_zmin[iw] = scan_ctl(argc, argv, "RETK_ZMIN", iw, "-999", NULL);
04257         ctl->retk_zmax[iw] = scan_ctl(argc, argv, "RETK_ZMAX", iw, "-999", NULL);
04258     }
04259
04260     /* Output flags... */
04261     ctl->write_bbt = (int) scan_ctl(argc, argv, "WRITE_BBT", -1, "0", NULL);
04262     ctl->write_matrix =
04263         (int) scan_ctl(argc, argv, "WRITE_MATRIX", -1, "0", NULL);
04264 }
04265
04266 /*****
04267
04268 void read_matrix(
04269     const char *dirname,
04270     const char *filename,
04271     gsl_matrix * matrix) {
04272
04273     FILE *in;
04274
04275     char dum[LEN], file[LEN], line[LEN];
04276
04277     double value;
04278
04279     int i, j;
04280
04281     /* Set filename... */
04282     if (dirname != NULL)
04283         sprintf(file, "%s/%s", dirname, filename);
04284     else
04285         sprintf(file, "%s", filename);
04286
04287     /* Write info... */
04288     printf("Read matrix: %s\n", file);
04289
04290     /* Open file... */
04291     if (!(in = fopen(file, "r")))
04292         ERRMSG("Cannot open file!");
04293
04294     /* Read data... */

```

```

04295     gsl_matrix_set_zero(matrix);
04296     while (fgets(line, LEN, in))
04297         if (sscanf(line, "%d %s %s %s %s %d %s %s %s %s %s %lg",
04298             &i, dum, dum, dum, dum, dum,
04299             &j, dum, dum, dum, dum, dum, &value) == 13)
04300         gsl_matrix_set(matrix, (size_t) i, (size_t) j, value);
04301
04302     /* Close file... */
04303     fclose(in);
04304 }
04305
04306 /*****
04307
04308 void read_obs(
04309     const char *dirname,
04310     const char *filename,
04311     ctl_t * ctl,
04312     obs_t * obs) {
04313
04314     FILE *in;
04315
04316     char file[LEN], line[LEN], *tok;
04317
04318     int id;
04319
04320     /* Init... */
04321     obs->nr = 0;
04322
04323     /* Set filename... */
04324     if (dirname != NULL)
04325         sprintf(file, "%s/%s", dirname, filename);
04326     else
04327         sprintf(file, "%s", filename);
04328
04329     /* Write info... */
04330     printf("Read observation data: %s\n", file);
04331
04332     /* Open file... */
04333     if (!(in = fopen(file, "r")))
04334         ERRMSG("Cannot open file!");
04335
04336     /* Read line... */
04337     while (fgets(line, LEN, in)) {
04338
04339         /* Read data... */
04340         TOK(line, tok, "%lg", obs->time[obs->nr]);
04341         TOK(NULL, tok, "%lg", obs->obsz[obs->nr]);
04342         TOK(NULL, tok, "%lg", obs->obslon[obs->nr]);
04343         TOK(NULL, tok, "%lg", obs->obslat[obs->nr]);
04344         TOK(NULL, tok, "%lg", obs->vpz[obs->nr]);
04345         TOK(NULL, tok, "%lg", obs->vplon[obs->nr]);
04346         TOK(NULL, tok, "%lg", obs->vplat[obs->nr]);
04347         TOK(NULL, tok, "%lg", obs->tpz[obs->nr]);
04348         TOK(NULL, tok, "%lg", obs->tplon[obs->nr]);
04349         TOK(NULL, tok, "%lg", obs->tplat[obs->nr]);
04350         for (id = 0; id < ctl->nd; id++)
04351             TOK(NULL, tok, "%lg", obs->rad[id][obs->nr]);
04352         for (id = 0; id < ctl->nd; id++)
04353             TOK(NULL, tok, "%lg", obs->tau[id][obs->nr]);
04354
04355         /* Increment counter... */
04356         if ((++obs->nr) > NR)
04357             ERRMSG("Too many rays!");
04358     }
04359
04360     /* Close file... */
04361     fclose(in);
04362
04363     /* Check number of points... */
04364     if (obs->nr < 1)
04365         ERRMSG("Could not read any data!");
04366 }
04367
04368 /*****
04369
04370 void read_shape(
04371     const char *filename,
04372     double *x,
04373     double *y,
04374     int *n) {
04375
04376     FILE *in;
04377
04378     char line[LEN];
04379
04380     /* Write info... */
04381     printf("Read shape function: %s\n", filename);

```

```

04382
04383 /* Open file... */
04384 if (!(in = fopen(filename, "r")))
04385     ERRMSG("Cannot open file!");
04386
04387 /* Read data... */
04388 *n = 0;
04389 while (fgets(line, LEN, in))
04390     if (sscanf(line, "%lg %lg", &x[*n], &y[*n]) == 2)
04391         if ((++(*n)) > NSHAPE)
04392             ERRMSG("Too many data points!");
04393
04394 /* Check number of points... */
04395 if (*n < 1)
04396     ERRMSG("Could not read any data!");
04397
04398 /* Close file... */
04399 fclose(in);
04400 }
04401
04402 /*****
04403
04404 double refractivity(
04405     double p,
04406     double t) {
04407
04408     /* Refractivity of air at 4 to 15 micron... */
04409     return 7.753e-05 * p / t;
04410 }
04411
04412 *****/
04413
04414 double scan_ctl(
04415     int argc,
04416     char *argv[],
04417     const char *varname,
04418     int arridx,
04419     const char *defvalue,
04420     char *value) {
04421
04422     FILE *in = NULL;
04423
04424     char dummy[LEN], fullname1[LEN], fullname2[LEN], line[LEN],
04425         msg[2 * LEN], rvarname[LEN], rval[LEN];
04426
04427     int contain = 0, i;
04428
04429     /* Open file... */
04430     if (argv[1][0] != '-')
04431         if (!(in = fopen(argv[1], "r")))
04432             ERRMSG("Cannot open file!");
04433
04434     /* Set full variable name... */
04435     if (arridx >= 0) {
04436         sprintf(fullname1, "%s[%d]", varname, arridx);
04437         sprintf(fullname2, "%s[*]", varname);
04438     } else {
04439         sprintf(fullname1, "%s", varname);
04440         sprintf(fullname2, "%s", varname);
04441     }
04442
04443     /* Read data... */
04444     if (in != NULL)
04445         while (fgets(line, LEN, in))
04446             if (sscanf(line, "%s %s %s", rvarname, dummy, rval) == 3)
04447                 if (strcasecmp(rvarname, fullname1) == 0 ||
04448                     strcasecmp(rvarname, fullname2) == 0) {
04449                     contain = 1;
04450                     break;
04451                 }
04452     for (i = 1; i < argc - 1; i++)
04453         if (strcasecmp(argv[i], fullname1) == 0 ||
04454             strcasecmp(argv[i], fullname2) == 0) {
04455             sprintf(rval, "%s", argv[i + 1]);
04456             contain = 1;
04457             break;
04458         }
04459
04460     /* Close file... */
04461     if (in != NULL)
04462         fclose(in);
04463
04464     /* Check for missing variables... */
04465     if (!contain) {
04466         if (strlen(defvalue) > 0)
04467             sprintf(rval, "%s", defvalue);
04468         else {

```

```

04469     sprintf(msg, "Missing variable %s!\n", fullnamel);
04470     ERRMSG(msg);
04471 }
04472 }
04473
04474 /* Write info... */
04475 printf("%s = %s\n", fullnamel, rval);
04476
04477 /* Return values... */
04478 if (value != NULL)
04479     sprintf(value, "%s", rval);
04480 return atof(rval);
04481 }
04482
04483 /*****
04484
04485 void tangent_point(
04486     los_t * los,
04487     double *tpz,
04488     double *tplon,
04489     double *tplat) {
04490
04491     double a, b, c, dummy, v[3], v0[3], v2[3], x, x1, x2, yy0, yy1, yy2;
04492
04493     size_t i, ip;
04494
04495     /* Find minimum altitude... */
04496     ip = gsl_stats_min_index(los->z, 1, (size_t) los->np);
04497
04498     /* Nadir or zenith... */
04499     if (ip <= 0 || ip >= (size_t) los->np - 1) {
04500         *tpz = los->z[los->np - 1];
04501         *tplon = los->lon[los->np - 1];
04502         *tplat = los->lat[los->np - 1];
04503     }
04504
04505     /* Limb... */
04506     else {
04507
04508         /* Determine interpolating polynomial y=a*x^2+b*x+c... */
04509         yy0 = los->z[ip - 1];
04510         yy1 = los->z[ip];
04511         yy2 = los->z[ip + 1];
04512         x1 = sqrt(POW2(los->ds[ip]) - POW2(yy1 - yy0));
04513         x2 = x1 + sqrt(POW2(los->ds[ip + 1]) - POW2(yy2 - yy1));
04514         a = 1 / (x1 - x2) * (-(yy0 - yy1) / x1 + (yy0 - yy2) / x2);
04515         b = -(yy0 - yy1) / x1 - a * x1;
04516         c = yy0;
04517
04518         /* Get tangent point location... */
04519         x = -b / (2 * a);
04520         *tpz = a * x * x + b * x + c;
04521         geo2cart(los->z[ip - 1], los->lon[ip - 1], los->lat[ip - 1], v0);
04522         geo2cart(los->z[ip + 1], los->lon[ip + 1], los->lat[ip + 1], v2);
04523         for (i = 0; i < 3; i++)
04524             v[i] = LIN(0.0, v0[i], x2, v2[i], x);
04525         cart2geo(v, &dummy, tplon, tplat);
04526     }
04527 }
04528
04529 /*****
04530
04531 void time2jsec(
04532     int year,
04533     int mon,
04534     int day,
04535     int hour,
04536     int min,
04537     int sec,
04538     double remain,
04539     double *jsec) {
04540
04541     struct tm t0, t1;
04542
04543     t0.tm_year = 100;
04544     t0.tm_mon = 0;
04545     t0.tm_mday = 1;
04546     t0.tm_hour = 0;
04547     t0.tm_min = 0;
04548     t0.tm_sec = 0;
04549
04550     t1.tm_year = year - 1900;
04551     t1.tm_mon = mon - 1;
04552     t1.tm_mday = day;
04553     t1.tm_hour = hour;
04554     t1.tm_min = min;
04555     t1.tm_sec = sec;

```



```

04556
04557     *jsec = (double) timegm(&t1) - (double) timegm(&t0) + remain;
04558 }
04559
04560 /*****
04561
04562 void timer(
04563     const char *name,
04564     const char *file,
04565     const char *func,
04566     int line,
04567     int mode) {
04568
04569     static double w0[10];
04570
04571     static int l0[10], nt;
04572
04573     /* Start new timer... */
04574     if (mode == 1) {
04575         w0[nt] = omp_get_wtime();
04576         l0[nt] = line;
04577         if ((++nt) >= 10)
04578             ERRMSG("Too many timers!");
04579     }
04580
04581     /* Write elapsed time... */
04582     else {
04583
04584         /* Check timer index... */
04585         if (nt - 1 < 0)
04586             ERRMSG("Coding error!");
04587
04588         /* Write elapsed time... */
04589         printf("Timer '%s' (%s, %s, l%d-%d): %.3f sec\n",
04590             name, file, func, l0[nt - 1], line, omp_get_wtime() - w0[nt - 1]);
04591     }
04592
04593     /* Stop timer... */
04594     if (mode == 3)
04595         nt--;
04596 }
04597
04598 /*****
04599
04600 void write_atm(
04601     const char *dirname,
04602     const char *filename,
04603     ctl_t * ctl,
04604     atm_t * atm) {
04605
04606     FILE *out;
04607
04608     char file[LEN];
04609
04610     int ig, ip, iw, n = 6;
04611
04612     /* Set filename... */
04613     if (dirname != NULL)
04614         sprintf(file, "%s/%s", dirname, filename);
04615     else
04616         sprintf(file, "%s", filename);
04617
04618     /* Write info... */
04619     printf("Write atmospheric data: %s\n", file);
04620
04621     /* Create file... */
04622     if (!(out = fopen(file, "w")))
04623         ERRMSG("Cannot create file!");
04624
04625     /* Write header... */
04626     fprintf(out,
04627         "# $1 = time (seconds since 2000-01-01T00:00Z)\n"
04628         "# $2 = altitude [km]\n"
04629         "# $3 = longitude [deg]\n"
04630         "# $4 = latitude [deg]\n"
04631         "# $5 = pressure [hPa]\n" "# $6 = temperature [K]\n");
04632     for (ig = 0; ig < ctl->ng; ig++)
04633         fprintf(out, "# $%d = %s volume mixing ratio\n", ++n, ctl->emitter[ig]);
04634     for (iw = 0; iw < ctl->nw; iw++)
04635         fprintf(out, "# $%d = window %d: extinction [1/km]\n", ++n, iw);
04636
04637     /* Write data... */
04638     for (ip = 0; ip < atm->np; ip++) {
04639         if (ip == 0 || atm->lat[ip] != atm->lat[ip - 1]
04640             || atm->lon[ip] != atm->lon[ip - 1])
04641             fprintf(out, "\n");
04642         fprintf(out, "%.2f %g %g %g %g %g", atm->time[ip], atm->z[ip],

```

```

04643         atm->lon[ip], atm->lat[ip], atm->p[ip], atm->t[ip]);
04644     for (ig = 0; ig < ctl->ng; ig++)
04645         fprintf(out, " %g", atm->q[ig][ip]);
04646     for (iw = 0; iw < ctl->nw; iw++)
04647         fprintf(out, " %g", atm->k[iw][ip]);
04648     fprintf(out, "\n");
04649 }
04650
04651 /* Close file... */
04652 fclose(out);
04653 }
04654
04655 /*****
04656
04657 void write_matrix(
04658     const char *dirname,
04659     const char *filename,
04660     ctl_t *ctl,
04661     gsl_matrix *matrix,
04662     atm_t *atm,
04663     obs_t *obs,
04664     const char *rowspace,
04665     const char *colspace,
04666     const char *sort) {
04667
04668     FILE *out;
04669
04670     char file[LEN], quantity[LEN];
04671
04672     int *cida, *ciqa, *cipa, *cira, *rida, *riqa, *ripa, *rira;
04673
04674     size_t i, j, nc, nr;
04675
04676     /* Check output flag... */
04677     if (!ctl->write_matrix)
04678         return;
04679
04680     /* Allocate... */
04681     ALLOC(cida, int, M);
04682     ALLOC(ciqa, int,
04683           N);
04684     ALLOC(cipa, int,
04685           N);
04686     ALLOC(cira, int,
04687           M);
04688     ALLOC(rida, int,
04689           M);
04690     ALLOC(riqa, int,
04691           N);
04692     ALLOC(ripa, int,
04693           N);
04694     ALLOC(rira, int,
04695           M);
04696
04697     /* Set filename... */
04698     if (dirname != NULL)
04699         sprintf(file, "%s/%s", dirname, filename);
04700     else
04701         sprintf(file, "%s", filename);
04702
04703     /* Write info... */
04704     printf("Write matrix: %s\n", file);
04705
04706     /* Create file... */
04707     if (!(out = fopen(file, "w")))
04708         ERRMSG("Cannot create file!");
04709
04710     /* Write header (row space)... */
04711     if (rowspace[0] == 'y') {
04712
04713         fprintf(out,
04714             "# $1 = Row: index (measurement space)\n"
04715             "# $2 = Row: channel wavenumber [cm^-1]\n"
04716             "# $3 = Row: time (seconds since 2000-01-01T00:00Z)\n"
04717             "# $4 = Row: view point altitude [km]\n"
04718             "# $5 = Row: view point longitude [deg]\n"
04719             "# $6 = Row: view point latitude [deg]\n");
04720
04721         /* Get number of rows... */
04722         nr = obs2y(ctl, obs, NULL, rida, rira);
04723     } else {
04724
04725         fprintf(out,
04726             "# $1 = Row: index (state space)\n"
04727             "# $2 = Row: name of quantity\n"
04728             "# $3 = Row: time (seconds since 2000-01-01T00:00Z)\n"

```

```

04730         "# $4 = Row: altitude [km]\n"
04731         "# $5 = Row: longitude [deg]\n" "# $6 = Row: latitude [deg]\n");
04732
04733     /* Get number of rows... */
04734     nr = atm2x(ctl, atm, NULL, rida, ripa);
04735 }
04736
04737 /* Write header (column space)... */
04738 if (colspace[0] == 'y') {
04739     fprintf(out,
04740         "# $7 = Col: index (measurement space)\n"
04741         "# $8 = Col: channel wavenumber [cm^-1]\n"
04742         "# $9 = Col: time (seconds since 2000-01-01T00:00Z)\n"
04743         "# $10 = Col: view point altitude [km]\n"
04744         "# $11 = Col: view point longitude [deg]\n"
04745         "# $12 = Col: view point latitude [deg]\n");
04746
04747     /* Get number of columns... */
04748     nc = obs2y(ctl, obs, NULL, cida, cira);
04749
04750 } else {
04751     fprintf(out,
04752         "# $7 = Col: index (state space)\n"
04753         "# $8 = Col: name of quantity\n"
04754         "# $9 = Col: time (seconds since 2000-01-01T00:00Z)\n"
04755         "# $10 = Col: altitude [km]\n"
04756         "# $11 = Col: longitude [deg]\n" "# $12 = Col: latitude [deg]\n");
04757
04758     /* Get number of columns... */
04759     nc = atm2x(ctl, atm, NULL, cida, cira);
04760 }
04761
04762 /* Write header entry... */
04763 fprintf(out, "# $13 = Matrix element\n\n");
04764
04765 /* Write matrix data... */
04766 i = j = 0;
04767 while (i < nr && j < nc) {
04768     /* Write info about the row... */
04769     if (rowspace[0] == 'y')
04770         fprintf(out, "%d %g %.2f %g %g %g",
04771             (int) i, ctl->nu[rda[i]],
04772             obs->time[rira[i]], obs->vpz[rira[i]],
04773             obs->vplon[rira[i]], obs->vplat[rira[i]]);
04774     else {
04775         idx2name(ctl, rida[i], quantity);
04776         fprintf(out, "%d %s %.2f %g %g %g", (int) i, quantity,
04777             atm->time[rida[i]], atm->z[rida[i]],
04778             atm->lon[rida[i]], atm->lat[rida[i]]);
04779     }
04780
04781     /* Write info about the column... */
04782     if (colspace[0] == 'y')
04783         fprintf(out, " %d %g %.2f %g %g %g",
04784             (int) j, ctl->nu[cida[j]],
04785             obs->time[cira[j]], obs->vpz[cira[j]],
04786             obs->vplon[cira[j]], obs->vplat[cira[j]]);
04787     else {
04788         idx2name(ctl, cida[j], quantity);
04789         fprintf(out, " %d %s %.2f %g %g %g", (int) j, quantity,
04790             atm->time[cida[j]], atm->z[cida[j]],
04791             atm->lon[cida[j]], atm->lat[cida[j]]);
04792     }
04793
04794     /* Write matrix entry... */
04795     fprintf(out, " %g\n", gsl_matrix_get(matrix, i, j));
04796
04797     /* Set matrix indices... */
04798     if (sort[0] == 'r') {
04799         j++;
04800         if (j >= nc) {
04801             j = 0;
04802             i++;
04803             fprintf(out, "\n");
04804         }
04805     } else {
04806         i++;
04807         if (i >= nr) {
04808             i = 0;
04809             j++;
04810             fprintf(out, "\n");
04811         }
04812     }
04813 }
04814 }
04815 }
04816 }

```

```

04817
04818 /* Close file... */
04819 fclose(out);
04820
04821 /* Free... */
04822 free(cida);
04823 free(ciga);
04824 free(cipa);
04825 free(cira);
04826 free(rida);
04827 free(riqa);
04828 free(ripa);
04829 free(rira);
04830 }
04831
04832 /*****
04833
04834 void write_obs(
04835     const char *dirname,
04836     const char *filename,
04837     ctl_t * ctl,
04838     obs_t * obs) {
04839
04840     FILE *out;
04841
04842     char file[LEN];
04843
04844     int id, ir, n = 10;
04845
04846     /* Set filename... */
04847     if (dirname != NULL)
04848         sprintf(file, "%s/%s", dirname, filename);
04849     else
04850         sprintf(file, "%s", filename);
04851
04852     /* Write info... */
04853     printf("Write observation data: %s\n", file);
04854
04855     /* Create file... */
04856     if (!(out = fopen(file, "w")))
04857         ERRMSG("Cannot create file!");
04858
04859     /* Write header... */
04860     fprintf(out,
04861         "# $1 = time (seconds since 2000-01-01T00:00Z)\n"
04862         "# $2 = observer altitude [km]\n"
04863         "# $3 = observer longitude [deg]\n"
04864         "# $4 = observer latitude [deg]\n"
04865         "# $5 = view point altitude [km]\n"
04866         "# $6 = view point longitude [deg]\n"
04867         "# $7 = view point latitude [deg]\n"
04868         "# $8 = tangent point altitude [km]\n"
04869         "# $9 = tangent point longitude [deg]\n"
04870         "# $10 = tangent point latitude [deg]\n");
04871     for (id = 0; id < ctl->nd; id++)
04872         fprintf(out, "# $%d = channel %g: radiance [W/(m^2 sr cm^-1)]\n",
04873             ++n, ctl->nu[id]);
04874     for (id = 0; id < ctl->nd; id++)
04875         fprintf(out, "# $%d = channel %g: transmittance\n", ++n, ctl->nu[id]);
04876
04877     /* Write data... */
04878     for (ir = 0; ir < obs->nr; ir++) {
04879         if (ir == 0 || obs->time[ir] != obs->time[ir - 1])
04880             fprintf(out, "\n");
04881         fprintf(out, "%.2f %g %g %g %g %g %g %g %g", obs->time[ir],
04882             obs->obsz[ir], obs->obslon[ir], obs->obslat[ir],
04883             obs->vpz[ir], obs->vplon[ir], obs->vplat[ir],
04884             obs->tpz[ir], obs->tplon[ir], obs->tplat[ir]);
04885         for (id = 0; id < ctl->nd; id++)
04886             fprintf(out, " %g", obs->rad[id][ir]);
04887         for (id = 0; id < ctl->nd; id++)
04888             fprintf(out, " %g", obs->tau[id][ir]);
04889         fprintf(out, "\n");
04890     }
04891
04892     /* Close file... */
04893     fclose(out);
04894 }
04895
04896 /*****
04897
04898 void x2atm(
04899     ctl_t * ctl,
04900     gsl_vector * x,
04901     atm_t * atm) {
04902
04903     int ig, iw;

```

```

04904
04905     size_t n = 0;
04906
04907     /* Set pressure... */
04908     x2atm_help(atm, ctl->retp_zmin, ctl->retp_zmax, atm->
04909 p, x, &n);
04909
04910     /* Set temperature... */
04911     x2atm_help(atm, ctl->rett_zmin, ctl->rett_zmax, atm->
04912 t, x, &n);
04912
04913     /* Set volume mixing ratio... */
04914     for (ig = 0; ig < ctl->ng; ig++)
04915         x2atm_help(atm, ctl->retq_zmin[ig], ctl->retq_zmax[ig],
04916 atm->q[ig], x, &n);
04917
04918     /* Set extinction... */
04919     for (iw = 0; iw < ctl->nw; iw++)
04920         x2atm_help(atm, ctl->retk_zmin[iw], ctl->retk_zmax[iw],
04921 atm->k[iw], x, &n);
04922 }
04923
04924 /*****
04925
04926 void x2atm_help(
04927     atm_t * atm,
04928     double zmin,
04929     double zmax,
04930     double *value,
04931     gsl_vector * x,
04932     size_t * n) {
04933
04934     int ip;
04935
04936     /* Extract state vector elements... */
04937     for (ip = 0; ip < atm->np; ip++)
04938         if (atm->z[ip] >= zmin && atm->z[ip] <= zmax) {
04939             value[ip] = gsl_vector_get(x, *n);
04940             (*n)++;
04941         }
04942 }
04943
04944 /*****
04945
04946 void y2obs(
04947     ctl_t * ctl,
04948     gsl_vector * y,
04949     obs_t * obs) {
04950
04951     int id, ir;
04952
04953     size_t m = 0;
04954
04955     /* Decompose measurement vector... */
04956     for (ir = 0; ir < obs->nr; ir++)
04957         for (id = 0; id < ctl->nd; id++)
04958             if (gsl_finite(obs->rad[id][ir])) {
04959                 obs->rad[id][ir] = gsl_vector_get(y, m);
04960                 m++;
04961             }
04962 }

```

## 5.23 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 ctmc02 (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.*
- `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_irr` (double \*xx, int n, double x)  
*Find array index for irregular grid.*
- int `locate_reg` (double \*xx, int n, double x)  
*Find array index for regular grid.*
- 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 \*rowsep, const char \*colsep, 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.*

### 5.23.1 Detailed Description

JURASSIC library declarations.

Definition in file [jurassic.h](#).

### 5.23.2 Function Documentation

#### 5.23.2.1 `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](#).

```

00034         {
00035
00036     int ig, iw;
00037
00038     size_t n = 0;
00039
00040     /* Add pressure... */
00041     atm2x_help(atm, ctl->retp_zmin, ctl->retp_zmax,
00042               atm->p, IDXP, x, iqa, ipa, &n);
00043
00044     /* Add temperature... */
00045     atm2x_help(atm, ctl->rett_zmin, ctl->rett_zmax,
00046               atm->t, IDXT, x, iqa, ipa, &n);
00047
00048     /* Add volume mixing ratios... */
00049     for (ig = 0; ig < ctl->ng; ig++)
00050         atm2x_help(atm, ctl->retq_zmin[ig], ctl->retq_zmax[ig],
00051                   atm->q[ig], IDXQ(ig), x, iqa, ipa, &n);
00052
00053     /* Add extinction... */
00054     for (iw = 0; iw < ctl->nw; iw++)
00055         atm2x_help(atm, ctl->retk_zmin[iw], ctl->retk_zmax[iw],
00056                   atm->k[iw], IDXK(iw), x, iqa, ipa, &n);
00057
00058     return n;
00059 }

```

Here is the call graph for this function:





**5.23.2.2** `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](#).

```
00072         {
00073
00074     int ip;
00075
00076     /* Add elements to state vector... */
00077     for (ip = 0; ip < atm->np; ip++)
00078         if (atm->z[ip] >= zmin && atm->z[ip] <= zmax) {
00079             if (x != NULL)
00080                 gsl_vector_set(x, *n, value[ip]);
00081             if (iqa != NULL)
00082                 iqa[*n] = val_iqa;
00083             if (ipa != NULL)
00084                 ipa[*n] = ip;
00085             (*n)++;
00086         }
00087 }
```

**5.23.2.3** `double brightness ( double rad, double nu )`

Compute brightness temperature.

Definition at line 91 of file [jurassic.c](#).

```
00093     {
00094
00095     return C2 * nu / gsl_log1p(C1 * POW3(nu) / rad);
00096 }
```

**5.23.2.4** `void cart2geo ( double * x, double * z, double * lon, double * lat )`

Convert Cartesian coordinates to geolocation.

Definition at line 101 of file [jurassic.c](#).

```
00105     {
00106
00107     double radius;
00108
00109     radius = NORM(x);
00110     *lat = asin(x[2] / radius) * 180 / M_PI;
00111     *lon = atan2(x[1], x[0]) * 180 / M_PI;
00112     *z = radius - RE;
00113 }
```

5.23.2.5 void climatology ( ctl\_t\* *ctl*, atm\_t\* *atm\_mean* )

Interpolate climatological data.

Definition at line 117 of file [jurassic.c](#).

```

00119         {
00120
00121     static double z[121] = {
00122         0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19,
00123         20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37,
00124         38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55,
00125         56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73,
00126         74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91,
00127         92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107,
00128         108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 120
00129     };
00130
00131     static double pre[121] = {
00132         1017, 901.083, 796.45, 702.227, 617.614, 541.644, 473.437, 412.288,
00133         357.603, 308.96, 265.994, 228.348, 195.619, 167.351, 143.039, 122.198,
00134         104.369, 89.141, 76.1528, 65.0804, 55.641, 47.591, 40.7233, 34.8637,
00135         29.8633, 25.5956, 21.9534, 18.8445, 16.1909, 13.9258, 11.9913,
00136         10.34, 8.92988, 7.72454, 6.6924, 5.80701, 5.04654, 4.39238, 3.82902,
00137         3.34337, 2.92413, 2.56128, 2.2464, 1.97258, 1.73384, 1.52519, 1.34242,
00138         1.18197, 1.04086, 0.916546, 0.806832, 0.709875, 0.624101, 0.548176,
00139         0.480974, 0.421507, 0.368904, 0.322408, 0.281386, 0.245249, 0.213465,
00140         0.185549, 0.161072, 0.139644, 0.120913, 0.104568, 0.0903249, 0.0779269,
00141         0.0671493, 0.0577962, 0.0496902, 0.0426736, 0.0366093, 0.0313743,
00142         0.0268598, 0.0229699, 0.0196206, 0.0167399, 0.0142646, 0.0121397,
00143         0.0103181, 0.00875775, 0.00742226, 0.00628076, 0.00530519, 0.00447183,
00144         0.00376124, 0.00315632, 0.00264248, 0.00220738, 0.00184003, 0.00153095,
00145         0.00127204, 0.00105608, 0.000876652, 0.00072798, 0.00060492,
00146         0.000503201, 0.000419226, 0.000349896, 0.000292659, 0.000245421,
00147         0.000206394, 0.000174125, 0.000147441, 0.000125333, 0.000106985,
00148         9.173e-05, 7.90172e-05, 6.84172e-05, 5.95574e-05, 5.21183e-05,
00149         4.58348e-05, 4.05127e-05, 3.59987e-05, 3.21583e-05, 2.88718e-05,
00150         2.60322e-05, 2.35687e-05, 2.14263e-05, 1.95489e-05
00151     };
00152
00153     static double tem[121] = {
00154         285.14, 279.34, 273.91, 268.3, 263.24, 256.55, 250.2, 242.82, 236.17,
00155         229.87, 225.04, 221.19, 218.85, 217.19, 216.2, 215.68, 215.42, 215.55,
00156         215.92, 216.4, 216.93, 217.45, 218, 218.68, 219.39, 220.25, 221.3,
00157         222.41, 223.88, 225.42, 227.2, 229.52, 231.89, 234.51, 236.85, 239.42,
00158         241.94, 244.57, 247.36, 250.32, 253.34, 255.82, 258.27, 260.39,
00159         262.03, 263.45, 264.2, 264.78, 264.67, 264.38, 263.24, 262.03, 260.02,
00160         258.09, 255.63, 253.28, 250.43, 247.81, 245.26, 242.77, 240.38,
00161         237.94, 235.79, 233.53, 231.5, 229.53, 227.6, 225.62, 223.77, 222.06,
00162         220.33, 218.69, 217.18, 215.64, 214.13, 212.52, 210.86, 209.25,
00163         207.49, 205.81, 204.11, 202.22, 200.32, 198.39, 195.92, 193.46,
00164         190.94, 188.31, 185.82, 183.57, 181.43, 179.74, 178.64, 178.1, 178.25,
00165         178.7, 179.41, 180.67, 182.31, 184.18, 186.6, 189.53, 192.66, 196.54,
00166         201.13, 205.93, 211.73, 217.86, 225, 233.53, 242.57, 252.14, 261.48,
00167         272.97, 285.26, 299.12, 312.2, 324.17, 338.34, 352.56, 365.28
00168     };
00169
00170     static double c2h2[121] = {
00171         1.352e-09, 2.83e-10, 1.269e-10, 6.926e-11, 4.346e-11, 2.909e-11,
00172         2.014e-11, 1.363e-11, 8.71e-12, 5.237e-12, 2.718e-12, 1.375e-12,
00173         5.786e-13, 2.16e-13, 7.317e-14, 2.551e-14, 1.055e-14, 4.758e-15,
00174         2.056e-15, 7.703e-16, 2.82e-16, 1.035e-16, 4.382e-17, 1.946e-17,
00175         9.638e-18, 5.2e-18, 2.811e-18, 1.494e-18, 7.925e-19, 4.213e-19,
00176         1.998e-19, 8.78e-20, 3.877e-20, 1.728e-20, 7.743e-21, 3.536e-21,
00177         1.623e-21, 7.508e-22, 3.508e-22, 1.65e-22, 7.837e-23, 3.733e-23,
00178         1.808e-23, 8.77e-24, 4.285e-24, 2.095e-24, 1.032e-24, 5.082e-25,
00179         2.506e-25, 1.236e-25, 6.088e-26, 2.996e-26, 1.465e-26, 0, 0, 0,
00180         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
00181         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
00182         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
00183     };
00184
00185     static double c2h6[121] = {
00186         2.667e-09, 2.02e-09, 1.658e-09, 1.404e-09, 1.234e-09, 1.109e-09,
00187         1.012e-09, 9.262e-10, 8.472e-10, 7.71e-10, 6.932e-10, 6.216e-10,
00188         5.503e-10, 4.87e-10, 4.342e-10, 3.861e-10, 3.347e-10, 2.772e-10,
00189         2.209e-10, 1.672e-10, 1.197e-10, 8.536e-11, 5.783e-11, 3.846e-11,
00190         2.495e-11, 1.592e-11, 1.017e-11, 6.327e-12, 3.895e-12, 2.403e-12,
00191         1.416e-12, 8.101e-13, 4.649e-13, 2.686e-13, 1.557e-13, 9.14e-14,
00192         5.386e-14, 3.19e-14, 1.903e-14, 1.14e-14, 6.875e-15, 4.154e-15,
00193         2.538e-15, 1.553e-15, 9.548e-16, 5.872e-16, 3.63e-16, 2.244e-16,
00194         1.388e-16, 8.587e-17, 5.308e-17, 3.279e-17, 2.017e-17, 1.238e-17,
00195         7.542e-18, 4.585e-18, 2.776e-18, 1.671e-18, 9.985e-19, 5.937e-19,

```

```
00196      3.518e-19, 2.07e-19, 1.215e-19, 7.06e-20, 4.097e-20, 2.37e-20,
00197      1.363e-20, 7.802e-21, 4.441e-21, 2.523e-21, 1.424e-21, 8.015e-22,
00198      4.497e-22, 2.505e-22, 1.391e-22, 7.691e-23, 4.238e-23, 2.331e-23,
00199      1.274e-23, 6.929e-24, 3.752e-24, 2.02e-24, 1.083e-24, 5.774e-25,
00200      3.041e-25, 1.593e-25, 8.308e-26, 4.299e-26, 2.195e-26, 1.112e-26,
00201      0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
00202      0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
00203  };
00204
00205  static double ccl4[121] = {
00206      1.075e-10, 1.075e-10, 1.075e-10, 1.075e-10, 1.075e-10, 1.075e-10,
00207      1.075e-10, 1.075e-10, 1.075e-10, 1.06e-10, 1.024e-10, 9.69e-11,
00208      8.93e-11, 8.078e-11, 7.213e-11, 6.307e-11, 5.383e-11, 4.49e-11,
00209      3.609e-11, 2.705e-11, 1.935e-11, 1.385e-11, 8.35e-12, 5.485e-12,
00210      3.853e-12, 2.22e-12, 5.875e-13, 3.445e-13, 1.015e-13, 6.075e-14,
00211      4.383e-14, 2.692e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14,
00212      1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14,
00213      1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14,
00214      1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14,
00215      1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14,
00216      1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14,
00217      1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14,
00218      1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14,
00219      1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14,
00220      1e-14, 1e-14, 1e-14
00221  };
00222
00223  static double ch4[121] = {
00224      1.864e-06, 1.835e-06, 1.819e-06, 1.805e-06, 1.796e-06, 1.788e-06,
00225      1.782e-06, 1.776e-06, 1.769e-06, 1.761e-06, 1.749e-06, 1.734e-06,
00226      1.716e-06, 1.692e-06, 1.654e-06, 1.61e-06, 1.567e-06, 1.502e-06,
00227      1.433e-06, 1.371e-06, 1.323e-06, 1.277e-06, 1.232e-06, 1.188e-06,
00228      1.147e-06, 1.108e-06, 1.07e-06, 1.027e-06, 9.854e-07, 9.416e-07,
00229      8.933e-07, 8.478e-07, 7.988e-07, 7.515e-07, 7.07e-07, 6.64e-07,
00230      6.239e-07, 5.864e-07, 5.512e-07, 5.184e-07, 4.87e-07, 4.571e-07,
00231      4.296e-07, 4.04e-07, 3.802e-07, 3.578e-07, 3.383e-07, 3.203e-07,
00232      3.032e-07, 2.889e-07, 2.76e-07, 2.635e-07, 2.519e-07, 2.409e-07,
00233      2.302e-07, 2.219e-07, 2.144e-07, 2.071e-07, 1.999e-07, 1.93e-07,
00234      1.862e-07, 1.795e-07, 1.731e-07, 1.668e-07, 1.607e-07, 1.548e-07,
00235      1.49e-07, 1.434e-07, 1.38e-07, 1.328e-07, 1.277e-07, 1.227e-07,
00236      1.18e-07, 1.134e-07, 1.089e-07, 1.046e-07, 1.004e-07, 9.635e-08,
00237      9.245e-08, 8.867e-08, 8.502e-08, 8.15e-08, 7.809e-08, 7.48e-08,
00238      7.159e-08, 6.849e-08, 6.55e-08, 6.262e-08, 5.98e-08, 5.708e-08,
00239      5.448e-08, 5.194e-08, 4.951e-08, 4.72e-08, 4.5e-08, 4.291e-08,
00240      4.093e-08, 3.905e-08, 3.729e-08, 3.563e-08, 3.408e-08, 3.265e-08,
00241      3.128e-08, 2.996e-08, 2.87e-08, 2.76e-08, 2.657e-08, 2.558e-08,
00242      2.467e-08, 2.385e-08, 2.307e-08, 2.234e-08, 2.168e-08, 2.108e-08,
00243      2.05e-08, 1.998e-08, 1.947e-08, 1.902e-08, 1.86e-08, 1.819e-08,
00244      1.782e-08
00245  };
00246
00247  static double clo[121] = {
00248      7.419e-15, 1.061e-14, 1.518e-14, 2.195e-14, 3.175e-14, 4.666e-14,
00249      6.872e-14, 1.03e-13, 1.553e-13, 2.375e-13, 3.664e-13, 5.684e-13,
00250      8.915e-13, 1.402e-12, 2.269e-12, 4.125e-12, 7.501e-12, 1.257e-11,
00251      2.048e-11, 3.338e-11, 5.44e-11, 8.846e-11, 1.008e-10, 1.082e-10,
00252      1.157e-10, 1.232e-10, 1.312e-10, 1.539e-10, 1.822e-10, 2.118e-10,
00253      2.387e-10, 2.687e-10, 2.875e-10, 3.031e-10, 3.23e-10, 3.648e-10,
00254      4.117e-10, 4.477e-10, 4.633e-10, 4.794e-10, 4.95e-10, 5.104e-10,
00255      5.259e-10, 5.062e-10, 4.742e-10, 4.443e-10, 4.051e-10, 3.659e-10,
00256      3.305e-10, 2.911e-10, 2.54e-10, 2.215e-10, 1.927e-10, 1.675e-10,
00257      1.452e-10, 1.259e-10, 1.09e-10, 9.416e-11, 8.119e-11, 6.991e-11,
00258      6.015e-11, 5.163e-11, 4.43e-11, 3.789e-11, 3.24e-11, 2.769e-11,
00259      2.361e-11, 2.011e-11, 1.71e-11, 1.453e-11, 1.233e-11, 1.045e-11,
00260      8.851e-12, 7.48e-12, 6.316e-12, 5.326e-12, 4.487e-12, 3.778e-12,
00261      3.176e-12, 2.665e-12, 2.234e-12, 1.87e-12, 1.563e-12, 1.304e-12,
00262      1.085e-12, 9.007e-13, 7.468e-13, 6.179e-13, 5.092e-13, 4.188e-13,
00263      3.442e-13, 2.816e-13, 2.304e-13, 1.885e-13, 1.542e-13, 1.263e-13,
00264      1.035e-13, 8.5e-14, 7.004e-14, 5.783e-14, 4.795e-14, 4.007e-14,
00265      3.345e-14, 2.792e-14, 2.33e-14, 1.978e-14, 1.686e-14, 1.438e-14,
00266      1.234e-14, 1.07e-14, 9.312e-15, 8.131e-15, 7.164e-15, 6.367e-15,
00267      5.67e-15, 5.088e-15, 4.565e-15, 4.138e-15, 3.769e-15, 3.432e-15,
00268      3.148e-15
00269  };
00270
00271  static double clono2[121] = {
00272      1.011e-13, 1.515e-13, 2.272e-13, 3.446e-13, 5.231e-13, 8.085e-13,
00273      1.253e-12, 1.979e-12, 3.149e-12, 5.092e-12, 8.312e-12, 1.366e-11,
00274      2.272e-11, 3.791e-11, 6.209e-11, 9.101e-11, 1.334e-10, 1.951e-10,
00275      2.853e-10, 3.94e-10, 4.771e-10, 5.771e-10, 6.675e-10, 7.665e-10,
00276      8.504e-10, 8.924e-10, 9.363e-10, 8.923e-10, 8.411e-10, 7.646e-10,
00277      6.525e-10, 5.576e-10, 4.398e-10, 3.403e-10, 2.612e-10, 1.915e-10,
00278      1.407e-10, 1.028e-10, 7.455e-11, 5.42e-11, 3.708e-11, 2.438e-11,
00279      1.618e-11, 1.075e-11, 7.17e-12, 4.784e-12, 3.205e-12, 2.147e-12,
00280      1.44e-12, 9.654e-13, 6.469e-13, 4.332e-13, 2.891e-13, 1.926e-13,
00281      1.274e-13, 8.422e-14, 5.547e-14, 3.636e-14, 2.368e-14, 1.536e-14,
00282      9.937e-15, 6.39e-15, 4.101e-15, 2.61e-15, 1.659e-15, 1.052e-15,
```

```
00283     6.638e-16, 4.172e-16, 2.61e-16, 1.63e-16, 1.013e-16, 6.275e-17,
00284     3.879e-17, 2.383e-17, 1.461e-17, 8.918e-18, 5.43e-18, 3.301e-18,
00285     1.997e-18, 1.203e-18, 7.216e-19, 4.311e-19, 2.564e-19, 1.519e-19,
00286     8.911e-20, 5.203e-20, 3.026e-20, 1.748e-20, 9.99e-21, 5.673e-21,
00287     3.215e-21, 1.799e-21, 1.006e-21, 5.628e-22, 3.146e-22, 1.766e-22,
00288     9.94e-23, 5.614e-23, 3.206e-23, 1.841e-23, 1.071e-23, 6.366e-24,
00289     3.776e-24, 2.238e-24, 1.326e-24, 8.253e-25, 5.201e-25, 3.279e-25,
00290     2.108e-25, 1.395e-25, 9.326e-26, 6.299e-26, 4.365e-26, 3.104e-26,
00291     2.219e-26, 1.621e-26, 1.185e-26, 8.92e-27, 6.804e-27, 5.191e-27,
00292     4.041e-27
00293 };
00294
00295 static double co[121] = {
00296     1.907e-07, 1.553e-07, 1.362e-07, 1.216e-07, 1.114e-07, 1.036e-07,
00297     9.737e-08, 9.152e-08, 8.559e-08, 7.966e-08, 7.277e-08, 6.615e-08,
00298     5.884e-08, 5.22e-08, 4.699e-08, 4.284e-08, 3.776e-08, 3.274e-08,
00299     2.845e-08, 2.479e-08, 2.246e-08, 2.054e-08, 1.991e-08, 1.951e-08,
00300     1.94e-08, 2.009e-08, 2.1e-08, 2.201e-08, 2.322e-08, 2.45e-08,
00301     2.602e-08, 2.73e-08, 2.867e-08, 2.998e-08, 3.135e-08, 3.255e-08,
00302     3.352e-08, 3.426e-08, 3.484e-08, 3.53e-08, 3.593e-08, 3.671e-08,
00303     3.759e-08, 3.945e-08, 4.192e-08, 4.49e-08, 5.03e-08, 5.703e-08,
00304     6.538e-08, 7.878e-08, 9.644e-08, 1.196e-07, 1.498e-07, 1.904e-07,
00305     2.422e-07, 3.055e-07, 3.804e-07, 4.747e-07, 5.899e-07, 7.272e-07,
00306     8.91e-07, 1.071e-06, 1.296e-06, 1.546e-06, 1.823e-06, 2.135e-06,
00307     2.44e-06, 2.714e-06, 2.967e-06, 3.189e-06, 3.391e-06, 3.58e-06,
00308     3.773e-06, 4.022e-06, 4.346e-06, 4.749e-06, 5.199e-06, 5.668e-06,
00309     6.157e-06, 6.688e-06, 7.254e-06, 7.867e-06, 8.539e-06, 9.26e-06,
00310     1.009e-05, 1.119e-05, 1.228e-05, 1.365e-05, 1.506e-05, 1.641e-05,
00311     1.784e-05, 1.952e-05, 2.132e-05, 2.323e-05, 2.531e-05, 2.754e-05,
00312     3.047e-05, 3.459e-05, 3.922e-05, 4.439e-05, 4.825e-05, 5.077e-05,
00313     5.34e-05, 5.618e-05, 5.909e-05, 6.207e-05, 6.519e-05, 6.845e-05,
00314     6.819e-05, 6.726e-05, 6.622e-05, 6.512e-05, 6.671e-05, 6.862e-05,
00315     7.048e-05, 7.264e-05, 7.3e-05, 7.3e-05, 7.3e-05, 7.3e-05, 7.3e-05
00316 };
00317
00318 static double cof2[121] = {
00319     7.5e-14, 1.055e-13, 1.485e-13, 2.111e-13, 3.001e-13, 4.333e-13,
00320     6.269e-13, 9.221e-13, 1.364e-12, 2.046e-12, 3.093e-12, 4.703e-12,
00321     7.225e-12, 1.113e-11, 1.66e-11, 2.088e-11, 2.626e-11, 3.433e-11,
00322     4.549e-11, 5.886e-11, 7.21e-11, 8.824e-11, 1.015e-10, 1.155e-10,
00323     1.288e-10, 1.388e-10, 1.497e-10, 1.554e-10, 1.606e-10, 1.639e-10,
00324     1.64e-10, 1.64e-10, 1.596e-10, 1.542e-10, 1.482e-10, 1.382e-10,
00325     1.289e-10, 1.198e-10, 1.109e-10, 1.026e-10, 9.484e-11, 8.75e-11,
00326     8.086e-11, 7.49e-11, 6.948e-11, 6.446e-11, 5.961e-11, 5.505e-11,
00327     5.085e-11, 4.586e-11, 4.1e-11, 3.665e-11, 3.235e-11, 2.842e-11,
00328     2.491e-11, 2.11e-11, 1.769e-11, 1.479e-11, 1.197e-11, 9.631e-12,
00329     7.74e-12, 6.201e-12, 4.963e-12, 3.956e-12, 3.151e-12, 2.507e-12,
00330     1.99e-12, 1.576e-12, 1.245e-12, 9.83e-13, 7.742e-13, 6.088e-13,
00331     4.782e-13, 3.745e-13, 2.929e-13, 2.286e-13, 1.782e-13, 1.388e-13,
00332     1.079e-13, 8.362e-14, 6.471e-14, 4.996e-14, 3.85e-14, 2.96e-14,
00333     2.265e-14, 1.729e-14, 1.317e-14, 9.998e-15, 7.549e-15, 5.683e-15,
00334     4.273e-15, 3.193e-15, 2.385e-15, 1.782e-15, 1.331e-15, 9.957e-16,
00335     7.461e-16, 5.601e-16, 4.228e-16, 3.201e-16, 2.438e-16, 1.878e-16,
00336     1.445e-16, 1.111e-16, 8.544e-17, 6.734e-17, 5.341e-17, 4.237e-17,
00337     3.394e-17, 2.759e-17, 2.254e-17, 1.851e-17, 1.54e-17, 1.297e-17,
00338     1.096e-17, 9.365e-18, 8e-18, 6.938e-18, 6.056e-18, 5.287e-18,
00339     4.662e-18
00340 };
00341
00342 static double f11[121] = {
00343     2.65e-10, 2.65e-10, 2.65e-10, 2.65e-10, 2.65e-10, 2.65e-10, 2.65e-10,
00344     2.65e-10, 2.65e-10, 2.65e-10, 2.65e-10, 2.65e-10, 2.635e-10, 2.536e-10,
00345     2.44e-10, 2.348e-10, 2.258e-10, 2.153e-10, 2.046e-10, 1.929e-10,
00346     1.782e-10, 1.648e-10, 1.463e-10, 1.291e-10, 1.1e-10, 8.874e-11,
00347     7.165e-11, 5.201e-11, 3.744e-11, 2.577e-11, 1.64e-11, 1.048e-11,
00348     5.993e-12, 3.345e-12, 1.839e-12, 9.264e-13, 4.688e-13, 2.329e-13,
00349     1.129e-13, 5.505e-14, 2.825e-14, 1.492e-14, 7.997e-15, 5.384e-15,
00350     3.988e-15, 2.955e-15, 2.196e-15, 1.632e-15, 1.214e-15, 9.025e-16,
00351     6.708e-16, 4.984e-16, 3.693e-16, 2.733e-16, 2.013e-16, 1.481e-16,
00352     1.087e-16, 7.945e-17, 5.782e-17, 4.195e-17, 3.038e-17, 2.19e-17,
00353     1.577e-17, 1.128e-17, 8.063e-18, 5.753e-18, 4.09e-18, 2.899e-18,
00354     2.048e-18, 1.444e-18, 1.015e-18, 7.12e-19, 4.985e-19, 3.474e-19,
00355     2.417e-19, 1.677e-19, 1.161e-19, 8.029e-20, 5.533e-20, 3.799e-20,
00356     2.602e-20, 1.776e-20, 1.209e-20, 8.202e-21, 5.522e-21, 3.707e-21,
00357     2.48e-21, 1.652e-21, 1.091e-21, 7.174e-22, 4.709e-22, 3.063e-22,
00358     1.991e-22, 1.294e-22, 8.412e-23, 5.483e-23, 3.581e-23, 2.345e-23,
00359     1.548e-23, 1.027e-23, 6.869e-24, 4.673e-24, 3.173e-24, 2.153e-24,
00360     1.461e-24, 1.028e-24, 7.302e-25, 5.188e-25, 3.739e-25, 2.753e-25,
00361     2.043e-25, 1.528e-25, 1.164e-25, 9.041e-26, 7.051e-26, 5.587e-26,
00362     4.428e-26, 3.588e-26, 2.936e-26, 2.402e-26, 1.995e-26
00363 };
00364
00365 static double f12[121] = {
00366     5.45e-10, 5.45e-10, 5.45e-10, 5.45e-10, 5.45e-10, 5.45e-10, 5.45e-10,
00367     5.45e-10, 5.45e-10, 5.45e-10, 5.45e-10, 5.45e-10, 5.429e-10, 5.291e-10,
00368     5.155e-10, 5.022e-10, 4.893e-10, 4.772e-10, 4.655e-10, 4.497e-10,
00369     4.249e-10, 4.015e-10, 3.632e-10, 3.261e-10, 2.858e-10, 2.408e-10,
```

```
00370      2.03e-10, 1.685e-10, 1.4e-10, 1.163e-10, 9.65e-11, 8.02e-11, 6.705e-11,
00371      5.624e-11, 4.764e-11, 4.249e-11, 3.792e-11, 3.315e-11, 2.819e-11,
00372      2.4e-11, 1.999e-11, 1.64e-11, 1.352e-11, 1.14e-11, 9.714e-12,
00373      8.28e-12, 7.176e-12, 6.251e-12, 5.446e-12, 4.72e-12, 4.081e-12,
00374      3.528e-12, 3.08e-12, 2.699e-12, 2.359e-12, 2.111e-12, 1.901e-12,
00375      1.709e-12, 1.534e-12, 1.376e-12, 1.233e-12, 1.103e-12, 9.869e-13,
00376      8.808e-13, 7.859e-13, 7.008e-13, 6.241e-13, 5.553e-13, 4.935e-13,
00377      4.383e-13, 3.889e-13, 3.447e-13, 3.054e-13, 2.702e-13, 2.389e-13,
00378      2.11e-13, 1.862e-13, 1.643e-13, 1.448e-13, 1.274e-13, 1.121e-13,
00379      9.844e-14, 8.638e-14, 7.572e-14, 6.62e-14, 5.782e-14, 5.045e-14,
00380      4.394e-14, 3.817e-14, 3.311e-14, 2.87e-14, 2.48e-14, 2.142e-14,
00381      1.851e-14, 1.599e-14, 1.383e-14, 1.196e-14, 1.036e-14, 9e-15,
00382      7.828e-15, 6.829e-15, 5.992e-15, 5.254e-15, 4.606e-15, 4.037e-15,
00383      3.583e-15, 3.19e-15, 2.841e-15, 2.542e-15, 2.291e-15, 2.07e-15,
00384      1.875e-15, 1.71e-15, 1.57e-15, 1.442e-15, 1.333e-15, 1.232e-15,
00385      1.147e-15, 1.071e-15, 1.001e-15, 9.396e-16
00386  };
00387
00388  static double f14[121] = {
00389      9e-11, 9e-11, 9e-11, 9e-11, 9e-11, 9e-11, 9e-11, 9e-11, 9e-11, 9e-11,
00390      9e-11, 9e-11, 9e-11, 9e-11, 9e-11, 8.91e-11, 8.73e-11, 8.46e-11,
00391      8.19e-11, 7.92e-11, 7.74e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11,
00392      7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11,
00393      7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11,
00394      7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11,
00395      7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11,
00396      7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11,
00397      7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11,
00398      7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11,
00399      7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11,
00400      7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11,
00401      7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11,
00402      7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11,
00403      7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11,
00404      7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11,
00405      7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11
00406  };
00407
00408  static double f22[121] = {
00409      1.4e-10, 1.4e-10, 1.4e-10, 1.4e-10, 1.4e-10, 1.4e-10, 1.4e-10, 1.4e-10,
00410      1.4e-10, 1.4e-10, 1.4e-10, 1.372e-10, 1.317e-10, 1.235e-10, 1.153e-10,
00411      1.075e-10, 1.002e-10, 9.332e-11, 8.738e-11, 8.194e-11, 7.7e-11,
00412      7.165e-11, 6.753e-11, 6.341e-11, 5.971e-11, 5.6e-11, 5.229e-11,
00413      4.859e-11, 4.488e-11, 4.118e-11, 3.83e-11, 3.568e-11, 3.308e-11,
00414      3.047e-11, 2.82e-11, 2.594e-11, 2.409e-11, 2.237e-11, 2.065e-11,
00415      1.894e-11, 1.771e-11, 1.647e-11, 1.532e-11, 1.416e-11, 1.332e-11,
00416      1.246e-11, 1.161e-11, 1.087e-11, 1.017e-11, 9.471e-12, 8.853e-12,
00417      8.235e-12, 7.741e-12, 7.247e-12, 6.836e-12, 6.506e-12, 6.176e-12,
00418      5.913e-12, 5.65e-12, 5.419e-12, 5.221e-12, 5.024e-12, 4.859e-12,
00419      4.694e-12, 4.546e-12, 4.414e-12, 4.282e-12, 4.15e-12, 4.019e-12,
00420      3.903e-12, 3.805e-12, 3.706e-12, 3.607e-12, 3.508e-12, 3.41e-12,
00421      3.31e-12, 3.212e-12, 3.129e-12, 3.047e-12, 2.964e-12, 2.882e-12,
00422      2.8e-12, 2.734e-12, 2.668e-12, 2.602e-12, 2.537e-12, 2.471e-12,
00423      2.421e-12, 2.372e-12, 2.322e-12, 2.273e-12, 2.224e-12, 2.182e-12,
00424      2.141e-12, 2.1e-12, 2.059e-12, 2.018e-12, 1.977e-12, 1.935e-12,
00425      1.894e-12, 1.853e-12, 1.812e-12, 1.77e-12, 1.73e-12, 1.688e-12,
00426      1.647e-12, 1.606e-12, 1.565e-12, 1.524e-12, 1.483e-12, 1.441e-12,
00427      1.4e-12, 1.359e-12, 1.317e-12, 1.276e-12, 1.235e-12, 1.194e-12,
00428      1.153e-12, 1.112e-12, 1.071e-12, 1.029e-12, 9.883e-13
00429  };
00430
00431  static double h2o[121] = {
00432      0.01166, 0.008269, 0.005742, 0.003845, 0.00277, 0.001897, 0.001272,
00433      0.000827, 0.000539, 0.0003469, 0.0001579, 3.134e-05, 1.341e-05,
00434      6.764e-06, 4.498e-06, 3.703e-06, 3.724e-06, 3.899e-06, 4.002e-06,
00435      4.122e-06, 4.277e-06, 4.438e-06, 4.558e-06, 4.673e-06, 4.763e-06,
00436      4.809e-06, 4.856e-06, 4.936e-06, 5.021e-06, 5.114e-06, 5.222e-06,
00437      5.331e-06, 5.414e-06, 5.488e-06, 5.563e-06, 5.633e-06, 5.704e-06,
00438      5.767e-06, 5.819e-06, 5.872e-06, 5.914e-06, 5.949e-06, 5.984e-06,
00439      6.015e-06, 6.044e-06, 6.073e-06, 6.104e-06, 6.136e-06, 6.167e-06,
00440      6.189e-06, 6.208e-06, 6.226e-06, 6.212e-06, 6.185e-06, 6.158e-06,
00441      6.114e-06, 6.066e-06, 6.018e-06, 5.877e-06, 5.728e-06, 5.582e-06,
00442      5.437e-06, 5.296e-06, 5.156e-06, 5.02e-06, 4.886e-06, 4.754e-06,
00443      4.625e-06, 4.498e-06, 4.374e-06, 4.242e-06, 4.096e-06, 3.955e-06,
00444      3.817e-06, 3.683e-06, 3.491e-06, 3.204e-06, 2.94e-06, 2.696e-06,
00445      2.47e-06, 2.252e-06, 2.019e-06, 1.808e-06, 1.618e-06, 1.445e-06,
00446      1.285e-06, 1.105e-06, 9.489e-07, 8.121e-07, 6.938e-07, 5.924e-07,
00447      5.04e-07, 4.288e-07, 3.648e-07, 3.103e-07, 2.642e-07, 2.252e-07,
00448      1.921e-07, 1.643e-07, 1.408e-07, 1.211e-07, 1.048e-07, 9.063e-08,
00449      7.835e-08, 6.774e-08, 5.936e-08, 5.221e-08, 4.592e-08, 4.061e-08,
00450      3.62e-08, 3.236e-08, 2.902e-08, 2.62e-08, 2.383e-08, 2.171e-08,
00451      1.989e-08, 1.823e-08, 1.684e-08, 1.562e-08, 1.449e-08, 1.351e-08
00452  };
00453
00454  static double h2o2[121] = {
00455      1.779e-10, 7.938e-10, 8.953e-10, 8.032e-10, 6.564e-10, 5.159e-10,
00456      4.003e-10, 3.026e-10, 2.222e-10, 1.58e-10, 1.044e-10, 6.605e-11,
```

```
00457     3.413e-11, 1.453e-11, 1.062e-11, 1.009e-11, 9.597e-12, 1.175e-11,
00458     1.572e-11, 2.091e-11, 2.746e-11, 3.603e-11, 4.791e-11, 6.387e-11,
00459     8.239e-11, 1.007e-10, 1.23e-10, 1.363e-10, 1.489e-10, 1.585e-10,
00460     1.608e-10, 1.632e-10, 1.576e-10, 1.502e-10, 1.423e-10, 1.302e-10,
00461     1.192e-10, 1.085e-10, 9.795e-11, 8.854e-11, 8.057e-11, 7.36e-11,
00462     6.736e-11, 6.362e-11, 6.087e-11, 5.825e-11, 5.623e-11, 5.443e-11,
00463     5.27e-11, 5.098e-11, 4.931e-11, 4.769e-11, 4.611e-11, 4.458e-11,
00464     4.308e-11, 4.102e-11, 3.887e-11, 3.682e-11, 3.521e-11, 3.369e-11,
00465     3.224e-11, 3.082e-11, 2.946e-11, 2.814e-11, 2.687e-11, 2.566e-11,
00466     2.449e-11, 2.336e-11, 2.227e-11, 2.123e-11, 2.023e-11, 1.927e-11,
00467     1.835e-11, 1.746e-11, 1.661e-11, 1.58e-11, 1.502e-11, 1.428e-11,
00468     1.357e-11, 1.289e-11, 1.224e-11, 1.161e-11, 1.102e-11, 1.045e-11,
00469     9.895e-12, 9.369e-12, 8.866e-12, 8.386e-12, 7.922e-12, 7.479e-12,
00470     7.06e-12, 6.656e-12, 6.274e-12, 5.914e-12, 5.575e-12, 5.257e-12,
00471     4.959e-12, 4.679e-12, 4.42e-12, 4.178e-12, 3.954e-12, 3.75e-12,
00472     3.557e-12, 3.372e-12, 3.198e-12, 3.047e-12, 2.908e-12, 2.775e-12,
00473     2.653e-12, 2.544e-12, 2.442e-12, 2.346e-12, 2.26e-12, 2.183e-12,
00474     2.11e-12, 2.044e-12, 1.98e-12, 1.924e-12, 1.871e-12, 1.821e-12,
00475     1.775e-12
00476 };
00477
00478 static double hcn[121] = {
00479     5.5e-10, 5.5e-10, 5.5e-10, 5.5e-10, 5.5e-10, 5.5e-10, 5.5e-10,
00480     5.5e-10, 5.5e-10, 5.5e-10, 5.5e-10, 5.498e-10, 5.495e-10, 5.493e-10,
00481     5.49e-10, 5.488e-10, 4.717e-10, 3.946e-10, 3.174e-10, 2.4e-10,
00482     1.626e-10, 1.619e-10, 1.612e-10, 1.602e-10, 1.593e-10, 1.582e-10,
00483     1.572e-10, 1.56e-10, 1.549e-10, 1.539e-10, 1.53e-10, 1.519e-10,
00484     1.506e-10, 1.487e-10, 1.467e-10, 1.449e-10, 1.43e-10, 1.413e-10,
00485     1.397e-10, 1.382e-10, 1.368e-10, 1.354e-10, 1.337e-10, 1.315e-10,
00486     1.292e-10, 1.267e-10, 1.241e-10, 1.215e-10, 1.19e-10, 1.165e-10,
00487     1.141e-10, 1.118e-10, 1.096e-10, 1.072e-10, 1.047e-10, 1.021e-10,
00488     9.968e-11, 9.739e-11, 9.539e-11, 9.339e-11, 9.135e-11, 8.898e-11,
00489     8.664e-11, 8.439e-11, 8.249e-11, 8.075e-11, 7.904e-11, 7.735e-11,
00490     7.565e-11, 7.399e-11, 7.245e-11, 7.109e-11, 6.982e-11, 6.863e-11,
00491     6.755e-11, 6.657e-11, 6.587e-11, 6.527e-11, 6.476e-11, 6.428e-11,
00492     6.382e-11, 6.343e-11, 6.307e-11, 6.272e-11, 6.238e-11, 6.205e-11,
00493     6.17e-11, 6.137e-11, 6.102e-11, 6.072e-11, 6.046e-11, 6.03e-11,
00494     6.018e-11, 6.01e-11, 6.001e-11, 5.992e-11, 5.984e-11, 5.975e-11,
00495     5.967e-11, 5.958e-11, 5.95e-11, 5.941e-11, 5.933e-11, 5.925e-11,
00496     5.916e-11, 5.908e-11, 5.899e-11, 5.891e-11, 5.883e-11, 5.874e-11,
00497     5.866e-11, 5.858e-11, 5.85e-11, 5.841e-11, 5.833e-11, 5.825e-11,
00498     5.817e-11, 5.808e-11, 5.8e-11, 5.792e-11, 5.784e-11
00499 };
00500
00501 static double hno3[121] = {
00502     1.809e-10, 7.234e-10, 5.899e-10, 4.342e-10, 3.277e-10, 2.661e-10,
00503     2.35e-10, 2.267e-10, 2.389e-10, 2.651e-10, 3.255e-10, 4.099e-10,
00504     5.42e-10, 6.978e-10, 8.807e-10, 1.112e-09, 1.405e-09, 2.04e-09,
00505     3.111e-09, 4.5e-09, 5.762e-09, 7.37e-09, 7.852e-09, 8.109e-09,
00506     8.067e-09, 7.554e-09, 7.076e-09, 6.268e-09, 5.524e-09, 4.749e-09,
00507     3.909e-09, 3.223e-09, 2.517e-09, 1.942e-09, 1.493e-09, 1.122e-09,
00508     8.449e-10, 6.361e-10, 4.787e-10, 3.611e-10, 2.804e-10, 2.215e-10,
00509     1.758e-10, 1.441e-10, 1.197e-10, 9.953e-11, 8.505e-11, 7.334e-11,
00510     6.325e-11, 5.625e-11, 5.058e-11, 4.548e-11, 4.122e-11, 3.748e-11,
00511     3.402e-11, 3.088e-11, 2.8e-11, 2.536e-11, 2.293e-11, 2.072e-11,
00512     1.871e-11, 1.687e-11, 1.52e-11, 1.368e-11, 1.23e-11, 1.105e-11,
00513     9.922e-12, 8.898e-12, 7.972e-12, 7.139e-12, 6.385e-12, 5.708e-12,
00514     5.099e-12, 4.549e-12, 4.056e-12, 3.613e-12, 3.216e-12, 2.862e-12,
00515     2.544e-12, 2.259e-12, 2.004e-12, 1.776e-12, 1.572e-12, 1.391e-12,
00516     1.227e-12, 1.082e-12, 9.528e-13, 8.379e-13, 7.349e-13, 6.436e-13,
00517     5.634e-13, 4.917e-13, 4.291e-13, 3.745e-13, 3.267e-13, 2.854e-13,
00518     2.494e-13, 2.181e-13, 1.913e-13, 1.68e-13, 1.479e-13, 1.31e-13,
00519     1.159e-13, 1.025e-13, 9.067e-14, 8.113e-14, 7.281e-14, 6.535e-14,
00520     5.892e-14, 5.348e-14, 4.867e-14, 4.439e-14, 4.073e-14, 3.76e-14,
00521     3.476e-14, 3.229e-14, 3e-14, 2.807e-14, 2.635e-14, 2.473e-14,
00522     2.332e-14
00523 };
00524
00525 static double hno4[121] = {
00526     6.118e-12, 3.594e-12, 2.807e-12, 3.04e-12, 4.458e-12, 7.986e-12,
00527     1.509e-11, 2.661e-11, 3.738e-11, 4.652e-11, 4.429e-11, 3.992e-11,
00528     3.347e-11, 3.005e-11, 3.173e-11, 4.055e-11, 5.812e-11, 8.489e-11,
00529     1.19e-10, 1.482e-10, 1.766e-10, 2.103e-10, 2.35e-10, 2.598e-10,
00530     2.801e-10, 2.899e-10, 3e-10, 2.817e-10, 2.617e-10, 2.332e-10,
00531     1.933e-10, 1.605e-10, 1.232e-10, 9.285e-11, 6.941e-11, 4.951e-11,
00532     3.539e-11, 2.402e-11, 1.522e-11, 9.676e-12, 6.056e-12, 3.745e-12,
00533     2.34e-12, 1.463e-12, 9.186e-13, 5.769e-13, 3.322e-13, 1.853e-13,
00534     1.035e-13, 7.173e-14, 5.382e-14, 4.036e-14, 3.401e-14, 2.997e-14,
00535     2.635e-14, 2.316e-14, 2.034e-14, 1.783e-14, 1.56e-14, 1.363e-14,
00536     1.19e-14, 1.037e-14, 9.032e-15, 7.846e-15, 6.813e-15, 5.912e-15,
00537     5.121e-15, 4.431e-15, 3.829e-15, 3.306e-15, 2.851e-15, 2.456e-15,
00538     2.114e-15, 1.816e-15, 1.559e-15, 1.337e-15, 1.146e-15, 9.811e-16,
00539     8.389e-16, 7.162e-16, 6.109e-16, 5.203e-16, 4.425e-16, 3.76e-16,
00540     3.184e-16, 2.692e-16, 2.274e-16, 1.917e-16, 1.61e-16, 1.35e-16,
00541     1.131e-16, 9.437e-17, 7.874e-17, 6.57e-17, 5.481e-17, 4.579e-17,
00542     3.828e-17, 3.204e-17, 2.691e-17, 2.264e-17, 1.912e-17, 1.626e-17,
00543     1.382e-17, 1.174e-17, 9.972e-18, 8.603e-18, 7.45e-18, 6.453e-18,
```

```
00544     5.623e-18, 4.944e-18, 4.361e-18, 3.859e-18, 3.443e-18, 3.096e-18,
00545     2.788e-18, 2.528e-18, 2.293e-18, 2.099e-18, 1.929e-18, 1.773e-18,
00546     1.64e-18
00547 };
00548
00549 static double hoc1[121] = {
00550     1.056e-12, 1.194e-12, 1.35e-12, 1.531e-12, 1.737e-12, 1.982e-12,
00551     2.263e-12, 2.599e-12, 2.991e-12, 3.459e-12, 4.012e-12, 4.662e-12,
00552     5.438e-12, 6.35e-12, 7.425e-12, 8.686e-12, 1.016e-11, 1.188e-11,
00553     1.389e-11, 1.659e-11, 2.087e-11, 2.621e-11, 3.265e-11, 4.064e-11,
00554     4.859e-11, 5.441e-11, 6.09e-11, 6.373e-11, 6.611e-11, 6.94e-11,
00555     7.44e-11, 7.97e-11, 8.775e-11, 9.722e-11, 1.064e-10, 1.089e-10,
00556     1.114e-10, 1.106e-10, 1.053e-10, 1.004e-10, 9.006e-11, 7.778e-11,
00557     6.739e-11, 5.636e-11, 4.655e-11, 3.845e-11, 3.042e-11, 2.368e-11,
00558     1.845e-11, 1.442e-11, 1.127e-11, 8.814e-12, 6.544e-12, 4.763e-12,
00559     3.449e-12, 2.612e-12, 1.999e-12, 1.526e-12, 1.16e-12, 8.793e-13,
00560     6.655e-13, 5.017e-13, 3.778e-13, 2.829e-13, 2.117e-13, 1.582e-13,
00561     1.178e-13, 8.755e-14, 6.486e-14, 4.799e-14, 3.54e-14, 2.606e-14,
00562     1.916e-14, 1.403e-14, 1.026e-14, 7.48e-15, 5.446e-15, 3.961e-15,
00563     2.872e-15, 2.076e-15, 1.498e-15, 1.077e-15, 7.726e-16, 5.528e-16,
00564     3.929e-16, 2.785e-16, 1.969e-16, 1.386e-16, 9.69e-17, 6.747e-17,
00565     4.692e-17, 3.236e-17, 2.232e-17, 1.539e-17, 1.061e-17, 7.332e-18,
00566     5.076e-18, 3.522e-18, 2.461e-18, 1.726e-18, 1.22e-18, 8.75e-19,
00567     6.264e-19, 4.482e-19, 3.207e-19, 2.368e-19, 1.762e-19, 1.312e-19,
00568     9.891e-20, 7.595e-20, 5.87e-20, 4.567e-20, 3.612e-20, 2.904e-20,
00569     2.343e-20, 1.917e-20, 1.568e-20, 1.308e-20, 1.1e-20, 9.25e-21,
00570     7.881e-21
00571 };
00572
00573 static double n2o[121] = {
00574     3.17e-07, 3.17e-07, 3.17e-07, 3.17e-07, 3.17e-07, 3.17e-07, 3.17e-07,
00575     3.17e-07, 3.17e-07, 3.17e-07, 3.124e-07, 3.077e-07, 3.03e-07,
00576     2.984e-07, 2.938e-07, 2.892e-07, 2.847e-07, 2.779e-07, 2.705e-07,
00577     2.631e-07, 2.557e-07, 2.484e-07, 2.345e-07, 2.201e-07, 2.01e-07,
00578     1.754e-07, 1.532e-07, 1.329e-07, 1.154e-07, 1.003e-07, 8.735e-08,
00579     7.617e-08, 6.512e-08, 5.547e-08, 4.709e-08, 3.915e-08, 3.259e-08,
00580     2.738e-08, 2.327e-08, 1.98e-08, 1.711e-08, 1.493e-08, 1.306e-08,
00581     1.165e-08, 1.049e-08, 9.439e-09, 8.375e-09, 7.391e-09, 6.525e-09,
00582     5.759e-09, 5.083e-09, 4.485e-09, 3.953e-09, 3.601e-09, 3.27e-09,
00583     2.975e-09, 2.757e-09, 2.556e-09, 2.37e-09, 2.195e-09, 2.032e-09,
00584     1.912e-09, 1.79e-09, 1.679e-09, 1.572e-09, 1.482e-09, 1.402e-09,
00585     1.326e-09, 1.254e-09, 1.187e-09, 1.127e-09, 1.071e-09, 1.02e-09,
00586     9.673e-10, 9.193e-10, 8.752e-10, 8.379e-10, 8.017e-10, 7.66e-10,
00587     7.319e-10, 7.004e-10, 6.721e-10, 6.459e-10, 6.199e-10, 5.942e-10,
00588     5.703e-10, 5.488e-10, 5.283e-10, 5.082e-10, 4.877e-10, 4.696e-10,
00589     4.52e-10, 4.355e-10, 4.198e-10, 4.039e-10, 3.888e-10, 3.754e-10,
00590     3.624e-10, 3.499e-10, 3.381e-10, 3.267e-10, 3.163e-10, 3.058e-10,
00591     2.959e-10, 2.864e-10, 2.77e-10, 2.686e-10, 2.604e-10, 2.534e-10,
00592     2.462e-10, 2.386e-10, 2.318e-10, 2.247e-10, 2.189e-10, 2.133e-10,
00593     2.071e-10, 2.014e-10, 1.955e-10, 1.908e-10, 1.86e-10, 1.817e-10
00594 };
00595
00596 static double n2o5[121] = {
00597     1.231e-11, 3.035e-12, 1.702e-12, 9.877e-13, 8.081e-13, 9.039e-13,
00598     1.169e-12, 1.474e-12, 1.651e-12, 1.795e-12, 1.998e-12, 2.543e-12,
00599     4.398e-12, 7.698e-12, 1.28e-11, 2.131e-11, 3.548e-11, 5.894e-11,
00600     7.645e-11, 1.089e-10, 1.391e-10, 1.886e-10, 2.386e-10, 2.986e-10,
00601     3.487e-10, 3.994e-10, 4.5e-10, 4.6e-10, 4.591e-10, 4.1e-10, 3.488e-10,
00602     2.846e-10, 2.287e-10, 1.696e-10, 1.011e-10, 6.428e-11, 4.324e-11,
00603     2.225e-11, 6.214e-12, 3.608e-12, 8.793e-13, 4.491e-13, 1.04e-13,
00604     6.1e-14, 3.436e-14, 6.671e-15, 1.171e-15, 5.848e-16, 1.212e-16,
00605     1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16,
00606     1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16,
00607     1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16,
00608     1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16,
00609     1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16,
00610     1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16,
00611     1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16,
00612     1e-16, 1e-16
00613 };
00614
00615 static double nh3[121] = {
00616     1e-10, 1e-10, 1e-10, 1e-10, 1e-10, 1e-10, 1e-10, 1e-10, 1e-10, 1e-10,
00617     1e-10, 1e-10, 1e-10, 1e-10, 9.444e-11, 8.488e-11, 7.241e-11, 5.785e-11,
00618     4.178e-11, 3.018e-11, 2.18e-11, 1.574e-11, 1.137e-11, 8.211e-12,
00619     5.973e-12, 4.327e-12, 3.118e-12, 2.234e-12, 1.573e-12, 1.04e-12,
00620     6.762e-13, 4.202e-13, 2.406e-13, 1.335e-13, 6.938e-14, 3.105e-14,
00621     1.609e-14, 1.033e-14, 6.432e-15, 4.031e-15, 2.555e-15, 1.656e-15,
00622     1.115e-15, 7.904e-16, 5.63e-16, 4.048e-16, 2.876e-16, 2.004e-16,
00623     1.356e-16, 9.237e-17, 6.235e-17, 4.223e-17, 3.009e-17, 2.328e-17,
00624     2.002e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17,
00625     1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17,
00626     1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17,
00627     1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17,
00628     1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17,
00629     1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17,
00630     1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17,
```

```
00631     1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17,
00632     1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17,
00633     1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17,
00634     1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17,
00635     1.914e-17
00636 };
00637
00638 static double no[121] = {
00639     2.586e-10, 4.143e-11, 1.566e-11, 9.591e-12, 8.088e-12, 8.462e-12,
00640     1.013e-11, 1.328e-11, 1.855e-11, 2.678e-11, 3.926e-11, 5.464e-11,
00641     7.012e-11, 8.912e-11, 1.127e-10, 1.347e-10, 1.498e-10, 1.544e-10,
00642     1.602e-10, 1.824e-10, 2.078e-10, 2.366e-10, 2.691e-10, 5.141e-10,
00643     8.259e-10, 1.254e-09, 1.849e-09, 2.473e-09, 3.294e-09, 4.16e-09,
00644     5.095e-09, 6.11e-09, 6.93e-09, 7.888e-09, 8.903e-09, 9.713e-09,
00645     1.052e-08, 1.115e-08, 1.173e-08, 1.21e-08, 1.228e-08, 1.239e-08,
00646     1.231e-08, 1.213e-08, 1.192e-08, 1.138e-08, 1.085e-08, 1.008e-08,
00647     9.224e-09, 8.389e-09, 7.262e-09, 6.278e-09, 5.335e-09, 4.388e-09,
00648     3.589e-09, 2.761e-09, 2.129e-09, 1.633e-09, 1.243e-09, 9.681e-10,
00649     8.355e-10, 7.665e-10, 7.442e-10, 8.584e-10, 9.732e-10, 1.063e-09,
00650     1.163e-09, 1.286e-09, 1.472e-09, 1.707e-09, 2.032e-09, 2.474e-09,
00651     2.977e-09, 3.506e-09, 4.102e-09, 5.013e-09, 6.493e-09, 8.414e-09,
00652     1.077e-08, 1.367e-08, 1.777e-08, 2.625e-08, 3.926e-08, 5.545e-08,
00653     7.195e-08, 9.464e-08, 1.404e-07, 2.183e-07, 3.329e-07, 4.535e-07,
00654     6.158e-07, 8.187e-07, 1.075e-06, 1.422e-06, 1.979e-06, 2.71e-06,
00655     3.58e-06, 4.573e-06, 5.951e-06, 7.999e-06, 1.072e-05, 1.372e-05,
00656     1.697e-05, 2.112e-05, 2.643e-05, 3.288e-05, 3.994e-05, 4.794e-05,
00657     5.606e-05, 6.383e-05, 7.286e-05, 8.156e-05, 8.883e-05, 9.469e-05,
00658     9.848e-05, 0.0001023, 0.0001066, 0.0001115, 0.0001145, 0.0001142,
00659     0.0001133
00660 };
00661
00662 static double no2[121] = {
00663     3.036e-09, 2.945e-10, 9.982e-11, 5.069e-11, 3.485e-11, 2.982e-11,
00664     2.947e-11, 3.164e-11, 3.714e-11, 4.586e-11, 6.164e-11, 8.041e-11,
00665     9.982e-11, 1.283e-10, 1.73e-10, 2.56e-10, 3.909e-10, 5.959e-10,
00666     9.081e-10, 1.384e-09, 1.788e-09, 2.189e-09, 2.686e-09, 3.091e-09,
00667     3.49e-09, 3.796e-09, 4.2e-09, 5.103e-09, 6.005e-09, 6.3e-09, 6.706e-09,
00668     7.07e-09, 7.434e-09, 7.663e-09, 7.788e-09, 7.8e-09, 7.597e-09,
00669     7.482e-09, 7.227e-09, 6.403e-09, 5.585e-09, 4.606e-09, 3.703e-09,
00670     2.984e-09, 2.183e-09, 1.48e-09, 8.441e-10, 5.994e-10, 3.799e-10,
00671     2.751e-10, 1.927e-10, 1.507e-10, 1.102e-10, 6.971e-11, 5.839e-11,
00672     3.904e-11, 3.087e-11, 2.176e-11, 1.464e-11, 1.209e-11, 8.497e-12,
00673     6.477e-12, 4.371e-12, 2.914e-12, 2.424e-12, 1.753e-12, 1.35e-12,
00674     9.417e-13, 6.622e-13, 5.148e-13, 3.841e-13, 3.446e-13, 3.01e-13,
00675     2.551e-13, 2.151e-13, 1.829e-13, 1.64e-13, 1.475e-13, 1.352e-13,
00676     1.155e-13, 9.963e-14, 9.771e-14, 9.577e-14, 9.384e-14, 9.186e-14,
00677     9e-14, 9e-14, 9e-14, 9e-14, 9e-14, 9e-14, 9e-14, 9e-14, 9e-14, 9e-14,
00678     9e-14, 9e-14, 9e-14, 9e-14, 9e-14, 9e-14, 9e-14, 9e-14, 9e-14,
00679     9e-14, 9e-14, 9e-14, 9e-14, 9e-14, 9e-14, 9e-14, 9e-14, 9e-14,
00680     9e-14, 9e-14, 9e-14, 9e-14, 9e-14, 9e-14
00681 };
00682
00683 static double o3[121] = {
00684     2.218e-08, 3.394e-08, 3.869e-08, 4.219e-08, 4.501e-08, 4.778e-08,
00685     5.067e-08, 5.402e-08, 5.872e-08, 6.521e-08, 7.709e-08, 9.461e-08,
00686     1.269e-07, 1.853e-07, 2.723e-07, 3.964e-07, 5.773e-07, 8.2e-07,
00687     1.155e-06, 1.59e-06, 2.076e-06, 2.706e-06, 3.249e-06, 3.848e-06,
00688     4.459e-06, 4.986e-06, 5.573e-06, 5.958e-06, 6.328e-06, 6.661e-06,
00689     6.9e-06, 7.146e-06, 7.276e-06, 7.374e-06, 7.447e-06, 7.383e-06,
00690     7.321e-06, 7.161e-06, 6.879e-06, 6.611e-06, 6.216e-06, 5.765e-06,
00691     5.355e-06, 4.905e-06, 4.471e-06, 4.075e-06, 3.728e-06, 3.413e-06,
00692     3.125e-06, 2.856e-06, 2.607e-06, 2.379e-06, 2.17e-06, 1.978e-06,
00693     1.8e-06, 1.646e-06, 1.506e-06, 1.376e-06, 1.233e-06, 1.102e-06,
00694     9.839e-07, 8.771e-07, 7.814e-07, 6.947e-07, 6.102e-07, 5.228e-07,
00695     4.509e-07, 3.922e-07, 3.501e-07, 3.183e-07, 2.909e-07, 2.686e-07,
00696     2.476e-07, 2.284e-07, 2.109e-07, 2.003e-07, 2.013e-07, 2.022e-07,
00697     2.032e-07, 2.042e-07, 2.097e-07, 2.361e-07, 2.656e-07, 2.989e-07,
00698     3.37e-07, 3.826e-07, 4.489e-07, 5.26e-07, 6.189e-07, 7.312e-07,
00699     8.496e-07, 8.444e-07, 8.392e-07, 8.339e-07, 8.286e-07, 8.234e-07,
00700     8.181e-07, 8.129e-07, 8.077e-07, 8.026e-07, 6.918e-07, 5.176e-07,
00701     3.865e-07, 2.885e-07, 2.156e-07, 1.619e-07, 1.219e-07, 9.161e-08,
00702     6.972e-08, 5.399e-08, 3.498e-08, 2.111e-08, 1.322e-08, 8.482e-09,
00703     5.527e-09, 3.423e-09, 2.071e-09, 1.314e-09, 8.529e-10, 5.503e-10,
00704     3.665e-10
00705 };
00706
00707 static double ocs[121] = {
00708     6e-10, 6e-10, 6e-10, 6e-10, 6e-10, 6e-10, 6e-10, 6e-10, 6e-10, 5.997e-10,
00709     5.989e-10, 5.881e-10, 5.765e-10, 5.433e-10, 5.074e-10, 4.567e-10,
00710     4.067e-10, 3.601e-10, 3.093e-10, 2.619e-10, 2.232e-10, 1.805e-10,
00711     1.46e-10, 1.187e-10, 8.03e-11, 5.435e-11, 3.686e-11, 2.217e-11,
00712     1.341e-11, 8.756e-12, 4.511e-12, 2.37e-12, 1.264e-12, 8.28e-13,
00713     5.263e-13, 3.209e-13, 1.717e-13, 9.068e-14, 4.709e-14, 2.389e-14,
00714     1.236e-14, 1.127e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14,
00715     1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14,
00716     1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14,
00717     1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14,
```



```

00718     1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14,
00719     1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14,
00720     1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14,
00721     1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14,
00722     1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14,
00723     1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14,
00724     1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14,
00725     1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14,
00726     1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14,
00727     1.091e-14, 1.091e-14, 1.091e-14
00728 };
00729
00730 static double sf6[121] = {
00731     4.103e-12, 4.103e-12, 4.103e-12, 4.103e-12, 4.103e-12, 4.103e-12,
00732     4.103e-12, 4.103e-12, 4.103e-12, 4.087e-12, 4.064e-12, 4.023e-12,
00733     3.988e-12, 3.941e-12, 3.884e-12, 3.755e-12, 3.622e-12, 3.484e-12,
00734     3.32e-12, 3.144e-12, 2.978e-12, 2.811e-12, 2.653e-12, 2.489e-12,
00735     2.332e-12, 2.199e-12, 2.089e-12, 2.013e-12, 1.953e-12, 1.898e-12,
00736     1.859e-12, 1.826e-12, 1.798e-12, 1.776e-12, 1.757e-12, 1.742e-12,
00737     1.728e-12, 1.717e-12, 1.707e-12, 1.698e-12, 1.691e-12, 1.685e-12,
00738     1.679e-12, 1.675e-12, 1.671e-12, 1.668e-12, 1.665e-12, 1.663e-12,
00739     1.661e-12, 1.659e-12, 1.658e-12, 1.657e-12, 1.656e-12, 1.655e-12,
00740     1.654e-12, 1.653e-12, 1.653e-12, 1.652e-12, 1.652e-12, 1.652e-12,
00741     1.651e-12, 1.651e-12, 1.651e-12, 1.651e-12, 1.651e-12, 1.651e-12,
00742     1.651e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12,
00743     1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12,
00744     1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12,
00745     1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12,
00746     1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12,
00747     1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12,
00748     1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12,
00749     1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12
00750 };
00751
00752 static double so2[121] = {
00753     1e-10, 1e-10, 1e-10, 1e-10, 1e-10, 1e-10, 1e-10, 1e-10, 1e-10, 1e-10,
00754     1e-10, 1e-10, 9.867e-11, 9.537e-11, 9e-11, 8.404e-11, 7.799e-11,
00755     7.205e-11, 6.616e-11, 6.036e-11, 5.475e-11, 5.007e-11, 4.638e-11,
00756     4.346e-11, 4.055e-11, 3.763e-11, 3.471e-11, 3.186e-11, 2.905e-11,
00757     2.631e-11, 2.358e-11, 2.415e-11, 2.949e-11, 3.952e-11, 5.155e-11,
00758     6.76e-11, 8.741e-11, 1.099e-10, 1.278e-10, 1.414e-10, 1.512e-10,
00759     1.607e-10, 1.699e-10, 1.774e-10, 1.832e-10, 1.871e-10, 1.907e-10,
00760     1.943e-10, 1.974e-10, 1.993e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10,
00761     2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10,
00762     2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10,
00763     2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10,
00764     2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10,
00765     2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10,
00766     2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10,
00767     2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10
00768 };
00769
00770 static int ig_co2 = -999;
00771
00772 double co2, *q[NG] = { NULL };
00773
00774 int ig, ip, iw, iz;
00775
00776 /* Find emitter index of CO2... */
00777 if (ig_co2 == -999)
00778     ig_co2 = find_emitter(ctl, "CO2");
00779
00780 /* Identify variable... */
00781 for (ig = 0; ig < ctl->ng; ig++) {
00782     q[ig] = NULL;
00783     if (strcasecmp(ctl->emitter[ig], "C2H2") == 0)
00784         q[ig] = c2h2;
00785     if (strcasecmp(ctl->emitter[ig], "C2H6") == 0)
00786         q[ig] = c2h6;
00787     if (strcasecmp(ctl->emitter[ig], "CCl4") == 0)
00788         q[ig] = ccl4;
00789     if (strcasecmp(ctl->emitter[ig], "CH4") == 0)
00790         q[ig] = ch4;
00791     if (strcasecmp(ctl->emitter[ig], "ClO") == 0)
00792         q[ig] = clo;
00793     if (strcasecmp(ctl->emitter[ig], "ClONO2") == 0)
00794         q[ig] = clono2;
00795     if (strcasecmp(ctl->emitter[ig], "CO") == 0)
00796         q[ig] = co;
00797     if (strcasecmp(ctl->emitter[ig], "COF2") == 0)
00798         q[ig] = cof2;
00799     if (strcasecmp(ctl->emitter[ig], "F11") == 0)
00800         q[ig] = f11;
00801     if (strcasecmp(ctl->emitter[ig], "F12") == 0)
00802         q[ig] = f12;
00803     if (strcasecmp(ctl->emitter[ig], "F14") == 0)
00804         q[ig] = f14;

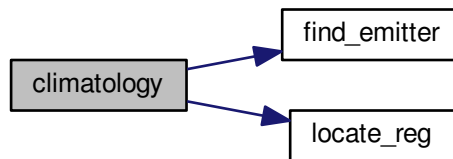
```

```

00805     if (strcasecmp(ctl->emitter[ig], "F22") == 0)
00806         q[ig] = f22;
00807     if (strcasecmp(ctl->emitter[ig], "H2O") == 0)
00808         q[ig] = h2o;
00809     if (strcasecmp(ctl->emitter[ig], "H2O2") == 0)
00810         q[ig] = h2o2;
00811     if (strcasecmp(ctl->emitter[ig], "HCN") == 0)
00812         q[ig] = hcn;
00813     if (strcasecmp(ctl->emitter[ig], "HNO3") == 0)
00814         q[ig] = hno3;
00815     if (strcasecmp(ctl->emitter[ig], "HNO4") == 0)
00816         q[ig] = hno4;
00817     if (strcasecmp(ctl->emitter[ig], "HOCl") == 0)
00818         q[ig] = hocl;
00819     if (strcasecmp(ctl->emitter[ig], "N2O") == 0)
00820         q[ig] = n2o;
00821     if (strcasecmp(ctl->emitter[ig], "N2O5") == 0)
00822         q[ig] = n2o5;
00823     if (strcasecmp(ctl->emitter[ig], "NH3") == 0)
00824         q[ig] = nh3;
00825     if (strcasecmp(ctl->emitter[ig], "NO") == 0)
00826         q[ig] = no;
00827     if (strcasecmp(ctl->emitter[ig], "NO2") == 0)
00828         q[ig] = no2;
00829     if (strcasecmp(ctl->emitter[ig], "O3") == 0)
00830         q[ig] = o3;
00831     if (strcasecmp(ctl->emitter[ig], "OCS") == 0)
00832         q[ig] = ocs;
00833     if (strcasecmp(ctl->emitter[ig], "SF6") == 0)
00834         q[ig] = sf6;
00835     if (strcasecmp(ctl->emitter[ig], "SO2") == 0)
00836         q[ig] = so2;
00837 }
00838
00839 /* Loop over atmospheric data points... */
00840 for (ip = 0; ip < atm->np; ip++) {
00841
00842     /* Get altitude index... */
00843     iz = locate_reg(z, 121, atm->z[ip]);
00844
00845     /* Interpolate pressure... */
00846     atm->p[ip] = EXP(z[iz], pre[iz], z[iz + 1], pre[iz + 1], atm->z[ip]);
00847
00848     /* Interpolate temperature... */
00849     atm->t[ip] = LIN(z[iz], tem[iz], z[iz + 1], tem[iz + 1], atm->z[ip]);
00850
00851     /* Interpolate trace gases... */
00852     for (ig = 0; ig < ctl->ng; ig++)
00853         if (q[ig] != NULL)
00854             atm->q[ig][ip] =
00855                 LIN(z[iz], q[ig][iz], z[iz + 1], q[ig][iz + 1], atm->z[ip]);
00856         else
00857             atm->q[ig][ip] = 0;
00858
00859     /* Set CO2... */
00860     if (ig_co2 >= 0) {
00861         co2 =
00862             371.789948e-6 + 2.026214e-6 * (atm->time[ip] - 63158400.) / 31557600.;
00863         atm->q[ig_co2][ip] = co2;
00864     }
00865
00866     /* Set extinction to zero... */
00867     for (iw = 0; iw < ctl->nw; iw++)
00868         atm->k[iw][ip] = 0;
00869 }
00870 }

```

Here is the call graph for this function:



#### 5.23.2.6 double ctmco2 ( double *nu*, double *p*, double *t*, double *u* )

Compute carbon dioxide continuum (optical depth).

Definition at line 874 of file [jurassic.c](#).

```

00878     {
00879
00880     static double co2296[2001] = { 9.3388e-5, 9.7711e-5, 1.0224e-4, 1.0697e-4,
00881     1.1193e-4, 1.1712e-4, 1.2255e-4, 1.2824e-4, 1.3419e-4, 1.4043e-4,
00882     1.4695e-4, 1.5378e-4, 1.6094e-4, 1.6842e-4, 1.7626e-4, 1.8447e-4,
00883     1.9307e-4, 2.0207e-4, 2.1149e-4, 2.2136e-4, 2.3169e-4, 2.4251e-4,
00884     2.5384e-4, 2.657e-4, 2.7813e-4, 2.9114e-4, 3.0477e-4, 3.1904e-4,
00885     3.3399e-4, 3.4965e-4, 3.6604e-4, 3.8322e-4, 4.0121e-4, 4.2006e-4,
00886     4.398e-4, 4.6047e-4, 4.8214e-4, 5.0483e-4, 5.286e-4, 5.535e-4,
00887     5.7959e-4, 6.0693e-4, 6.3557e-4, 6.6558e-4, 6.9702e-4, 7.2996e-4,
00888     7.6449e-4, 8.0066e-4, 8.3856e-4, 8.7829e-4, 9.1991e-4, 9.6354e-4,
00889     .0010093, .0010572, .0011074, .00116, .0012152, .001273,
00890     .0013336, .0013972, .0014638, .0015336, .0016068, .0016835,
00891     .001764, .0018483, .0019367, .0020295, .0021267, .0022286,
00892     .0023355, .0024476, .0025652, .0026885, .0028178, .0029534,
00893     .0030956, .0032448, .0034012, .0035654, .0037375, .0039181,
00894     .0041076, .0043063, .0045148, .0047336, .0049632, .005204,
00895     .0054567, .0057219, .0060002, .0062923, .0065988, .0069204,
00896     .007258, .0076123, .0079842, .0083746, .0087844, .0092146,
00897     .0096663, .01014, .010638, .011161, .01171, .012286, .012891,
00898     .013527, .014194, .014895, .015631, .016404, .017217, .01807,
00899     .018966, .019908, .020897, .021936, .023028, .024176, .025382,
00900     .026649, .027981, .02938, .030851, .032397, .034023, .035732,
00901     .037528, .039416, .041402, .04349, .045685, .047994, .050422,
00902     .052975, .055661, .058486, .061458, .064584, .067873, .071334,
00903     .074975, .078807, .082839, .087082, .091549, .096249, .1012,
00904     .10641, .11189, .11767, .12375, .13015, .13689, .14399, .15147,
00905     .15935, .16765, .17639, .18561, .19531, .20554, .21632, .22769,
00906     .23967, .25229, .2656, .27964, .29443, .31004, .3265, .34386,
00907     .36218, .3815, .40188, .42339, .44609, .47004, .49533, .52202,
00908     .5502, .57995, .61137, .64455, .6796, .71663, .75574, .79707,
00909     .84075, .88691, .9357, .98728, 1.0418, 1.0995, 1.1605, 1.225,
00910     1.2932, 1.3654, 1.4418, 1.5227, 1.6083, 1.6989, 1.7948, 1.8964,
00911     2.004, 2.118, 2.2388, 2.3668, 2.5025, 2.6463, 2.7988, 2.9606,
00912     3.1321, 3.314, 3.5071, 3.712, 3.9296, 4.1605, 4.4058, 4.6663,
00913     4.9431, 5.2374, 5.5501, 5.8818, 6.2353, 6.6114, 7.0115, 7.4372,
00914     7.8905, 8.3731, 8.8871, 9.4349, 10.019, 10.641, 11.305, 12.013,
00915     12.769, 13.576, 14.437, 15.358, 16.342, 17.39, 18.513, 19.716,
00916     21.003, 22.379, 23.854, 25.436, 27.126, 28.942, 30.89, 32.973,
00917     35.219, 37.634, 40.224, 43.021, 46.037, 49.29, 52.803, 56.447,
00918     60.418, 64.792, 69.526, 74.637, 80.182, 86.193, 92.713, 99.786,
00919     107.47, 115.84, 124.94, 134.86, 145.69, 157.49, 170.3, 184.39,
00920     199.83, 216.4, 234.55, 254.72, 276.82, 299.85, 326.16, 354.99,
00921     386.51, 416.68, 449.89, 490.12, 534.35, 578.25, 632.26, 692.61,
00922     756.43, 834.75, 924.11, 1016.9, 996.96, 1102.7, 1219.2, 1351.9,
00923     1494.3, 1654.1, 1826.5, 2027.9, 2249., 2453.8, 2714.4, 2999.4,
00924     3209.5, 3509., 3840.4, 3907.5, 4190.7, 4533.5, 4648.3, 5059.1,
00925     5561.6, 6191.4, 6820.8, 7905.9, 9362.2, 2431.3, 2211.3, 2046.8,
00926     2023.8, 1985.9, 1905.9, 1491.1, 1369.8, 1262.2, 1200.7, 887.74,
00927     820.25, 885.23, 887.21, 816.73, 1126.9, 1216.2, 1272.4, 1579.5,
00928     1634.2, 1656.3, 1657.9, 1789.5, 1670.8, 1509.5, 8474.6, 7489.2,
00929     6793.6, 6117., 5574.1, 5141.2, 5084.6, 4745.1, 4413.2, 4102.8,
  
```

```
00930 4024.7, 3715., 3398.6, 3100.8, 2900.4, 2629.2, 2374., 2144.7,
00931 1955.8, 1760.8, 1591.2, 1435.2, 1296.2, 1174., 1065.1, 967.76,
00932 999.48, 897.45, 809.23, 732.77, 670.26, 611.93, 560.11, 518.77,
00933 476.84, 438.8, 408.48, 380.21, 349.24, 322.71, 296.65, 272.85,
00934 251.96, 232.04, 213.88, 197.69, 182.41, 168.41, 155.79, 144.05,
00935 133.31, 123.48, 114.5, 106.21, 98.591, 91.612, 85.156, 79.204,
00936 73.719, 68.666, 63.975, 59.637, 56.35, 52.545, 49.042, 45.788,
00937 42.78, 39.992, 37.441, 35.037, 32.8, 30.744, 28.801, 26.986,
00938 25.297, 23.731, 22.258, 20.883, 19.603, 18.403, 17.295, 16.249,
00939 15.271, 14.356, 13.501, 12.701, 11.954, 11.254, 10.6, 9.9864,
00940 9.4118, 8.8745, 8.3714, 7.8997, 7.4578, 7.0446, 6.6573, 6.2949,
00941 5.9577, 5.6395, 5.3419, 5.063, 4.8037, 4.5608, 4.3452, 4.1364,
00942 3.9413, 3.7394, 3.562, 3.3932, 3.2325, 3.0789, 2.9318, 2.7898,
00943 2.6537, 2.5225, 2.3958, 2.2305, 2.1215, 2.0245, 1.9427, 1.8795,
00944 1.8336, 1.7604, 1.7016, 1.6419, 1.5282, 1.4611, 1.3443, 1.27,
00945 1.1675, 1.0824, 1.0534, .99833, .95854, .92981, .90887, .89346,
00946 .88113, .87068, .86102, .85096, .88262, .86151, .83565, .80518,
00947 .77045, .73736, .74744, .74954, .75773, .82267, .83493, .89402,
00948 .89725, .93426, .95564, .94045, .94174, .93404, .92035, .90456,
00949 .88621, .86673, .78117, .7515, .72056, .68822, .65658, .62764,
00950 .55984, .55598, .57407, .60963, .63763, .66198, .61132, .60972,
00951 .52496, .50649, .41872, .3964, .32422, .27276, .24048, .23772,
00952 .2286, .22711, .23999, .32038, .34371, .36621, .38561, .39953,
00953 .40636, .44913, .42716, .3919, .35477, .33935, .3351, .39746,
00954 .40993, .49398, .49956, .56157, .54742, .57295, .57386, .55417,
00955 .50745, .471, .43446, .39102, .34993, .31269, .27888, .24912,
00956 .22291, .19994, .17972, .16197, .14633, .13252, .12029, .10942,
00957 .099745, .091118, .083404, .076494, .070292, .064716, .059697,
00958 .055173, .051093, .047411, .044089, .041092, .038392, .035965,
00959 .033789, .031846, .030122, .028607, .02729, .026169, .025209,
00960 .024405, .023766, .023288, .022925, .022716, .022681, .022685,
00961 .022768, .023133, .023325, .023486, .024004, .024126, .024083,
00962 .023785, .024023, .023029, .021649, .021108, .019454, .017809,
00963 .017292, .016635, .017037, .018068, .018977, .018756, .017847,
00964 .016557, .016142, .014459, .012869, .012381, .010875, .0098701,
00965 .009285, .0091698, .0091701, .0096145, .010553, .01106, .012613,
00966 .014362, .015017, .016507, .017741, .01768, .017784, .0171,
00967 .016357, .016172, .017257, .018978, .020935, .021741, .023567,
00968 .025183, .025589, .026732, .027648, .028278, .028215, .02856,
00969 .029015, .029062, .028851, .028497, .027825, .027801, .026523,
00970 .02487, .022967, .022168, .020194, .018605, .017903, .018439,
00971 .019697, .020311, .020855, .020057, .018608, .016738, .015963,
00972 .013844, .011801, .011134, .0097573, .0086007, .0086226,
00973 .0083721, .0090978, .0097616, .0098426, .011317, .012853, .01447,
00974 .014657, .015771, .016351, .016079, .014829, .013431, .013185,
00975 .013207, .01448, .016176, .017971, .018265, .019526, .020455,
00976 .019797, .019802, .0194, .018176, .017505, .016197, .015339,
00977 .014401, .013213, .012203, .011186, .010236, .0093288, .0084854,
00978 .0076837, .0069375, .0062614, .0056628, .0051153, .0046015,
00979 .0041501, .003752, .0033996, .0030865, .0028077, .0025586,
00980 .0023355, .0021353, .0019553, .0017931, .0016466, .0015141,
00981 .0013941, .0012852, .0011862, .0010962, .0010142, 9.3935e-4,
00982 8.71e-4, 8.0851e-4, 7.5132e-4, 6.9894e-4, 6.5093e-4, 6.0689e-4,
00983 5.6647e-4, 5.2935e-4, 4.9525e-4, 4.6391e-4, 4.3509e-4, 4.086e-4,
00984 3.8424e-4, 3.6185e-4, 3.4126e-4, 3.2235e-4, 3.0498e-4, 2.8904e-4,
00985 2.7444e-4, 2.6106e-4, 2.4883e-4, 2.3766e-4, 2.275e-4, 2.1827e-4,
00986 2.0992e-4, 2.0239e-4, 1.9563e-4, 1.896e-4, 1.8427e-4, 1.796e-4,
00987 1.7555e-4, 1.7209e-4, 1.692e-4, 1.6687e-4, 1.6505e-4, 1.6375e-4,
00988 1.6294e-4, 1.6261e-4, 1.6274e-4, 1.6334e-4, 1.6438e-4, 1.6587e-4,
00989 1.678e-4, 1.7017e-4, 1.7297e-4, 1.762e-4, 1.7988e-4, 1.8399e-4,
00990 1.8855e-4, 1.9355e-4, 1.9902e-4, 2.0494e-4, 2.1134e-4, 2.1823e-4,
00991 2.2561e-4, 2.335e-4, 2.4192e-4, 2.5088e-4, 2.604e-4, 2.705e-4,
00992 2.8119e-4, 2.9251e-4, 3.0447e-4, 3.171e-4, 3.3042e-4, 3.4447e-4,
00993 3.5927e-4, 3.7486e-4, 3.9127e-4, 4.0854e-4, 4.267e-4, 4.4579e-4,
00994 4.6586e-4, 4.8696e-4, 5.0912e-4, 5.324e-4, 5.5685e-4, 5.8253e-4,
00995 6.0949e-4, 6.378e-4, 6.6753e-4, 6.9873e-4, 7.3149e-4, 7.6588e-4,
00996 8.0198e-4, 8.3987e-4, 8.7964e-4, 9.2139e-4, 9.6522e-4, .0010112,
00997 .0010595, .0011102, .0011634, .0012193, .001278, .0013396,
00998 .0014043, .0014722, .0015436, .0016185, .0016972, .0017799,
00999 .0018668, .001958, .0020539, .0021547, .0022606, .0023719,
01000 .002489, .002612, .0027414, .0028775, .0030206, .0031712,
01001 .0033295, .0034962, .0036716, .0038563, .0040506, .0042553,
01002 .0044709, .004698, .0049373, .0051894, .0054552, .0057354,
01003 .006031, .0063427, .0066717, .0070188, .0073854, .0077726,
01004 .0081816, .0086138, .0090709, .0095543, .010066, .010607,
01005 .011181, .011789, .012433, .013116, .013842, .014613, .015432,
01006 .016304, .017233, .018224, .019281, .020394, .021574, .022836,
01007 .024181, .025594, .027088, .028707, .030401, .032245, .034219,
01008 .036262, .038539, .040987, .043578, .04641, .04949, .052726,
01009 .056326, .0602, .064093, .068521, .073278, .077734, .083064,
01010 .088731, .093885, .1003, .1072, .11365, .12187, .13078, .13989,
01011 .15095, .16299, .17634, .19116, .20628, .22419, .24386, .26587,
01012 .28811, .31399, .34321, .36606, .39675, .42742, .44243, .47197,
01013 .49993, .49027, .51147, .52803, .48931, .49729, .5026, .43854,
01014 .441, .44766, .43414, .46151, .50029, .55247, .43855, .32115,
01015 .32607, .3431, .36119, .38029, .41179, .43996, .47144, .51853,
01016 .55362, .59122, .66338, .69877, .74001, .82923, .86907, .90361,
```

01017 1.0025, 1.031, 1.0559, 1.104, 1.1178, 1.1341, 1.1547, 1.351,  
 01018 1.4772, 1.4812, 1.4907, 1.512, 1.5442, 1.5853, 1.6358, 1.6963,  
 01019 1.7674, 1.8474, 1.9353, 2.0335, 2.143, 2.2592, 2.3853, 2.5217,  
 01020 2.6686, 2.8273, 2.9998, 3.183, 3.3868, 3.6109, 3.8564, 4.1159,  
 01021 4.4079, 4.7278, 5.0497, 5.3695, 5.758, 6.0834, 6.4976, 6.9312,  
 01022 7.38, 7.5746, 7.9833, 8.3791, 8.3956, 8.7501, 9.1067, 9.072,  
 01023 9.4649, 9.9112, 10.402, 10.829, 11.605, 12.54, 12.713, 10.443,  
 01024 10.825, 11.375, 11.955, 12.623, 13.326, 14.101, 15.041, 15.547,  
 01025 16.461, 17.439, 18.716, 19.84, 21.036, 22.642, 23.901, 25.244,  
 01026 27.03, 28.411, 29.871, 31.403, 33.147, 34.744, 36.456, 39.239,  
 01027 43.605, 45.162, 47.004, 49.093, 51.391, 53.946, 56.673, 59.629,  
 01028 63.167, 66.576, 70.254, 74.222, 78.477, 83.034, 87.914, 93.18,  
 01029 98.77, 104.74, 111.15, 117.95, 125.23, 133.01, 141.33, 150.21,  
 01030 159.71, 169.89, 180.93, 192.54, 204.99, 218.34, 232.65, 248.,  
 01031 264.47, 282.14, 301.13, 321.53, 343.48, 367.08, 392.5, 419.88,  
 01032 449.4, 481.26, 515.64, 552.79, 592.99, 636.48, 683.61, 734.65,  
 01033 789.99, 850.02, 915.14, 985.81, 1062.5, 1147.1, 1237.8, 1336.4,  
 01034 1443.2, 1558.9, 1684.2, 1819.2, 1965.2, 2122.6, 2291.7, 2470.8,  
 01035 2665.7, 2874.9, 3099.4, 3337.9, 3541., 3813.3, 4111.9, 4439.3,  
 01036 4798.9, 5196., 5639.2, 6087.5, 6657.7, 7306.7, 8040.7, 8845.5,  
 01037 9702.2, 10670., 11739., 12842., 14141., 15498., 17068., 18729.,  
 01038 20557., 22559., 25248., 27664., 30207., 32915., 35611., 38081.,  
 01039 40715., 43191., 41651., 42750., 43785., 44353., 44366., 44189.,  
 01040 43618., 42862., 41878., 35133., 35215., 36383., 39420., 44055.,  
 01041 44155., 45850., 46853., 39197., 38274., 29942., 28553., 21792.,  
 01042 21228., 17106., 14955., 18181., 19557., 21427., 23728., 26301.,  
 01043 28584., 30775., 32536., 33867., 40089., 39204., 37329., 34452.,  
 01044 31373., 33921., 34800., 36043., 44415., 45162., 52181., 50895.,  
 01045 54140., 50840., 50468., 48302., 44915., 40910., 36754., 32755.,  
 01046 29093., 25860., 22962., 20448., 18247., 16326., 14645., 13165.,  
 01047 11861., 10708., 9686.9, 8779.7, 7971.9, 7250.8, 6605.7, 6027.2,  
 01048 5507.3, 5039.1, 4616.6, 4234.8, 3889., 3575.4, 3290.5, 3031.3,  
 01049 2795.2, 2579.9, 2383.1, 2203.3, 2038.6, 1887.6, 1749.1, 1621.9,  
 01050 1505., 1397.4, 1298.3, 1207., 1122.8, 1045., 973.1, 906.64,  
 01051 845.16, 788.22, 735.48, 686.57, 641.21, 599.1, 559.99, 523.64,  
 01052 489.85, 458.42, 429.16, 401.92, 376.54, 352.88, 330.82, 310.24,  
 01053 291.03, 273.09, 256.34, 240.69, 226.05, 212.37, 199.57, 187.59,  
 01054 176.37, 165.87, 156.03, 146.82, 138.17, 130.07, 122.47, 115.34,  
 01055 108.65, 102.37, 96.473, 90.934, 85.73, 80.84, 76.243, 71.922,  
 01056 67.858, 64.034, 60.438, 57.052, 53.866, 50.866, 48.04, 45.379,  
 01057 42.872, 40.51, 38.285, 36.188, 34.211, 32.347, 30.588, 28.929,  
 01058 27.362, 25.884, 24.489, 23.171, 21.929, 20.755, 19.646, 18.599,  
 01059 17.61, 16.677, 15.795, 14.961, 14.174, 13.43, 12.725, 12.06,  
 01060 11.431, 10.834, 10.27, 9.7361, 9.2302, 8.7518, 8.2997, 7.8724,  
 01061 7.4674, 7.0848, 6.7226, 6.3794, 6.054, 5.745, 5.4525, 5.1752,  
 01062 4.9121, 4.6625, 4.4259, 4.2015, 3.9888, 3.7872, 3.5961, 3.4149,  
 01063 3.2431, 3.0802, 2.9257, 2.7792, 2.6402, 2.5084, 2.3834, 2.2648,  
 01064 2.1522, 2.0455, 1.9441, 1.848, 1.7567, 1.6701, 1.5878, 1.5097,  
 01065 1.4356, 1.3651, 1.2981, 1.2345, 1.174, 1.1167, 1.062, 1.0101,  
 01066 .96087, .91414, .86986, .82781, .78777, .74971, .71339, .67882,  
 01067 .64604, .61473, .58507, .55676, .52987, .5044, .48014, .45715,  
 01068 .43527, .41453, .3948, .37609, .35831, .34142, .32524, .30995,  
 01069 .29536, .28142, .26807, .25527, .24311, .23166, .22077, .21053,  
 01070 .20081, .19143, .18261, .17407, .16603, .15833, .15089, .14385,  
 01071 .13707, .13065, .12449, .11865, .11306, .10774, .10266, .097818,  
 01072 .093203, .088815, .084641, .080671, .076892, .073296, .069873,  
 01073 .066613, .06351, .060555, .05774, .055058, .052504, .050071,  
 01074 .047752, .045543, .043438, .041432, .039521, .037699, .035962,  
 01075 .034307, .032729, .031225, .029791, .028423, .02712, .025877,  
 01076 .024692, .023563, .022485, .021458, .020478, .019543, .018652,  
 01077 .017802, .016992, .016219, .015481, .014778, .014107, .013467,  
 01078 .012856, .012274, .011718, .011188, .010682, .0102, .0097393,  
 01079 .0093001, .008881, .0084812, .0080997, .0077358, .0073885,  
 01080 .0070571, .0067409, .0064393, .0061514, .0058768, .0056147,  
 01081 .0053647, .0051262, .0048987, .0046816, .0044745, .0042769,  
 01082 .0040884, .0039088, .0037373, .0035739, .003418, .0032693,  
 01083 .0031277, .0029926, .0028639, .0027413, .0026245, .0025133,  
 01084 .0024074, .0023066, .0022108, .0021196, .002033, .0019507,  
 01085 .0018726, .0017985, .0017282, .0016617, .0015988, .0015394,  
 01086 .0014834, .0014306, .0013811, .0013346, .0012911, .0012506,  
 01087 .0012131, .0011784, .0011465, .0011175, .0010912, .0010678,  
 01088 .0010472, .0010295, .0010147, .001003, 9.9428e-4, 9.8883e-4,  
 01089 9.8673e-4, 9.8821e-4, 9.9343e-4, .0010027, .0010164, .0010348,  
 01090 .0010586, .0010882, .0011245, .0011685, .0012145, .0012666,  
 01091 .0013095, .0013688, .0014048, .0014663, .0015309, .0015499,  
 01092 .0016144, .0016312, .001705, .0017892, .0018499, .0019715,  
 01093 .0021102, .0022442, .0024284, .0025893, .0027703, .0029445,  
 01094 .0031193, .003346, .0034552, .0036906, .0037584, .0040084,  
 01095 .0041934, .0044587, .0047093, .0049759, .0053421, .0055134,  
 01096 .0059048, .0058663, .0061036, .0063259, .0059657, .0060653,  
 01097 .0060972, .0055539, .0055653, .0055772, .005331, .0054953,  
 01098 .0055919, .0058684, .006183, .0066675, .0069808, .0075142,  
 01099 .0078536, .0084282, .0089454, .0094625, .0093703, .0095857,  
 01100 .0099283, .010063, .010521, .0097778, .0098175, .010379, .010447,  
 01101 .0105, .010617, .010706, .01078, .011177, .011212, .011304,  
 01102 .011446, .011603, .011816, .012165, .012545, .013069, .013539,  
 01103 .01411, .014776, .016103, .017016, .017994, .018978, .01998,

```

01104 .021799, .022745, .023681, .024627, .025562, .026992, .027958,
01105 .029013, .030154, .031402, .03228, .033651, .035272, .037088,
01106 .039021, .041213, .043597, .045977, .04877, .051809, .054943,
01107 .058064, .061528, .06537, .069309, .071928, .075752, .079589,
01108 .083352, .084096, .087497, .090817, .091198, .094966, .099045,
01109 .10429, .10867, .11518, .12269, .13126, .14087, .15161, .16388,
01110 .16423, .1759, .18721, .19994, .21275, .22513, .23041, .24231,
01111 .25299, .25396, .26396, .27696, .27929, .2908, .30595, .31433,
01112 .3282, .3429, .35944, .37467, .39277, .41245, .43326, .45649,
01113 .48152, .51897, .54686, .57877, .61263, .64962, .68983, .73945,
01114 .78619, .83537, .89622, .95002, 1.0067, 1.0742, 1.1355, 1.2007,
01115 1.2738, 1.347, 1.4254, 1.5094, 1.6009, 1.6976, 1.8019, 1.9148,
01116 2.0357, 2.166, 2.3066, 2.4579, 2.6208, 2.7966, 2.986, 3.188,
01117 3.4081, 3.6456, 3.9, 4.1747, 4.4712, 4.7931, 5.1359, 5.5097,
01118 5.9117, 6.3435, 6.8003, 7.3001, 7.8385, 8.3945, 9.011, 9.6869,
01119 10.392, 11.18, 12.036, 12.938, 13.944, 14.881, 16.029, 17.255,
01120 18.574, 19.945, 21.38, 22.9, 24.477, 26.128, 27.87, 29.037,
01121 30.988, 33.145, 35.506, 37.76, 40.885, 44.487, 48.505, 52.911,
01122 57.56, 61.964, 67.217, 72.26, 78.343, 85.08, 91.867, 99.435,
01123 107.68, 116.97, 127.12, 138.32, 150.26, 163.04, 174.81, 189.26,
01124 205.61, 224.68, 240.98, 261.88, 285.1, 307.58, 334.35, 363.53,
01125 394.68, 427.85, 458.85, 489.25, 472.87, 486.93, 496.27, 501.52,
01126 501.57, 497.14, 488.09, 476.32, 393.76, 388.51, 393.42, 414.45,
01127 455.12, 514.62, 520.38, 547.42, 562.6, 487.47, 480.83, 391.06,
01128 376.92, 303.7, 295.91, 256.03, 236.73, 280.38, 310.71, 335.53,
01129 367.88, 401.94, 435.52, 469.13, 497.94, 588.82, 597.94, 597.2,
01130 588.28, 571.2, 555.75, 603.56, 638.15, 680.75, 801.72, 848.01,
01131 962.15, 990.06, 1068.1, 1076.2, 1115.3, 1134.2, 1136.6, 1119.1,
01132 1108.9, 1090.6, 1068.7, 1041.9, 1005.4, 967.98, 927.08, 780.1,
01133 751.41, 733.12, 742.65, 785.56, 855.16, 852.45, 878.1, 784.59,
01134 777.81, 765.13, 622.93, 498.09, 474.89, 386.9, 378.48, 336.17,
01135 322.04, 329.57, 350.5, 383.38, 420.02, 462.39, 499.71, 531.98,
01136 654.99, 653.43, 639.99, 605.16, 554.16, 504.42, 540.64, 552.33,
01137 679.46, 699.51, 713.91, 832.17, 919.91, 884.96, 907.57, 846.56,
01138 818.56, 768.93, 706.71, 642.17, 575.95, 515.38, 459.07, 409.02,
01139 364.61, 325.46, 291.1, 260.89, 234.39, 211.01, 190.38, 172.11,
01140 155.91, 141.49, 128.63, 117.13, 106.84, 97.584, 89.262, 81.756,
01141 74.975, 68.842, 63.28, 58.232, 53.641, 49.46, 45.649, 42.168,
01142 38.991, 36.078, 33.409, 30.96, 28.71, 26.642, 24.737, 22.985,
01143 21.37, 19.882, 18.512, 17.242, 16.073, 14.987, 13.984, 13.05,
01144 12.186, 11.384, 10.637, 9.9436, 9.2988, 8.6991, 8.141, 7.6215,
01145 7.1378, 6.6872, 6.2671, 5.8754, 5.51, 5.1691, 4.851, 4.5539,
01146 4.2764, 4.0169, 3.7742, 3.5472, 3.3348, 3.1359, 2.9495, 2.7749,
01147 2.6113, 2.4578, 2.3139, 2.1789, 2.0523, 1.9334, 1.8219, 1.7171,
01148 1.6188, 1.5263, 1.4395, 1.3579, 1.2812, 1.209, 1.1411, 1.0773,
01149 1.0171, .96048, .90713, .85684, .80959, .76495, .72282, .68309,
01150 .64563, .61035, .57707, .54573, .51622, .48834, .46199, .43709,
01151 .41359, .39129, .37034, .35064, .33198, .31442, .29784, .28218,
01152 .26732, .25337, .24017, .22774, .21601, .20479, .19426
01153 };
01154
01155 static double co2260[2001] = { 5.7971e-5, 6.0733e-5, 6.3628e-5, 6.6662e-5,
01156 6.9843e-5, 7.3176e-5, 7.6671e-5, 8.0334e-5, 8.4175e-5, 8.8201e-5,
01157 9.2421e-5, 9.6846e-5, 1.0149e-4, 1.0635e-4, 1.1145e-4, 1.1679e-4,
01158 1.224e-4, 1.2828e-4, 1.3444e-4, 1.409e-4, 1.4768e-4, 1.5479e-4,
01159 1.6224e-4, 1.7006e-4, 1.7826e-4, 1.8685e-4, 1.9587e-4, 2.0532e-4,
01160 2.1524e-4, 2.2565e-4, 2.3656e-4, 2.48e-4, 2.6001e-4, 2.7261e-4,
01161 2.8582e-4, 2.9968e-4, 3.1422e-4, 3.2948e-4, 3.4548e-4, 3.6228e-4,
01162 3.799e-4, 3.9838e-4, 4.1778e-4, 4.3814e-4, 4.595e-4, 4.8191e-4,
01163 5.0543e-4, 5.3012e-4, 5.5603e-4, 5.8321e-4, 6.1175e-4, 6.417e-4,
01164 6.7314e-4, 7.0614e-4, 7.4078e-4, 7.7714e-4, 8.1531e-4, 8.5538e-4,
01165 8.9745e-4, 9.4162e-4, 9.8798e-4, .0010367, .0010878, .0011415,
01166 .0011978, .001257, .0013191, .0013844, .001453, .0015249,
01167 .0016006, .00168, .0017634, .001851, .001943, .0020397, .0021412,
01168 .0022479, .00236, .0024778, .0026015, .0027316, .0028682,
01169 .0030117, .0031626, .0033211, .0034877, .0036628, .0038469,
01170 .0040403, .0042436, .0044574, .004682, .0049182, .0051665,
01171 .0054276, .0057021, .0059907, .0062942, .0066133, .0069489,
01172 .0073018, .0076729, .0080632, .0084738, .0089056, .0093599,
01173 .0098377, .01034, .010869, .011426, .012011, .012627, .013276,
01174 .013958, .014676, .015431, .016226, .017063, .017944, .018872,
01175 .019848, .020876, .021958, .023098, .024298, .025561, .026892,
01176 .028293, .029769, .031323, .032961, .034686, .036503, .038418,
01177 .040435, .042561, .044801, .047161, .049649, .052271, .055035,
01178 .057948, .061019, .064256, .06767, .07127, .075066, .079069,
01179 .083291, .087744, .092441, .097396, .10262, .10814, .11396,
01180 .1201, .12658, .13342, .14064, .14826, .1563, .1648, .17376,
01181 .18323, .19324, .2038, .21496, .22674, .23919, .25234, .26624,
01182 .28093, .29646, .31287, .33021, .34855, .36794, .38844, .41012,
01183 .43305, .45731, .48297, .51011, .53884, .56924, .60141, .63547,
01184 .67152, .70969, .75012, .79292, .83826, .8863, .93718, .99111,
01185 1.0482, 1.1088, 1.173, 1.2411, 1.3133, 1.3898, 1.471, 1.5571,
01186 1.6485, 1.7455, 1.8485, 1.9577, 2.0737, 2.197, 2.3278, 2.4668,
01187 2.6145, 2.7715, 2.9383, 3.1156, 3.3042, 3.5047, 3.7181, 3.9451,
01188 4.1866, 4.4437, 4.7174, 5.0089, 5.3192, 5.65, 6.0025, 6.3782,
01189 6.7787, 7.206, 7.6617, 8.1479, 8.6669, 9.221, 9.8128, 10.445,
01190 11.12, 11.843, 12.615, 13.441, 14.325, 15.271, 16.283, 17.367,

```

01191 18.529, 19.776, 21.111, 22.544, 24.082, 25.731, 27.504, 29.409,  
01192 31.452, 33.654, 36.024, 38.573, 41.323, 44.29, 47.492, 50.951,  
01193 54.608, 58.588, 62.929, 67.629, 72.712, 78.226, 84.207, 90.699,  
01194 97.749, 105.42, 113.77, 122.86, 132.78, 143.61, 155.44, 168.33,  
01195 182.48, 198.01, 214.87, 233.39, 253.86, 276.34, 300.3, 327.28,  
01196 356.89, 389.48, 422.29, 458.99, 501.39, 548.13, 595.62, 652.74,  
01197 716.54, 784.57, 866.78, 960.59, 1062.8, 1072.5, 1189.5, 1319.4,  
01198 1467.6, 1630.2, 1813.7, 2016.9, 2253., 2515.3, 2773.5, 3092.8,  
01199 3444.4, 3720.4, 4104.3, 4527.5, 4645.9, 5021.7, 5462.2, 5597.,  
01200 6110.6, 6732.5, 7513.8, 8270.6, 9640.6, 11487., 2796.1, 2680.1,  
01201 2441.6, 2404.2, 2334.8, 2215.2, 1642.5, 1477.9, 1328.1, 1223.5,  
01202 843.34, 766.96, 831.65, 834.84, 774.85, 1156.3, 1275.6, 1366.1,  
01203 1795.6, 1885., 1936.5, 1953.4, 2154.4, 2002.7, 1789.8, 10381.,  
01204 9040., 8216.5, 7384.7, 6721.9, 6187.7, 6143.8, 5703.9, 5276.6,  
01205 4873.1, 4736., 4325.3, 3927., 3554.1, 3286.1, 2950.1, 2642.4,  
01206 2368.7, 2138.9, 1914., 1719.6, 1543.9, 1388.6, 1252.1, 1132.2,  
01207 1024.1, 1025.4, 920.58, 829.59, 750.54, 685.01, 624.25, 570.14,  
01208 525.81, 481.85, 441.95, 408.71, 377.23, 345.86, 318.51, 292.26,  
01209 268.34, 247.04, 227.14, 209.02, 192.69, 177.59, 163.78, 151.26,  
01210 139.73, 129.19, 119.53, 110.7, 102.57, 95.109, 88.264, 81.948,  
01211 76.13, 70.768, 65.827, 61.251, 57.022, 53.495, 49.824, 46.443,  
01212 43.307, 40.405, 37.716, 35.241, 32.923, 30.77, 28.78, 26.915,  
01213 25.177, 23.56, 22.059, 20.654, 19.345, 18.126, 16.988, 15.93,  
01214 14.939, 14.014, 13.149, 12.343, 11.589, 10.884, 10.225, 9.6093,  
01215 9.0327, 8.4934, 7.9889, 7.5166, 7.0744, 6.6604, 6.2727, 5.9098,  
01216 5.5701, 5.2529, 4.955, 4.676, 4.4148, 4.171, 3.9426, 3.7332,  
01217 3.5347, 3.3493, 3.1677, 3.0025, 2.8466, 2.6994, 2.5601, 2.4277,  
01218 2.3016, 2.1814, 2.0664, 1.9564, 1.8279, 1.7311, 1.6427, 1.5645,  
01219 1.4982, 1.443, 1.374, 1.3146, 1.2562, 1.17, 1.1105, 1.0272,  
01220 .96863, .89718, .83654, .80226, .75908, .72431, .69573, .67174,  
01221 .65126, .63315, .61693, .60182, .58715, .59554, .57649, .55526,  
01222 .53177, .50622, .48176, .4813, .47642, .47492, .50273, .50293,  
01223 .52687, .52239, .53419, .53814, .52626, .52211, .51492, .50622,  
01224 .49746, .48841, .4792, .43534, .41999, .40349, .38586, .36799,  
01225 .35108, .31089, .30803, .3171, .33599, .35041, .36149, .32924,  
01226 .32462, .27309, .25961, .20922, .19504, .15683, .13098, .11588,  
01227 .11478, .11204, .11363, .12135, .16423, .17785, .19094, .20236,  
01228 .21084, .2154, .24108, .22848, .20871, .18797, .17963, .17834,  
01229 .21552, .22284, .26945, .27052, .30108, .28977, .29772, .29224,  
01230 .27658, .24956, .22777, .20654, .18392, .16338, .1452, .12916,  
01231 .1152, .10304, .092437, .083163, .075031, .067878, .061564,  
01232 .055976, .051018, .046609, .042679, .03917, .036032, .033223,  
01233 .030706, .02845, .026428, .024617, .022998, .021554, .02027,  
01234 .019136, .018141, .017278, .016541, .015926, .015432, .015058,  
01235 .014807, .014666, .014635, .014728, .014947, .01527, .015728,  
01236 .016345, .017026, .017798, .018839, .019752, .020636, .021886,  
01237 .022695, .02327, .023478, .024292, .023544, .022222, .021932,  
01238 .020052, .018143, .017722, .017031, .017782, .01938, .020734,  
01239 .020476, .019255, .017477, .016878, .014617, .012489, .011765,  
01240 .0099077, .0086446, .0079446, .0078644, .0079763, .008671,  
01241 .01001, .0108, .012933, .015349, .016341, .018484, .020254,  
01242 .020254, .020478, .019591, .018595, .018385, .019913, .022254,  
01243 .024847, .025809, .028053, .029924, .030212, .031367, .03222,  
01244 .032739, .032537, .03286, .033344, .033507, .033499, .033339,  
01245 .032809, .033041, .031723, .029837, .027511, .026603, .024032,  
01246 .021914, .020948, .021701, .023425, .024259, .024987, .023818,  
01247 .021768, .019223, .018144, .015282, .012604, .01163, .0097907,  
01248 .008336, .0082473, .0079582, .0088077, .009779, .010129, .012145,  
01249 .014378, .016761, .01726, .018997, .019998, .019809, .01819,  
01250 .016358, .016099, .01617, .017939, .020223, .022521, .02277,  
01251 .024279, .025247, .024222, .023989, .023224, .021493, .020362,  
01252 .018596, .017309, .015975, .014466, .013171, .011921, .01078,  
01253 .0097229, .0087612, .0078729, .0070682, .0063494, .0057156,  
01254 .0051459, .0046273, .0041712, .0037686, .0034119, .003095,  
01255 .0028126, .0025603, .0023342, .0021314, .0019489, .0017845,  
01256 .001636, .0015017, .00138, .0012697, .0011694, .0010782,  
01257 9.9507e-4, 9.1931e-4, 8.5013e-4, 7.869e-4, 7.2907e-4, 6.7611e-4,  
01258 6.2758e-4, 5.8308e-4, 5.4223e-4, 5.0473e-4, 4.7027e-4, 4.3859e-4,  
01259 4.0946e-4, 3.8265e-4, 3.5798e-4, 3.3526e-4, 3.1436e-4, 2.9511e-4,  
01260 2.7739e-4, 2.6109e-4, 2.4609e-4, 2.3229e-4, 2.1961e-4, 2.0797e-4,  
01261 1.9729e-4, 1.875e-4, 1.7855e-4, 1.7038e-4, 1.6294e-4, 1.5619e-4,  
01262 1.5007e-4, 1.4456e-4, 1.3961e-4, 1.3521e-4, 1.3131e-4, 1.2789e-4,  
01263 1.2494e-4, 1.2242e-4, 1.2032e-4, 1.1863e-4, 1.1733e-4, 1.1641e-4,  
01264 1.1585e-4, 1.1565e-4, 1.158e-4, 1.1629e-4, 1.1712e-4, 1.1827e-4,  
01265 1.1976e-4, 1.2158e-4, 1.2373e-4, 1.262e-4, 1.2901e-4, 1.3214e-4,  
01266 1.3562e-4, 1.3944e-4, 1.4361e-4, 1.4814e-4, 1.5303e-4, 1.5829e-4,  
01267 1.6394e-4, 1.6999e-4, 1.7644e-4, 1.8332e-4, 1.9063e-4, 1.984e-4,  
01268 2.0663e-4, 2.1536e-4, 2.246e-4, 2.3436e-4, 2.4468e-4, 2.5558e-4,  
01269 2.6708e-4, 2.7921e-4, 2.92e-4, 3.0548e-4, 3.1968e-4, 3.3464e-4,  
01270 3.5039e-4, 3.6698e-4, 3.8443e-4, 4.0281e-4, 4.2214e-4, 4.4248e-4,  
01271 4.6389e-4, 4.864e-4, 5.1009e-4, 5.3501e-4, 5.6123e-4, 5.888e-4,  
01272 6.1781e-4, 6.4833e-4, 6.8043e-4, 7.142e-4, 7.4973e-4, 7.8711e-4,  
01273 8.2644e-4, 8.6783e-4, 9.1137e-4, 9.5721e-4, .0010054, .0010562,  
01274 .0011096, .0011659, .0012251, .0012875, .0013532, .0014224,  
01275 .0014953, .001572, .0016529, .0017381, .0018279, .0019226,  
01276 .0020224, .0021277, .0022386, .0023557, .0024792, .0026095,  
01277 .002747, .0028921, .0030453, .0032071, .003378, .0035586,

```
01278 .0037494, .003951, .0041642, .0043897, .0046282, .0048805,
01279 .0051476, .0054304, .00573, .0060473, .0063837, .0067404,
01280 .0071188, .0075203, .0079466, .0083994, .0088806, .0093922,
01281 .0099366, .010516, .011134, .011792, .012494, .013244, .014046,
01282 .014898, .015808, .016781, .017822, .018929, .020108, .02138,
01283 .022729, .02419, .02576, .027412, .029233, .031198, .033301,
01284 .035594, .038092, .040767, .04372, .046918, .050246, .053974,
01285 .058009, .061976, .066586, .071537, .076209, .081856, .087998,
01286 .093821, .10113, .10913, .11731, .12724, .13821, .15025, .1639,
01287 .17807, .19472, .21356, .23496, .25758, .28387, .31389, .34104,
01288 .37469, .40989, .43309, .46845, .5042, .5023, .52981, .55275,
01289 .51075, .51976, .52457, .44779, .44721, .4503, .4243, .45244,
01290 .49491, .55399, .39021, .24802, .2501, .2618, .27475, .28879,
01291 .31317, .33643, .36257, .4018, .43275, .46525, .53333, .56599,
01292 .60557, .70142, .74194, .77736, .88567, .91182, .93294, .98407,
01293 .98772, .99176, .9995, 1.2405, 1.3602, 1.338, 1.3255, 1.3267,
01294 1.3404, 1.3634, 1.3967, 1.4407, 1.4961, 1.5603, 1.6328, 1.7153,
01295 1.8094, 1.9091, 2.018, 2.1367, 2.264, 2.4035, 2.5562, 2.7179,
01296 2.9017, 3.1052, 3.3304, 3.5731, 3.8488, 4.1553, 4.4769, 4.7818,
01297 5.1711, 5.5204, 5.9516, 6.4097, 6.8899, 7.1118, 7.5469, 7.9735,
01298 7.9511, 8.3014, 8.6418, 8.4757, 8.8256, 9.2294, 9.6923, 10.033,
01299 10.842, 11.851, 11.78, 8.8435, 9.1381, 9.5956, 10.076, 10.629,
01300 11.22, 11.883, 12.69, 13.163, 13.974, 14.846, 16.027, 17.053,
01301 18.148, 19.715, 20.907, 22.163, 23.956, 25.235, 26.566, 27.94,
01302 29.576, 30.956, 32.432, 35.337, 39.911, 41.128, 42.625, 44.386,
01303 46.369, 48.619, 51.031, 53.674, 56.825, 59.921, 63.286, 66.929,
01304 70.859, 75.081, 79.618, 84.513, 89.739, 95.335, 101.35, 107.76,
01305 114.63, 121.98, 129.87, 138.3, 147.34, 157.04, 167.56, 178.67,
01306 190.61, 203.43, 217.19, 231.99, 247.88, 264.98, 283.37, 303.17,
01307 324.49, 347.47, 372.25, 398.98, 427.85, 459.06, 492.8, 529.31,
01308 568.89, 611.79, 658.35, 708.91, 763.87, 823.65, 888.72, 959.58,
01309 1036.8, 1121.8, 1213.9, 1314.3, 1423.8, 1543., 1672.8, 1813.4,
01310 1966.1, 2131.4, 2309.5, 2499.3, 2705., 2925.7, 3161.6, 3411.3,
01311 3611.5, 3889.2, 4191.1, 4519.3, 4877.9, 5272.9, 5712.9, 6142.7,
01312 6719.6, 7385., 8145., 8977.7, 9831.9, 10827., 11934., 13063.,
01313 14434., 15878., 17591., 19435., 21510., 23835., 26835., 29740.,
01314 32878., 36305., 39830., 43273., 46931., 50499., 49586., 51598.,
01315 53429., 54619., 55081., 55102., 54485., 53487., 52042., 42689.,
01316 42607., 44020., 47994., 54169., 53916., 55808., 56642., 46049.,
01317 44243., 32929., 30658., 21963., 20835., 15962., 13679., 17652.,
01318 19680., 22388., 25625., 29184., 32520., 35720., 38414., 40523.,
01319 49228., 48173., 45678., 41768., 37600., 41313., 42654., 44465.,
01320 55736., 56630., 65409., 63308., 66572., 61845., 60379., 56777.,
01321 51920., 46601., 41367., 36529., 32219., 28470., 25192., 22362.,
01322 19907., 17772., 15907., 14273., 12835., 11567., 10445., 9450.2,
01323 8565.1, 7776., 7070.8, 6439.2, 5872.3, 5362.4, 4903., 4488.3,
01324 4113.4, 3773.8, 3465.8, 3186.1, 2931.7, 2700.1, 2488.8, 2296.,
01325 2119.8, 1958.6, 1810.9, 1675.6, 1551.4, 1437.3, 1332.4, 1236.,
01326 1147.2, 1065.3, 989.86, 920.22, 855.91, 796.48, 741.53, 690.69,
01327 643.62, 600.02, 559.6, 522.13, 487.35, 455.06, 425.08, 397.21,
01328 371.3, 347.2, 324.78, 303.9, 284.46, 266.34, 249.45, 233.7,
01329 219.01, 205.3, 192.5, 180.55, 169.38, 158.95, 149.2, 140.07,
01330 131.54, 123.56, 116.09, 109.09, 102.54, 96.405, 90.655, 85.266,
01331 80.213, 75.475, 71.031, 66.861, 62.948, 59.275, 55.827, 52.587,
01332 49.544, 46.686, 43.998, 41.473, 39.099, 36.867, 34.768, 32.795,
01333 30.939, 29.192, 27.546, 25.998, 24.539, 23.164, 21.869, 20.65,
01334 19.501, 18.419, 17.399, 16.438, 15.532, 14.678, 13.874, 13.115,
01335 12.4, 11.726, 11.088, 10.488, 9.921, 9.3846, 8.8784, 8.3996,
01336 7.9469, 7.5197, 7.1174, 6.738, 6.379, 6.0409, 5.7213, 5.419,
01337 5.1327, 4.8611, 4.6046, 4.3617, 4.1316, 3.9138, 3.7077, 3.5125,
01338 3.3281, 3.1536, 2.9885, 2.8323, 2.6846, 2.5447, 2.4124, 2.2871,
01339 2.1686, 2.0564, 1.9501, 1.8495, 1.7543, 1.6641, 1.5787, 1.4978,
01340 1.4212, 1.3486, 1.2799, 1.2147, 1.1529, 1.0943, 1.0388, .98602,
01341 .93596, .8886, .84352, .80078, .76029, .722, .68585, .65161,
01342 .61901, .58808, .55854, .53044, .5039, .47853, .45459, .43173,
01343 .41008, .38965, .37021, .35186, .33444, .31797, .30234, .28758,
01344 .2736, .26036, .24764, .2357, .22431, .21342, .20295, .19288,
01345 .18334, .17444, .166, .15815, .15072, .14348, .13674, .13015,
01346 .12399, .11807, .11231, .10689, .10164, .096696, .091955,
01347 .087476, .083183, .079113, .075229, .071536, .068026, .064698,
01348 .06154, .058544, .055699, .052997, .050431, .047993, .045676,
01349 .043475, .041382, .039392, .037501, .035702, .033991, .032364,
01350 .030817, .029345, .027945, .026613, .025345, .024139, .022991,
01351 .021899, .02086, .019871, .018929, .018033, .01718, .016368,
01352 .015595, .014859, .014158, .013491, .012856, .012251, .011675,
01353 .011126, .010604, .010107, .0096331, .009182, .0087523, .0083431,
01354 .0079533, .0075821, .0072284, .0068915, .0065706, .0062649,
01355 .0059737, .0056963, .005432, .0051802, .0049404, .0047118,
01356 .0044941, .0042867, .0040891, .0039009, .0037216, .0035507,
01357 .003388, .0032329, .0030852, .0029445, .0028105, .0026829,
01358 .0025613, .0024455, .0023353, .0022303, .0021304, .0020353,
01359 .0019448, .0018587, .0017767, .0016988, .0016247, .0015543,
01360 .0014874, .0014238, .0013635, .0013062, .0012519, .0012005,
01361 .0011517, .0011057, .0010621, .001021, 9.8233e-4, 9.4589e-4,
01362 9.1167e-4, 8.7961e-4, 8.4964e-4, 8.2173e-4, 7.9582e-4, 7.7189e-4,
01363 7.499e-4, 7.2983e-4, 7.1167e-4, 6.9542e-4, 6.8108e-4, 6.6866e-4,
01364 6.5819e-4, 6.4971e-4, 6.4328e-4, 6.3895e-4, 6.3681e-4, 6.3697e-4,
```



```

01365 6.3956e-4, 6.4472e-4, 6.5266e-4, 6.6359e-4, 6.778e-4, 6.9563e-4,
01366 7.1749e-4, 7.4392e-4, 7.7556e-4, 8.1028e-4, 8.4994e-4, 8.8709e-4,
01367 9.3413e-4, 9.6953e-4, .0010202, .0010738, .0010976, .0011507,
01368 .0011686, .0012264, .001291, .0013346, .0014246, .0015293,
01369 .0016359, .0017824, .0019255, .0020854, .002247, .0024148,
01370 .0026199, .0027523, .0029704, .0030702, .0033047, .0035013,
01371 .0037576, .0040275, .0043089, .0046927, .0049307, .0053486,
01372 .0053809, .0056699, .0059325, .0055488, .005634, .0056392,
01373 .004946, .0048855, .0048208, .0044386, .0045498, .0046377,
01374 .0048939, .0052396, .0057324, .0060859, .0066906, .0071148,
01375 .0077224, .0082687, .008769, .0084471, .008572, .0087729,
01376 .008775, .0090742, .0080704, .0080288, .0085747, .0086087,
01377 .0086408, .0088752, .0089381, .0089757, .0093532, .0092824,
01378 .0092566, .0092645, .0092735, .009342, .0095806, .0097991,
01379 .010213, .010611, .011129, .011756, .013237, .01412, .015034,
01380 .015936, .01682, .018597, .019315, .019995, .020658, .021289,
01381 .022363, .022996, .023716, .024512, .025434, .026067, .027118,
01382 .028396, .029865, .031442, .033253, .03525, .037296, .039701,
01383 .042356, .045154, .048059, .051294, .054893, .058636, .061407,
01384 .065172, .068974, .072676, .073379, .076547, .079556, .079134,
01385 .082308, .085739, .090192, .09359, .099599, .10669, .11496,
01386 .1244, .13512, .14752, .14494, .15647, .1668, .17863, .19029,
01387 .20124, .20254, .21179, .21982, .21625, .22364, .23405, .23382,
01388 .2434, .25708, .26406, .27621, .28909, .30395, .31717, .33271,
01389 .3496, .36765, .38774, .40949, .446, .46985, .49846, .5287, .562,
01390 .59841, .64598, .68834, .7327, .78978, .8373, .88708, .94744,
01391 1.0006, 1.0574, 1.1215, 1.1856, 1.2546, 1.3292, 1.4107, 1.4974,
01392 1.5913, 1.6931, 1.8028, 1.9212, 2.0492, 2.1874, 2.3365, 2.4978,
01393 2.6718, 2.8588, 3.062, 3.2818, 3.5188, 3.7752, 4.0527, 4.3542,
01394 4.6782, 5.0312, 5.4123, 5.8246, 6.2639, 6.7435, 7.2636, 7.8064,
01395 8.4091, 9.0696, 9.7677, 10.548, 11.4, 12.309, 13.324, 14.284,
01396 15.445, 16.687, 18.019, 19.403, 20.847, 22.366, 23.925, 25.537,
01397 27.213, 28.069, 29.864, 31.829, 33.988, 35.856, 38.829, 42.321,
01398 46.319, 50.606, 55.126, 59.126, 64.162, 68.708, 74.615, 81.176,
01399 87.739, 95.494, 103.83, 113.38, 123.99, 135.8, 148.7, 162.58,
01400 176.32, 192.6, 211.47, 232.7, 252.64, 277.41, 305.38, 333.44,
01401 366.42, 402.66, 442.14, 484.53, 526.42, 568.15, 558.78, 582.6,
01402 600.98, 613.94, 619.44, 618.24, 609.84, 595.96, 484.86, 475.59,
01403 478.49, 501.56, 552.19, 628.44, 630.39, 658.92, 671.96, 562.7,
01404 545.88, 423.43, 400.14, 306.59, 294.13, 246.8, 226.51, 278.21,
01405 314.39, 347.22, 389.13, 433.16, 477.48, 521.67, 560.54, 683.6,
01406 696.37, 695.91, 683.1, 658.24, 634.89, 698.85, 742.87, 796.66,
01407 954.49, 1009.5, 1150.5, 1179.1, 1267.9, 1272.4, 1312.7, 1330.4,
01408 1331.6, 1315.8, 1308.3, 1293.3, 1274.6, 1249.5, 1213.2, 1172.1,
01409 1124.4, 930.33, 893.36, 871.27, 883.54, 940.76, 1036, 1025.6,
01410 1053.1, 914.51, 894.15, 865.03, 670.63, 508.41, 475.15, 370.85,
01411 361.06, 319.38, 312.75, 331.87, 367.13, 415, 467.94, 525.49,
01412 578.41, 624.66, 794.82, 796.97, 780.29, 736.49, 670.18, 603.75,
01413 659.67, 679.8, 857.12, 884.05, 900.65, 1046.1, 1141.9, 1083,
01414 1089.2, 1e3, 947.08, 872.31, 787.91, 704.75, 624.93, 553.68,
01415 489.91, 434.21, 385.64, 343.3, 306.42, 274.18, 245.94, 221.11,
01416 199.23, 179.88, 162.73, 147.48, 133.88, 121.73, 110.86, 101.1,
01417 92.323, 84.417, 77.281, 70.831, 64.991, 59.694, 54.884, 50.509,
01418 46.526, 42.893, 39.58, 36.549, 33.776, 31.236, 28.907, 26.77,
01419 24.805, 23, 21.339, 19.81, 18.404, 17.105, 15.909, 14.801,
01420 13.778, 12.83, 11.954, 11.142, 10.389, 9.691, 9.0434, 8.4423,
01421 7.8842, 7.3657, 6.8838, 6.4357, 6.0189, 5.6308, 5.2696, 4.9332,
01422 4.6198, 4.3277, 4.0553, 3.8012, 3.5639, 3.3424, 3.1355, 2.9422,
01423 2.7614, 2.5924, 2.4343, 2.2864, 2.148, 2.0184, 1.8971, 1.7835,
01424 1.677, 1.5773, 1.4838, 1.3961, 1.3139, 1.2369, 1.1645, 1.0966,
01425 1.0329, .97309, .91686, .86406, .81439, .76767, .72381, .68252,
01426 .64359, .60695, .57247, .54008, .50957, .48092, .45401, .42862,
01427 .40465, .38202, .36072, .34052, .3216, .30386, .28711, .27135,
01428 .25651, .24252, .2293, .21689, .20517, .19416, .18381, .17396,
01429 .16469
01430 };
01431
01432 static double co2230[2001] = { 2.743e-5, 2.8815e-5, 3.027e-5, 3.1798e-5,
01433 3.3405e-5, 3.5094e-5, 3.6869e-5, 3.8734e-5, 4.0694e-5, 4.2754e-5,
01434 4.492e-5, 4.7196e-5, 4.9588e-5, 5.2103e-5, 5.4747e-5, 5.7525e-5,
01435 6.0446e-5, 6.3516e-5, 6.6744e-5, 7.0137e-5, 7.3704e-5, 7.7455e-5,
01436 8.1397e-5, 8.5543e-5, 8.9901e-5, 9.4484e-5, 9.9302e-5, 1.0437e-4,
01437 1.097e-4, 1.153e-4, 1.2119e-4, 1.2738e-4, 1.3389e-4, 1.4074e-4,
01438 1.4795e-4, 1.5552e-4, 1.6349e-4, 1.7187e-4, 1.8068e-4, 1.8995e-4,
01439 1.997e-4, 2.0996e-4, 2.2075e-4, 2.321e-4, 2.4403e-4, 2.5659e-4,
01440 2.698e-4, 2.837e-4, 2.9832e-4, 3.137e-4, 3.2988e-4, 3.4691e-4,
01441 3.6483e-4, 3.8368e-4, 4.0351e-4, 4.2439e-4, 4.4635e-4, 4.6947e-4,
01442 4.9379e-4, 5.1939e-4, 5.4633e-4, 5.7468e-4, 6.0452e-4, 6.3593e-4,
01443 6.69e-4, 7.038e-4, 7.4043e-4, 7.79e-4, 8.1959e-4, 8.6233e-4,
01444 9.0732e-4, 9.5469e-4, .0010046, .0010571, .0011124, .0011706,
01445 .0012319, .0012964, .0013644, .001436, .0015114, .0015908,
01446 .0016745, .0017625, .0018553, .0019531, .002056, .0021645,
01447 .0022788, .0023992, .002526, .0026596, .0028004, .0029488,
01448 .0031052, .0032699, .0034436, .0036265, .0038194, .0040227,
01449 .0042369, .0044628, .0047008, .0049518, .0052164, .0054953,
01450 .0057894, .0060995, .0064265, .0067713, .007135, .0075184,
01451 .0079228, .0083494, .0087993, .0092738, .0097745, .010303,

```

```

01452 .01086, .011448, .012068, .012722, .013413, .014142, .014911,
01453 .015723, .01658, .017484, .018439, .019447, .020511, .021635,
01454 .022821, .024074, .025397, .026794, .02827, .029829, .031475,
01455 .033215, .035052, .036994, .039045, .041213, .043504, .045926,
01456 .048485, .05119, .05405, .057074, .060271, .063651, .067225,
01457 .071006, .075004, .079233, .083708, .088441, .093449, .098749,
01458 .10436, .11029, .11657, .12322, .13026, .13772, .14561, .15397,
01459 .16282, .1722, .18214, .19266, .20381, .21563, .22816, .24143,
01460 .2555, .27043, .28625, .30303, .32082, .3397, .35972, .38097,
01461 .40352, .42746, .45286, .47983, .50847, .53888, .57119, .6055,
01462 .64196, .6807, .72187, .76564, .81217, .86165, .91427, .97025,
01463 1.0298, 1.0932, 1.1606, 1.2324, 1.3088, 1.3902, 1.477, 1.5693,
01464 1.6678, 1.7727, 1.8845, 2.0038, 2.131, 2.2666, 2.4114, 2.5659,
01465 2.7309, 2.907, 3.0951, 3.2961, 3.5109, 3.7405, 3.986, 4.2485,
01466 4.5293, 4.8299, 5.1516, 5.4961, 5.8651, 6.2605, 6.6842, 7.1385,
01467 7.6256, 8.1481, 8.7089, 9.3109, 9.9573, 10.652, 11.398, 12.2,
01468 13.063, 13.992, 14.99, 16.064, 17.222, 18.469, 19.813, 21.263,
01469 22.828, 24.516, 26.34, 28.31, 30.437, 32.738, 35.226, 37.914,
01470 40.824, 43.974, 47.377, 51.061, 55.011, 59.299, 63.961, 69.013,
01471 74.492, 80.444, 86.919, 93.836, 101.23, 109.25, 117.98, 127.47,
01472 137.81, 149.07, 161.35, 174.75, 189.42, 205.49, 223.02, 242.26,
01473 263.45, 286.75, 311.94, 340.01, 370.86, 404.92, 440.44, 480.27,
01474 525.17, 574.71, 626.22, 686.8, 754.38, 827.07, 913.38, 1011.7,
01475 1121.5, 1161.6, 1289.5, 1432.2, 1595.4, 1777., 1983.3, 2216.1,
01476 2485.7, 2788.3, 3101.5, 3481., 3902.1, 4257.1, 4740., 5272.8,
01477 5457.9, 5946.2, 6505.3, 6668.4, 7302.4, 8061.6, 9015.8, 9908.3,
01478 11613., 13956., 3249.6, 3243., 2901.5, 2841.3, 2729.6, 2558.2,
01479 1797.8, 1583.2, 1386., 1233.5, 787.74, 701.46, 761.66, 767.21,
01480 722.83, 1180.6, 1332.1, 1461.6, 2032.9, 2166., 2255.9, 2294.7,
01481 2587.2, 2396.5, 2122.4, 12553., 10784., 9832.5, 8827.3, 8029.1,
01482 7377.9, 7347.1, 6783.8, 6239.1, 5721.1, 5503., 4975.1, 4477.8,
01483 4021.3, 3676.8, 3275.3, 2914.9, 2597.4, 2328.2, 2075.4, 1857.6,
01484 1663.6, 1493.3, 1343.8, 1213.3, 1095.6, 1066.5, 958.91, 865.15,
01485 783.31, 714.35, 650.77, 593.98, 546.2, 499.9, 457.87, 421.75,
01486 387.61, 355.25, 326.62, 299.7, 275.21, 253.17, 232.83, 214.31,
01487 197.5, 182.08, 167.98, 155.12, 143.32, 132.5, 122.58, 113.48,
01488 105.11, 97.415, 90.182, 83.463, 77.281, 71.587, 66.341, 61.493,
01489 57.014, 53.062, 49.21, 45.663, 42.38, 39.348, 36.547, 33.967,
01490 31.573, 29.357, 27.314, 25.415, 23.658, 22.03, 20.524, 19.125,
01491 17.829, 16.627, 15.511, 14.476, 13.514, 12.618, 11.786, 11.013,
01492 10.294, 9.6246, 9.0018, 8.4218, 7.8816, 7.3783, 6.9092, 6.4719,
01493 6.0641, 5.6838, 5.3289, 4.998, 4.6893, 4.4014, 4.1325, 3.8813,
01494 3.6469, 3.4283, 3.2241, 3.035, 2.8576, 2.6922, 2.5348, 2.3896,
01495 2.2535, 2.1258, 2.0059, 1.8929, 1.7862, 1.6854, 1.5898, 1.4992,
01496 1.4017, 1.3218, 1.2479, 1.1809, 1.1215, 1.0693, 1.0116, .96016,
01497 .9105, .84859, .80105, .74381, .69982, .65127, .60899, .57843,
01498 .54592, .51792, .49336, .47155, .45201, .43426, .41807, .40303,
01499 .38876, .3863, .37098, .35492, .33801, .32032, .30341, .29874,
01500 .29193, .28689, .29584, .29155, .29826, .29195, .29287, .2904,
01501 .28199, .27709, .27162, .26622, .26133, .25676, .25235, .23137,
01502 .22365, .21519, .20597, .19636, .18699, .16485, .16262, .16643,
01503 .17542, .18198, .18631, .16759, .16338, .13505, .1267, .10053,
01504 .092554, .074093, .062159, .055523, .054849, .05401, .05528,
01505 .058982, .07952, .08647, .093244, .099285, .10393, .10661,
01506 .12072, .11417, .10396, .093265, .089137, .088909, .10902,
01507 .11277, .13625, .13565, .14907, .14167, .1428, .13744, .12768,
01508 .11382, .10244, .091686, .08109, .071739, .063616, .056579,
01509 .050504, .045251, .040689, .036715, .033237, .030181, .027488,
01510 .025107, .022998, .021125, .01946, .017979, .016661, .015489,
01511 .014448, .013526, .012712, .011998, .011375, .010839, .010384,
01512 .010007, .0097053, .0094783, .0093257, .0092489, .0092504,
01513 .0093346, .0095077, .0097676, .01012, .01058, .011157, .011844,
01514 .012672, .013665, .014766, .015999, .017509, .018972, .020444,
01515 .022311, .023742, .0249, .025599, .026981, .026462, .025143,
01516 .025066, .022814, .020458, .020026, .019142, .020189, .022371,
01517 .024163, .023728, .02199, .019506, .018591, .015576, .012784,
01518 .011744, .0094777, .0079148, .0070652, .006986, .0071758,
01519 .008086, .0098025, .01087, .013609, .016764, .018137, .021061,
01520 .023498, .023576, .023965, .022828, .021519, .021283, .023364,
01521 .026457, .029782, .030856, .033486, .035515, .035543, .036558,
01522 .037198, .037472, .037045, .037284, .03777, .038085, .038366,
01523 .038526, .038282, .038915, .037697, .035667, .032941, .031959,
01524 .028692, .025918, .024596, .025592, .027873, .028935, .02984,
01525 .028148, .025305, .021912, .020454, .016732, .013357, .01205,
01526 .009731, .0079881, .0077704, .0074387, .0083895, .0096776,
01527 .010326, .01293, .015955, .019247, .020145, .02267, .024231,
01528 .024184, .022131, .019784, .01955, .01971, .022119, .025116,
01529 .027978, .028107, .029808, .030701, .029164, .028551, .027286,
01530 .024946, .023259, .020982, .019221, .017471, .015643, .014074,
01531 .01261, .011301, .010116, .0090582, .0081036, .0072542, .0065034,
01532 .0058436, .0052571, .0047321, .0042697, .0038607, .0034977,
01533 .0031747, .0028864, .0026284, .002397, .002189, .0020017,
01534 .0018326, .0016798, .0015414, .0014159, .0013019, .0011983,
01535 .0011039, .0010177, 9.391e-4, 8.6717e-4, 8.0131e-4, 7.4093e-4,
01536 6.8553e-4, 6.3464e-4, 5.8787e-4, 5.4487e-4, 5.0533e-4, 4.69e-4,
01537 4.3556e-4, 4.0474e-4, 3.7629e-4, 3.5e-4, 3.2569e-4, 3.032e-4,
01538 2.8239e-4, 2.6314e-4, 2.4535e-4, 2.2891e-4, 2.1374e-4, 1.9975e-4,

```

01539 1.8685e-4, 1.7498e-4, 1.6406e-4, 1.5401e-4, 1.4479e-4, 1.3633e-4,  
 01540 1.2858e-4, 1.2148e-4, 1.1499e-4, 1.0907e-4, 1.0369e-4, 9.8791e-5,  
 01541 9.4359e-5, 9.0359e-5, 8.6766e-5, 8.3555e-5, 8.0703e-5, 7.8192e-5,  
 01542 7.6003e-5, 7.4119e-5, 7.2528e-5, 7.1216e-5, 7.0171e-5, 6.9385e-5,  
 01543 6.8848e-5, 6.8554e-5, 6.8496e-5, 6.8669e-5, 6.9069e-5, 6.9694e-5,  
 01544 7.054e-5, 7.1608e-5, 7.2896e-5, 7.4406e-5, 7.6139e-5, 7.8097e-5,  
 01545 8.0283e-5, 8.2702e-5, 8.5357e-5, 8.8255e-5, 9.1402e-5, 9.4806e-5,  
 01546 9.8473e-5, 1.0241e-4, 1.0664e-4, 1.1115e-4, 1.1598e-4, 1.2112e-4,  
 01547 1.2659e-4, 1.3241e-4, 1.3859e-4, 1.4515e-4, 1.521e-4, 1.5947e-4,  
 01548 1.6728e-4, 1.7555e-4, 1.8429e-4, 1.9355e-4, 2.0334e-4, 2.1369e-4,  
 01549 2.2463e-4, 2.3619e-4, 2.4841e-4, 2.6132e-4, 2.7497e-4, 2.8938e-4,  
 01550 3.0462e-4, 3.2071e-4, 3.3771e-4, 3.5567e-4, 3.7465e-4, 3.947e-4,  
 01551 4.1588e-4, 4.3828e-4, 4.6194e-4, 4.8695e-4, 5.1338e-4, 5.4133e-4,  
 01552 5.7087e-4, 6.0211e-4, 6.3515e-4, 6.701e-4, 7.0706e-4, 7.4617e-4,  
 01553 7.8756e-4, 8.3136e-4, 8.7772e-4, 9.2681e-4, 9.788e-4, .0010339,  
 01554 .0010922, .001154, .0012195, .0012889, .0013626, .0014407,  
 01555 .0015235, .0016114, .0017048, .0018038, .001909, .0020207,  
 01556 .0021395, .0022657, .0023998, .0025426, .0026944, .002856,  
 01557 .0030281, .0032114, .0034068, .003615, .0038371, .004074,  
 01558 .004327, .0045971, .0048857, .0051942, .0055239, .0058766,  
 01559 .0062538, .0066573, .0070891, .007551, .0080455, .0085747,  
 01560 .0091412, .0097481, .010397, .011092, .011837, .012638, .013495,  
 01561 .014415, .01541, .016475, .017621, .018857, .020175, .02162,  
 01562 .023185, .024876, .02672, .028732, .030916, .033319, .035939,  
 01563 .038736, .041847, .04524, .048715, .052678, .056977, .061203,  
 01564 .066184, .07164, .076952, .083477, .090674, .098049, .10697,  
 01565 .1169, .1277, .14011, .15323, .1684, .18601, .20626, .22831,  
 01566 .25417, .28407, .31405, .34957, .38823, .41923, .46026, .50409,  
 01567 .51227, .54805, .57976, .53818, .55056, .557, .46741, .46403,  
 01568 .4636, .42265, .45166, .49852, .56663, .34306, .17779, .17697,  
 01569 .18346, .19129, .20014, .21778, .23604, .25649, .28676, .31238,  
 01570 .33856, .39998, .4288, .46568, .56654, .60786, .64473, .76466,  
 01571 .7897, .80778, .86443, .85736, .84798, .84157, 1.1385, 1.2446,  
 01572 1.1923, 1.1552, 1.1338, 1.1266, 1.1292, 1.1431, 1.1683, 1.2059,  
 01573 1.2521, 1.3069, 1.3712, 1.4471, 1.5275, 1.6165, 1.7145, 1.8189,  
 01574 1.9359, 2.065, 2.2007, 2.3591, 2.5362, 2.7346, 2.9515, 3.2021,  
 01575 3.4851, 3.7935, 4.0694, 4.4463, 4.807, 5.2443, 5.7178, 6.2231,  
 01576 6.4796, 6.9461, 7.4099, 7.3652, 7.7182, 8.048, 7.7373, 8.0363,  
 01577 8.3855, 8.8044, 9.0257, 9.8574, 10.948, 10.563, 6.8979, 7.0744,  
 01578 7.4121, 7.7663, 8.1768, 8.6243, 9.1437, 9.7847, 10.182, 10.849,  
 01579 11.572, 12.602, 13.482, 14.431, 15.907, 16.983, 18.11, 19.884,  
 01580 21.02, 22.18, 23.355, 24.848, 25.954, 27.13, 30.186, 34.893,  
 01581 35.682, 36.755, 38.111, 39.703, 41.58, 43.606, 45.868, 48.573,  
 01582 51.298, 54.291, 57.559, 61.116, 64.964, 69.124, 73.628, 78.471,  
 01583 83.683, 89.307, 95.341, 101.84, 108.83, 116.36, 124.46, 133.18,  
 01584 142.57, 152.79, 163.69, 175.43, 188.11, 201.79, 216.55, 232.51,  
 01585 249.74, 268.38, 288.54, 310.35, 333.97, 359.55, 387.26, 417.3,  
 01586 449.88, 485.2, 523.54, 565.14, 610.28, 659.31, 712.56, 770.43,  
 01587 833.36, 901.82, 976.36, 1057.6, 1146.8, 1243.8, 1350., 1466.3,  
 01588 1593.6, 1732.7, 1884.1, 2049.1, 2228.2, 2421.9, 2629.4, 2853.7,  
 01589 3094.4, 3351.1, 3622.3, 3829.8, 4123.1, 4438.3, 4777.2, 5144.1,  
 01590 5545.4, 5990.5, 6404.5, 6996.8, 7687.6, 8482.9, 9349.4, 10203.,  
 01591 11223., 12358., 13493., 14916., 16416., 18236., 20222., 22501.,  
 01592 25102., 28358., 31707., 35404., 39538., 43911., 48391., 53193.,  
 01593 58028., 58082., 61276., 64193., 66294., 67480., 67921., 67423.,  
 01594 66254., 64341., 51737., 51420., 53072., 58145., 66195., 65358.,  
 01595 67377., 67869., 53509., 50553., 35737., 32425., 21704., 19974.,  
 01596 14457., 12142., 16798., 19489., 23049., 27270., 31910., 36457.,  
 01597 40877., 44748., 47876., 59793., 58626., 55454., 50337., 44893.,  
 01598 50228., 52216., 54747., 69541., 70455., 81014., 77694., 80533.,  
 01599 73953., 70927., 65539., 59002., 52281., 45953., 40292., 35360.,  
 01600 31124., 27478., 24346., 21647., 19308., 17271., 15491., 13927.,  
 01601 12550., 11331., 10250., 9288.8, 8431.4, 7664.9, 6978.3, 6361.8,  
 01602 5807.4, 5307.7, 4856.8, 4449., 4079.8, 3744.9, 3440.8, 3164.2,  
 01603 2912.3, 2682.7, 2473., 2281.4, 2106., 1945.3, 1797.9, 1662.5,  
 01604 1538.1, 1423.6, 1318.1, 1221., 1131.5, 1049., 972.99, 902.87,  
 01605 838.01, 777.95, 722.2, 670.44, 622.35, 577.68, 536.21, 497.76,  
 01606 462.12, 429.13, 398.61, 370.39, 344.29, 320.16, 297.85, 277.2,  
 01607 258.08, 240.38, 223.97, 208.77, 194.66, 181.58, 169.43, 158.15,  
 01608 147.67, 137.92, 128.86, 120.44, 112.6, 105.3, 98.499, 92.166,  
 01609 86.264, 80.763, 75.632, 70.846, 66.381, 62.213, 58.321, 54.685,  
 01610 51.288, 48.114, 45.145, 42.368, 39.772, 37.341, 35.065, 32.937,  
 01611 30.943, 29.077, 27.33, 25.693, 24.158, 22.717, 21.367, 20.099,  
 01612 18.909, 17.792, 16.744, 15.761, 14.838, 13.971, 13.157, 12.393,  
 01613 11.676, 11.003, 10.369, 9.775, 9.2165, 8.6902, 8.1963, 7.7314,  
 01614 7.2923, 6.8794, 6.4898, 6.122, 5.7764, 5.4525, 5.1484, 4.8611,  
 01615 4.5918, 4.3379, 4.0982, 3.8716, 3.6567, 3.4545, 3.2634, 3.0828,  
 01616 2.9122, 2.7512, 2.5993, 2.4561, 2.3211, 2.1938, 2.0737, 1.9603,  
 01617 1.8534, 1.7525, 1.6572, 1.5673, 1.4824, 1.4022, 1.3265, 1.2551,  
 01618 1.1876, 1.1239, 1.0637, 1.0069, .9532, .90248, .85454, .80921,  
 01619 .76631, .72569, .6872, .65072, .61635, .5836, .55261, .52336,  
 01620 .49581, .46998, .44559, .42236, .40036, .37929, .35924, .34043,  
 01621 .32238, .30547, .28931, .27405, .25975, .24616, .23341, .22133,  
 01622 .20997, .19924, .18917, .17967, .17075, .16211, .15411, .14646,  
 01623 .13912, .13201, .12509, .11857, .11261, .10698, .10186, .097039,  
 01624 .092236, .087844, .083443, .07938, .075452, .071564, .067931,  
 01625 .064389, .061078, .057901, .054921, .052061, .049364, .046789,

```
01626 .04435, .042044, .039866, .037808, .035863, .034023, .032282,
01627 .030634, .029073, .027595, .026194, .024866, .023608, .022415,
01628 .021283, .02021, .019193, .018228, .017312, .016443, .015619,
01629 .014837, .014094, .01339, .012721, .012086, .011483, .010911,
01630 .010368, .009852, .0093623, .0088972, .0084556, .0080362,
01631 .0076379, .0072596, .0069003, .006559, .0062349, .0059269,
01632 .0056344, .0053565, .0050925, .0048417, .0046034, .004377,
01633 .0041618, .0039575, .0037633, .0035788, .0034034, .0032368,
01634 .0030785, .002928, .0027851, .0026492, .0025201, .0023975,
01635 .0022809, .0021701, .0020649, .0019649, .0018699, .0017796,
01636 .0016938, .0016122, .0015348, .0014612, .0013913, .001325,
01637 .0012619, .0012021, .0011452, .0010913, .0010401, 9.9149e-4,
01638 9.454e-4, 9.0169e-4, 8.6024e-4, 8.2097e-4, 7.8377e-4, 7.4854e-4,
01639 7.1522e-4, 6.8371e-4, 6.5393e-4, 6.2582e-4, 5.9932e-4, 5.7435e-4,
01640 5.5087e-4, 5.2882e-4, 5.0814e-4, 4.8881e-4, 4.7076e-4, 4.5398e-4,
01641 4.3843e-4, 4.2407e-4, 4.109e-4, 3.9888e-4, 3.88e-4, 3.7826e-4,
01642 3.6963e-4, 3.6213e-4, 3.5575e-4, 3.505e-4, 3.464e-4, 3.4346e-4,
01643 3.4173e-4, 3.4125e-4, 3.4206e-4, 3.4424e-4, 3.4787e-4, 3.5303e-4,
01644 3.5986e-4, 3.6847e-4, 3.7903e-4, 3.9174e-4, 4.0681e-4, 4.2455e-4,
01645 4.4527e-4, 4.6942e-4, 4.9637e-4, 5.2698e-4, 5.5808e-4, 5.9514e-4,
01646 6.2757e-4, 6.689e-4, 7.1298e-4, 7.3955e-4, 7.8403e-4, 8.0449e-4,
01647 8.5131e-4, 9.0256e-4, 9.3692e-4, .0010051, .0010846, .0011678,
01648 .001282, .0014016, .0015355, .0016764, .0018272, .0020055,
01649 .0021455, .0023421, .0024615, .0026786, .0028787, .0031259,
01650 .0034046, .0036985, .0040917, .0043902, .0048349, .0049531,
01651 .0052989, .0056148, .0052452, .0053357, .005333, .0045069,
01652 .0043851, .004253, .003738, .0038084, .0039013, .0041505,
01653 .0045372, .0050569, .0054507, .0061267, .0066122, .0072449,
01654 .0078012, .0082651, .0076538, .0076573, .0076806, .0075227,
01655 .0076269, .0063758, .006254, .0067749, .0067909, .0068231,
01656 .0072143, .0072762, .0072954, .007679, .0075107, .0073658,
01657 .0072441, .0071074, .0070378, .007176, .0072472, .0075844,
01658 .0079291, .008412, .0090165, .010688, .011535, .012375, .013166,
01659 .013895, .015567, .016011, .016392, .016737, .017043, .017731,
01660 .018031, .018419, .018877, .019474, .019868, .020604, .021538,
01661 .022653, .023869, .025288, .026879, .028547, .030524, .03274,
01662 .035132, .03769, .040567, .043793, .047188, .049962, .053542,
01663 .057205, .060776, .061489, .064419, .067124, .065945, .068487,
01664 .071209, .074783, .077039, .082444, .08902, .09692, .10617,
01665 .11687, .12952, .12362, .13498, .14412, .15492, .16519, .1744,
01666 .17096, .17714, .18208, .17363, .17813, .18564, .18295, .19045,
01667 .20252, .20815, .21844, .22929, .24229, .25321, .26588, .2797,
01668 .29465, .31136, .32961, .36529, .38486, .41027, .43694, .4667,
01669 .49943, .54542, .58348, .62303, .67633, .71755, .76054, .81371,
01670 .85934, .90841, .96438, 1.0207, 1.0821, 1.1491, 1.2226, 1.3018,
01671 1.388, 1.4818, 1.5835, 1.6939, 1.8137, 1.9435, 2.0843, 2.237,
01672 2.4026, 2.5818, 2.7767, 2.9885, 3.2182, 3.4679, 3.7391, 4.0349,
01673 4.3554, 4.7053, 5.0849, 5.4986, 5.9436, 6.4294, 6.9598, 7.5203,
01674 8.143, 8.8253, 9.5568, 10.371, 11.267, 12.233, 13.31, 14.357,
01675 15.598, 16.93, 18.358, 19.849, 21.408, 23.04, 24.706, 26.409,
01676 28.153, 28.795, 30.549, 32.43, 34.49, 36.027, 38.955, 42.465,
01677 46.565, 50.875, 55.378, 59.002, 63.882, 67.949, 73.693, 80.095,
01678 86.403, 94.264, 102.65, 112.37, 123.3, 135.54, 149.14, 163.83,
01679 179.17, 196.89, 217.91, 240.94, 264.13, 292.39, 324.83, 358.21,
01680 397.16, 440.5, 488.6, 541.04, 595.3, 650.43, 652.03, 688.74,
01681 719.47, 743.54, 757.68, 762.35, 756.43, 741.42, 595.43, 580.97,
01682 580.83, 605.68, 667.88, 764.49, 759.93, 789.12, 798.17, 645.66,
01683 615.65, 455.05, 421.09, 306.45, 289.14, 235.7, 215.52, 274.57,
01684 316.53, 357.73, 409.89, 465.06, 521.84, 579.02, 630.64, 794.46,
01685 813., 813.56, 796.25, 761.57, 727.97, 812.14, 866.75, 932.5,
01686 1132.8, 1194.8, 1362.2, 1387.2, 1482.3, 1479.7, 1517.9, 1533.1,
01687 1534.2, 1523.3, 1522.5, 1515.5, 1505.2, 1486.5, 1454., 1412.,
01688 1358.8, 1107.8, 1060.9, 1033.5, 1048.2, 1122.4, 1248.9, 1227.1,
01689 1255.4, 1058.9, 1020.7, 970.59, 715.24, 512.56, 468.47, 349.3,
01690 338.26, 299.22, 301.26, 332.38, 382.08, 445.49, 515.87, 590.85,
01691 662.3, 726.05, 955.59, 964.11, 945.17, 891.48, 807.11, 720.9,
01692 803.36, 834.46, 1073.9, 1107.1, 1123.6, 1296., 1393.7, 1303.1,
01693 1284.3, 1161.8, 1078.8, 976.13, 868.72, 767.4, 674.72, 593.73,
01694 523.12, 462.24, 409.75, 364.34, 325., 290.73, 260.76, 234.46,
01695 211.28, 190.78, 172.61, 156.44, 142.01, 129.12, 117.57, 107.2,
01696 97.877, 89.47, 81.882, 75.021, 68.807, 63.171, 58.052, 53.396,
01697 49.155, 45.288, 41.759, 38.531, 35.576, 32.868, 30.384, 28.102,
01698 26.003, 24.071, 22.293, 20.655, 19.147, 17.756, 16.476, 15.292,
01699 14.198, 13.183, 12.241, 11.367, 10.554, 9.7989, 9.0978, 8.4475,
01700 7.845, 7.2868, 6.7704, 6.2927, 5.8508, 5.4421, 5.064, 4.714,
01701 4.3902, 4.0902, 3.8121, 3.5543, 3.315, 3.093, 2.8869, 2.6953,
01702 2.5172, 2.3517, 2.1977, 2.0544, 1.9211, 1.7969, 1.6812, 1.5735,
01703 1.4731, 1.3794, 1.2921, 1.2107, 1.1346, 1.0637, .99744, .93554,
01704 .87771, .82368, .77313, .72587, .6816, .64014, .60134, .565,
01705 .53086, .49883, .46881, .44074, .4144, .38979, .36679, .34513,
01706 .32474, .30552, .28751, .27045, .25458, .23976, .22584, .21278,
01707 .20051, .18899, .17815, .16801, .15846, .14954, .14117, .13328,
01708 .12584
01709 };
01710
01711 double xw, dw, ew, cw296, cw260, cw230, dt230, dt260, dt296, ctw, ctmph;
01712
```

```

01713     int iw;
01714
01715     /* Get CO2 continuum absorption... */
01716     xw = nu / 2 + 1;
01717     if (xw >= 1 && xw < 2001) {
01718         iw = (int) xw;
01719         dw = xw - iw;
01720         ew = 1 - dw;
01721         cw296 = ew * co2296[iw - 1] + dw * co2296[iw];
01722         cw260 = ew * co2260[iw - 1] + dw * co2260[iw];
01723         cw230 = ew * co2230[iw - 1] + dw * co2230[iw];
01724         dt230 = t - 230;
01725         dt260 = t - 260;
01726         dt296 = t - 296;
01727         ctw = dt260 * 5.050505e-4 * dt296 * cw230 - dt230 * 9.259259e-4
01728             * dt296 * cw260 + dt230 * 4.208754e-4 * dt260 * cw296;
01729         ctmph = u / NA / 1000 * p / P0 * ctw;
01730     } else
01731         ctmph = 0;
01732     return ctmph;
01733 }

```

### 5.23.2.7 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](#).

```

01742     {
01743
01744     static double h2o296[2001] = { .17, .1695, .172, .168, .1687, .1624, .1606,
01745     .1508, .1447, .1344, .1214, .1133, .1009, .09217, .08297, .06989,
01746     .06513, .05469, .05056, .04417, .03779, .03484, .02994, .0272,
01747     .02325, .02063, .01818, .01592, .01405, .01251, .0108, .009647,
01748     .008424, .007519, .006555, .00588, .005136, .004511, .003989,
01749     .003509, .003114, .00274, .002446, .002144, .001895, .001676,
01750     .001486, .001312, .001164, .001031, 9.129e-4, 8.106e-4, 7.213e-4,
01751     6.4e-4, 5.687e-4, 5.063e-4, 4.511e-4, 4.029e-4, 3.596e-4,
01752     3.22e-4, 2.889e-4, 2.597e-4, 2.337e-4, 2.108e-4, 1.907e-4,
01753     1.728e-4, 1.57e-4, 1.43e-4, 1.305e-4, 1.195e-4, 1.097e-4,
01754     1.009e-4, 9.307e-5, 8.604e-5, 7.971e-5, 7.407e-5, 6.896e-5,
01755     6.433e-5, 6.013e-5, 5.631e-5, 5.283e-5, 4.963e-5, 4.669e-5,
01756     4.398e-5, 4.148e-5, 3.917e-5, 3.702e-5, 3.502e-5, 3.316e-5,
01757     3.142e-5, 2.978e-5, 2.825e-5, 2.681e-5, 2.546e-5, 2.419e-5,
01758     2.299e-5, 2.186e-5, 2.079e-5, 1.979e-5, 1.884e-5, 1.795e-5,
01759     1.711e-5, 1.633e-5, 1.559e-5, 1.49e-5, 1.426e-5, 1.367e-5,
01760     1.312e-5, 1.263e-5, 1.218e-5, 1.178e-5, 1.143e-5, 1.112e-5,
01761     1.088e-5, 1.07e-5, 1.057e-5, 1.05e-5, 1.051e-5, 1.059e-5,
01762     1.076e-5, 1.1e-5, 1.133e-5, 1.18e-5, 1.237e-5, 1.308e-5,
01763     1.393e-5, 1.483e-5, 1.614e-5, 1.758e-5, 1.93e-5, 2.123e-5,
01764     2.346e-5, 2.647e-5, 2.93e-5, 3.279e-5, 3.745e-5, 4.152e-5,
01765     4.813e-5, 5.477e-5, 6.203e-5, 7.331e-5, 8.056e-5, 9.882e-5,
01766     1.05e-4, 1.21e-4, 1.341e-4, 1.572e-4, 1.698e-4, 1.968e-4,
01767     2.175e-4, 2.431e-4, 2.735e-4, 2.867e-4, 3.19e-4, 3.371e-4,
01768     3.554e-4, 3.726e-4, 3.837e-4, 3.878e-4, 3.864e-4, 3.858e-4,
01769     3.841e-4, 3.852e-4, 3.815e-4, 3.762e-4, 3.618e-4, 3.579e-4,
01770     3.45e-4, 3.202e-4, 3.018e-4, 2.785e-4, 2.602e-4, 2.416e-4,
01771     2.097e-4, 1.939e-4, 1.689e-4, 1.498e-4, 1.308e-4, 1.17e-4,
01772     1.011e-4, 9.237e-5, 7.909e-5, 7.006e-5, 6.112e-5, 5.401e-5,
01773     4.914e-5, 4.266e-5, 3.963e-5, 3.316e-5, 3.037e-5, 2.598e-5,
01774     2.294e-5, 2.066e-5, 1.813e-5, 1.583e-5, 1.423e-5, 1.247e-5,
01775     1.116e-5, 9.76e-6, 8.596e-6, 7.72e-6, 6.825e-6, 6.108e-6,
01776     5.366e-6, 4.733e-6, 4.229e-6, 3.731e-6, 3.346e-6, 2.972e-6,
01777     2.628e-6, 2.356e-6, 2.102e-6, 1.878e-6, 1.678e-6, 1.507e-6,
01778     1.348e-6, 1.21e-6, 1.089e-6, 9.806e-7, 8.857e-7, 8.004e-7,
01779     7.261e-7, 6.599e-7, 6.005e-7, 5.479e-7, 5.011e-7, 4.595e-7,
01780     4.219e-7, 3.885e-7, 3.583e-7, 3.314e-7, 3.071e-7, 2.852e-7,
01781     2.654e-7, 2.474e-7, 2.311e-7, 2.162e-7, 2.026e-7, 1.902e-7,
01782     1.788e-7, 1.683e-7, 1.587e-7, 1.497e-7, 1.415e-7, 1.338e-7,
01783     1.266e-7, 1.2e-7, 1.138e-7, 1.08e-7, 1.027e-7, 9.764e-8,
01784     9.296e-8, 8.862e-8, 8.458e-8, 8.087e-8, 7.744e-8, 7.429e-8,
01785     7.145e-8, 6.893e-8, 6.664e-8, 6.468e-8, 6.322e-8, 6.162e-8,
01786     6.07e-8, 5.992e-8, 5.913e-8, 5.841e-8, 5.796e-8, 5.757e-8,
01787     5.746e-8, 5.731e-8, 5.679e-8, 5.577e-8, 5.671e-8, 5.656e-8,
01788     5.594e-8, 5.593e-8, 5.602e-8, 5.62e-8, 5.693e-8, 5.725e-8,
01789     5.858e-8, 6.037e-8, 6.249e-8, 6.535e-8, 6.899e-8, 7.356e-8,
01790     7.918e-8, 8.618e-8, 9.385e-8, 1.039e-7, 1.158e-7, 1.29e-7,
01791     1.437e-7, 1.65e-7, 1.871e-7, 2.121e-7, 2.427e-7, 2.773e-7,
01792     3.247e-7, 3.677e-7, 4.037e-7, 4.776e-7, 5.101e-7, 6.214e-7,
01793     6.936e-7, 7.581e-7, 8.486e-7, 9.355e-7, 9.942e-7, 1.063e-6,

```

```
01794 1.123e-6, 1.191e-6, 1.215e-6, 1.247e-6, 1.26e-6, 1.271e-6,
01795 1.284e-6, 1.317e-6, 1.323e-6, 1.349e-6, 1.353e-6, 1.362e-6,
01796 1.344e-6, 1.329e-6, 1.336e-6, 1.327e-6, 1.325e-6, 1.359e-6,
01797 1.374e-6, 1.415e-6, 1.462e-6, 1.526e-6, 1.619e-6, 1.735e-6,
01798 1.863e-6, 2.034e-6, 2.265e-6, 2.482e-6, 2.756e-6, 3.103e-6,
01799 3.466e-6, 3.832e-6, 4.378e-6, 4.913e-6, 5.651e-6, 6.311e-6,
01800 7.169e-6, 8.057e-6, 9.253e-6, 1.047e-5, 1.212e-5, 1.36e-5,
01801 1.569e-5, 1.776e-5, 2.02e-5, 2.281e-5, 2.683e-5, 2.994e-5,
01802 3.488e-5, 3.896e-5, 4.499e-5, 5.175e-5, 6.035e-5, 6.34e-5,
01803 7.281e-5, 7.923e-5, 8.348e-5, 9.631e-5, 1.044e-4, 1.102e-4,
01804 1.176e-4, 1.244e-4, 1.283e-4, 1.326e-4, 1.4e-4, 1.395e-4,
01805 1.387e-4, 1.363e-4, 1.314e-4, 1.241e-4, 1.228e-4, 1.148e-4,
01806 1.086e-4, 1.018e-4, 8.89e-5, 8.316e-5, 7.292e-5, 6.452e-5,
01807 5.625e-5, 5.045e-5, 4.38e-5, 3.762e-5, 3.29e-5, 2.836e-5,
01808 2.485e-5, 2.168e-5, 1.895e-5, 1.659e-5, 1.453e-5, 1.282e-5,
01809 1.132e-5, 1.001e-5, 8.836e-6, 7.804e-6, 6.922e-6, 6.116e-6,
01810 5.429e-6, 4.824e-6, 4.278e-6, 3.788e-6, 3.371e-6, 2.985e-6,
01811 2.649e-6, 2.357e-6, 2.09e-6, 1.858e-6, 1.647e-6, 1.462e-6,
01812 1.299e-6, 1.155e-6, 1.028e-6, 9.142e-7, 8.132e-7, 7.246e-7,
01813 6.451e-7, 5.764e-7, 5.151e-7, 4.603e-7, 4.121e-7, 3.694e-7,
01814 3.318e-7, 2.985e-7, 2.69e-7, 2.428e-7, 2.197e-7, 1.992e-7,
01815 1.81e-7, 1.649e-7, 1.506e-7, 1.378e-7, 1.265e-7, 1.163e-7,
01816 1.073e-7, 9.918e-8, 9.191e-8, 8.538e-8, 7.949e-8, 7.419e-8,
01817 6.94e-8, 6.508e-8, 6.114e-8, 5.761e-8, 5.437e-8, 5.146e-8,
01818 4.89e-8, 4.636e-8, 4.406e-8, 4.201e-8, 4.015e-8, 3.84e-8,
01819 3.661e-8, 3.51e-8, 3.377e-8, 3.242e-8, 3.13e-8, 3.015e-8,
01820 2.918e-8, 2.83e-8, 2.758e-8, 2.707e-8, 2.656e-8, 2.619e-8,
01821 2.609e-8, 2.615e-8, 2.63e-8, 2.675e-8, 2.745e-8, 2.842e-8,
01822 2.966e-8, 3.125e-8, 3.318e-8, 3.565e-8, 3.85e-8, 4.191e-8,
01823 4.59e-8, 5.059e-8, 5.607e-8, 6.239e-8, 6.958e-8, 7.796e-8,
01824 8.773e-8, 9.88e-8, 1.114e-7, 1.258e-7, 1.422e-7, 1.61e-7,
01825 1.822e-7, 2.06e-7, 2.337e-7, 2.645e-7, 2.996e-7, 3.393e-7,
01826 3.843e-7, 4.363e-7, 4.935e-7, 5.607e-7, 6.363e-7, 7.242e-7,
01827 8.23e-7, 9.411e-7, 1.071e-6, 1.232e-6, 1.402e-6, 1.6e-6, 1.82e-6,
01828 2.128e-6, 2.386e-6, 2.781e-6, 3.242e-6, 3.653e-6, 4.323e-6,
01829 4.747e-6, 5.321e-6, 5.919e-6, 6.681e-6, 7.101e-6, 7.983e-6,
01830 8.342e-6, 8.741e-6, 9.431e-6, 9.952e-6, 1.026e-5, 1.055e-5,
01831 1.095e-5, 1.095e-5, 1.087e-5, 1.056e-5, 1.026e-5, 9.715e-6,
01832 9.252e-6, 8.452e-6, 7.958e-6, 7.268e-6, 6.295e-6, 6.003e-6, 5e-6,
01833 4.591e-6, 3.983e-6, 3.479e-6, 3.058e-6, 2.667e-6, 2.293e-6,
01834 1.995e-6, 1.747e-6, 1.517e-6, 1.335e-6, 1.165e-6, 1.028e-6,
01835 9.007e-7, 7.956e-7, 7.015e-7, 6.192e-7, 5.491e-7, 4.859e-7,
01836 4.297e-7, 3.799e-7, 3.38e-7, 3.002e-7, 2.659e-7, 2.366e-7,
01837 2.103e-7, 1.861e-7, 1.655e-7, 1.469e-7, 1.309e-7, 1.162e-7,
01838 1.032e-7, 9.198e-8, 8.181e-8, 7.294e-8, 6.516e-8, 5.787e-8,
01839 5.163e-8, 4.612e-8, 4.119e-8, 3.695e-8, 3.308e-8, 2.976e-8,
01840 2.67e-8, 2.407e-8, 2.171e-8, 1.965e-8, 1.78e-8, 1.617e-8,
01841 1.47e-8, 1.341e-8, 1.227e-8, 1.125e-8, 1.033e-8, 9.524e-9,
01842 8.797e-9, 8.162e-9, 7.565e-9, 7.04e-9, 6.56e-9, 6.129e-9,
01843 5.733e-9, 5.376e-9, 5.043e-9, 4.75e-9, 4.466e-9, 4.211e-9,
01844 3.977e-9, 3.759e-9, 3.558e-9, 3.373e-9, 3.201e-9, 3.043e-9,
01845 2.895e-9, 2.76e-9, 2.635e-9, 2.518e-9, 2.411e-9, 2.314e-9,
01846 2.23e-9, 2.151e-9, 2.087e-9, 2.035e-9, 1.988e-9, 1.946e-9,
01847 1.927e-9, 1.916e-9, 1.916e-9, 1.933e-9, 1.966e-9, 2.018e-9,
01848 2.09e-9, 2.182e-9, 2.299e-9, 2.442e-9, 2.623e-9, 2.832e-9,
01849 3.079e-9, 3.368e-9, 3.714e-9, 4.104e-9, 4.567e-9, 5.091e-9,
01850 5.701e-9, 6.398e-9, 7.194e-9, 8.127e-9, 9.141e-9, 1.035e-8,
01851 1.177e-8, 1.338e-8, 1.508e-8, 1.711e-8, 1.955e-8, 2.216e-8,
01852 2.534e-8, 2.871e-8, 3.291e-8, 3.711e-8, 4.285e-8, 4.868e-8,
01853 5.509e-8, 6.276e-8, 7.262e-8, 8.252e-8, 9.4e-8, 1.064e-7,
01854 1.247e-7, 1.411e-7, 1.626e-7, 1.827e-7, 2.044e-7, 2.284e-7,
01855 2.452e-7, 2.854e-7, 3.026e-7, 3.278e-7, 3.474e-7, 3.693e-7,
01856 3.93e-7, 4.104e-7, 4.22e-7, 4.439e-7, 4.545e-7, 4.778e-7,
01857 4.812e-7, 5.018e-7, 4.899e-7, 5.075e-7, 5.073e-7, 5.171e-7,
01858 5.131e-7, 5.25e-7, 5.617e-7, 5.846e-7, 6.239e-7, 6.696e-7,
01859 7.398e-7, 8.073e-7, 9.15e-7, 1.009e-6, 1.116e-6, 1.264e-6,
01860 1.439e-6, 1.644e-6, 1.856e-6, 2.147e-6, 2.317e-6, 2.713e-6,
01861 2.882e-6, 2.99e-6, 3.489e-6, 3.581e-6, 4.033e-6, 4.26e-6,
01862 4.543e-6, 4.84e-6, 4.826e-6, 5.013e-6, 5.252e-6, 5.277e-6,
01863 5.306e-6, 5.236e-6, 5.123e-6, 5.171e-6, 4.843e-6, 4.615e-6,
01864 4.385e-6, 3.97e-6, 3.693e-6, 3.231e-6, 2.915e-6, 2.495e-6,
01865 2.144e-6, 1.91e-6, 1.639e-6, 1.417e-6, 1.226e-6, 1.065e-6,
01866 9.29e-7, 8.142e-7, 7.161e-7, 6.318e-7, 5.581e-7, 4.943e-7,
01867 4.376e-7, 3.884e-7, 3.449e-7, 3.06e-7, 2.712e-7, 2.412e-7,
01868 2.139e-7, 1.903e-7, 1.689e-7, 1.499e-7, 1.331e-7, 1.183e-7,
01869 1.05e-7, 9.362e-8, 8.306e-8, 7.403e-8, 6.578e-8, 5.853e-8,
01870 5.216e-8, 4.632e-8, 4.127e-8, 3.678e-8, 3.279e-8, 2.923e-8,
01871 2.612e-8, 2.339e-8, 2.094e-8, 1.877e-8, 1.686e-8, 1.516e-8,
01872 1.366e-8, 1.234e-8, 1.114e-8, 1.012e-8, 9.182e-9, 8.362e-9,
01873 7.634e-9, 6.981e-9, 6.406e-9, 5.888e-9, 5.428e-9, 5.021e-9,
01874 4.65e-9, 4.326e-9, 4.033e-9, 3.77e-9, 3.536e-9, 3.327e-9,
01875 3.141e-9, 2.974e-9, 2.825e-9, 2.697e-9, 2.584e-9, 2.488e-9,
01876 2.406e-9, 2.34e-9, 2.292e-9, 2.259e-9, 2.244e-9, 2.243e-9,
01877 2.272e-9, 2.31e-9, 2.378e-9, 2.454e-9, 2.618e-9, 2.672e-9,
01878 2.831e-9, 3.05e-9, 3.225e-9, 3.425e-9, 3.677e-9, 3.968e-9,
01879 4.221e-9, 4.639e-9, 4.96e-9, 5.359e-9, 5.649e-9, 6.23e-9,
01880 6.716e-9, 7.218e-9, 7.746e-9, 7.988e-9, 8.627e-9, 8.999e-9,
```

01881 9.442e-9, 9.82e-9, 1.015e-8, 1.06e-8, 1.079e-8, 1.109e-8,  
01882 1.137e-8, 1.186e-8, 1.18e-8, 1.187e-8, 1.194e-8, 1.192e-8,  
01883 1.224e-8, 1.245e-8, 1.246e-8, 1.318e-8, 1.377e-8, 1.471e-8,  
01884 1.582e-8, 1.713e-8, 1.853e-8, 2.063e-8, 2.27e-8, 2.567e-8,  
01885 2.891e-8, 3.264e-8, 3.744e-8, 4.286e-8, 4.915e-8, 5.623e-8,  
01886 6.336e-8, 7.293e-8, 8.309e-8, 9.319e-8, 1.091e-7, 1.243e-7,  
01887 1.348e-7, 1.449e-7, 1.62e-7, 1.846e-7, 1.937e-7, 2.04e-7,  
01888 2.179e-7, 2.298e-7, 2.433e-7, 2.439e-7, 2.464e-7, 2.611e-7,  
01889 2.617e-7, 2.582e-7, 2.453e-7, 2.401e-7, 2.349e-7, 2.203e-7,  
01890 2.066e-7, 1.939e-7, 1.78e-7, 1.558e-7, 1.391e-7, 1.203e-7,  
01891 1.048e-7, 9.464e-8, 8.306e-8, 7.239e-8, 6.317e-8, 5.52e-8,  
01892 4.847e-8, 4.282e-8, 3.796e-8, 3.377e-8, 2.996e-8, 2.678e-8,  
01893 2.4e-8, 2.134e-8, 1.904e-8, 1.705e-8, 1.523e-8, 1.35e-8,  
01894 1.204e-8, 1.07e-8, 9.408e-9, 8.476e-9, 7.47e-9, 6.679e-9,  
01895 5.929e-9, 5.267e-9, 4.711e-9, 4.172e-9, 3.761e-9, 3.288e-9,  
01896 2.929e-9, 2.609e-9, 2.315e-9, 2.042e-9, 1.844e-9, 1.64e-9,  
01897 1.47e-9, 1.31e-9, 1.176e-9, 1.049e-9, 9.377e-10, 8.462e-10,  
01898 7.616e-10, 6.854e-10, 6.191e-10, 5.596e-10, 5.078e-10, 4.611e-10,  
01899 4.197e-10, 3.83e-10, 3.505e-10, 3.215e-10, 2.956e-10, 2.726e-10,  
01900 2.521e-10, 2.338e-10, 2.173e-10, 2.026e-10, 1.895e-10, 1.777e-10,  
01901 1.672e-10, 1.579e-10, 1.496e-10, 1.423e-10, 1.358e-10, 1.302e-10,  
01902 1.254e-10, 1.216e-10, 1.187e-10, 1.163e-10, 1.147e-10, 1.145e-10,  
01903 1.15e-10, 1.17e-10, 1.192e-10, 1.25e-10, 1.298e-10, 1.345e-10,  
01904 1.405e-10, 1.538e-10, 1.648e-10, 1.721e-10, 1.872e-10, 1.968e-10,  
01905 2.089e-10, 2.172e-10, 2.317e-10, 2.389e-10, 2.503e-10, 2.585e-10,  
01906 2.686e-10, 2.8e-10, 2.895e-10, 3.019e-10, 3.037e-10, 3.076e-10,  
01907 3.146e-10, 3.198e-10, 3.332e-10, 3.397e-10, 3.54e-10, 3.667e-10,  
01908 3.895e-10, 4.071e-10, 4.565e-10, 4.983e-10, 5.439e-10, 5.968e-10,  
01909 6.676e-10, 7.456e-10, 8.405e-10, 9.478e-10, 1.064e-9, 1.218e-9,  
01910 1.386e-9, 1.581e-9, 1.787e-9, 2.032e-9, 2.347e-9, 2.677e-9,  
01911 3.008e-9, 3.544e-9, 4.056e-9, 4.687e-9, 5.331e-9, 6.227e-9,  
01912 6.854e-9, 8.139e-9, 8.945e-9, 9.865e-9, 1.125e-8, 1.178e-8,  
01913 1.364e-8, 1.436e-8, 1.54e-8, 1.672e-8, 1.793e-8, 1.906e-8,  
01914 2.036e-8, 2.144e-8, 2.292e-8, 2.371e-8, 2.493e-8, 2.606e-8,  
01915 2.706e-8, 2.866e-8, 3.036e-8, 3.136e-8, 3.405e-8, 3.665e-8,  
01916 3.837e-8, 4.229e-8, 4.748e-8, 5.32e-8, 5.763e-8, 6.677e-8,  
01917 7.216e-8, 7.716e-8, 8.958e-8, 9.419e-8, 1.036e-7, 1.108e-7,  
01918 1.189e-7, 1.246e-7, 1.348e-7, 1.31e-7, 1.361e-7, 1.364e-7,  
01919 1.363e-7, 1.343e-7, 1.293e-7, 1.254e-7, 1.235e-7, 1.158e-7,  
01920 1.107e-7, 9.961e-8, 9.011e-8, 7.91e-8, 6.916e-8, 6.338e-8,  
01921 5.564e-8, 4.827e-8, 4.198e-8, 3.695e-8, 3.276e-8, 2.929e-8,  
01922 2.633e-8, 2.391e-8, 2.192e-8, 2.021e-8, 1.89e-8, 1.772e-8,  
01923 1.667e-8, 1.603e-8, 1.547e-8, 1.537e-8, 1.492e-8, 1.515e-8,  
01924 1.479e-8, 1.45e-8, 1.513e-8, 1.495e-8, 1.529e-8, 1.565e-8,  
01925 1.564e-8, 1.553e-8, 1.569e-8, 1.584e-8, 1.57e-8, 1.538e-8,  
01926 1.513e-8, 1.472e-8, 1.425e-8, 1.349e-8, 1.328e-8, 1.249e-8,  
01927 1.17e-8, 1.077e-8, 9.514e-9, 8.614e-9, 7.46e-9, 6.621e-9,  
01928 5.775e-9, 5.006e-9, 4.308e-9, 3.747e-9, 3.24e-9, 2.84e-9,  
01929 2.481e-9, 2.184e-9, 1.923e-9, 1.71e-9, 1.504e-9, 1.334e-9,  
01930 1.187e-9, 1.053e-9, 9.367e-10, 8.306e-10, 7.419e-10, 6.63e-10,  
01931 5.918e-10, 5.277e-10, 4.717e-10, 4.222e-10, 3.783e-10, 3.39e-10,  
01932 3.036e-10, 2.729e-10, 2.455e-10, 2.211e-10, 1.995e-10, 1.804e-10,  
01933 1.635e-10, 1.485e-10, 1.355e-10, 1.24e-10, 1.139e-10, 1.051e-10,  
01934 9.757e-11, 9.114e-11, 8.577e-11, 8.139e-11, 7.792e-11, 7.52e-11,  
01935 7.39e-11, 7.311e-11, 7.277e-11, 7.482e-11, 7.698e-11, 8.162e-11,  
01936 8.517e-11, 8.968e-11, 9.905e-11, 1.075e-10, 1.187e-10, 1.291e-10,  
01937 1.426e-10, 1.573e-10, 1.734e-10, 1.905e-10, 2.097e-10, 2.28e-10,  
01938 2.473e-10, 2.718e-10, 2.922e-10, 3.128e-10, 3.361e-10, 3.641e-10,  
01939 3.91e-10, 4.196e-10, 4.501e-10, 4.932e-10, 5.258e-10, 5.755e-10,  
01940 6.253e-10, 6.664e-10, 7.344e-10, 7.985e-10, 8.877e-10, 1.005e-9,  
01941 1.118e-9, 1.251e-9, 1.428e-9, 1.61e-9, 1.888e-9, 2.077e-9,  
01942 2.331e-9, 2.751e-9, 3.061e-9, 3.522e-9, 3.805e-9, 4.181e-9,  
01943 4.575e-9, 5.167e-9, 5.634e-9, 6.007e-9, 6.501e-9, 6.829e-9,  
01944 7.211e-9, 7.262e-9, 7.696e-9, 7.832e-9, 7.799e-9, 7.651e-9,  
01945 7.304e-9, 7.15e-9, 6.977e-9, 6.603e-9, 6.209e-9, 5.69e-9,  
01946 5.432e-9, 4.764e-9, 4.189e-9, 3.64e-9, 3.203e-9, 2.848e-9,  
01947 2.51e-9, 2.194e-9, 1.946e-9, 1.75e-9, 1.567e-9, 1.426e-9,  
01948 1.302e-9, 1.197e-9, 1.109e-9, 1.035e-9, 9.719e-10, 9.207e-10,  
01949 8.957e-10, 8.578e-10, 8.262e-10, 8.117e-10, 7.987e-10, 7.875e-10,  
01950 7.741e-10, 7.762e-10, 7.537e-10, 7.424e-10, 7.474e-10, 7.294e-10,  
01951 7.216e-10, 7.233e-10, 7.075e-10, 6.892e-10, 6.618e-10, 6.314e-10,  
01952 6.208e-10, 5.689e-10, 5.55e-10, 4.984e-10, 4.6e-10, 4.078e-10,  
01953 3.879e-10, 3.459e-10, 2.982e-10, 2.626e-10, 2.329e-10, 1.988e-10,  
01954 1.735e-10, 1.487e-10, 1.297e-10, 1.133e-10, 9.943e-11, 8.736e-11,  
01955 7.726e-11, 6.836e-11, 6.053e-11, 5.384e-11, 4.789e-11, 4.267e-11,  
01956 3.804e-11, 3.398e-11, 3.034e-11, 2.71e-11, 2.425e-11, 2.173e-11,  
01957 1.95e-11, 1.752e-11, 1.574e-11, 1.418e-11, 1.278e-11, 1.154e-11,  
01958 1.044e-11, 9.463e-12, 8.602e-12, 7.841e-12, 7.171e-12, 6.584e-12,  
01959 6.073e-12, 5.631e-12, 5.254e-12, 4.937e-12, 4.679e-12, 4.476e-12,  
01960 4.328e-12, 4.233e-12, 4.194e-12, 4.211e-12, 4.286e-12, 4.424e-12,  
01961 4.628e-12, 4.906e-12, 5.262e-12, 5.708e-12, 6.254e-12, 6.914e-12,  
01962 7.714e-12, 8.677e-12, 9.747e-12, 1.101e-11, 1.256e-11, 1.409e-11,  
01963 1.597e-11, 1.807e-11, 2.034e-11, 2.316e-11, 2.622e-11, 2.962e-11,  
01964 3.369e-11, 3.819e-11, 4.329e-11, 4.932e-11, 5.589e-11, 6.364e-11,  
01965 7.284e-11, 8.236e-11, 9.447e-11, 1.078e-10, 1.229e-10, 1.417e-10,  
01966 1.614e-10, 1.843e-10, 2.107e-10, 2.406e-10, 2.728e-10, 3.195e-10,  
01967 3.595e-10, 4.153e-10, 4.736e-10, 5.41e-10, 6.088e-10, 6.769e-10,

```
01968 7.691e-10, 8.545e-10, 9.621e-10, 1.047e-9, 1.161e-9, 1.296e-9,
01969 1.424e-9, 1.576e-9, 1.739e-9, 1.893e-9, 2.08e-9, 2.336e-9,
01970 2.604e-9, 2.76e-9, 3.001e-9, 3.365e-9, 3.55e-9, 3.895e-9,
01971 4.183e-9, 4.614e-9, 4.846e-9, 5.068e-9, 5.427e-9, 5.541e-9,
01972 5.864e-9, 5.997e-9, 5.997e-9, 6.061e-9, 5.944e-9, 5.855e-9,
01973 5.661e-9, 5.523e-9, 5.374e-9, 4.94e-9, 4.688e-9, 4.17e-9,
01974 3.913e-9, 3.423e-9, 2.997e-9, 2.598e-9, 2.253e-9, 1.946e-9,
01975 1.71e-9, 1.507e-9, 1.336e-9, 1.19e-9, 1.068e-9, 9.623e-10,
01976 8.772e-10, 8.007e-10, 7.42e-10, 6.884e-10, 6.483e-10, 6.162e-10,
01977 5.922e-10, 5.688e-10, 5.654e-10, 5.637e-10, 5.701e-10, 5.781e-10,
01978 5.874e-10, 6.268e-10, 6.357e-10, 6.525e-10, 7.137e-10, 7.441e-10,
01979 8.024e-10, 8.485e-10, 9.143e-10, 9.536e-10, 9.717e-10, 1.018e-9,
01980 1.042e-9, 1.054e-9, 1.092e-9, 1.079e-9, 1.064e-9, 1.043e-9,
01981 1.02e-9, 9.687e-10, 9.273e-10, 9.208e-10, 9.068e-10, 7.687e-10,
01982 7.385e-10, 6.595e-10, 5.87e-10, 5.144e-10, 4.417e-10, 3.804e-10,
01983 3.301e-10, 2.866e-10, 2.509e-10, 2.202e-10, 1.947e-10, 1.719e-10,
01984 1.525e-10, 1.361e-10, 1.21e-10, 1.084e-10, 9.8e-11, 8.801e-11,
01985 7.954e-11, 7.124e-11, 6.335e-11, 5.76e-11, 5.132e-11, 4.601e-11,
01986 4.096e-11, 3.657e-11, 3.25e-11, 2.909e-11, 2.587e-11, 2.297e-11,
01987 2.05e-11, 1.828e-11, 1.632e-11, 1.462e-11, 1.314e-11, 1.185e-11,
01988 1.073e-11, 9.76e-12, 8.922e-12, 8.206e-12, 7.602e-12, 7.1e-12,
01989 6.694e-12, 6.378e-12, 6.149e-12, 6.004e-12, 5.941e-12, 5.962e-12,
01990 6.069e-12, 6.265e-12, 6.551e-12, 6.935e-12, 7.457e-12, 8.074e-12,
01991 8.811e-12, 9.852e-12, 1.086e-11, 1.207e-11, 1.361e-11, 1.553e-11,
01992 1.737e-11, 1.93e-11, 2.175e-11, 2.41e-11, 2.706e-11, 3.023e-11,
01993 3.313e-11, 3.657e-11, 4.118e-11, 4.569e-11, 5.025e-11, 5.66e-11,
01994 6.231e-11, 6.881e-11, 7.996e-11, 8.526e-11, 9.694e-11, 1.106e-10,
01995 1.222e-10, 1.355e-10, 1.525e-10, 1.775e-10, 1.924e-10, 2.181e-10,
01996 2.379e-10, 2.662e-10, 2.907e-10, 3.154e-10, 3.366e-10, 3.579e-10,
01997 3.858e-10, 4.046e-10, 4.196e-10, 4.166e-10, 4.457e-10, 4.466e-10,
01998 4.404e-10, 4.337e-10, 4.15e-10, 4.083e-10, 3.91e-10, 3.723e-10,
01999 3.514e-10, 3.303e-10, 2.847e-10, 2.546e-10, 2.23e-10, 1.994e-10,
02000 1.733e-10, 1.488e-10, 1.297e-10, 1.144e-10, 1.004e-10, 8.741e-11,
02001 7.928e-11, 7.034e-11, 6.323e-11, 5.754e-11, 5.25e-11, 4.85e-11,
02002 4.502e-11, 4.286e-11, 4.028e-11, 3.899e-11, 3.824e-11, 3.761e-11,
02003 3.804e-11, 3.839e-11, 3.845e-11, 4.244e-11, 4.382e-11, 4.582e-11,
02004 4.847e-11, 5.209e-11, 5.384e-11, 5.887e-11, 6.371e-11, 6.737e-11,
02005 7.168e-11, 7.415e-11, 7.827e-11, 8.037e-11, 8.12e-11, 8.071e-11,
02006 8.008e-11, 7.851e-11, 7.544e-11, 7.377e-11, 7.173e-11, 6.801e-11,
02007 6.267e-11, 5.727e-11, 5.288e-11, 4.853e-11, 4.082e-11, 3.645e-11,
02008 3.136e-11, 2.672e-11, 2.304e-11, 1.986e-11, 1.725e-11, 1.503e-11,
02009 1.315e-11, 1.153e-11, 1.014e-11, 8.942e-12, 7.901e-12, 6.993e-12,
02010 6.199e-12, 5.502e-12, 4.89e-12, 4.351e-12, 3.878e-12, 3.461e-12,
02011 3.094e-12, 2.771e-12, 2.488e-12, 2.241e-12, 2.025e-12, 1.838e-12,
02012 1.677e-12, 1.541e-12, 1.427e-12, 1.335e-12, 1.262e-12, 1.209e-12,
02013 1.176e-12, 1.161e-12, 1.165e-12, 1.189e-12, 1.234e-12, 1.3e-12,
02014 1.389e-12, 1.503e-12, 1.644e-12, 1.814e-12, 2.017e-12, 2.255e-12,
02015 2.534e-12, 2.858e-12, 3.231e-12, 3.661e-12, 4.153e-12, 4.717e-12,
02016 5.36e-12, 6.094e-12, 6.93e-12, 7.882e-12, 8.966e-12, 1.02e-11,
02017 1.162e-11, 1.324e-11, 1.51e-11, 1.72e-11, 1.965e-11, 2.237e-11,
02018 2.56e-11, 2.927e-11, 3.371e-11, 3.842e-11, 4.429e-11, 5.139e-11,
02019 5.798e-11, 6.697e-11, 7.626e-11, 8.647e-11, 1.022e-10, 1.136e-10,
02020 1.3e-10, 1.481e-10, 1.672e-10, 1.871e-10, 2.126e-10, 2.357e-10,
02021 2.583e-10, 2.997e-10, 3.289e-10, 3.702e-10, 4.012e-10, 4.319e-10,
02022 4.527e-10, 5.001e-10, 5.448e-10, 5.611e-10, 5.76e-10, 5.965e-10,
02023 6.079e-10, 6.207e-10, 6.276e-10, 6.222e-10, 6.137e-10, 6e-10,
02024 5.814e-10, 5.393e-10, 5.35e-10, 4.947e-10, 4.629e-10, 4.117e-10,
02025 3.712e-10, 3.372e-10, 2.923e-10, 2.55e-10, 2.232e-10, 1.929e-10,
02026 1.679e-10, 1.46e-10, 1.289e-10, 1.13e-10, 9.953e-11, 8.763e-11,
02027 7.76e-11, 6.9e-11, 6.16e-11, 5.525e-11, 4.958e-11, 4.489e-11,
02028 4.072e-11, 3.728e-11, 3.438e-11, 3.205e-11, 3.006e-11, 2.848e-11,
02029 2.766e-11, 2.688e-11, 2.664e-11, 2.67e-11, 2.696e-11, 2.786e-11,
02030 2.861e-11, 3.009e-11, 3.178e-11, 3.389e-11, 3.587e-11, 3.819e-11,
02031 4.054e-11, 4.417e-11, 4.703e-11, 5.137e-11, 5.46e-11, 6.055e-11,
02032 6.333e-11, 6.771e-11, 7.219e-11, 7.717e-11, 8.131e-11, 8.491e-11,
02033 8.574e-11, 9.01e-11, 9.017e-11, 8.999e-11, 8.959e-11, 8.838e-11,
02034 8.579e-11, 8.162e-11, 8.098e-11, 7.472e-11, 7.108e-11, 6.559e-11,
02035 5.994e-11, 5.172e-11, 4.424e-11, 3.951e-11, 3.34e-11, 2.902e-11,
02036 2.541e-11, 2.215e-11, 1.945e-11, 1.716e-11, 1.503e-11, 1.339e-11,
02037 1.185e-11, 1.05e-11, 9.336e-12, 8.307e-12, 7.312e-12, 6.55e-12,
02038 5.836e-12, 5.178e-12, 4.6e-12, 4.086e-12, 3.639e-12, 3.247e-12,
02039 2.904e-12, 2.604e-12, 2.341e-12, 2.112e-12, 1.914e-12, 1.744e-12,
02040 1.598e-12, 1.476e-12, 1.374e-12, 1.293e-12, 1.23e-12, 1.185e-12,
02041 1.158e-12, 1.147e-12, 1.154e-12, 1.177e-12, 1.219e-12, 1.28e-12,
02042 1.36e-12, 1.463e-12, 1.591e-12, 1.75e-12, 1.94e-12, 2.156e-12,
02043 2.43e-12, 2.748e-12, 3.052e-12, 3.533e-12, 3.967e-12, 4.471e-12,
02044 5.041e-12, 5.86e-12, 6.664e-12, 7.522e-12, 8.342e-12, 9.412e-12,
02045 1.072e-11, 1.213e-11, 1.343e-11, 1.496e-11, 1.664e-11, 1.822e-11,
02046 2.029e-11, 2.233e-11, 2.457e-11, 2.709e-11, 2.928e-11, 3.115e-11,
02047 3.356e-11, 3.592e-11, 3.818e-11, 3.936e-11, 4.061e-11, 4.149e-11,
02048 4.299e-11, 4.223e-11, 4.251e-11, 4.287e-11, 4.177e-11, 4.094e-11,
02049 3.942e-11, 3.772e-11, 3.614e-11, 3.394e-11, 3.222e-11, 2.791e-11,
02050 2.665e-11, 2.309e-11, 2.032e-11, 1.74e-11, 1.535e-11, 1.323e-11,
02051 1.151e-11, 9.803e-12, 8.65e-12, 7.54e-12, 6.619e-12, 5.832e-12,
02052 5.113e-12, 4.503e-12, 3.975e-12, 3.52e-12, 3.112e-12, 2.797e-12,
02053 2.5e-12, 2.24e-12, 2.013e-12, 1.819e-12, 1.653e-12, 1.513e-12,
02054 1.395e-12, 1.299e-12, 1.225e-12, 1.168e-12, 1.124e-12, 1.148e-12,
```



```

02055    1.107e-12, 1.128e-12, 1.169e-12, 1.233e-12, 1.307e-12, 1.359e-12,
02056    1.543e-12, 1.686e-12, 1.794e-12, 2.028e-12, 2.21e-12, 2.441e-12,
02057    2.653e-12, 2.828e-12, 3.093e-12, 3.28e-12, 3.551e-12, 3.677e-12,
02058    3.803e-12, 3.844e-12, 4.068e-12, 4.093e-12, 4.002e-12, 3.904e-12,
02059    3.624e-12, 3.633e-12, 3.622e-12, 3.443e-12, 3.184e-12, 2.934e-12,
02060    2.476e-12, 2.212e-12, 1.867e-12, 1.594e-12, 1.37e-12, 1.192e-12,
02061    1.045e-12, 9.211e-13, 8.17e-13, 7.29e-13, 6.55e-13, 5.929e-13,
02062    5.415e-13, 4.995e-13, 4.661e-13, 4.406e-13, 4.225e-13, 4.116e-13,
02063    4.075e-13, 4.102e-13, 4.198e-13, 4.365e-13, 4.606e-13, 4.925e-13,
02064    5.326e-13, 5.818e-13, 6.407e-13, 7.104e-13, 7.92e-13, 8.868e-13,
02065    9.964e-13, 1.123e-12, 1.268e-12, 1.434e-12, 1.626e-12, 1.848e-12,
02066    2.107e-12, 2.422e-12, 2.772e-12, 3.145e-12, 3.704e-12, 4.27e-12,
02067    4.721e-12, 5.361e-12, 6.083e-12, 7.095e-12, 7.968e-12, 9.228e-12,
02068    1.048e-11, 1.187e-11, 1.336e-11, 1.577e-11, 1.772e-11, 2.017e-11,
02069    2.25e-11, 2.63e-11, 2.911e-11, 3.356e-11, 3.82e-11, 4.173e-11,
02070    4.811e-11, 5.254e-11, 5.839e-11, 6.187e-11, 6.805e-11, 7.118e-11,
02071    7.369e-11, 7.664e-11, 7.794e-11, 7.947e-11, 8.036e-11, 7.954e-11,
02072    7.849e-11, 7.518e-11, 7.462e-11, 6.926e-11, 6.531e-11, 6.197e-11,
02073    5.421e-11, 4.777e-11, 4.111e-11, 3.679e-11, 3.166e-11, 2.786e-11,
02074    2.436e-11, 2.144e-11, 1.859e-11, 1.628e-11, 1.414e-11, 1.237e-11,
02075    1.093e-11, 9.558e-12
02076    };
02077
02078    static double h2o260[2001] = { .2752, .2732, .2749, .2676, .2667, .2545,
02079    .2497, .2327, .2218, .2036, .1825, .1694, .1497, .1353, .121,
02080    .1014, .09405, .07848, .07195, .06246, .05306, .04853, .04138,
02081    .03735, .03171, .02785, .02431, .02111, .01845, .0164, .01405,
02082    .01255, .01098, .009797, .008646, .007779, .006898, .006099,
02083    .005453, .004909, .004413, .003959, .003581, .003199, .002871,
02084    .002583, .00233, .002086, .001874, .001684, .001512, .001361,
02085    .001225, .0011, 9.89e-4, 8.916e-4, 8.039e-4, 7.256e-4, 6.545e-4,
02086    5.918e-4, 5.359e-4, 4.867e-4, 4.426e-4, 4.033e-4, 3.682e-4,
02087    3.366e-4, 3.085e-4, 2.833e-4, 2.605e-4, 2.403e-4, 2.221e-4,
02088    2.055e-4, 1.908e-4, 1.774e-4, 1.653e-4, 1.544e-4, 1.443e-4,
02089    1.351e-4, 1.267e-4, 1.19e-4, 1.119e-4, 1.053e-4, 9.922e-5,
02090    9.355e-5, 8.831e-5, 8.339e-5, 7.878e-5, 7.449e-5, 7.043e-5,
02091    6.664e-5, 6.307e-5, 5.969e-5, 5.654e-5, 5.357e-5, 5.075e-5,
02092    4.81e-5, 4.56e-5, 4.322e-5, 4.102e-5, 3.892e-5, 3.696e-5,
02093    3.511e-5, 3.339e-5, 3.177e-5, 3.026e-5, 2.886e-5, 2.756e-5,
02094    2.636e-5, 2.527e-5, 2.427e-5, 2.337e-5, 2.257e-5, 2.185e-5,
02095    2.127e-5, 2.08e-5, 2.041e-5, 2.013e-5, 2e-5, 1.997e-5, 2.009e-5,
02096    2.031e-5, 2.068e-5, 2.124e-5, 2.189e-5, 2.267e-5, 2.364e-5,
02097    2.463e-5, 2.618e-5, 2.774e-5, 2.937e-5, 3.144e-5, 3.359e-5,
02098    3.695e-5, 4.002e-5, 4.374e-5, 4.947e-5, 5.431e-5, 6.281e-5,
02099    7.169e-5, 8.157e-5, 9.728e-5, 1.079e-4, 1.337e-4, 1.442e-4,
02100    1.683e-4, 1.879e-4, 2.223e-4, 2.425e-4, 2.838e-4, 3.143e-4,
02101    3.527e-4, 4.012e-4, 4.237e-4, 4.747e-4, 5.057e-4, 5.409e-4,
02102    5.734e-4, 5.944e-4, 6.077e-4, 6.175e-4, 6.238e-4, 6.226e-4,
02103    6.248e-4, 6.192e-4, 6.098e-4, 5.818e-4, 5.709e-4, 5.465e-4,
02104    5.043e-4, 4.699e-4, 4.294e-4, 3.984e-4, 3.672e-4, 3.152e-4,
02105    2.883e-4, 2.503e-4, 2.211e-4, 1.92e-4, 1.714e-4, 1.485e-4,
02106    1.358e-4, 1.156e-4, 1.021e-4, 8.887e-5, 7.842e-5, 7.12e-5,
02107    6.186e-5, 5.73e-5, 4.792e-5, 4.364e-5, 3.72e-5, 3.28e-5,
02108    2.946e-5, 2.591e-5, 2.261e-5, 2.048e-5, 1.813e-5, 1.63e-5,
02109    1.447e-5, 1.282e-5, 1.167e-5, 1.041e-5, 9.449e-6, 8.51e-6,
02110    7.596e-6, 6.961e-6, 6.272e-6, 5.728e-6, 5.198e-6, 4.667e-6,
02111    4.288e-6, 3.897e-6, 3.551e-6, 3.235e-6, 2.952e-6, 2.688e-6,
02112    2.449e-6, 2.241e-6, 2.05e-6, 1.879e-6, 1.722e-6, 1.582e-6,
02113    1.456e-6, 1.339e-6, 1.236e-6, 1.144e-6, 1.06e-6, 9.83e-7,
02114    9.149e-7, 8.535e-7, 7.973e-7, 7.466e-7, 6.999e-7, 6.574e-7,
02115    6.18e-7, 5.821e-7, 5.487e-7, 5.18e-7, 4.896e-7, 4.631e-7,
02116    4.386e-7, 4.16e-7, 3.945e-7, 3.748e-7, 3.562e-7, 3.385e-7,
02117    3.222e-7, 3.068e-7, 2.922e-7, 2.788e-7, 2.659e-7, 2.539e-7,
02118    2.425e-7, 2.318e-7, 2.219e-7, 2.127e-7, 2.039e-7, 1.958e-7,
02119    1.885e-7, 1.818e-7, 1.758e-7, 1.711e-7, 1.662e-7, 1.63e-7,
02120    1.605e-7, 1.58e-7, 1.559e-7, 1.545e-7, 1.532e-7, 1.522e-7,
02121    1.51e-7, 1.495e-7, 1.465e-7, 1.483e-7, 1.469e-7, 1.448e-7,
02122    1.444e-7, 1.436e-7, 1.426e-7, 1.431e-7, 1.425e-7, 1.445e-7,
02123    1.477e-7, 1.515e-7, 1.567e-7, 1.634e-7, 1.712e-7, 1.802e-7,
02124    1.914e-7, 2.024e-7, 2.159e-7, 2.295e-7, 2.461e-7, 2.621e-7,
02125    2.868e-7, 3.102e-7, 3.394e-7, 3.784e-7, 4.223e-7, 4.864e-7,
02126    5.501e-7, 6.039e-7, 7.193e-7, 7.728e-7, 9.514e-7, 1.073e-6,
02127    1.18e-6, 1.333e-6, 1.472e-6, 1.566e-6, 1.677e-6, 1.784e-6,
02128    1.904e-6, 1.953e-6, 2.02e-6, 2.074e-6, 2.128e-6, 2.162e-6,
02129    2.219e-6, 2.221e-6, 2.249e-6, 2.239e-6, 2.235e-6, 2.185e-6,
02130    2.141e-6, 2.124e-6, 2.09e-6, 2.068e-6, 2.1e-6, 2.104e-6,
02131    2.142e-6, 2.181e-6, 2.257e-6, 2.362e-6, 2.5e-6, 2.664e-6,
02132    2.884e-6, 3.189e-6, 3.48e-6, 3.847e-6, 4.313e-6, 4.79e-6,
02133    5.25e-6, 5.989e-6, 6.692e-6, 7.668e-6, 8.52e-6, 9.606e-6,
02134    1.073e-5, 1.225e-5, 1.377e-5, 1.582e-5, 1.761e-5, 2.029e-5,
02135    2.284e-5, 2.602e-5, 2.94e-5, 3.483e-5, 3.928e-5, 4.618e-5,
02136    5.24e-5, 6.132e-5, 7.183e-5, 8.521e-5, 9.111e-5, 1.07e-4,
02137    1.184e-4, 1.264e-4, 1.475e-4, 1.612e-4, 1.704e-4, 1.818e-4,
02138    1.924e-4, 1.994e-4, 2.061e-4, 2.18e-4, 2.187e-4, 2.2e-4,
02139    2.196e-4, 2.131e-4, 2.015e-4, 1.988e-4, 1.847e-4, 1.729e-4,
02140    1.597e-4, 1.373e-4, 1.262e-4, 1.087e-4, 9.439e-5, 8.061e-5,
02141    7.093e-5, 6.049e-5, 5.12e-5, 4.435e-5, 3.817e-5, 3.34e-5,

```

```
02142 2.927e-5, 2.573e-5, 2.291e-5, 2.04e-5, 1.827e-5, 1.636e-5,
02143 1.463e-5, 1.309e-5, 1.17e-5, 1.047e-5, 9.315e-6, 8.328e-6,
02144 7.458e-6, 6.665e-6, 5.94e-6, 5.316e-6, 4.752e-6, 4.252e-6,
02145 3.825e-6, 3.421e-6, 3.064e-6, 2.746e-6, 2.465e-6, 2.216e-6,
02146 1.99e-6, 1.79e-6, 1.609e-6, 1.449e-6, 1.306e-6, 1.177e-6,
02147 1.063e-6, 9.607e-7, 8.672e-7, 7.855e-7, 7.118e-7, 6.46e-7,
02148 5.871e-7, 5.34e-7, 4.868e-7, 4.447e-7, 4.068e-7, 3.729e-7,
02149 3.423e-7, 3.151e-7, 2.905e-7, 2.686e-7, 2.484e-7, 2.306e-7,
02150 2.142e-7, 1.995e-7, 1.86e-7, 1.738e-7, 1.626e-7, 1.522e-7,
02151 1.427e-7, 1.338e-7, 1.258e-7, 1.183e-7, 1.116e-7, 1.056e-7,
02152 9.972e-8, 9.46e-8, 9.007e-8, 8.592e-8, 8.195e-8, 7.816e-8,
02153 7.483e-8, 7.193e-8, 6.892e-8, 6.642e-8, 6.386e-8, 6.154e-8,
02154 5.949e-8, 5.764e-8, 5.622e-8, 5.479e-8, 5.364e-8, 5.301e-8,
02155 5.267e-8, 5.263e-8, 5.313e-8, 5.41e-8, 5.55e-8, 5.745e-8,
02156 6.003e-8, 6.311e-8, 6.713e-8, 7.173e-8, 7.724e-8, 8.368e-8,
02157 9.121e-8, 9.986e-8, 1.097e-7, 1.209e-7, 1.338e-7, 1.486e-7,
02158 1.651e-7, 1.837e-7, 2.048e-7, 2.289e-7, 2.557e-7, 2.857e-7,
02159 3.195e-7, 3.587e-7, 4.015e-7, 4.497e-7, 5.049e-7, 5.665e-7,
02160 6.366e-7, 7.121e-7, 7.996e-7, 8.946e-7, 1.002e-6, 1.117e-6,
02161 1.262e-6, 1.416e-6, 1.611e-6, 1.807e-6, 2.056e-6, 2.351e-6,
02162 2.769e-6, 3.138e-6, 3.699e-6, 4.386e-6, 5.041e-6, 6.074e-6,
02163 6.812e-6, 7.79e-6, 8.855e-6, 1.014e-5, 1.095e-5, 1.245e-5,
02164 1.316e-5, 1.39e-5, 1.504e-5, 1.583e-5, 1.617e-5, 1.652e-5,
02165 1.713e-5, 1.724e-5, 1.715e-5, 1.668e-5, 1.629e-5, 1.552e-5,
02166 1.478e-5, 1.34e-5, 1.245e-5, 1.121e-5, 9.575e-6, 8.956e-6,
02167 7.345e-6, 6.597e-6, 5.612e-6, 4.818e-6, 4.165e-6, 3.579e-6,
02168 3.041e-6, 2.623e-6, 2.29e-6, 1.984e-6, 1.748e-6, 1.534e-6,
02169 1.369e-6, 1.219e-6, 1.092e-6, 9.8e-7, 8.762e-7, 7.896e-7,
02170 7.104e-7, 6.364e-7, 5.691e-7, 5.107e-7, 4.575e-7, 4.09e-7,
02171 3.667e-7, 3.287e-7, 2.931e-7, 2.633e-7, 2.356e-7, 2.111e-7,
02172 1.895e-7, 1.697e-7, 1.525e-7, 1.369e-7, 1.233e-7, 1.114e-7,
02173 9.988e-8, 9.004e-8, 8.149e-8, 7.352e-8, 6.662e-8, 6.03e-8,
02174 5.479e-8, 4.974e-8, 4.532e-8, 4.129e-8, 3.781e-8, 3.462e-8,
02175 3.176e-8, 2.919e-8, 2.687e-8, 2.481e-8, 2.292e-8, 2.119e-8,
02176 1.967e-8, 1.828e-8, 1.706e-8, 1.589e-8, 1.487e-8, 1.393e-8,
02177 1.307e-8, 1.228e-8, 1.156e-8, 1.089e-8, 1.028e-8, 9.696e-9,
02178 9.159e-9, 8.658e-9, 8.187e-9, 7.746e-9, 7.34e-9, 6.953e-9,
02179 6.594e-9, 6.259e-9, 5.948e-9, 5.66e-9, 5.386e-9, 5.135e-9,
02180 4.903e-9, 4.703e-9, 4.515e-9, 4.362e-9, 4.233e-9, 4.117e-9,
02181 4.017e-9, 3.962e-9, 3.924e-9, 3.905e-9, 3.922e-9, 3.967e-9,
02182 4.046e-9, 4.165e-9, 4.32e-9, 4.522e-9, 4.769e-9, 5.083e-9,
02183 5.443e-9, 5.872e-9, 6.366e-9, 6.949e-9, 7.601e-9, 8.371e-9,
02184 9.22e-9, 1.02e-8, 1.129e-8, 1.251e-8, 1.393e-8, 1.542e-8,
02185 1.72e-8, 1.926e-8, 2.152e-8, 2.392e-8, 2.678e-8, 3.028e-8,
02186 3.39e-8, 3.836e-8, 4.309e-8, 4.9e-8, 5.481e-8, 6.252e-8,
02187 7.039e-8, 7.883e-8, 8.849e-8, 1.012e-7, 1.142e-7, 1.3e-7,
02188 1.475e-7, 1.732e-7, 1.978e-7, 2.304e-7, 2.631e-7, 2.988e-7,
02189 3.392e-7, 3.69e-7, 4.355e-7, 4.672e-7, 5.11e-7, 5.461e-7,
02190 5.828e-7, 6.233e-7, 6.509e-7, 6.672e-7, 6.969e-7, 7.104e-7,
02191 7.439e-7, 7.463e-7, 7.708e-7, 7.466e-7, 7.668e-7, 7.549e-7,
02192 7.586e-7, 7.384e-7, 7.439e-7, 7.785e-7, 7.915e-7, 8.31e-7,
02193 8.745e-7, 9.558e-7, 1.038e-6, 1.173e-6, 1.304e-6, 1.452e-6,
02194 1.671e-6, 1.931e-6, 2.239e-6, 2.578e-6, 3.032e-6, 3.334e-6,
02195 3.98e-6, 4.3e-6, 4.518e-6, 5.321e-6, 5.508e-6, 6.211e-6, 6.59e-6,
02196 7.046e-6, 7.555e-6, 7.558e-6, 7.875e-6, 8.319e-6, 8.433e-6,
02197 8.59e-6, 8.503e-6, 8.304e-6, 8.336e-6, 7.739e-6, 7.301e-6,
02198 6.827e-6, 6.078e-6, 5.551e-6, 4.762e-6, 4.224e-6, 3.538e-6,
02199 2.984e-6, 2.619e-6, 2.227e-6, 1.923e-6, 1.669e-6, 1.462e-6,
02200 1.294e-6, 1.155e-6, 1.033e-6, 9.231e-7, 8.238e-7, 7.36e-7,
02201 6.564e-7, 5.869e-7, 5.236e-7, 4.673e-7, 4.174e-7, 3.736e-7,
02202 3.33e-7, 2.976e-7, 2.657e-7, 2.367e-7, 2.106e-7, 1.877e-7,
02203 1.671e-7, 1.494e-7, 1.332e-7, 1.192e-7, 1.065e-7, 9.558e-8,
02204 8.586e-8, 7.717e-8, 6.958e-8, 6.278e-8, 5.666e-8, 5.121e-8,
02205 4.647e-8, 4.213e-8, 3.815e-8, 3.459e-8, 3.146e-8, 2.862e-8,
02206 2.604e-8, 2.375e-8, 2.162e-8, 1.981e-8, 1.817e-8, 1.67e-8,
02207 1.537e-8, 1.417e-8, 1.31e-8, 1.215e-8, 1.128e-8, 1.05e-8,
02208 9.793e-9, 9.158e-9, 8.586e-9, 8.068e-9, 7.595e-9, 7.166e-9,
02209 6.778e-9, 6.427e-9, 6.108e-9, 5.826e-9, 5.571e-9, 5.347e-9,
02210 5.144e-9, 4.968e-9, 4.822e-9, 4.692e-9, 4.589e-9, 4.506e-9,
02211 4.467e-9, 4.44e-9, 4.466e-9, 4.515e-9, 4.718e-9, 4.729e-9,
02212 4.937e-9, 5.249e-9, 5.466e-9, 5.713e-9, 6.03e-9, 6.436e-9,
02213 6.741e-9, 7.33e-9, 7.787e-9, 8.414e-9, 8.908e-9, 9.868e-9,
02214 1.069e-8, 1.158e-8, 1.253e-8, 1.3e-8, 1.409e-8, 1.47e-8,
02215 1.548e-8, 1.612e-8, 1.666e-8, 1.736e-8, 1.763e-8, 1.812e-8,
02216 1.852e-8, 1.923e-8, 1.897e-8, 1.893e-8, 1.888e-8, 1.868e-8,
02217 1.895e-8, 1.899e-8, 1.876e-8, 1.96e-8, 2.02e-8, 2.121e-8,
02218 2.239e-8, 2.379e-8, 2.526e-8, 2.766e-8, 2.994e-8, 3.332e-8,
02219 3.703e-8, 4.158e-8, 4.774e-8, 5.499e-8, 6.355e-8, 7.349e-8,
02220 8.414e-8, 9.846e-8, 1.143e-7, 1.307e-7, 1.562e-7, 1.817e-7,
02221 2.011e-7, 2.192e-7, 2.485e-7, 2.867e-7, 3.035e-7, 3.223e-7,
02222 3.443e-7, 3.617e-7, 3.793e-7, 3.793e-7, 3.839e-7, 4.081e-7,
02223 4.117e-7, 4.085e-7, 3.92e-7, 3.851e-7, 3.754e-7, 3.49e-7,
02224 3.229e-7, 2.978e-7, 2.691e-7, 2.312e-7, 2.029e-7, 1.721e-7,
02225 1.472e-7, 1.308e-7, 1.132e-7, 9.736e-8, 8.458e-8, 7.402e-8,
02226 6.534e-8, 5.811e-8, 5.235e-8, 4.762e-8, 4.293e-8, 3.896e-8,
02227 3.526e-8, 3.165e-8, 2.833e-8, 2.551e-8, 2.288e-8, 2.036e-8,
02228 1.82e-8, 1.626e-8, 1.438e-8, 1.299e-8, 1.149e-8, 1.03e-8,
```

02229 9.148e-9, 8.122e-9, 7.264e-9, 6.425e-9, 5.777e-9, 5.06e-9,  
 02230 4.502e-9, 4.013e-9, 3.567e-9, 3.145e-9, 2.864e-9, 2.553e-9,  
 02231 2.311e-9, 2.087e-9, 1.886e-9, 1.716e-9, 1.556e-9, 1.432e-9,  
 02232 1.311e-9, 1.202e-9, 1.104e-9, 1.013e-9, 9.293e-10, 8.493e-10,  
 02233 7.79e-10, 7.185e-10, 6.642e-10, 6.141e-10, 5.684e-10, 5.346e-10,  
 02234 5.032e-10, 4.725e-10, 4.439e-10, 4.176e-10, 3.93e-10, 3.714e-10,  
 02235 3.515e-10, 3.332e-10, 3.167e-10, 3.02e-10, 2.887e-10, 2.769e-10,  
 02236 2.665e-10, 2.578e-10, 2.503e-10, 2.436e-10, 2.377e-10, 2.342e-10,  
 02237 2.305e-10, 2.296e-10, 2.278e-10, 2.321e-10, 2.355e-10, 2.402e-10,  
 02238 2.478e-10, 2.67e-10, 2.848e-10, 2.982e-10, 3.263e-10, 3.438e-10,  
 02239 3.649e-10, 3.829e-10, 4.115e-10, 4.264e-10, 4.473e-10, 4.63e-10,  
 02240 4.808e-10, 4.995e-10, 5.142e-10, 5.313e-10, 5.318e-10, 5.358e-10,  
 02241 5.452e-10, 5.507e-10, 5.698e-10, 5.782e-10, 5.983e-10, 6.164e-10,  
 02242 6.532e-10, 6.811e-10, 7.624e-10, 8.302e-10, 9.067e-10, 9.937e-10,  
 02243 1.104e-9, 1.221e-9, 1.361e-9, 1.516e-9, 1.675e-9, 1.883e-9,  
 02244 2.101e-9, 2.349e-9, 2.614e-9, 2.92e-9, 3.305e-9, 3.724e-9,  
 02245 4.142e-9, 4.887e-9, 5.614e-9, 6.506e-9, 7.463e-9, 8.817e-9,  
 02246 9.849e-9, 1.187e-8, 1.321e-8, 1.474e-8, 1.698e-8, 1.794e-8,  
 02247 2.09e-8, 2.211e-8, 2.362e-8, 2.556e-8, 2.729e-8, 2.88e-8,  
 02248 3.046e-8, 3.167e-8, 3.367e-8, 3.457e-8, 3.59e-8, 3.711e-8,  
 02249 3.826e-8, 4.001e-8, 4.211e-8, 4.315e-8, 4.661e-8, 5.01e-8,  
 02250 5.249e-8, 5.84e-8, 6.628e-8, 7.512e-8, 8.253e-8, 9.722e-8,  
 02251 1.067e-7, 1.153e-7, 1.347e-7, 1.428e-7, 1.577e-7, 1.694e-7,  
 02252 1.833e-7, 1.938e-7, 2.108e-7, 2.059e-7, 2.157e-7, 2.185e-7,  
 02253 2.208e-7, 2.182e-7, 2.093e-7, 2.014e-7, 1.962e-7, 1.819e-7,  
 02254 1.713e-7, 1.51e-7, 1.34e-7, 1.154e-7, 9.89e-8, 8.88e-8, 7.673e-8,  
 02255 6.599e-8, 5.73e-8, 5.081e-8, 4.567e-8, 4.147e-8, 3.773e-8,  
 02256 3.46e-8, 3.194e-8, 2.953e-8, 2.759e-8, 2.594e-8, 2.442e-8,  
 02257 2.355e-8, 2.283e-8, 2.279e-8, 2.231e-8, 2.279e-8, 2.239e-8,  
 02258 2.21e-8, 2.309e-8, 2.293e-8, 2.352e-8, 2.415e-8, 2.43e-8,  
 02259 2.426e-8, 2.465e-8, 2.5e-8, 2.496e-8, 2.465e-8, 2.445e-8,  
 02260 2.383e-8, 2.299e-8, 2.165e-8, 2.113e-8, 1.968e-8, 1.819e-8,  
 02261 1.644e-8, 1.427e-8, 1.27e-8, 1.082e-8, 9.428e-9, 8.091e-9,  
 02262 6.958e-9, 5.988e-9, 5.246e-9, 4.601e-9, 4.098e-9, 3.664e-9,  
 02263 3.287e-9, 2.942e-9, 2.656e-9, 2.364e-9, 2.118e-9, 1.903e-9,  
 02264 1.703e-9, 1.525e-9, 1.365e-9, 1.229e-9, 1.107e-9, 9.96e-10,  
 02265 8.945e-10, 8.08e-10, 7.308e-10, 6.616e-10, 5.994e-10, 5.422e-10,  
 02266 4.929e-10, 4.478e-10, 4.07e-10, 3.707e-10, 3.379e-10, 3.087e-10,  
 02267 2.823e-10, 2.592e-10, 2.385e-10, 2.201e-10, 2.038e-10, 1.897e-10,  
 02268 1.774e-10, 1.667e-10, 1.577e-10, 1.502e-10, 1.437e-10, 1.394e-10,  
 02269 1.358e-10, 1.324e-10, 1.329e-10, 1.324e-10, 1.36e-10, 1.39e-10,  
 02270 1.424e-10, 1.544e-10, 1.651e-10, 1.817e-10, 1.984e-10, 2.195e-10,  
 02271 2.438e-10, 2.7e-10, 2.991e-10, 3.322e-10, 3.632e-10, 3.957e-10,  
 02272 4.36e-10, 4.701e-10, 5.03e-10, 5.381e-10, 5.793e-10, 6.19e-10,  
 02273 6.596e-10, 7.004e-10, 7.561e-10, 7.934e-10, 8.552e-10, 9.142e-10,  
 02274 9.57e-10, 1.027e-9, 1.097e-9, 1.193e-9, 1.334e-9, 1.47e-9,  
 02275 1.636e-9, 1.871e-9, 2.122e-9, 2.519e-9, 2.806e-9, 3.203e-9,  
 02276 3.846e-9, 4.362e-9, 5.114e-9, 5.643e-9, 6.305e-9, 6.981e-9,  
 02277 7.983e-9, 8.783e-9, 9.419e-9, 1.017e-8, 1.063e-8, 1.121e-8,  
 02278 1.13e-8, 1.201e-8, 1.225e-8, 1.232e-8, 1.223e-8, 1.177e-8,  
 02279 1.151e-8, 1.116e-8, 1.047e-8, 9.698e-9, 8.734e-9, 8.202e-9,  
 02280 7.041e-9, 6.074e-9, 5.172e-9, 4.468e-9, 3.913e-9, 3.414e-9,  
 02281 2.975e-9, 2.65e-9, 2.406e-9, 2.173e-9, 2.009e-9, 1.861e-9,  
 02282 1.727e-9, 1.612e-9, 1.514e-9, 1.43e-9, 1.362e-9, 1.333e-9,  
 02283 1.288e-9, 1.249e-9, 1.238e-9, 1.228e-9, 1.217e-9, 1.202e-9,  
 02284 1.209e-9, 1.177e-9, 1.157e-9, 1.165e-9, 1.142e-9, 1.131e-9,  
 02285 1.138e-9, 1.117e-9, 1.1e-9, 1.069e-9, 1.023e-9, 1.005e-9,  
 02286 9.159e-10, 8.863e-10, 7.865e-10, 7.153e-10, 6.247e-10, 5.846e-10,  
 02287 5.133e-10, 4.36e-10, 3.789e-10, 3.335e-10, 2.833e-10, 2.483e-10,  
 02288 2.155e-10, 1.918e-10, 1.709e-10, 1.529e-10, 1.374e-10, 1.235e-10,  
 02289 1.108e-10, 9.933e-11, 8.932e-11, 8.022e-11, 7.224e-11, 6.52e-11,  
 02290 5.896e-11, 5.328e-11, 4.813e-11, 4.365e-11, 3.961e-11, 3.594e-11,  
 02291 3.266e-11, 2.967e-11, 2.701e-11, 2.464e-11, 2.248e-11, 2.054e-11,  
 02292 1.878e-11, 1.721e-11, 1.579e-11, 1.453e-11, 1.341e-11, 1.241e-11,  
 02293 1.154e-11, 1.078e-11, 1.014e-11, 9.601e-12, 9.167e-12, 8.838e-12,  
 02294 8.614e-12, 8.493e-12, 8.481e-12, 8.581e-12, 8.795e-12, 9.131e-12,  
 02295 9.601e-12, 1.021e-11, 1.097e-11, 1.191e-11, 1.303e-11, 1.439e-11,  
 02296 1.601e-11, 1.778e-11, 1.984e-11, 2.234e-11, 2.474e-11, 2.766e-11,  
 02297 3.085e-11, 3.415e-11, 3.821e-11, 4.261e-11, 4.748e-11, 5.323e-11,  
 02298 5.935e-11, 6.619e-11, 7.418e-11, 8.294e-11, 9.26e-11, 1.039e-10,  
 02299 1.156e-10, 1.297e-10, 1.46e-10, 1.641e-10, 1.858e-10, 2.1e-10,  
 02300 2.383e-10, 2.724e-10, 3.116e-10, 3.538e-10, 4.173e-10, 4.727e-10,  
 02301 5.503e-10, 6.337e-10, 7.32e-10, 8.298e-10, 9.328e-10, 1.059e-9,  
 02302 1.176e-9, 1.328e-9, 1.445e-9, 1.593e-9, 1.77e-9, 1.954e-9,  
 02303 2.175e-9, 2.405e-9, 2.622e-9, 2.906e-9, 3.294e-9, 3.713e-9,  
 02304 3.98e-9, 4.384e-9, 4.987e-9, 5.311e-9, 5.874e-9, 6.337e-9,  
 02305 7.027e-9, 7.39e-9, 7.769e-9, 8.374e-9, 8.605e-9, 9.165e-9,  
 02306 9.415e-9, 9.511e-9, 9.704e-9, 9.588e-9, 9.45e-9, 9.086e-9,  
 02307 8.798e-9, 8.469e-9, 7.697e-9, 7.168e-9, 6.255e-9, 5.772e-9,  
 02308 4.97e-9, 4.271e-9, 3.653e-9, 3.154e-9, 2.742e-9, 2.435e-9,  
 02309 2.166e-9, 1.936e-9, 1.731e-9, 1.556e-9, 1.399e-9, 1.272e-9,  
 02310 1.157e-9, 1.066e-9, 9.844e-10, 9.258e-10, 8.787e-10, 8.421e-10,  
 02311 8.083e-10, 8.046e-10, 8.067e-10, 8.181e-10, 8.325e-10, 8.517e-10,  
 02312 9.151e-10, 9.351e-10, 9.677e-10, 1.071e-9, 1.126e-9, 1.219e-9,  
 02313 1.297e-9, 1.408e-9, 1.476e-9, 1.517e-9, 1.6e-9, 1.649e-9,  
 02314 1.678e-9, 1.746e-9, 1.742e-9, 1.728e-9, 1.699e-9, 1.655e-9,  
 02315 1.561e-9, 1.48e-9, 1.451e-9, 1.411e-9, 1.171e-9, 1.106e-9,

```
02316 9.714e-10, 8.523e-10, 7.346e-10, 6.241e-10, 5.371e-10, 4.704e-10,
02317 4.144e-10, 3.683e-10, 3.292e-10, 2.942e-10, 2.62e-10, 2.341e-10,
02318 2.104e-10, 1.884e-10, 1.7e-10, 1.546e-10, 1.394e-10, 1.265e-10,
02319 1.14e-10, 1.019e-10, 9.279e-11, 8.283e-11, 7.458e-11, 6.668e-11,
02320 5.976e-11, 5.33e-11, 4.794e-11, 4.289e-11, 3.841e-11, 3.467e-11,
02321 3.13e-11, 2.832e-11, 2.582e-11, 2.356e-11, 2.152e-11, 1.97e-11,
02322 1.808e-11, 1.664e-11, 1.539e-11, 1.434e-11, 1.344e-11, 1.269e-11,
02323 1.209e-11, 1.162e-11, 1.129e-11, 1.108e-11, 1.099e-11, 1.103e-11,
02324 1.119e-11, 1.148e-11, 1.193e-11, 1.252e-11, 1.329e-11, 1.421e-11,
02325 1.555e-11, 1.685e-11, 1.839e-11, 2.054e-11, 2.317e-11, 2.571e-11,
02326 2.839e-11, 3.171e-11, 3.49e-11, 3.886e-11, 4.287e-11, 4.645e-11,
02327 5.047e-11, 5.592e-11, 6.109e-11, 6.628e-11, 7.381e-11, 8.088e-11,
02328 8.966e-11, 1.045e-10, 1.12e-10, 1.287e-10, 1.486e-10, 1.662e-10,
02329 1.866e-10, 2.133e-10, 2.524e-10, 2.776e-10, 3.204e-10, 3.559e-10,
02330 4.028e-10, 4.448e-10, 4.882e-10, 5.244e-10, 5.605e-10, 6.018e-10,
02331 6.328e-10, 6.579e-10, 6.541e-10, 7.024e-10, 7.074e-10, 7.068e-10,
02332 7.009e-10, 6.698e-10, 6.545e-10, 6.209e-10, 5.834e-10, 5.412e-10,
02333 5.001e-10, 4.231e-10, 3.727e-10, 3.211e-10, 2.833e-10, 2.447e-10,
02334 2.097e-10, 1.843e-10, 1.639e-10, 1.449e-10, 1.27e-10, 1.161e-10,
02335 1.033e-10, 9.282e-11, 8.407e-11, 7.639e-11, 7.023e-11, 6.474e-11,
02336 6.142e-11, 5.76e-11, 5.568e-11, 5.472e-11, 5.39e-11, 5.455e-11,
02337 5.54e-11, 5.587e-11, 6.23e-11, 6.49e-11, 6.868e-11, 7.382e-11,
02338 8.022e-11, 8.372e-11, 9.243e-11, 1.004e-10, 1.062e-10, 1.13e-10,
02339 1.176e-10, 1.244e-10, 1.279e-10, 1.298e-10, 1.302e-10, 1.312e-10,
02340 1.295e-10, 1.244e-10, 1.211e-10, 1.167e-10, 1.098e-10, 9.927e-11,
02341 8.854e-11, 8.011e-11, 7.182e-11, 5.923e-11, 5.212e-11, 4.453e-11,
02342 3.832e-11, 3.371e-11, 2.987e-11, 2.651e-11, 2.354e-11, 2.093e-11,
02343 1.863e-11, 1.662e-11, 1.486e-11, 1.331e-11, 1.193e-11, 1.071e-11,
02344 9.628e-12, 8.66e-12, 7.801e-12, 7.031e-12, 6.347e-12, 5.733e-12,
02345 5.182e-12, 4.695e-12, 4.26e-12, 3.874e-12, 3.533e-12, 3.235e-12,
02346 2.979e-12, 2.76e-12, 2.579e-12, 2.432e-12, 2.321e-12, 2.246e-12,
02347 2.205e-12, 2.196e-12, 2.223e-12, 2.288e-12, 2.387e-12, 2.525e-12,
02348 2.704e-12, 2.925e-12, 3.191e-12, 3.508e-12, 3.876e-12, 4.303e-12,
02349 4.793e-12, 5.347e-12, 5.978e-12, 6.682e-12, 7.467e-12, 8.34e-12,
02350 9.293e-12, 1.035e-11, 1.152e-11, 1.285e-11, 1.428e-11, 1.586e-11,
02351 1.764e-11, 1.972e-11, 2.214e-11, 2.478e-11, 2.776e-11, 3.151e-11,
02352 3.591e-11, 4.103e-11, 4.66e-11, 5.395e-11, 6.306e-11, 7.172e-11,
02353 8.358e-11, 9.67e-11, 1.11e-10, 1.325e-10, 1.494e-10, 1.736e-10,
02354 2.007e-10, 2.296e-10, 2.608e-10, 3.004e-10, 3.361e-10, 3.727e-10,
02355 4.373e-10, 4.838e-10, 5.483e-10, 6.006e-10, 6.535e-10, 6.899e-10,
02356 7.687e-10, 8.444e-10, 8.798e-10, 9.135e-10, 9.532e-10, 9.757e-10,
02357 9.968e-10, 1.006e-9, 9.949e-10, 9.789e-10, 9.564e-10, 9.215e-10,
02358 8.51e-10, 8.394e-10, 7.707e-10, 7.152e-10, 6.274e-10, 5.598e-10,
02359 5.028e-10, 4.3e-10, 3.71e-10, 3.245e-10, 2.809e-10, 2.461e-10,
02360 2.154e-10, 1.91e-10, 1.685e-10, 1.487e-10, 1.313e-10, 1.163e-10,
02361 1.031e-10, 9.172e-11, 8.221e-11, 7.382e-11, 6.693e-11, 6.079e-11,
02362 5.581e-11, 5.167e-11, 4.811e-11, 4.506e-11, 4.255e-11, 4.083e-11,
02363 3.949e-11, 3.881e-11, 3.861e-11, 3.858e-11, 3.951e-11, 4.045e-11,
02364 4.24e-11, 4.487e-11, 4.806e-11, 5.133e-11, 5.518e-11, 5.919e-11,
02365 6.533e-11, 7.031e-11, 7.762e-11, 8.305e-11, 9.252e-11, 9.727e-11,
02366 1.045e-10, 1.117e-10, 1.2e-10, 1.275e-10, 1.341e-10, 1.362e-10,
02367 1.438e-10, 1.45e-10, 1.455e-10, 1.455e-10, 1.434e-10, 1.381e-10,
02368 1.301e-10, 1.276e-10, 1.163e-10, 1.089e-10, 9.911e-11, 8.943e-11,
02369 7.618e-11, 6.424e-11, 5.717e-11, 4.866e-11, 4.257e-11, 3.773e-11,
02370 3.331e-11, 2.958e-11, 2.629e-11, 2.316e-11, 2.073e-11, 1.841e-11,
02371 1.635e-11, 1.464e-11, 1.31e-11, 1.16e-11, 1.047e-11, 9.408e-12,
02372 8.414e-12, 7.521e-12, 6.705e-12, 5.993e-12, 5.371e-12, 4.815e-12,
02373 4.338e-12, 3.921e-12, 3.567e-12, 3.265e-12, 3.01e-12, 2.795e-12,
02374 2.613e-12, 2.464e-12, 2.346e-12, 2.256e-12, 2.195e-12, 2.165e-12,
02375 2.166e-12, 2.198e-12, 2.262e-12, 2.364e-12, 2.502e-12, 2.682e-12,
02376 2.908e-12, 3.187e-12, 3.533e-12, 3.946e-12, 4.418e-12, 5.013e-12,
02377 5.708e-12, 6.379e-12, 7.43e-12, 8.39e-12, 9.51e-12, 1.078e-11,
02378 1.259e-11, 1.438e-11, 1.63e-11, 1.814e-11, 2.055e-11, 2.348e-11,
02379 2.664e-11, 2.956e-11, 3.3e-11, 3.677e-11, 4.032e-11, 4.494e-11,
02380 4.951e-11, 5.452e-11, 6.014e-11, 6.5e-11, 6.915e-11, 7.45e-11,
02381 7.971e-11, 8.468e-11, 8.726e-11, 8.995e-11, 9.182e-11, 9.509e-11,
02382 9.333e-11, 9.386e-11, 9.457e-11, 9.21e-11, 9.019e-11, 8.68e-11,
02383 8.298e-11, 7.947e-11, 7.46e-11, 7.082e-11, 6.132e-11, 5.855e-11,
02384 5.073e-11, 4.464e-11, 3.825e-11, 3.375e-11, 2.911e-11, 2.535e-11,
02385 2.16e-11, 1.907e-11, 1.665e-11, 1.463e-11, 1.291e-11, 1.133e-11,
02386 9.997e-12, 8.836e-12, 7.839e-12, 6.943e-12, 6.254e-12, 5.6e-12,
02387 5.029e-12, 4.529e-12, 4.102e-12, 3.737e-12, 3.428e-12, 3.169e-12,
02388 2.959e-12, 2.798e-12, 2.675e-12, 2.582e-12, 2.644e-12, 2.557e-12,
02389 2.614e-12, 2.717e-12, 2.874e-12, 3.056e-12, 3.187e-12, 3.631e-12,
02390 3.979e-12, 4.248e-12, 4.817e-12, 5.266e-12, 5.836e-12, 6.365e-12,
02391 6.807e-12, 7.47e-12, 7.951e-12, 8.636e-12, 8.972e-12, 9.314e-12,
02392 9.445e-12, 1.003e-11, 1.013e-11, 9.937e-12, 9.729e-12, 9.064e-12,
02393 9.119e-12, 9.124e-12, 8.704e-12, 8.078e-12, 7.47e-12, 6.329e-12,
02394 5.674e-12, 4.808e-12, 4.119e-12, 3.554e-12, 3.103e-12, 2.731e-12,
02395 2.415e-12, 2.15e-12, 1.926e-12, 1.737e-12, 1.578e-12, 1.447e-12,
02396 1.34e-12, 1.255e-12, 1.191e-12, 1.146e-12, 1.121e-12, 1.114e-12,
02397 1.126e-12, 1.156e-12, 1.207e-12, 1.278e-12, 1.372e-12, 1.49e-12,
02398 1.633e-12, 1.805e-12, 2.01e-12, 2.249e-12, 2.528e-12, 2.852e-12,
02399 3.228e-12, 3.658e-12, 4.153e-12, 4.728e-12, 5.394e-12, 6.176e-12,
02400 7.126e-12, 8.188e-12, 9.328e-12, 1.103e-11, 1.276e-11, 1.417e-11,
02401 1.615e-11, 1.814e-11, 2.155e-11, 2.429e-11, 2.826e-11, 3.222e-11,
02402 3.664e-11, 4.14e-11, 4.906e-11, 5.536e-11, 6.327e-11, 7.088e-11,
```

```
02403      8.316e-11, 9.242e-11, 1.07e-10, 1.223e-10, 1.341e-10, 1.553e-10,
02404      1.703e-10, 1.9e-10, 2.022e-10, 2.233e-10, 2.345e-10, 2.438e-10,
02405      2.546e-10, 2.599e-10, 2.661e-10, 2.703e-10, 2.686e-10, 2.662e-10,
02406      2.56e-10, 2.552e-10, 2.378e-10, 2.252e-10, 2.146e-10, 1.885e-10,
02407      1.668e-10, 1.441e-10, 1.295e-10, 1.119e-10, 9.893e-11, 8.687e-11,
02408      7.678e-11, 6.685e-11, 5.879e-11, 5.127e-11, 4.505e-11, 3.997e-11,
02409      3.511e-11
02410  };
02411
02412  static double h2ofrn[2001] = { .01095, .01126, .01205, .01322, .0143,
02413      .01506, .01548, .01534, .01486, .01373, .01262, .01134, .01001,
02414      .008702, .007475, .006481, .00548, .0046, .003833, .00311,
02415      .002543, .002049, .00168, .001374, .001046, 8.193e-4, 6.267e-4,
02416      4.968e-4, 3.924e-4, 2.983e-4, 2.477e-4, 1.997e-4, 1.596e-4,
02417      1.331e-4, 1.061e-4, 8.942e-5, 7.168e-5, 5.887e-5, 4.848e-5,
02418      3.817e-5, 3.17e-5, 2.579e-5, 2.162e-5, 1.768e-5, 1.49e-5,
02419      1.231e-5, 1.013e-5, 8.555e-6, 7.328e-6, 6.148e-6, 5.207e-6,
02420      4.387e-6, 3.741e-6, 3.22e-6, 2.753e-6, 2.346e-6, 1.985e-6,
02421      1.716e-6, 1.475e-6, 1.286e-6, 1.122e-6, 9.661e-7, 8.284e-7,
02422      7.057e-7, 6.119e-7, 5.29e-7, 4.571e-7, 3.948e-7, 3.432e-7,
02423      2.983e-7, 2.589e-7, 2.265e-7, 1.976e-7, 1.704e-7, 1.456e-7,
02424      1.26e-7, 1.101e-7, 9.648e-8, 8.415e-8, 7.34e-8, 6.441e-8,
02425      5.643e-8, 4.94e-8, 4.276e-8, 3.703e-8, 3.227e-8, 2.825e-8,
02426      2.478e-8, 2.174e-8, 1.898e-8, 1.664e-8, 1.458e-8, 1.278e-8,
02427      1.126e-8, 9.891e-9, 8.709e-9, 7.652e-9, 6.759e-9, 5.975e-9,
02428      5.31e-9, 4.728e-9, 4.214e-9, 3.792e-9, 3.463e-9, 3.226e-9,
02429      2.992e-9, 2.813e-9, 2.749e-9, 2.809e-9, 2.913e-9, 3.037e-9,
02430      3.413e-9, 3.738e-9, 4.189e-9, 4.808e-9, 5.978e-9, 7.088e-9,
02431      8.071e-9, 9.61e-9, 1.21e-8, 1.5e-8, 1.764e-8, 2.221e-8, 2.898e-8,
02432      3.948e-8, 5.068e-8, 6.227e-8, 7.898e-8, 1.033e-7, 1.437e-7,
02433      1.889e-7, 2.589e-7, 3.59e-7, 4.971e-7, 7.156e-7, 9.983e-7,
02434      1.381e-6, 1.929e-6, 2.591e-6, 3.453e-6, 4.57e-6, 5.93e-6,
02435      7.552e-6, 9.556e-6, 1.183e-5, 1.425e-5, 1.681e-5, 1.978e-5,
02436      2.335e-5, 2.668e-5, 3.022e-5, 3.371e-5, 3.715e-5, 3.967e-5,
02437      4.06e-5, 4.01e-5, 3.809e-5, 3.491e-5, 3.155e-5, 2.848e-5,
02438      2.678e-5, 2.66e-5, 2.811e-5, 3.071e-5, 3.294e-5, 3.459e-5,
02439      3.569e-5, 3.56e-5, 3.434e-5, 3.186e-5, 2.916e-5, 2.622e-5,
02440      2.275e-5, 1.918e-5, 1.62e-5, 1.373e-5, 1.182e-5, 1.006e-5,
02441      8.556e-6, 7.26e-6, 6.107e-6, 5.034e-6, 4.211e-6, 3.426e-6,
02442      2.865e-6, 2.446e-6, 1.998e-6, 1.628e-6, 1.242e-6, 1.005e-6,
02443      7.853e-7, 6.21e-7, 5.071e-7, 4.156e-7, 3.548e-7, 2.825e-7,
02444      2.261e-7, 1.916e-7, 1.51e-7, 1.279e-7, 1.059e-7, 9.14e-8,
02445      7.707e-8, 6.17e-8, 5.311e-8, 4.263e-8, 3.518e-8, 2.961e-8,
02446      2.457e-8, 2.119e-8, 1.712e-8, 1.439e-8, 1.201e-8, 1.003e-8,
02447      8.564e-9, 7.199e-9, 6.184e-9, 5.206e-9, 4.376e-9, 3.708e-9,
02448      3.157e-9, 2.725e-9, 2.361e-9, 2.074e-9, 1.797e-9, 1.562e-9,
02449      1.364e-9, 1.196e-9, 1.042e-9, 8.862e-10, 7.648e-10, 6.544e-10,
02450      5.609e-10, 4.791e-10, 4.108e-10, 3.531e-10, 3.038e-10, 2.618e-10,
02451      2.268e-10, 1.969e-10, 1.715e-10, 1.496e-10, 1.308e-10, 1.147e-10,
02452      1.008e-10, 8.894e-11, 7.885e-11, 7.031e-11, 6.355e-11, 5.854e-11,
02453      5.534e-11, 5.466e-11, 5.725e-11, 6.447e-11, 7.943e-11, 1.038e-10,
02454      1.437e-10, 2.04e-10, 2.901e-10, 4.051e-10, 5.556e-10, 7.314e-10,
02455      9.291e-10, 1.134e-9, 1.321e-9, 1.482e-9, 1.596e-9, 1.669e-9,
02456      1.715e-9, 1.762e-9, 1.817e-9, 1.828e-9, 1.848e-9, 1.873e-9,
02457      1.902e-9, 1.894e-9, 1.864e-9, 1.841e-9, 1.797e-9, 1.704e-9,
02458      1.559e-9, 1.382e-9, 1.187e-9, 1.001e-9, 8.468e-10, 7.265e-10,
02459      6.521e-10, 6.381e-10, 6.66e-10, 7.637e-10, 9.705e-10, 1.368e-9,
02460      1.856e-9, 2.656e-9, 3.954e-9, 5.96e-9, 8.72e-9, 1.247e-8,
02461      1.781e-8, 2.491e-8, 3.311e-8, 4.272e-8, 5.205e-8, 6.268e-8,
02462      7.337e-8, 8.277e-8, 9.185e-8, 1.004e-7, 1.091e-7, 1.159e-7,
02463      1.188e-7, 1.175e-7, 1.124e-7, 1.033e-7, 9.381e-8, 8.501e-8,
02464      7.956e-8, 7.894e-8, 8.331e-8, 9.102e-8, 9.836e-8, 1.035e-7,
02465      1.064e-7, 1.06e-7, 1.032e-7, 9.808e-8, 9.139e-8, 8.442e-8,
02466      7.641e-8, 6.881e-8, 6.161e-8, 5.404e-8, 4.804e-8, 4.446e-8,
02467      4.328e-8, 4.259e-8, 4.421e-8, 4.673e-8, 4.985e-8, 5.335e-8,
02468      5.796e-8, 6.542e-8, 7.714e-8, 8.827e-8, 1.04e-7, 1.238e-7,
02469      1.499e-7, 1.829e-7, 2.222e-7, 2.689e-7, 3.303e-7, 3.981e-7,
02470      4.84e-7, 5.91e-7, 7.363e-7, 9.087e-7, 1.139e-6, 1.455e-6,
02471      1.866e-6, 2.44e-6, 3.115e-6, 3.941e-6, 4.891e-6, 5.992e-6,
02472      7.111e-6, 8.296e-6, 9.21e-6, 9.987e-6, 1.044e-5, 1.073e-5,
02473      1.092e-5, 1.106e-5, 1.138e-5, 1.171e-5, 1.186e-5, 1.186e-5,
02474      1.179e-5, 1.166e-5, 1.151e-5, 1.16e-5, 1.197e-5, 1.241e-5,
02475      1.268e-5, 1.26e-5, 1.184e-5, 1.063e-5, 9.204e-6, 7.584e-6,
02476      6.053e-6, 4.482e-6, 3.252e-6, 2.337e-6, 1.662e-6, 1.18e-6,
02477      8.15e-7, 5.95e-7, 4.354e-7, 3.302e-7, 2.494e-7, 1.93e-7,
02478      1.545e-7, 1.25e-7, 1.039e-7, 8.602e-8, 7.127e-8, 5.897e-8,
02479      4.838e-8, 4.018e-8, 3.28e-8, 2.72e-8, 2.307e-8, 1.972e-8,
02480      1.654e-8, 1.421e-8, 1.174e-8, 1.004e-8, 8.739e-9, 7.358e-9,
02481      6.242e-9, 5.303e-9, 4.567e-9, 3.94e-9, 3.375e-9, 2.864e-9,
02482      2.422e-9, 2.057e-9, 1.75e-9, 1.505e-9, 1.294e-9, 1.101e-9,
02483      9.401e-10, 8.018e-10, 6.903e-10, 5.965e-10, 5.087e-10, 4.364e-10,
02484      3.759e-10, 3.247e-10, 2.809e-10, 2.438e-10, 2.123e-10, 1.853e-10,
02485      1.622e-10, 1.426e-10, 1.26e-10, 1.125e-10, 1.022e-10, 9.582e-11,
02486      9.388e-11, 9.801e-11, 1.08e-10, 1.276e-10, 1.551e-10, 1.903e-10,
02487      2.291e-10, 2.724e-10, 3.117e-10, 3.4e-10, 3.562e-10, 3.625e-10,
02488      3.619e-10, 3.429e-10, 3.221e-10, 2.943e-10, 2.645e-10, 2.338e-10,
02489      2.062e-10, 1.901e-10, 1.814e-10, 1.827e-10, 1.906e-10, 1.984e-10,
```

```
02490 2.04e-10, 2.068e-10, 2.075e-10, 2.018e-10, 1.959e-10, 1.897e-10,
02491 1.852e-10, 1.791e-10, 1.696e-10, 1.634e-10, 1.598e-10, 1.561e-10,
02492 1.518e-10, 1.443e-10, 1.377e-10, 1.346e-10, 1.342e-10, 1.375e-10,
02493 1.525e-10, 1.767e-10, 2.108e-10, 2.524e-10, 2.981e-10, 3.477e-10,
02494 4.262e-10, 5.326e-10, 6.646e-10, 8.321e-10, 1.069e-9, 1.386e-9,
02495 1.743e-9, 2.216e-9, 2.808e-9, 3.585e-9, 4.552e-9, 5.907e-9,
02496 7.611e-9, 9.774e-9, 1.255e-8, 1.666e-8, 2.279e-8, 3.221e-8,
02497 4.531e-8, 6.4e-8, 9.187e-8, 1.295e-7, 1.825e-7, 2.431e-7,
02498 3.181e-7, 4.009e-7, 4.941e-7, 5.88e-7, 6.623e-7, 7.155e-7,
02499 7.451e-7, 7.594e-7, 7.541e-7, 7.467e-7, 7.527e-7, 7.935e-7,
02500 8.461e-7, 8.954e-7, 9.364e-7, 9.843e-7, 1.024e-6, 1.05e-6,
02501 1.059e-6, 1.074e-6, 1.072e-6, 1.043e-6, 9.789e-7, 8.803e-7,
02502 7.662e-7, 6.378e-7, 5.133e-7, 3.958e-7, 2.914e-7, 2.144e-7,
02503 1.57e-7, 1.14e-7, 8.47e-8, 6.2e-8, 4.657e-8, 3.559e-8, 2.813e-8,
02504 2.222e-8, 1.769e-8, 1.391e-8, 1.125e-8, 9.186e-9, 7.704e-9,
02505 6.447e-9, 5.381e-9, 4.442e-9, 3.669e-9, 3.057e-9, 2.564e-9,
02506 2.153e-9, 1.784e-9, 1.499e-9, 1.281e-9, 1.082e-9, 9.304e-10,
02507 8.169e-10, 6.856e-10, 5.866e-10, 5.043e-10, 4.336e-10, 3.731e-10,
02508 3.175e-10, 2.745e-10, 2.374e-10, 2.007e-10, 1.737e-10, 1.508e-10,
02509 1.302e-10, 1.13e-10, 9.672e-11, 8.375e-11, 7.265e-11, 6.244e-11,
02510 5.343e-11, 4.64e-11, 3.975e-11, 3.488e-11, 3.097e-11, 2.834e-11,
02511 2.649e-11, 2.519e-11, 2.462e-11, 2.443e-11, 2.44e-11, 2.398e-11,
02512 2.306e-11, 2.183e-11, 2.021e-11, 1.821e-11, 1.599e-11, 1.403e-11,
02513 1.196e-11, 1.023e-11, 8.728e-12, 7.606e-12, 6.941e-12, 6.545e-12,
02514 6.484e-12, 6.6e-12, 6.718e-12, 6.785e-12, 6.746e-12, 6.724e-12,
02515 6.764e-12, 6.995e-12, 7.144e-12, 7.32e-12, 7.33e-12, 7.208e-12,
02516 6.789e-12, 6.09e-12, 5.337e-12, 4.62e-12, 4.037e-12, 3.574e-12,
02517 3.311e-12, 3.346e-12, 3.566e-12, 3.836e-12, 4.076e-12, 4.351e-12,
02518 4.691e-12, 5.114e-12, 5.427e-12, 6.167e-12, 7.436e-12, 8.842e-12,
02519 1.038e-11, 1.249e-11, 1.54e-11, 1.915e-11, 2.48e-11, 3.256e-11,
02520 4.339e-11, 5.611e-11, 7.519e-11, 1.037e-10, 1.409e-10, 1.883e-10,
02521 2.503e-10, 3.38e-10, 4.468e-10, 5.801e-10, 7.335e-10, 8.98e-10,
02522 1.11e-9, 1.363e-9, 1.677e-9, 2.104e-9, 2.681e-9, 3.531e-9,
02523 4.621e-9, 6.106e-9, 8.154e-9, 1.046e-8, 1.312e-8, 1.607e-8,
02524 1.948e-8, 2.266e-8, 2.495e-8, 2.655e-8, 2.739e-8, 2.739e-8,
02525 2.662e-8, 2.589e-8, 2.59e-8, 2.664e-8, 2.833e-8, 3.023e-8,
02526 3.305e-8, 3.558e-8, 3.793e-8, 3.961e-8, 4.056e-8, 4.102e-8,
02527 4.025e-8, 3.917e-8, 3.706e-8, 3.493e-8, 3.249e-8, 3.096e-8,
02528 3.011e-8, 3.111e-8, 3.395e-8, 3.958e-8, 4.875e-8, 6.066e-8,
02529 7.915e-8, 1.011e-7, 1.3e-7, 1.622e-7, 2.003e-7, 2.448e-7,
02530 2.863e-7, 3.317e-7, 3.655e-7, 3.96e-7, 4.098e-7, 4.168e-7,
02531 4.198e-7, 4.207e-7, 4.289e-7, 4.384e-7, 4.471e-7, 4.524e-7,
02532 4.574e-7, 4.633e-7, 4.785e-7, 5.028e-7, 5.371e-7, 5.727e-7,
02533 5.955e-7, 5.998e-7, 5.669e-7, 5.082e-7, 4.397e-7, 3.596e-7,
02534 2.814e-7, 2.074e-7, 1.486e-7, 1.057e-7, 7.25e-8, 4.946e-8,
02535 3.43e-8, 2.447e-8, 1.793e-8, 1.375e-8, 1.096e-8, 9.091e-9,
02536 7.709e-9, 6.631e-9, 5.714e-9, 4.886e-9, 4.205e-9, 3.575e-9,
02537 3.07e-9, 2.631e-9, 2.284e-9, 2.002e-9, 1.745e-9, 1.509e-9,
02538 1.284e-9, 1.084e-9, 9.163e-10, 7.663e-10, 6.346e-10, 5.283e-10,
02539 4.354e-10, 3.59e-10, 2.982e-10, 2.455e-10, 2.033e-10, 1.696e-10,
02540 1.432e-10, 1.211e-10, 1.02e-10, 8.702e-11, 7.38e-11, 6.293e-11,
02541 5.343e-11, 4.532e-11, 3.907e-11, 3.365e-11, 2.945e-11, 2.558e-11,
02542 2.192e-11, 1.895e-11, 1.636e-11, 1.42e-11, 1.228e-11, 1.063e-11,
02543 9.348e-12, 8.2e-12, 7.231e-12, 6.43e-12, 5.702e-12, 5.052e-12,
02544 4.469e-12, 4e-12, 3.679e-12, 3.387e-12, 3.197e-12, 3.158e-12,
02545 3.327e-12, 3.675e-12, 4.292e-12, 5.437e-12, 7.197e-12, 1.008e-11,
02546 1.437e-11, 2.035e-11, 2.905e-11, 4.062e-11, 5.528e-11, 7.177e-11,
02547 9.064e-11, 1.109e-10, 1.297e-10, 1.473e-10, 1.652e-10, 1.851e-10,
02548 2.079e-10, 2.313e-10, 2.619e-10, 2.958e-10, 3.352e-10, 3.796e-10,
02549 4.295e-10, 4.923e-10, 5.49e-10, 5.998e-10, 6.388e-10, 6.645e-10,
02550 6.712e-10, 6.549e-10, 6.38e-10, 6.255e-10, 6.253e-10, 6.459e-10,
02551 6.977e-10, 7.59e-10, 8.242e-10, 8.92e-10, 9.403e-10, 9.701e-10,
02552 9.483e-10, 9.135e-10, 8.617e-10, 7.921e-10, 7.168e-10, 6.382e-10,
02553 5.677e-10, 5.045e-10, 4.572e-10, 4.312e-10, 4.145e-10, 4.192e-10,
02554 4.541e-10, 5.368e-10, 6.771e-10, 8.962e-10, 1.21e-9, 1.659e-9,
02555 2.33e-9, 3.249e-9, 4.495e-9, 5.923e-9, 7.642e-9, 9.607e-9,
02556 1.178e-8, 1.399e-8, 1.584e-8, 1.73e-8, 1.816e-8, 1.87e-8,
02557 1.868e-8, 1.87e-8, 1.884e-8, 1.99e-8, 2.15e-8, 2.258e-8,
02558 2.364e-8, 2.473e-8, 2.602e-8, 2.689e-8, 2.731e-8, 2.816e-8,
02559 2.859e-8, 2.839e-8, 2.703e-8, 2.451e-8, 2.149e-8, 1.787e-8,
02560 1.449e-8, 1.111e-8, 8.282e-9, 6.121e-9, 4.494e-9, 3.367e-9,
02561 2.487e-9, 1.885e-9, 1.503e-9, 1.249e-9, 1.074e-9, 9.427e-10,
02562 8.439e-10, 7.563e-10, 6.772e-10, 6.002e-10, 5.254e-10, 4.588e-10,
02563 3.977e-10, 3.449e-10, 3.003e-10, 2.624e-10, 2.335e-10, 2.04e-10,
02564 1.771e-10, 1.534e-10, 1.296e-10, 1.097e-10, 9.173e-11, 7.73e-11,
02565 6.547e-11, 5.191e-11, 4.198e-11, 3.361e-11, 2.732e-11, 2.244e-11,
02566 1.791e-11, 1.509e-11, 1.243e-11, 1.035e-11, 8.969e-12, 7.394e-12,
02567 6.323e-12, 5.282e-12, 4.543e-12, 3.752e-12, 3.14e-12, 2.6e-12,
02568 2.194e-12, 1.825e-12, 1.511e-12, 1.245e-12, 1.024e-12, 8.539e-13,
02569 7.227e-13, 6.102e-13, 5.189e-13, 4.43e-13, 3.774e-13, 3.236e-13,
02570 2.8e-13, 2.444e-13, 2.156e-13, 1.932e-13, 1.775e-13, 1.695e-13,
02571 1.672e-13, 1.704e-13, 1.825e-13, 2.087e-13, 2.614e-13, 3.377e-13,
02572 4.817e-13, 6.989e-13, 1.062e-12, 1.562e-12, 2.288e-12, 3.295e-12,
02573 4.55e-12, 5.965e-12, 7.546e-12, 9.395e-12, 1.103e-11, 1.228e-11,
02574 1.318e-11, 1.38e-11, 1.421e-11, 1.39e-11, 1.358e-11, 1.336e-11,
02575 1.342e-11, 1.356e-11, 1.424e-11, 1.552e-11, 1.73e-11, 1.951e-11,
02576 2.128e-11, 2.249e-11, 2.277e-11, 2.226e-11, 2.111e-11, 1.922e-11,
```

02577 1.775e-11, 1.661e-11, 1.547e-11, 1.446e-11, 1.323e-11, 1.21e-11,  
 02578 1.054e-11, 9.283e-12, 8.671e-12, 8.67e-12, 9.429e-12, 1.062e-11,  
 02579 1.255e-11, 1.506e-11, 1.818e-11, 2.26e-11, 2.831e-11, 3.723e-11,  
 02580 5.092e-11, 6.968e-11, 9.826e-11, 1.349e-10, 1.87e-10, 2.58e-10,  
 02581 3.43e-10, 4.424e-10, 5.521e-10, 6.812e-10, 8.064e-10, 9.109e-10,  
 02582 9.839e-10, 1.028e-9, 1.044e-9, 1.029e-9, 1.005e-9, 1.002e-9,  
 02583 1.038e-9, 1.122e-9, 1.233e-9, 1.372e-9, 1.524e-9, 1.665e-9,  
 02584 1.804e-9, 1.908e-9, 2.015e-9, 2.117e-9, 2.219e-9, 2.336e-9,  
 02585 2.531e-9, 2.805e-9, 3.189e-9, 3.617e-9, 4.208e-9, 4.911e-9,  
 02586 5.619e-9, 6.469e-9, 7.188e-9, 7.957e-9, 8.503e-9, 9.028e-9,  
 02587 9.571e-9, 9.99e-9, 1.055e-8, 1.102e-8, 1.132e-8, 1.141e-8,  
 02588 1.145e-8, 1.145e-8, 1.176e-8, 1.224e-8, 1.304e-8, 1.388e-8,  
 02589 1.445e-8, 1.453e-8, 1.368e-8, 1.22e-8, 1.042e-8, 8.404e-9,  
 02590 6.403e-9, 4.643e-9, 3.325e-9, 2.335e-9, 1.638e-9, 1.19e-9,  
 02591 9.161e-10, 7.412e-10, 6.226e-10, 5.516e-10, 5.068e-10, 4.831e-10,  
 02592 4.856e-10, 5.162e-10, 5.785e-10, 6.539e-10, 7.485e-10, 8.565e-10,  
 02593 9.534e-10, 1.052e-9, 1.115e-9, 1.173e-9, 1.203e-9, 1.224e-9,  
 02594 1.243e-9, 1.248e-9, 1.261e-9, 1.265e-9, 1.25e-9, 1.217e-9,  
 02595 1.176e-9, 1.145e-9, 1.153e-9, 1.199e-9, 1.278e-9, 1.366e-9,  
 02596 1.426e-9, 1.444e-9, 1.365e-9, 1.224e-9, 1.051e-9, 8.539e-10,  
 02597 6.564e-10, 4.751e-10, 3.404e-10, 2.377e-10, 1.631e-10, 1.114e-10,  
 02598 7.87e-11, 5.793e-11, 4.284e-11, 3.3e-11, 2.62e-11, 2.152e-11,  
 02599 1.777e-11, 1.496e-11, 1.242e-11, 1.037e-11, 8.725e-12, 7.004e-12,  
 02600 5.718e-12, 4.769e-12, 3.952e-12, 3.336e-12, 2.712e-12, 2.213e-12,  
 02601 1.803e-12, 1.492e-12, 1.236e-12, 1.006e-12, 8.384e-13, 7.063e-13,  
 02602 5.879e-13, 4.93e-13, 4.171e-13, 3.569e-13, 3.083e-13, 2.688e-13,  
 02603 2.333e-13, 2.035e-13, 1.82e-13, 1.682e-13, 1.635e-13, 1.674e-13,  
 02604 1.769e-13, 2.022e-13, 2.485e-13, 3.127e-13, 4.25e-13, 5.928e-13,  
 02605 8.514e-13, 1.236e-12, 1.701e-12, 2.392e-12, 3.231e-12, 4.35e-12,  
 02606 5.559e-12, 6.915e-12, 8.519e-12, 1.013e-11, 1.146e-11, 1.24e-11,  
 02607 1.305e-11, 1.333e-11, 1.318e-11, 1.263e-11, 1.238e-11, 1.244e-11,  
 02608 1.305e-11, 1.432e-11, 1.623e-11, 1.846e-11, 2.09e-11, 2.328e-11,  
 02609 2.526e-11, 2.637e-11, 2.702e-11, 2.794e-11, 2.889e-11, 2.989e-11,  
 02610 3.231e-11, 3.68e-11, 4.375e-11, 5.504e-11, 7.159e-11, 9.502e-11,  
 02611 1.279e-10, 1.645e-10, 2.098e-10, 2.618e-10, 3.189e-10, 3.79e-10,  
 02612 4.303e-10, 4.753e-10, 5.027e-10, 5.221e-10, 5.293e-10, 5.346e-10,  
 02613 5.467e-10, 5.796e-10, 6.2e-10, 6.454e-10, 6.705e-10, 6.925e-10,  
 02614 7.233e-10, 7.35e-10, 7.538e-10, 7.861e-10, 8.077e-10, 8.132e-10,  
 02615 7.749e-10, 7.036e-10, 6.143e-10, 5.093e-10, 4.089e-10, 3.092e-10,  
 02616 2.299e-10, 1.705e-10, 1.277e-10, 9.723e-11, 7.533e-11, 6.126e-11,  
 02617 5.154e-11, 4.428e-11, 3.913e-11, 3.521e-11, 3.297e-11, 3.275e-11,  
 02618 3.46e-11, 3.798e-11, 4.251e-11, 4.745e-11, 5.232e-11, 5.606e-11,  
 02619 5.82e-11, 5.88e-11, 5.79e-11, 5.661e-11, 5.491e-11, 5.366e-11,  
 02620 5.341e-11, 5.353e-11, 5.336e-11, 5.293e-11, 5.248e-11, 5.235e-11,  
 02621 5.208e-11, 5.322e-11, 5.521e-11, 5.725e-11, 5.827e-11, 5.685e-11,  
 02622 5.245e-11, 4.612e-11, 3.884e-11, 3.129e-11, 2.404e-11, 1.732e-11,  
 02623 1.223e-11, 8.574e-12, 5.888e-12, 3.986e-12, 2.732e-12, 1.948e-12,  
 02624 1.414e-12, 1.061e-12, 8.298e-13, 6.612e-13, 5.413e-13, 4.472e-13,  
 02625 3.772e-13, 3.181e-13, 2.645e-13, 2.171e-13, 1.778e-13, 1.464e-13,  
 02626 1.183e-13, 9.637e-14, 7.991e-14, 6.668e-14, 5.57e-14, 4.663e-14,  
 02627 3.848e-14, 3.233e-14, 2.706e-14, 2.284e-14, 1.944e-14, 1.664e-14,  
 02628 1.43e-14, 1.233e-14, 1.066e-14, 9.234e-15, 8.023e-15, 6.993e-15,  
 02629 6.119e-15, 5.384e-15, 4.774e-15, 4.283e-15, 3.916e-15, 3.695e-15,  
 02630 3.682e-15, 4.004e-15, 4.912e-15, 6.853e-15, 1.056e-14, 1.712e-14,  
 02631 2.804e-14, 4.516e-14, 7.113e-14, 1.084e-13, 1.426e-13, 1.734e-13,  
 02632 1.978e-13, 2.194e-13, 2.388e-13, 2.489e-13, 2.626e-13, 2.865e-13,  
 02633 3.105e-13, 3.387e-13, 3.652e-13, 3.984e-13, 4.398e-13, 4.906e-13,  
 02634 5.55e-13, 6.517e-13, 7.813e-13, 9.272e-13, 1.164e-12, 1.434e-12,  
 02635 1.849e-12, 2.524e-12, 3.328e-12, 4.523e-12, 6.108e-12, 8.207e-12,  
 02636 1.122e-11, 1.477e-11, 1.9e-11, 2.412e-11, 2.984e-11, 3.68e-11,  
 02637 4.353e-11, 4.963e-11, 5.478e-11, 5.903e-11, 6.233e-11, 6.483e-11,  
 02638 6.904e-11, 7.569e-11, 8.719e-11, 1.048e-10, 1.278e-10, 1.557e-10,  
 02639 1.869e-10, 2.218e-10, 2.61e-10, 2.975e-10, 3.371e-10, 3.746e-10,  
 02640 4.065e-10, 4.336e-10, 4.503e-10, 4.701e-10, 4.8e-10, 4.917e-10,  
 02641 5.038e-10, 5.128e-10, 5.143e-10, 5.071e-10, 5.019e-10, 5.025e-10,  
 02642 5.183e-10, 5.496e-10, 5.877e-10, 6.235e-10, 6.42e-10, 6.234e-10,  
 02643 5.698e-10, 4.916e-10, 4.022e-10, 3.126e-10, 2.282e-10, 1.639e-10,  
 02644 1.142e-10, 7.919e-11, 5.69e-11, 4.313e-11, 3.413e-11, 2.807e-11,  
 02645 2.41e-11, 2.166e-11, 2.024e-11, 1.946e-11, 1.929e-11, 1.963e-11,  
 02646 2.035e-11, 2.162e-11, 2.305e-11, 2.493e-11, 2.748e-11, 3.048e-11,  
 02647 3.413e-11, 3.754e-11, 4.155e-11, 4.635e-11, 5.11e-11, 5.734e-11,  
 02648 6.338e-11, 6.99e-11, 7.611e-11, 8.125e-11, 8.654e-11, 8.951e-11,  
 02649 9.182e-11, 9.31e-11, 9.273e-11, 9.094e-11, 8.849e-11, 8.662e-11,  
 02650 8.67e-11, 8.972e-11, 9.566e-11, 1.025e-10, 1.083e-10, 1.111e-10,  
 02651 1.074e-10, 9.771e-11, 8.468e-11, 6.958e-11, 5.47e-11, 4.04e-11,  
 02652 2.94e-11, 2.075e-11, 1.442e-11, 1.01e-11, 7.281e-12, 5.409e-12,  
 02653 4.138e-12, 3.304e-12, 2.784e-12, 2.473e-12, 2.273e-12, 2.186e-12,  
 02654 2.118e-12, 2.066e-12, 1.958e-12, 1.818e-12, 1.675e-12, 1.509e-12,  
 02655 1.349e-12, 1.171e-12, 9.838e-13, 8.213e-13, 6.765e-13, 5.378e-13,  
 02656 4.161e-13, 3.119e-13, 2.279e-13, 1.637e-13, 1.152e-13, 8.112e-14,  
 02657 5.919e-14, 4.47e-14, 3.492e-14, 2.811e-14, 2.319e-14, 1.948e-14,  
 02658 1.66e-14, 1.432e-14, 1.251e-14, 1.109e-14, 1.006e-14, 9.45e-15,  
 02659 9.384e-15, 1.012e-14, 1.216e-14, 1.636e-14, 2.305e-14, 3.488e-14,  
 02660 5.572e-14, 8.479e-14, 1.265e-13, 1.905e-13, 2.73e-13, 3.809e-13,  
 02661 4.955e-13, 6.303e-13, 7.861e-13, 9.427e-13, 1.097e-12, 1.212e-12,  
 02662 1.328e-12, 1.415e-12, 1.463e-12, 1.495e-12, 1.571e-12, 1.731e-12,  
 02663 1.981e-12, 2.387e-12, 2.93e-12, 3.642e-12, 4.584e-12, 5.822e-12,

```
02664    7.278e-12, 9.193e-12, 1.135e-11, 1.382e-11, 1.662e-11, 1.958e-11,
02665    2.286e-11, 2.559e-11, 2.805e-11, 2.988e-11, 3.106e-11, 3.182e-11,
02666    3.2e-11, 3.258e-11, 3.362e-11, 3.558e-11, 3.688e-11, 3.8e-11,
02667    3.929e-11, 4.062e-11, 4.186e-11, 4.293e-11, 4.48e-11, 4.643e-11,
02668    4.704e-11, 4.571e-11, 4.206e-11, 3.715e-11, 3.131e-11, 2.541e-11,
02669    1.978e-11, 1.508e-11, 1.146e-11, 8.7e-12, 6.603e-12, 5.162e-12,
02670    4.157e-12, 3.408e-12, 2.829e-12, 2.405e-12, 2.071e-12, 1.826e-12,
02671    1.648e-12, 1.542e-12, 1.489e-12, 1.485e-12, 1.493e-12, 1.545e-12,
02672    1.637e-12, 1.814e-12, 2.061e-12, 2.312e-12, 2.651e-12, 3.03e-12,
02673    3.46e-12, 3.901e-12, 4.306e-12, 4.721e-12, 5.008e-12, 5.281e-12,
02674    5.541e-12, 5.791e-12, 6.115e-12, 6.442e-12, 6.68e-12, 6.791e-12,
02675    6.831e-12, 6.839e-12, 6.946e-12, 7.128e-12, 7.537e-12, 8.036e-12,
02676    8.392e-12, 8.526e-12, 8.11e-12, 7.325e-12, 6.329e-12, 5.183e-12,
02677    4.081e-12, 2.985e-12, 2.141e-12, 1.492e-12, 1.015e-12, 6.684e-13,
02678    4.414e-13, 2.987e-13, 2.038e-13, 1.391e-13, 9.86e-14, 7.24e-14,
02679    5.493e-14, 4.288e-14, 3.427e-14, 2.787e-14, 2.296e-14, 1.909e-14,
02680    1.598e-14, 1.344e-14, 1.135e-14, 9.616e-15, 8.169e-15, 6.957e-15,
02681    5.938e-15, 5.08e-15, 4.353e-15, 3.738e-15, 3.217e-15, 2.773e-15,
02682    2.397e-15, 2.077e-15, 1.805e-15, 1.575e-15, 1.382e-15, 1.221e-15,
02683    1.09e-15, 9.855e-16, 9.068e-16, 8.537e-16, 8.27e-16, 8.29e-16,
02684    8.634e-16, 9.359e-16, 1.055e-15, 1.233e-15, 1.486e-15, 1.839e-15,
02685    2.326e-15, 2.998e-15, 3.934e-15, 5.256e-15, 7.164e-15, 9.984e-15,
02686    1.427e-14, 2.099e-14, 3.196e-14, 5.121e-14, 7.908e-14, 1.131e-13,
02687    1.602e-13, 2.239e-13, 3.075e-13, 4.134e-13, 5.749e-13, 7.886e-13,
02688    1.071e-12, 1.464e-12, 2.032e-12, 2.8e-12, 3.732e-12, 4.996e-12,
02689    6.483e-12, 8.143e-12, 1.006e-11, 1.238e-11, 1.484e-11, 1.744e-11,
02690    2.02e-11, 2.274e-11, 2.562e-11, 2.848e-11, 3.191e-11, 3.617e-11,
02691    4.081e-11, 4.577e-11, 4.937e-11, 5.204e-11, 5.401e-11, 5.462e-11,
02692    5.507e-11, 5.51e-11, 5.605e-11, 5.686e-11, 5.739e-11, 5.766e-11,
02693    5.74e-11, 5.754e-11, 5.761e-11, 5.777e-11, 5.712e-11, 5.51e-11,
02694    5.088e-11, 4.438e-11, 3.728e-11, 2.994e-11, 2.305e-11, 1.715e-11,
02695    1.256e-11, 9.208e-12, 6.745e-12, 5.014e-12, 3.785e-12, 2.9e-12,
02696    2.239e-12, 1.757e-12, 1.414e-12, 1.142e-12, 9.482e-13, 8.01e-13,
02697    6.961e-13, 6.253e-13, 5.735e-13, 5.433e-13, 5.352e-13, 5.493e-13,
02698    5.706e-13, 6.068e-13, 6.531e-13, 7.109e-13, 7.767e-13, 8.59e-13,
02699    9.792e-13, 1.142e-12, 1.371e-12, 1.65e-12, 1.957e-12, 2.302e-12,
02700    2.705e-12, 3.145e-12, 3.608e-12, 4.071e-12, 4.602e-12, 5.133e-12,
02701    5.572e-12, 5.987e-12, 6.248e-12, 6.533e-12, 6.757e-12, 6.935e-12,
02702    7.224e-12, 7.422e-12, 7.538e-12, 7.547e-12, 7.495e-12, 7.543e-12,
02703    7.725e-12, 8.139e-12, 8.627e-12, 9.146e-12, 9.443e-12, 9.318e-12,
02704    8.649e-12, 7.512e-12, 6.261e-12, 4.915e-12, 3.647e-12, 2.597e-12,
02705    1.785e-12, 1.242e-12, 8.66e-13, 6.207e-13, 4.61e-13, 3.444e-13,
02706    2.634e-13, 2.1e-13, 1.725e-13, 1.455e-13, 1.237e-13, 1.085e-13,
02707    9.513e-14, 7.978e-14, 6.603e-14, 5.288e-14, 4.084e-14, 2.952e-14,
02708    2.157e-14, 1.593e-14, 1.199e-14, 9.267e-15, 7.365e-15, 6.004e-15,
02709    4.995e-15, 4.218e-15, 3.601e-15, 3.101e-15, 2.692e-15, 2.36e-15,
02710    2.094e-15, 1.891e-15, 1.755e-15, 1.699e-15, 1.755e-15, 1.987e-15,
02711    2.506e-15, 3.506e-15, 5.289e-15, 8.311e-15, 1.325e-14, 2.129e-14,
02712    3.237e-14, 4.595e-14, 6.441e-14, 8.433e-14, 1.074e-13, 1.383e-13,
02713    1.762e-13, 2.281e-13, 2.831e-13, 3.523e-13, 4.38e-13, 5.304e-13,
02714    6.29e-13, 7.142e-13, 8.032e-13, 8.934e-13, 9.888e-13, 1.109e-12,
02715    1.261e-12, 1.462e-12, 1.74e-12, 2.099e-12, 2.535e-12, 3.008e-12,
02716    3.462e-12, 3.856e-12, 4.098e-12, 4.239e-12, 4.234e-12, 4.132e-12,
02717    3.986e-12, 3.866e-12, 3.829e-12, 3.742e-12, 3.705e-12, 3.694e-12,
02718    3.765e-12, 3.849e-12, 3.929e-12, 4.056e-12, 4.092e-12, 4.047e-12,
02719    3.792e-12, 3.407e-12, 2.953e-12, 2.429e-12, 1.931e-12, 1.46e-12,
02720    1.099e-12, 8.199e-13, 6.077e-13, 4.449e-13, 3.359e-13, 2.524e-13,
02721    1.881e-13, 1.391e-13, 1.02e-13, 7.544e-14, 5.555e-14, 4.22e-14,
02722    3.321e-14, 2.686e-14, 2.212e-14, 1.78e-14, 1.369e-14, 1.094e-14,
02723    9.13e-15, 8.101e-15, 7.828e-15, 8.393e-15, 1.012e-14, 1.259e-14,
02724    1.538e-14, 1.961e-14, 2.619e-14, 3.679e-14, 5.049e-14, 6.917e-14,
02725    8.88e-14, 1.115e-13, 1.373e-13, 1.619e-13, 1.878e-13, 2.111e-13,
02726    2.33e-13, 2.503e-13, 2.613e-13, 2.743e-13, 2.826e-13, 2.976e-13,
02727    3.162e-13, 3.36e-13, 3.491e-13, 3.541e-13, 3.595e-13, 3.608e-13,
02728    3.709e-13, 3.869e-13, 4.12e-13, 4.366e-13, 4.504e-13, 4.379e-13,
02729    3.955e-13, 3.385e-13, 2.741e-13, 2.089e-13, 1.427e-13, 9.294e-14,
02730    5.775e-14, 3.565e-14, 2.21e-14, 1.398e-14, 9.194e-15, 6.363e-15,
02731    4.644e-15, 3.55e-15, 2.808e-15, 2.274e-15, 1.871e-15, 1.557e-15,
02732    1.308e-15, 1.108e-15, 9.488e-16, 8.222e-16, 7.238e-16, 6.506e-16,
02733    6.008e-16, 5.742e-16, 5.724e-16, 5.991e-16, 6.625e-16, 7.775e-16,
02734    9.734e-16, 1.306e-15, 1.88e-15, 2.879e-15, 4.616e-15, 7.579e-15,
02735    1.248e-14, 2.03e-14, 3.244e-14, 5.171e-14, 7.394e-14, 9.676e-14,
02736    1.199e-13, 1.467e-13, 1.737e-13, 2.02e-13, 2.425e-13, 3.016e-13,
02737    3.7e-13, 4.617e-13, 5.949e-13, 7.473e-13, 9.378e-13, 1.191e-12,
02738    1.481e-12, 1.813e-12, 2.232e-12, 2.722e-12, 3.254e-12, 3.845e-12,
02739    4.458e-12, 5.048e-12, 5.511e-12, 5.898e-12, 6.204e-12, 6.293e-12,
02740    6.386e-12, 6.467e-12, 6.507e-12, 6.466e-12, 6.443e-12, 6.598e-12,
02741    6.873e-12, 7.3e-12, 7.816e-12, 8.368e-12, 8.643e-12, 8.466e-12,
02742    7.871e-12, 6.853e-12, 5.714e-12, 4.482e-12, 3.392e-12, 2.613e-12,
02743    2.008e-12, 1.562e-12, 1.228e-12, 9.888e-13, 7.646e-13, 5.769e-13,
02744    4.368e-13, 3.324e-13, 2.508e-13, 1.916e-13
02745    };
02746
02747    static double xfcrev[15] =
02748    { 1.003, 1.009, 1.015, 1.023, 1.029, 1.033, 1.037,
02749      1.039, 1.04, 1.046, 1.036, 1.027, 1.01, 1.002, 1.
02750    };
```



```

02751
02752 double a1, a2, a3, dw, ew, dx, xw, xx, vf2, vf6, cw260, cw296,
02753 sfac, fscal, cwfrn, ctmph, ctwfrn, ctwsf;
02754
02755 int iw, ix;
02756
02757 /* Get H2O continuum absorption... */
02758 xw = nu / 10 + 1;
02759 if (xw >= 1 && xw < 2001) {
02760     iw = (int) xw;
02761     dw = xw - iw;
02762     ew = 1 - dw;
02763     cw296 = ew * h2o296[iw - 1] + dw * h2o296[iw];
02764     cw260 = ew * h2o260[iw - 1] + dw * h2o260[iw];
02765     cwfrn = ew * h2ofrn[iw - 1] + dw * h2ofrn[iw];
02766     if (nu <= 820 || nu >= 960) {
02767         sfac = 1;
02768     } else {
02769         xx = (nu - 820) / 10;
02770         ix = (int) xx;
02771         dx = xx - ix;
02772         sfac = (1 - dx) * xfcrev[ix] + dx * xfcrev[ix + 1];
02773     }
02774     ctwsf = sfac * cw296 * pow(cw260 / cw296, (296 - t) / (296 - 260));
02775     vf2 = POW2(nu - 370);
02776     vf6 = POW3(vf2);
02777     fscal = 36100 / (vf2 + vf6 * 1e-8 + 36100) * -.25 + 1;
02778     ctwfrn = cwfrn * fscal;
02779     a1 = nu * u * tanh(.7193876 / t * nu);
02780     a2 = 296 / t;
02781     a3 = p / P0 * (q * ctwsf + (1 - q) * ctwfrn) * 1e-20;
02782     ctmph = a1 * a2 * a3;
02783 } else
02784     ctmph = 0;
02785 return ctmph;
02786 }

```

### 5.23.2.8 double ctmn2 ( double nu, double p, double t )

Compute nitrogen continuum (absorption coefficient).

Definition at line 2790 of file [jurassic.c](#).

```

02793     {
02794
02795 static double ba[98] = { 0., 4.45e-8, 5.22e-8, 6.46e-8, 7.75e-8, 9.03e-8,
02796 1.06e-7, 1.21e-7, 1.37e-7, 1.57e-7, 1.75e-7, 2.01e-7, 2.3e-7,
02797 2.59e-7, 2.95e-7, 3.26e-7, 3.66e-7, 4.05e-7, 4.47e-7, 4.92e-7,
02798 5.34e-7, 5.84e-7, 6.24e-7, 6.67e-7, 7.14e-7, 7.26e-7, 7.54e-7,
02799 7.84e-7, 8.09e-7, 8.42e-7, 8.62e-7, 8.87e-7, 9.11e-7, 9.36e-7,
02800 9.76e-7, 1.03e-6, 1.11e-6, 1.23e-6, 1.39e-6, 1.61e-6, 1.76e-6,
02801 1.94e-6, 1.97e-6, 1.87e-6, 1.75e-6, 1.56e-6, 1.42e-6, 1.35e-6,
02802 1.32e-6, 1.29e-6, 1.29e-6, 1.29e-6, 1.3e-6, 1.32e-6, 1.33e-6,
02803 1.34e-6, 1.35e-6, 1.33e-6, 1.31e-6, 1.29e-6, 1.24e-6, 1.2e-6,
02804 1.16e-6, 1.1e-6, 1.04e-6, 9.96e-7, 9.38e-7, 8.63e-7, 7.98e-7,
02805 7.26e-7, 6.55e-7, 5.94e-7, 5.35e-7, 4.74e-7, 4.24e-7, 3.77e-7,
02806 3.33e-7, 2.96e-7, 2.63e-7, 2.34e-7, 2.08e-7, 1.85e-7, 1.67e-7,
02807 1.47e-7, 1.32e-7, 1.2e-7, 1.09e-7, 9.85e-8, 9.08e-8, 8.18e-8,
02808 7.56e-8, 6.85e-8, 6.14e-8, 5.83e-8, 5.77e-8, 5e-8, 4.32e-8, 0.
02809 };
02810
02811 static double betaa[98] = { 802., 802., 761., 722., 679., 646., 609., 562.,
02812 511., 472., 436., 406., 377., 355., 338., 319., 299., 278., 255.,
02813 233., 208., 184., 149., 107., 66., 25., -13., -49., -82., -104.,
02814 -119., -130., -139., -144., -146., -146., -147., -148., -150.,
02815 -153., -160., -169., -181., -189., -195., -200., -205., -209.,
02816 -211., -210., -210., -209., -205., -199., -190., -180., -168.,
02817 -157., -143., -126., -108., -89., -63., -32., 1., 35., 65., 95.,
02818 121., 141., 152., 161., 164., 164., 155., 148., 143., 137.,
02819 133., 131., 133., 139., 150., 165., 187., 213., 248., 284., 321.,
02820 372., 449., 514., 569., 609., 642., 673., 673.
02821 };
02822
02823 static double nua[98] = { 2120., 2125., 2130., 2135., 2140., 2145., 2150.,
02824 2155., 2160., 2165., 2170., 2175., 2180., 2185., 2190., 2195.,
02825 2200., 2205., 2210., 2215., 2220., 2225., 2230., 2235., 2240.,
02826 2245., 2250., 2255., 2260., 2265., 2270., 2275., 2280., 2285.,
02827 2290., 2295., 2300., 2305., 2310., 2315., 2320., 2325., 2330.,
02828 2335., 2340., 2345., 2350., 2355., 2360., 2365., 2370., 2375.,
02829 2380., 2385., 2390., 2395., 2400., 2405., 2410., 2415., 2420.,

```

```

02830     2425., 2430., 2435., 2440., 2445., 2450., 2455., 2460., 2465.,
02831     2470., 2475., 2480., 2485., 2490., 2495., 2500., 2505., 2510.,
02832     2515., 2520., 2525., 2530., 2535., 2540., 2545., 2550., 2555.,
02833     2560., 2565., 2570., 2575., 2580., 2585., 2590., 2595., 2600., 2605.
02834 };
02835
02836 double b, beta, q_n2 = 0.79, t0 = 273, tr = 296;
02837
02838 int idx;
02839
02840 /* Check wavenumber range... */
02841 if (nu < nua[0] || nu > nua[97])
02842     return 0;
02843
02844 /* Interpolate B and beta... */
02845 idx = locate_reg(nua, 98, nu);
02846 b = LIN(nua[idx], ba[idx], nua[idx + 1], ba[idx + 1], nu);
02847 beta = LIN(nua[idx], betaa[idx], nua[idx + 1], betaa[idx + 1], nu);
02848
02849 /* Compute absorption coefficient... */
02850 return 0.1 * POW2(p / P0 * t0 / t) * exp(beta * (1 / tr - 1 / t))
02851     * q_n2 * b * (q_n2 + (1 - q_n2) * (1.294 - 0.4545 * t / tr));
02852 }

```

Here is the call graph for this function:



#### 5.23.2.9 double ctmo2 ( double nu, double p, double t )

Compute oxygen continuum (absorption coefficient).

Definition at line 2856 of file [jurassic.c](#).

```

02859     {
02860
02861     static double ba[90] = { 0., .061, .074, .084, .096, .12, .162, .208, .246,
02862     .285, .314, .38, .444, .5, .571, .673, .768, .853, .966, 1.097,
02863     1.214, 1.333, 1.466, 1.591, 1.693, 1.796, 1.922, 2.037, 2.154,
02864     2.264, 2.375, 2.508, 2.671, 2.847, 3.066, 3.417, 3.828, 4.204,
02865     4.453, 4.599, 4.528, 4.284, 3.955, 3.678, 3.477, 3.346, 3.29,
02866     3.251, 3.231, 3.226, 3.212, 3.192, 3.108, 3.033, 2.911, 2.798,
02867     2.646, 2.508, 2.322, 2.13, 1.928, 1.757, 1.588, 1.417, 1.253,
02868     1.109, .99, .888, .791, .678, .587, .524, .464, .403, .357, .32,
02869     .29, .267, .242, .215, .182, .16, .146, .128, .103, .087, .081,
02870     .071, .064, 0.
02871 };
02872
02873     static double betaa[90] = { 467., 467., 400., 315., 379., 368., 475., 521.,
02874     531., 512., 442., 444., 430., 381., 335., 324., 296., 248., 215.,
02875     193., 158., 127., 101., 71., 31., -6., -26., -47., -63., -79.,
02876     -88., -88., -87., -90., -98., -99., -109., -134., -160., -167.,
02877     -164., -158., -153., -151., -156., -166., -168., -173., -170.,
02878     -161., -145., -126., -108., -84., -59., -29., 4., 41., 73., 97.,
02879     123., 159., 198., 220., 242., 256., 281., 311., 334., 319., 313.,
02880     321., 323., 310., 315., 320., 335., 361., 378., 373., 338., 319.,
02881     346., 322., 291., 290., 350., 371., 504., 504.
02882 };
02883
02884     static double nua[90] = { 1360., 1365., 1370., 1375., 1380., 1385., 1390.,
02885     1395., 1400., 1405., 1410., 1415., 1420., 1425., 1430., 1435.,
02886     1440., 1445., 1450., 1455., 1460., 1465., 1470., 1475., 1480.,
02887     1485., 1490., 1495., 1500., 1505., 1510., 1515., 1520., 1525.,
02888     1530., 1535., 1540., 1545., 1550., 1555., 1560., 1565., 1570.,

```

```

02889      1575., 1580., 1585., 1590., 1595., 1600., 1605., 1610., 1615.,
02890      1620., 1625., 1630., 1635., 1640., 1645., 1650., 1655., 1660.,
02891      1665., 1670., 1675., 1680., 1685., 1690., 1695., 1700., 1705.,
02892      1710., 1715., 1720., 1725., 1730., 1735., 1740., 1745., 1750.,
02893      1755., 1760., 1765., 1770., 1775., 1780., 1785., 1790., 1795.,
02894      1800., 1805.
02895  };
02896
02897  double b, beta, q_o2 = 0.21, t0 = 273, tr = 296;
02898
02899  int idx;
02900
02901  /* Check wavenumber range... */
02902  if (nu < nua[0] || nu > nua[89])
02903      return 0;
02904
02905  /* Interpolate B and beta... */
02906  idx = locate_reg(nua, 90, nu);
02907  b = LIN(nua[idx], ba[idx], nua[idx + 1], ba[idx + 1], nu);
02908  beta = LIN(nua[idx], betaa[idx], nua[idx + 1], betaa[idx + 1], nu);
02909
02910  /* Compute absorption coefficient... */
02911  return 0.1 * POW2(p / P0 * t0 / t) * exp(beta * (1 / tr - 1 / t)) * q_o2 *
02912      b;
02913 }

```

Here is the call graph for this function:



#### 5.23.2.10 void copy\_atm ( ctl\_t \*ctl, atm\_t \*atm\_dest, atm\_t \*atm\_src, int init )

Copy and initialize atmospheric data.

Definition at line 2917 of file [jurassic.c](#).

```

02921      {
02922
02923      int ig, ip, iw;
02924
02925      size_t s;
02926
02927      /* Data size... */
02928      s = (size_t) atm_src->np * sizeof(double);
02929
02930      /* Copy data... */
02931      atm_dest->np = atm_src->np;
02932      memcpy(atm_dest->time, atm_src->time, s);
02933      memcpy(atm_dest->z, atm_src->z, s);
02934      memcpy(atm_dest->lon, atm_src->lon, s);
02935      memcpy(atm_dest->lat, atm_src->lat, s);
02936      memcpy(atm_dest->p, atm_src->p, s);
02937      memcpy(atm_dest->t, atm_src->t, s);
02938      for (ig = 0; ig < ctl->ng; ig++)
02939          memcpy(atm_dest->q[ig], atm_src->q[ig], s);
02940      for (iw = 0; iw < ctl->nw; iw++)
02941          memcpy(atm_dest->k[iw], atm_src->k[iw], s);
02942
02943      /* Initialize... */
02944      if (init)
02945          for (ip = 0; ip < atm_dest->np; ip++) {
02946              atm_dest->p[ip] = 0;
02947              atm_dest->t[ip] = 0;
02948              for (ig = 0; ig < ctl->ng; ig++)
02949                  atm_dest->q[ig][ip] = 0;
02950              for (iw = 0; iw < ctl->nw; iw++)
02951                  atm_dest->k[iw][ip] = 0;
02952          }
02953 }

```

#### 5.23.2.11 void copy\_obs ( ctl\_t \* *ctl*, obs\_t \* *obs\_dest*, obs\_t \* *obs\_src*, int *init* )

Copy and initialize observation data.

Definition at line 2957 of file [jurassic.c](#).

```

02961         {
02962
02963     int id, ir;
02964
02965     size_t s;
02966
02967     /* Data size... */
02968     s = (size_t) obs_src->nr * sizeof(double);
02969
02970     /* Copy data... */
02971     obs_dest->nr = obs_src->nr;
02972     memcpy(obs_dest->time, obs_src->time, s);
02973     memcpy(obs_dest->obsz, obs_src->obsz, s);
02974     memcpy(obs_dest->obslon, obs_src->obslon, s);
02975     memcpy(obs_dest->obslat, obs_src->obslat, s);
02976     memcpy(obs_dest->vpz, obs_src->vpz, s);
02977     memcpy(obs_dest->vplon, obs_src->vplon, s);
02978     memcpy(obs_dest->vplat, obs_src->vplat, s);
02979     memcpy(obs_dest->tpz, obs_src->tpz, s);
02980     memcpy(obs_dest->tplon, obs_src->tplon, s);
02981     memcpy(obs_dest->tplat, obs_src->tplat, s);
02982     for (id = 0; id < ctl->nd; id++)
02983         memcpy(obs_dest->rad[id], obs_src->rad[id], s);
02984     for (id = 0; id < ctl->nd; id++)
02985         memcpy(obs_dest->tau[id], obs_src->tau[id], s);
02986
02987     /* Initialize... */
02988     if (init)
02989         for (id = 0; id < ctl->nd; id++)
02990             for (ir = 0; ir < obs_dest->nr; ir++)
02991                 if (gsl_finite(obs_dest->rad[id][ir])) {
02992                     obs_dest->rad[id][ir] = 0;
02993                     obs_dest->tau[id][ir] = 0;
02994                 }
02995 }
```

#### 5.23.2.12 int find\_emitter ( ctl\_t \* *ctl*, const char \* *emitter* )

Find index of an emitter.

Definition at line 2999 of file [jurassic.c](#).

```

03001         {
03002
03003     int ig;
03004
03005     for (ig = 0; ig < ctl->ng; ig++)
03006         if (strcasecmp(ctl->emitter[ig], emitter) == 0)
03007             return ig;
03008
03009     return -1;
03010 }
```

#### 5.23.2.13 void formod ( ctl\_t \* *ctl*, atm\_t \* *atm*, obs\_t \* *obs* )

Determine ray paths and compute radiative transfer.

Definition at line 3014 of file [jurassic.c](#).



5.23.2.14 void formod\_continua ( ctl\_t \* *ctl*, los\_t \* *los*, int *ip*, double \* *beta* )

Compute absorption coefficient of continua.

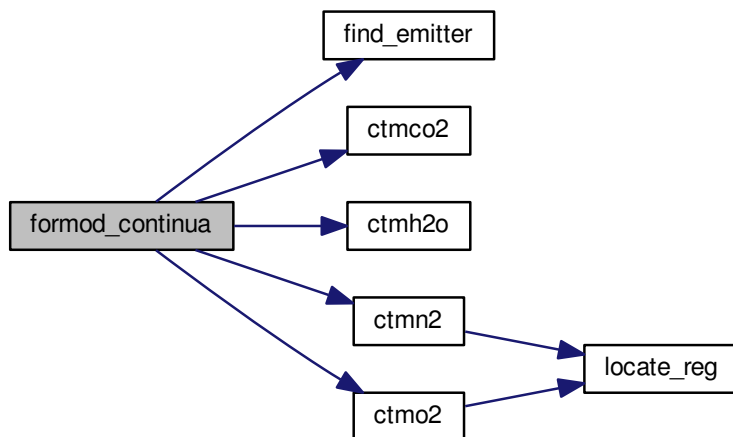
Definition at line 3058 of file [jurassic.c](#).

```

03062     {
03063
03064     static int ig_co2 = -999, ig_h2o = -999;
03065
03066     int id;
03067
03068     /* Extinction... */
03069     for (id = 0; id < ctl->nd; id++)
03070         beta[id] = los->k[ctl->window[id]][ip];
03071
03072     /* CO2 continuum... */
03073     if (ctl->ctm_co2) {
03074         if (ig_co2 == -999)
03075             ig_co2 = find_emitter(ctl, "CO2");
03076         if (ig_co2 >= 0)
03077             for (id = 0; id < ctl->nd; id++)
03078                 beta[id] += ctmco2(ctl->nu[id], los->p[ip], los->t[ip],
03079                                     los->u[ig_co2][ip]) / los->ds[ip];
03080     }
03081
03082     /* H2O continuum... */
03083     if (ctl->ctm_h2o) {
03084         if (ig_h2o == -999)
03085             ig_h2o = find_emitter(ctl, "H2O");
03086         if (ig_h2o >= 0)
03087             for (id = 0; id < ctl->nd; id++)
03088                 beta[id] += ctmh2o(ctl->nu[id], los->p[ip], los->t[ip],
03089                                     los->q[ig_h2o][ip],
03090                                     los->u[ig_h2o][ip]) / los->ds[ip];
03091     }
03092
03093     /* N2 continuum... */
03094     if (ctl->ctm_n2)
03095         for (id = 0; id < ctl->nd; id++)
03096             beta[id] += ctmn2(ctl->nu[id], los->p[ip], los->t[ip]);
03097
03098     /* O2 continuum... */
03099     if (ctl->ctm_o2)
03100         for (id = 0; id < ctl->nd; id++)
03101             beta[id] += ctmo2(ctl->nu[id], los->p[ip], los->t[ip]);
03102 }

```

Here is the call graph for this function:



### 5.23.2.15 void formod\_fov ( ctl\_t \*ctl, obs\_t \*obs )

Apply field of view convolution.

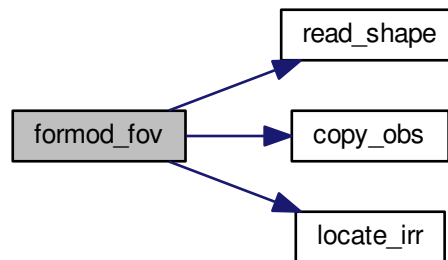
Definition at line 3106 of file [jurassic.c](#).

```

03108         {
03109
03110     static double dz[NSHAPE], w[NSHAPE];
03111
03112     static int init = 0, n;
03113
03114     obs_t *obs2;
03115
03116     double rad[ND][NR], tau[ND][NR], wsum, z[NR], zfov;
03117
03118     int i, id, idx, ir, ir2, nz;
03119
03120     /* Do not take into account FOV... */
03121     if (ctl->fov[0] == '-')
03122         return;
03123
03124     /* Initialize FOV data... */
03125     if (!init) {
03126         init = 1;
03127         read_shape(ctl->fov, dz, w, &n);
03128     }
03129
03130     /* Allocate... */
03131     ALLOC(obs2, obs_t, 1);
03132
03133     /* Copy observation data... */
03134     copy_obs(ctl, obs2, obs, 0);
03135
03136     /* Loop over ray paths... */
03137     for (ir = 0; ir < obs->nr; ir++) {
03138
03139         /* Get radiance and transmittance profiles... */
03140         nz = 0;
03141         for (ir2 = GSL_MAX(ir - NFOV, 0); ir2 < GSL_MIN(ir + 1 + NFOV, obs->nr);
03142             ir2++)
03143             if (obs->time[ir2] == obs->time[ir]) {
03144                 z[nz] = obs2->vpz[ir2];
03145                 for (id = 0; id < ctl->nd; id++) {
03146                     rad[id][nz] = obs2->rad[id][ir2];
03147                     tau[id][nz] = obs2->tau[id][ir2];
03148                 }
03149                 nz++;
03150             }
03151         if (nz < 2)
03152             ERRMSG("Cannot apply FOV convolution!");
03153
03154         /* Convolute profiles with FOV... */
03155         wsum = 0;
03156         for (id = 0; id < ctl->nd; id++) {
03157             obs->rad[id][ir] = 0;
03158             obs->tau[id][ir] = 0;
03159         }
03160         for (i = 0; i < n; i++) {
03161             zfov = obs->vpz[ir] + dz[i];
03162             idx = locate_irr(z, nz, zfov);
03163             for (id = 0; id < ctl->nd; id++) {
03164                 obs->rad[id][ir] += w[i]
03165                     * LIN(z[idx], rad[id][idx], z[idx + 1], rad[id][idx + 1], zfov);
03166                 obs->tau[id][ir] += w[i]
03167                     * LIN(z[idx], tau[id][idx], z[idx + 1], tau[id][idx + 1], zfov);
03168             }
03169             wsum += w[i];
03170         }
03171         for (id = 0; id < ctl->nd; id++) {
03172             obs->rad[id][ir] /= wsum;
03173             obs->tau[id][ir] /= wsum;
03174         }
03175     }
03176
03177     /* Free... */
03178     free(obs2);
03179 }

```

Here is the call graph for this function:



#### 5.23.2.16 void formod\_pencil ( ctl\_t \* *ctl*, atm\_t \* *atm*, obs\_t \* *obs*, int *ir* )

Compute radiative transfer for a pencil beam.

Definition at line 3183 of file [jurassic.c](#).

```

03187     {
03188
03189     static tbl_t *tbl;
03190
03191     static int init = 0;
03192
03193     los_t *los;
03194
03195     double beta_ctm[ND], eps, src_planck[ND], tau_path[NG][ND], tau_gas[ND];
03196
03197     int id, ip;
03198
03199     /* Initialize look-up tables... */
03200     if (!init) {
03201         init = 1;
03202         ALLOC(tbl, tbl_t, 1);
03203         init_tbl(ctl, tbl);
03204     }
03205
03206     /* Allocate... */
03207     ALLOC(los, los_t, 1);
03208
03209     /* Initialize... */
03210     for (id = 0; id < ctl->nd; id++) {
03211         obs->rad[id][ir] = 0;
03212         obs->tau[id][ir] = 1;
03213     }
03214
03215     /* Raytracing... */
03216     raytrace(ctl, atm, obs, los, ir);
03217
03218     /* Loop over LOS points... */
03219     for (ip = 0; ip < los->np; ip++) {
03220
03221         /* Get trace gas transmittance... */
03222         intpol_tbl(ctl, tbl, los, ip, tau_path, tau_gas);
03223
03224         /* Get continuum absorption... */
03225         formod_continua(ctl, los, ip, beta_ctm);
03226
03227         /* Compute Planck function... */
03228         formod_srcfunc(ctl, tbl, los->t[ip], src_planck);
03229
03230         /* Loop over channels... */
03231         for (id = 0; id < ctl->nd; id++)
03232             if (tau_gas[id] > 0) {
03233

```

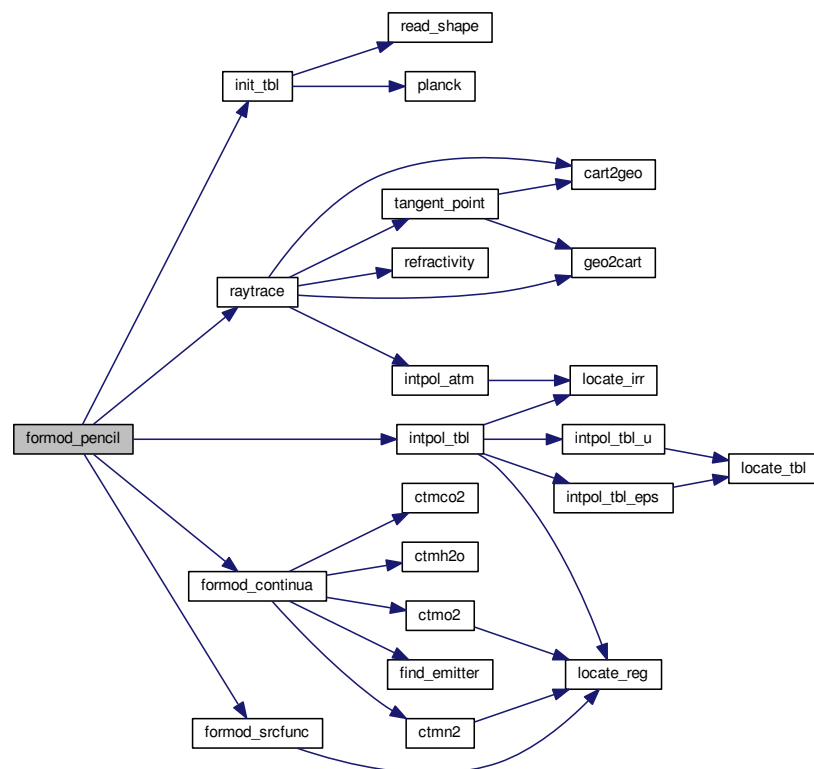


```

03234      /* Get segment emissivity... */
03235      eps = 1 - tau_gas[id] * exp(-beta_ctm[id] * los->ds[ip]);
03236
03237      /* Compute radiance... */
03238      obs->rad[id][ir] += src_planck[id] * eps * obs->tau[id][ir];
03239
03240      /* Compute path transmittance... */
03241      obs->tau[id][ir] *= (1 - eps);
03242  }
03243  }
03244
03245  /* Add surface... */
03246  if (los->tsurf > 0) {
03247      formod_srcfunc(ctl, tbl, los->tsurf, src_planck);
03248      for (id = 0; id < ctl->nd; id++)
03249          obs->rad[id][ir] += src_planck[id] * obs->tau[id][ir];
03250  }
03251
03252  /* Free... */
03253  free(los);
03254  }

```

Here is the call graph for this function:



**5.23.2.17** void formod\_srcfunc ( ctl\_t \* *ctl*, tbl\_t \* *tbl*, double *t*, double \* *src* )

Compute Planck source function.

Definition at line 3258 of file [jurassic.c](#).

```

03262      {
03263
03264      int id, it;

```

```

03265
03266  /* Determine index in temperature array... */
03267  it = locate_reg(tbl->st, TBLNS, t);
03268
03269  /* Interpolate Planck function value... */
03270  for (id = 0; id < ctl->nd; id++)
03271      src[id] = LIN(tbl->st[it], tbl->sr[id][it],
03272                  tbl->st[it + 1], tbl->sr[id][it + 1], t);
03273 }

```

Here is the call graph for this function:



#### 5.23.2.18 void geo2cart ( double z, double lon, double lat, double \* x )

Convert geolocation to Cartesian coordinates.

Definition at line 3277 of file [jurassic.c](#).

```

03281      {
03282
03283      double radius;
03284
03285      radius = z + RE;
03286      x[0] = radius * cos(lat / 180 * M_PI) * cos(lon / 180 * M_PI);
03287      x[1] = radius * cos(lat / 180 * M_PI) * sin(lon / 180 * M_PI);
03288      x[2] = radius * sin(lat / 180 * M_PI);
03289 }

```

#### 5.23.2.19 void hydrostatic ( ctl\_t \* ctl, atm\_t \* atm )

Set hydrostatic equilibrium.

Definition at line 3293 of file [jurassic.c](#).

```

03295      {
03296
03297      static int ig_h2o = -999;
03298
03299      double dzmin = 1e99, e = 0, mean, mmair = 28.96456e-3, mmh2o = 18.0153e-3;
03300
03301      int i, ip, ipref = 0, ipt = 20;
03302
03303      /* Check reference height... */
03304      if (ctl->hydz < 0)
03305          return;
03306
03307      /* Determine emitter index of H2O... */
03308      if (ig_h2o == -999)
03309          ig_h2o = find_emitter(ctl, "H2O");
03310
03311      /* Find air parcel next to reference height... */
03312      for (ip = 0; ip < atm->np; ip++)
03313          if (fabs(atm->z[ip] - ctl->hydz) < dzmin) {
03314              dzmin = fabs(atm->z[ip] - ctl->hydz);
03315              ipref = ip;
03316          }

```

```

03317
03318 /* Upper part of profile... */
03319 for (ip = ipref + 1; ip < atm->np; ip++) {
03320     mean = 0;
03321     for (i = 0; i < ipt; i++) {
03322         if (ig_h2o >= 0)
03323             e = LIN(0.0, atm->q[ig_h2o][ip - 1],
03324                     ipt - 1.0, atm->q[ig_h2o][ip], (double) i);
03325         mean += (e * mmh2o + (1 - e) * mmair)
03326             * G0 / RI
03327             / LIN(0.0, atm->t[ip - 1], ipt - 1.0, atm->t[ip], (double) i) / ipt;
03328     }
03329
03330 /* Compute p(z,T)... */
03331 atm->p[ip] =
03332     exp(log(atm->p[ip - 1]) - mean * 1000 * (atm->z[ip] - atm->z[ip - 1]));
03333 }
03334
03335 /* Lower part of profile... */
03336 for (ip = ipref - 1; ip >= 0; ip--) {
03337     mean = 0;
03338     for (i = 0; i < ipt; i++) {
03339         if (ig_h2o >= 0)
03340             e = LIN(0.0, atm->q[ig_h2o][ip + 1],
03341                     ipt - 1.0, atm->q[ig_h2o][ip], (double) i);
03342         mean += (e * mmh2o + (1 - e) * mmair)
03343             * G0 / RI
03344             / LIN(0.0, atm->t[ip + 1], ipt - 1.0, atm->t[ip], (double) i) / ipt;
03345     }
03346
03347 /* Compute p(z,T)... */
03348 atm->p[ip] =
03349     exp(log(atm->p[ip + 1]) - mean * 1000 * (atm->z[ip] - atm->z[ip + 1]));
03350 }
03351 }

```

Here is the call graph for this function:



#### 5.23.2.20 void idx2name( ctl\_t \*ctl, int idx, char \*quantity )

Determine name of state vector quantity for given index.

Definition at line 3355 of file [jurassic.c](#).

```

03358     {
03359         int ig, iw;
03360
03361         if (idx == IDXP)
03362             sprintf(quantity, "PRESSURE");
03363
03364         if (idx == IDXT)
03365             sprintf(quantity, "TEMPERATURE");
03366
03367         for (ig = 0; ig < ctl->ng; ig++)
03368             if (idx == IDXQ(ig))
03369                 sprintf(quantity, "%s", ctl->emitter[ig]);
03370
03371         for (iw = 0; iw < ctl->nw; iw++)
03372             if (idx == IDXK(iw))
03373                 sprintf(quantity, "EXTINCT_WINDOW%d", iw);
03374     }
03375 }

```

## 5.23.2.21 void init\_tbl ( ctl\_t \* ctl, tbl\_t \* tbl )

Initialize look-up tables.

Definition at line 3379 of file [jurassic.c](#).

```

03381         {
03382
03383     FILE *in;
03384
03385     char filename[2 * LEN], line[LEN];
03386
03387     double eps, eps_old, press, press_old, temp, temp_old, u, u_old,
03388            f[NSHAPE], fsum, nu[NSHAPE];
03389
03390     int i, id, ig, ip, it, n;
03391
03392     /* Loop over trace gases and channels... */
03393     for (ig = 0; ig < ctl->ng; ig++)
03394 #pragma omp parallel for default(none) shared(ctl,tbl,ig) private(in,filename,line,eps,eps_old,press,
03395                                press_old,temp,temp_old,u,u_old,id,ip,it)
03395         for (id = 0; id < ctl->nd; id++) {
03396
03397             /* Initialize... */
03398             tbl->np[ig][id] = -1;
03399             eps_old = -999;
03400             press_old = -999;
03401             temp_old = -999;
03402             u_old = -999;
03403
03404             /* Try to open file... */
03405             sprintf(filename, "%s%.4f%s.tab",
03406                     ctl->tblbase, ctl->nu[id], ctl->emitter[ig]);
03407             if (!(in = fopen(filename, "r"))) {
03408                 printf("Missing emissivity table: %s\n", filename);
03409                 continue;
03410             }
03411             printf("Read emissivity table: %s\n", filename);
03412
03413             /* Read data... */
03414             while (fgets(line, LEN, in)) {
03415
03416                 /* Parse line... */
03417                 if (sscanf(line, "%lg %lg %lg %lg", &press, &temp, &u, &eps) != 4)
03418                     continue;
03419
03420                 /* Determine pressure index... */
03421                 if (press != press_old) {
03422                     press_old = press;
03423                     if ((++tbl->np[ig][id]) >= TBLNP)
03424                         ERRMSG("Too many pressure levels!");
03425                     tbl->nt[ig][id][tbl->np[ig][id]] = -1;
03426                 }
03427
03428                 /* Determine temperature index... */
03429                 if (temp != temp_old) {
03430                     temp_old = temp;
03431                     if ((++tbl->nt[ig][id][tbl->np[ig][id]]) >= TBLNT)
03432                         ERRMSG("Too many temperatures!");
03433                     tbl->nu[ig][id][tbl->np[ig][id]]
03434                     [tbl->nt[ig][id][tbl->np[ig][id]]] = -1;
03435                 }
03436
03437                 /* Determine column density index... */
03438                 if ((eps > eps_old && u > u_old) || tbl->nu[ig][id][tbl->np[ig][id]]
03439                     [tbl->nt[ig][id][tbl->np[ig][id]]] < 0) {
03440                     eps_old = eps;
03441                     u_old = u;
03442                     if ((++tbl->nu[ig][id][tbl->np[ig][id]]
03443                         [tbl->nt[ig][id][tbl->np[ig][id]]]) >= TBLNU) {
03444                         tbl->nu[ig][id][tbl->np[ig][id]]
03445                         [tbl->nt[ig][id][tbl->np[ig][id]]]--;
03446                         continue;
03447                     }
03448                 }
03449
03450                 /* Store data... */
03451                 tbl->p[ig][id][tbl->np[ig][id]] = press;
03452                 tbl->t[ig][id][tbl->np[ig][id]][tbl->nt[ig][id][tbl->np[ig][id]]]
03453                 = temp;
03454                 tbl->u[ig][id][tbl->np[ig][id]][tbl->nt[ig][id][tbl->np[ig][id]]]
03455                 [tbl->nu[ig][id][tbl->np[ig][id]]]
03456                 [tbl->nt[ig][id][tbl->np[ig][id]]] = (float) u;

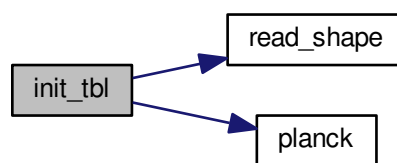
```

```

03457         tbl->eps[ig][id][tbl->np[ig][id]][tbl->nt[ig][id][tbl->np[ig][id]]]
03458         [tbl->nu[ig][id][tbl->np[ig][id]]]
03459         [tbl->nt[ig][id][tbl->np[ig][id]]] = (float) eps;
03460     }
03461
03462     /* Increment counters... */
03463     tbl->np[ig][id]++;
03464     for (ip = 0; ip < tbl->np[ig][id]; ip++) {
03465         tbl->nt[ig][id][ip]++;
03466         for (it = 0; it < tbl->nt[ig][id][ip]; it++)
03467             tbl->nu[ig][id][ip][it]++;
03468     }
03469
03470     /* Close file... */
03471     fclose(in);
03472 }
03473
03474 /* Write info... */
03475 printf("Initialize source function table...\n");
03476
03477 /* Loop over channels... */
03478 #pragma omp parallel for default(none) shared(ctl,tbl,ig) private(filename,it,i,n,f,fsum,nu)
03479 for (id = 0; id < ctl->nd; id++) {
03480
03481     /* Read filter function... */
03482     sprintf(filename, "%s_%.4f.filt", ctl->tblbase, ctl->nu[id]);
03483     read_shape(filename, nu, f, &n);
03484
03485     /* Compute source function table... */
03486     for (it = 0; it < TBLNS; it++) {
03487
03488         /* Set temperature... */
03489         tbl->st[it] = LIN(0.0, TMIN, TBLNS - 1.0, TMAX, (double) it);
03490
03491         /* Integrate Planck function... */
03492         fsum = 0;
03493         tbl->sr[id][it] = 0;
03494         for (i = 0; i < n; i++) {
03495             fsum += f[i];
03496             tbl->sr[id][it] += f[i] * planck(tbl->st[it], nu[i]);
03497         }
03498         tbl->sr[id][it] /= fsum;
03499     }
03500 }
03501 }

```

Here is the call graph for this function:



**5.23.2.22** 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 3505 of file [jurassic.c](#).

```

03512     {
03513
03514         int ig, ip, iw;
03515

```

```

03516  /* Get array index... */
03517  ip = locate_irr(atm->z, atm->np, z);
03518
03519  /* Interpolate... */
03520  *p = EXP(atm->z[ip], atm->p[ip], atm->z[ip + 1], atm->p[ip + 1], z);
03521  *t = LIN(atm->z[ip], atm->t[ip], atm->z[ip + 1], atm->t[ip + 1], z);
03522  for (ig = 0; ig < ctl->ng; ig++)
03523      q[ig] =
03524          LIN(atm->z[ip], atm->q[ig][ip], atm->z[ip + 1], atm->q[ig][ip + 1], z);
03525  for (iw = 0; iw < ctl->nw; iw++)
03526      k[iw] =
03527          LIN(atm->z[ip], atm->k[iw][ip], atm->z[ip + 1], atm->k[iw][ip + 1], z);
03528  }

```

Here is the call graph for this function:



**5.23.2.23** 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 3532 of file [jurassic.c](#).

```

03538      {
03539
03540      double eps, eps00, eps01, eps10, eps11, u;
03541
03542      int id, ig, ipr, it0, it1;
03543
03544      /* Initialize... */
03545      if (ip <= 0)
03546          for (ig = 0; ig < ctl->ng; ig++)
03547              for (id = 0; id < ctl->nd; id++)
03548                  tau_path[ig][id] = 1;
03549
03550      /* Loop over channels... */
03551      for (id = 0; id < ctl->nd; id++) {
03552
03553          /* Initialize... */
03554          tau_seg[id] = 1;
03555
03556          /* Loop over emitters... */
03557          for (ig = 0; ig < ctl->ng; ig++) {
03558
03559              /* Check size of table (pressure)... */
03560              if (tbl->np[ig][id] < 2)
03561                  eps = 0;
03562
03563              /* Check transmittance... */
03564              else if (tau_path[ig][id] < 1e-9)
03565                  eps = 1;
03566
03567              /* Interpolate... */
03568              else {
03569
03570                  /* Determine pressure and temperature indices... */
03571                  ipr = locate_irr(tbl->p[ig][id], tbl->np[ig][id], los->p[ip]);
03572                  it0 =
03573                      locate_irr(tbl->t[ig][id][ipr], tbl->nt[ig][id][ipr], los->
03574                          t[ip]);
03575                  it1 =
03576                      locate_reg(tbl->t[ig][id][ipr + 1], tbl->nt[ig][id][ipr + 1],
03577                          los->t[ip]);

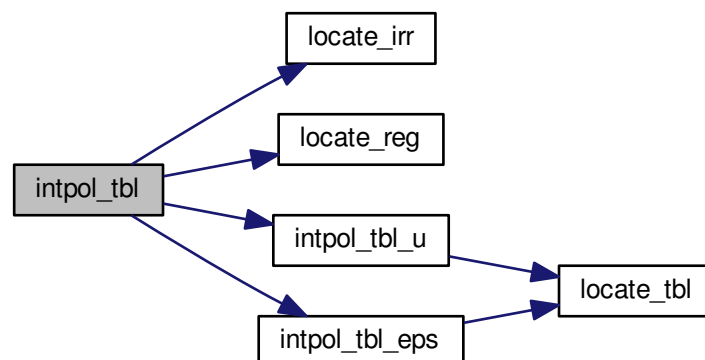
```

```

03577
03578 /* Check size of table (temperature and column density)... */
03579 if (tbl->nt[ig][id][ipr] < 2 || tbl->nt[ig][id][ipr + 1] < 2
03580     || tbl->nu[ig][id][ipr][it0] < 2
03581     || tbl->nu[ig][id][ipr][it0 + 1] < 2
03582     || tbl->nu[ig][id][ipr + 1][it1] < 2
03583     || tbl->nu[ig][id][ipr + 1][it1 + 1] < 2)
03584     eps = 0;
03585
03586 else {
03587
03588     /* Get emissivities of extended path... */
03589     u = intpol_tbl_u(tbl, ig, id, ipr, it0, 1 - tau_path[ig][id]);
03590     eps00 = intpol_tbl_eps(tbl, ig, id, ipr, it0, u + los->u[ig][ip]);
03591
03592     u = intpol_tbl_u(tbl, ig, id, ipr, it0 + 1, 1 - tau_path[ig][id]);
03593     eps01 =
03594         intpol_tbl_eps(tbl, ig, id, ipr, it0 + 1, u + los->u[ig][ip]);
03595
03596     u = intpol_tbl_u(tbl, ig, id, ipr + 1, it1, 1 - tau_path[ig][id]);
03597     eps10 =
03598         intpol_tbl_eps(tbl, ig, id, ipr + 1, it1, u + los->u[ig][ip]);
03599
03600     u =
03601         intpol_tbl_u(tbl, ig, id, ipr + 1, it1 + 1, 1 - tau_path[ig][id]);
03602     eps11 =
03603         intpol_tbl_eps(tbl, ig, id, ipr + 1, it1 + 1, u + los->
03604 u[ig][ip]);
03605
03606     /* Interpolate with respect to temperature... */
03607     eps00 = LIN(tbl->t[ig][id][ipr][it0], eps00,
03608                 tbl->t[ig][id][ipr][it0 + 1], eps01, los->t[ip]);
03609     eps11 = LIN(tbl->t[ig][id][ipr + 1][it1], eps10,
03610                 tbl->t[ig][id][ipr + 1][it1 + 1], eps11, los->t[ip]);
03611
03612     /* Interpolate with respect to pressure... */
03613     eps00 = LIN(tbl->p[ig][id][ipr], eps00,
03614                 tbl->p[ig][id][ipr + 1], eps11, los->p[ip]);
03615
03616     /* Check emssivity range... */
03617     eps00 = GSL_MAX(GSL_MIN(eps00, 1), 0);
03618
03619     /* Determine segment emissivity... */
03620     eps = 1 - (1 - eps00) / tau_path[ig][id];
03621 }
03622
03623 /* Get transmittance of extended path... */
03624 tau_path[ig][id] *= (1 - eps);
03625
03626 /* Get segment transmittance... */
03627 tau_seg[id] *= (1 - eps);
03628 }
03629 }
03630 }

```

Here is the call graph for this function:



#### 5.23.2.24 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 3634 of file [jurassic.c](#).

```

03640         {
03641
03642     int idx;
03643
03644     /* Lower boundary... */
03645     if (u < tbl->u[ig][id][ip][it][0])
03646         return LIN(0, 0, tbl->u[ig][id][ip][it][0], tbl->eps[ig][id][ip][it][0],
03647             u);
03648
03649     /* Upper boundary... */
03650     else if (u > tbl->u[ig][id][ip][it][tbl->nu[ig][id][ip][it] - 1])
03651         return LIN(tbl->u[ig][id][ip][it][tbl->nu[ig][id][ip][it] - 1],
03652             tbl->eps[ig][id][ip][it][tbl->nu[ig][id][ip][it] - 1],
03653             1e30, 1, u);
03654
03655     /* Interpolation... */
03656     else {
03657
03658         /* Get index... */
03659         idx = locate_tbl(tbl->u[ig][id][ip][it], tbl->nu[ig][id][ip][it], u);
03660
03661         /* Interpolate... */
03662         return
03663             LIN(tbl->u[ig][id][ip][it][idx], tbl->eps[ig][id][ip][it][idx],
03664                 tbl->u[ig][id][ip][it][idx + 1], tbl->eps[ig][id][ip][it][idx + 1],
03665                 u);
03666     }
03667 }

```

Here is the call graph for this function:



#### 5.23.2.25 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 3671 of file [jurassic.c](#).

```

03677         {
03678
03679     int idx;
03680
03681     /* Lower boundary... */
03682     if (eps < tbl->eps[ig][id][ip][it][0])
03683         return LIN(0, 0, tbl->eps[ig][id][ip][it][0], tbl->u[ig][id][ip][it][0],
03684             eps);
03685
03686     /* Upper boundary... */
03687     else if (eps > tbl->eps[ig][id][ip][it][tbl->nu[ig][id][ip][it] - 1])
03688         return LIN(tbl->eps[ig][id][ip][it][tbl->nu[ig][id][ip][it] - 1],
03689             tbl->u[ig][id][ip][it][tbl->nu[ig][id][ip][it] - 1],
03690             1, 1e30, eps);
03691

```



```

03692  /* Interpolation... */
03693  else {
03694
03695      /* Get index... */
03696      idx = locate_tbl(tbl->eps[ig][id][ip][it], tbl->nu[ig][id][ip][it], eps);
03697
03698      /* Interpolate... */
03699      return
03700      LIN(tbl->eps[ig][id][ip][it][idx], tbl->u[ig][id][ip][it][idx],
03701         tbl->eps[ig][id][ip][it][idx + 1], tbl->u[ig][id][ip][it][idx + 1],
03702         eps);
03703  }
03704 }

```

Here is the call graph for this function:



**5.23.2.26** 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 3708 of file [jurassic.c](#).

```

03716      {
03717
03718      struct tm t0, *t1;
03719
03720      time_t jsec0;
03721
03722      t0.tm_year = 100;
03723      t0.tm_mon = 0;
03724      t0.tm_mday = 1;
03725      t0.tm_hour = 0;
03726      t0.tm_min = 0;
03727      t0.tm_sec = 0;
03728
03729      jsec0 = (time_t) jsec + timegm(&t0);
03730      t1 = gmtime(&jsec0);
03731
03732      *year = t1->tm_year + 1900;
03733      *mon = t1->tm_mon + 1;
03734      *day = t1->tm_mday;
03735      *hour = t1->tm_hour;
03736      *min = t1->tm_min;
03737      *sec = t1->tm_sec;
03738      *remain = jsec - floor(jsec);
03739  }

```

**5.23.2.27** void kernel ( ctl\_t \* *ctl*, atm\_t \* *atm*, obs\_t \* *obs*, gsl\_matrix \* *k* )

Compute Jacobians.

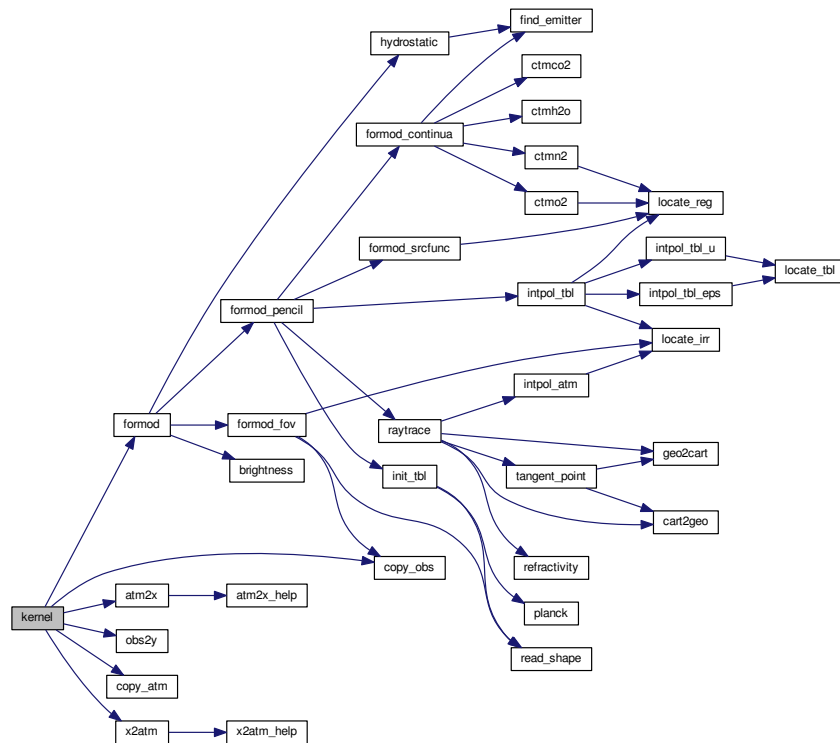
Definition at line 3743 of file [jurassic.c](#).

```

03747         {
03748
03749     atm_t *atml;
03750     obs_t *obs1;
03751
03752     gsl_vector *x0, *x1, *yy0, *yy1;
03753
03754     int *iqa, j;
03755
03756     double h;
03757
03758     size_t i, n, m;
03759
03760     /* Get sizes... */
03761     m = k->size1;
03762     n = k->size2;
03763
03764     /* Allocate... */
03765     x0 = gsl_vector_alloc(n);
03766     yy0 = gsl_vector_alloc(m);
03767     ALLOC(iqa, int,
03768           N);
03769
03770     /* Compute radiance for undisturbed atmospheric data... */
03771     formod(ctl, atm, obs);
03772
03773     /* Compose vectors... */
03774     atm2x(ctl, atm, x0, iqa, NULL);
03775     obs2y(ctl, obs, yy0, NULL, NULL);
03776
03777     /* Initialize kernel matrix... */
03778     gsl_matrix_set_zero(k);
03779
03780     /* Loop over state vector elements... */
03781 #pragma omp parallel for default(none) shared(ctl,atm,obs,k,x0,yy0,n,m,iqa) private(i, j, h, x1, yy1, atml,
03782     obs1)
03783     for (j = 0; j < (int) n; j++) {
03784
03785         /* Allocate... */
03786         x1 = gsl_vector_alloc(n);
03787         yy1 = gsl_vector_alloc(m);
03788         ALLOC(atml, atm_t, 1);
03789         ALLOC(obs1, obs_t, 1);
03790
03791         /* Set perturbation size... */
03792         if (iqa[j] == IDXP)
03793             h = GSL_MAX(fabs(0.01 * gsl_vector_get(x0, (size_t) j)), 1e-7);
03794         else if (iqa[j] == IDXT)
03795             h = 1;
03796         else if (iqa[j] >= IDXQ(0) && iqa[j] < IDXQ(ctl->ng))
03797             h = GSL_MAX(fabs(0.01 * gsl_vector_get(x0, (size_t) j)), 1e-15);
03798         else if (iqa[j] >= IDXK(0) && iqa[j] < IDXK(ctl->nw))
03799             h = 1e-4;
03800         else
03801             ERRMSG("Cannot set perturbation size!");
03802
03803         /* Disturb state vector element... */
03804         gsl_vector_memcpy(x1, x0);
03805         gsl_vector_set(x1, (size_t) j, gsl_vector_get(x1, (size_t) j) + h);
03806         copy_atm(ctl, atml, atm, 0);
03807         copy_obs(ctl, obs1, obs, 0);
03808         x2atm(ctl, x1, atml);
03809
03810         /* Compute radiance for disturbed atmospheric data... */
03811         formod(ctl, atml, obs1);
03812
03813         /* Compose measurement vector for disturbed radiance data... */
03814         obs2y(ctl, obs1, yy1, NULL, NULL);
03815
03816         /* Compute derivatives... */
03817         for (i = 0; i < m; i++)
03818             gsl_matrix_set(k, i, (size_t) j,
03819                           (gsl_vector_get(yy1, i) - gsl_vector_get(yy0, i)) / h);
03820
03821         /* Free... */
03822         gsl_vector_free(x1);
03823         gsl_vector_free(yy1);
03824         free(atml);
03825         free(obs1);
03826     }
03827
03828     /* Free... */
03829     gsl_vector_free(x0);
03830     gsl_vector_free(yy0);
03831     free(iqa);
03832 }

```

Here is the call graph for this function:



#### 5.23.2.28 int locate\_irr ( double \* xx, int n, double x )

Find array index for irregular grid.

Definition at line 3835 of file [jurassic.c](#).

```

03838     {
03839
03840     int i, ilo, ihi;
03841
03842     ilo = 0;
03843     ihi = n - 1;
03844     i = (ihi + ilo) >> 1;
03845
03846     if (xx[i] < xx[i + 1])
03847         while (ihi > ilo + 1) {
03848             i = (ihi + ilo) >> 1;
03849             if (xx[i] > x)
03850                 ihi = i;
03851             else
03852                 ilo = i;
03853         } else
03854             while (ihi > ilo + 1) {
03855                 i = (ihi + ilo) >> 1;
03856                 if (xx[i] <= x)
03857                     ihi = i;
03858                 else
03859                     ilo = i;
03860             }
03861     return ilo;
03862 }
03863 }
```

## 5.23.2.29 int locate\_reg ( double \* xx, int n, double x )

Find array index for regular grid.

Definition at line 3867 of file [jurassic.c](#).

```

03870         {
03871
03872     int i;
03873
03874     /* Calculate index... */
03875     i = (int) ((x - xx[0]) / (xx[1] - xx[0]));
03876
03877     /* Check range... */
03878     if (i < 0)
03879         i = 0;
03880     else if (i >= n - 2)
03881         i = n - 2;
03882
03883     return i;
03884 }
```

## 5.23.2.30 int locate\_tbl ( float \* xx, int n, double x )

Find array index in float array.

Definition at line 3888 of file [jurassic.c](#).

```

03891         {
03892
03893     int i, ilo, ihi;
03894
03895     ilo = 0;
03896     ihi = n - 1;
03897     i = (ihi + ilo) >> 1;
03898
03899     while (ihi > ilo + 1) {
03900         i = (ihi + ilo) >> 1;
03901         if (xx[i] > x)
03902             ihi = i;
03903         else
03904             ilo = i;
03905     }
03906
03907     return ilo;
03908 }
```

## 5.23.2.31 size\_t obs2y ( ctl\_t \* ctl, obs\_t \* obs, gsl\_vector \* y, int \* ida, int \* ira )

Compose measurement vector.

Definition at line 3912 of file [jurassic.c](#).

```

03917         {
03918
03919     int id, ir;
03920
03921     size_t m = 0;
03922
03923     /* Determine measurement vector... */
03924     for (ir = 0; ir < obs->nr; ir++)
03925         for (id = 0; id < ctl->nd; id++)
03926             if (gsl_finite(obs->rad[id][ir])) {
03927                 if (y != NULL)
03928                     gsl_vector_set(y, m, obs->rad[id][ir]);
03929                 if (ida != NULL)
03930                     ida[m] = id;
03931                 if (ira != NULL)
03932                     ira[m] = ir;
03933                 m++;
03934             }
03935
03936     return m;
03937 }
```

### 5.23.2.32 double planck ( double *t*, double *nu* )

Compute Planck function.

Definition at line 3941 of file [jurassic.c](#).

```
03943     {
03944
03945     return C1 * POW3(nu) / gsl_expml(C2 * nu / t);
03946 }
```

### 5.23.2.33 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 3950 of file [jurassic.c](#).

```
03955     {
03956
03957     double cosa, d, dmax, dmin = 0, ds, ex0[3], ex1[3], frac, h = 0.02, k[NW],
03958     lat, lon, n, naux, ng[3], norm, p, q[NG], t, x[3], xh[3],
03959     xobs[3], xvp[3], z = 1e99, zmax, zmin, zrefrac = 60;
03960
03961     int i, ig, ip, iw, stop = 0;
03962
03963     /* Initialize... */
03964     los->np = 0;
03965     los->tsurf = -999;
03966     obs->tpz[ir] = obs->vpz[ir];
03967     obs->tplon[ir] = obs->vplon[ir];
03968     obs->tplat[ir] = obs->vplat[ir];
03969
03970     /* Get altitude range of atmospheric data... */
03971     gsl_stats_minmax(&zmin, &zmax, atm->z, 1, (size_t) atm->np);
03972
03973     /* Check observer altitude... */
03974     if (obs->obsz[ir] < zmin)
03975         ERRMSG("Observer below surface!");
03976
03977     /* Check view point altitude... */
03978     if (obs->vpz[ir] > zmax)
03979         return;
03980
03981     /* Determine Cartesian coordinates for observer and view point... */
03982     geo2cart(obs->obsz[ir], obs->obslon[ir], obs->obslat[ir], xobs);
03983     geo2cart(obs->vpz[ir], obs->vplon[ir], obs->vplat[ir], xvp);
03984
03985     /* Determine initial tangent vector... */
03986     for (i = 0; i < 3; i++)
03987         ex0[i] = xvp[i] - xobs[i];
03988     norm = NORM(ex0);
03989     for (i = 0; i < 3; i++)
03990         ex0[i] /= norm;
03991
03992     /* Observer within atmosphere... */
03993     for (i = 0; i < 3; i++)
03994         x[i] = xobs[i];
03995
03996     /* Observer above atmosphere (search entry point)... */
03997     if (obs->obsz[ir] > zmax) {
03998         dmax = norm;
03999         while (fabs(dmin - dmax) > 0.001) {
04000             d = (dmax + dmin) / 2;
04001             for (i = 0; i < 3; i++)
04002                 x[i] = xobs[i] + d * ex0[i];
04003             cart2geo(x, &z, &lon, &lat);
04004             if (z <= zmax && z > zmax - 0.001)
04005                 break;
04006             if (z < zmax - 0.0005)
04007                 dmax = d;
04008             else
04009                 dmin = d;
04010         }
04011     }
04012
04013     /* Ray-tracing... */
```

```

04014 while (1) {
04015
04016     /* Set step length... */
04017     ds = ctl->rayds;
04018     if (ctl->raydz > 0) {
04019         norm = NORM(x);
04020         for (i = 0; i < 3; i++)
04021             xh[i] = x[i] / norm;
04022         cosa = fabs(DOTP(ex0, xh));
04023         if (cosa != 0)
04024             ds = GSL_MIN(ctl->rayds, ctl->raydz / cosa);
04025     }
04026
04027     /* Determine geolocation... */
04028     cart2geo(x, &z, &lon, &lat);
04029
04030     /* Check if LOS hits the ground or has left atmosphere... */
04031     if (z < zmin || z > zmax) {
04032         stop = (z < zmin ? 2 : 1);
04033         frac =
04034             ((z <
04035              zmin ? zmin : zmax) - los->z[los->np - 1]) / (z - los->z[los->np -
04036                                                           1]);
04037         geo2cart(los->z[los->np - 1], los->lon[los->np - 1],
04038                 los->lat[los->np - 1], xh);
04039         for (i = 0; i < 3; i++)
04040             x[i] = xh[i] + frac * (x[i] - xh[i]);
04041         cart2geo(x, &z, &lon, &lat);
04042         los->ds[los->np - 1] = ds * frac;
04043         ds = 0;
04044     }
04045
04046     /* Interpolate atmospheric data... */
04047     intpol_atm(ctl, atm, z, &p, &t, q, k);
04048
04049     /* Save data... */
04050     los->lon[los->np] = lon;
04051     los->lat[los->np] = lat;
04052     los->z[los->np] = z;
04053     los->p[los->np] = p;
04054     los->t[los->np] = t;
04055     for (ig = 0; ig < ctl->ng; ig++)
04056         los->q[ig][los->np] = q[ig];
04057     for (iw = 0; iw < ctl->nw; iw++)
04058         los->k[iw][los->np] = k[iw];
04059     los->ds[los->np] = ds;
04060
04061     /* Increment and check number of LOS points... */
04062     if ((++los->np) > NLOS)
04063         ERRMSG("Too many LOS points!");
04064
04065     /* Check stop flag... */
04066     if (stop) {
04067         los->tsurf = (stop == 2 ? t : -999);
04068         break;
04069     }
04070
04071     /* Determine refractivity... */
04072     if (ctl->refrac && z <= zrefrac)
04073         n = 1 + refractivity(p, t);
04074     else
04075         n = 1;
04076
04077     /* Construct new tangent vector (first term)... */
04078     for (i = 0; i < 3; i++)
04079         ex1[i] = ex0[i] * n;
04080
04081     /* Compute gradient of refractivity... */
04082     if (ctl->refrac && z <= zrefrac) {
04083         for (i = 0; i < 3; i++)
04084             xh[i] = x[i] + 0.5 * ds * ex0[i];
04085         cart2geo(xh, &z, &lon, &lat);
04086         intpol_atm(ctl, atm, z, &p, &t, q, k);
04087         n = refractivity(p, t);
04088         for (i = 0; i < 3; i++) {
04089             xh[i] += h;
04090             cart2geo(xh, &z, &lon, &lat);
04091             intpol_atm(ctl, atm, z, &p, &t, q, k);
04092             naux = refractivity(p, t);
04093             ng[i] = (naux - n) / h;
04094             xh[i] -= h;
04095         }
04096     } else
04097         for (i = 0; i < 3; i++)
04098             ng[i] = 0;
04099
04100     /* Construct new tangent vector (second term)... */

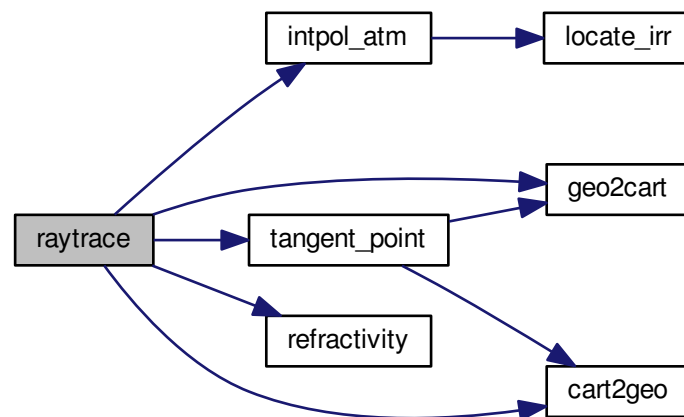
```

```

04101     for (i = 0; i < 3; i++)
04102         exl[i] += ds * ng[i];
04103
04104     /* Normalize new tangent vector... */
04105     norm = NORM(exl);
04106     for (i = 0; i < 3; i++)
04107         exl[i] /= norm;
04108
04109     /* Determine next point of LOS... */
04110     for (i = 0; i < 3; i++)
04111         x[i] += 0.5 * ds * (ex0[i] + exl[i]);
04112
04113     /* Copy tangent vector... */
04114     for (i = 0; i < 3; i++)
04115         ex0[i] = exl[i];
04116 }
04117
04118 /* Get tangent point (to be done before changing segment lengths!)... */
04119 tangent_point(los, &obs->tpz[ir], &obs->tplon[ir], &obs->
04120             tplat[ir]);
04121
04122 /* Change segment lengths according to trapezoid rule... */
04123 for (ip = los->np - 1; ip >= 1; ip--)
04124     los->ds[ip] = 0.5 * (los->ds[ip - 1] + los->ds[ip]);
04125 los->ds[0] *= 0.5;
04126
04127 /* Compute column density... */
04128 for (ip = 0; ip < los->np; ip++)
04129     for (ig = 0; ig < ctl->ng; ig++)
04130         los->u[ig][ip] = 10 * los->q[ig][ip] * los->p[ip]
04131         / (KB * los->t[ip] * los->ds[ip];
04132 }

```

Here is the call graph for this function:



#### 5.23.2.34 void read\_atm ( const char \* dirname, const char \* filename, ctl\_t \* ctl, atm\_t \* atm )

Read atmospheric data.

Definition at line 4135 of file [jurassic.c](#).

```

04139     {
04140
04141     FILE *in;
04142
04143     char file[LEN], line[LEN], *tok;

```

```

04144
04145     int ig, iw;
04146
04147     /* Init... */
04148     atm->np = 0;
04149
04150     /* Set filename... */
04151     if (dirname != NULL)
04152         sprintf(file, "%s/%s", dirname, filename);
04153     else
04154         sprintf(file, "%s", filename);
04155
04156     /* Write info... */
04157     printf("Read atmospheric data: %s\n", file);
04158
04159     /* Open file... */
04160     if (!(in = fopen(file, "r")))
04161         ERRMSG("Cannot open file!");
04162
04163     /* Read line... */
04164     while (fgets(line, LEN, in)) {
04165
04166         /* Read data... */
04167         TOK(line, tok, "%lg", atm->time[atm->np]);
04168         TOK(NULL, tok, "%lg", atm->z[atm->np]);
04169         TOK(NULL, tok, "%lg", atm->lon[atm->np]);
04170         TOK(NULL, tok, "%lg", atm->lat[atm->np]);
04171         TOK(NULL, tok, "%lg", atm->p[atm->np]);
04172         TOK(NULL, tok, "%lg", atm->t[atm->np]);
04173         for (ig = 0; ig < ctl->ng; ig++)
04174             TOK(NULL, tok, "%lg", atm->q[ig][atm->np]);
04175         for (iw = 0; iw < ctl->nw; iw++)
04176             TOK(NULL, tok, "%lg", atm->k[iw][atm->np]);
04177
04178         /* Increment data point counter... */
04179         if ((++atm->np) > NP)
04180             ERRMSG("Too many data points!");
04181     }
04182
04183     /* Close file... */
04184     fclose(in);
04185
04186     /* Check number of points... */
04187     if (atm->np < 1)
04188         ERRMSG("Could not read any data!");
04189 }

```

#### 5.23.2.35 void read\_ctl ( int argc, char \* argv[], ctl\_t \* ctl )

Read forward model control parameters.

Definition at line 4193 of file [jurassic.c](#).

```

04196     {
04197
04198     int id, ig, iw;
04199
04200     /* Write info... */
04201     printf("\nJuelich Rapid Spectral Simulation Code (JURASSIC)\n"
04202           "(executable: %s | compiled: %s, %s)\n\n",
04203           argv[0], __DATE__, __TIME__);
04204
04205     /* Emitters... */
04206     ctl->ng = (int) scan_ctl(argc, argv, "NG", -1, "0", NULL);
04207     if (ctl->ng < 0 || ctl->ng > NG)
04208         ERRMSG("Set 0 <= NG <= MAX!");
04209     for (ig = 0; ig < ctl->ng; ig++)
04210         scan_ctl(argc, argv, "EMITTER", ig, "", ctl->emitter[ig]);
04211
04212     /* Radiance channels... */
04213     ctl->nd = (int) scan_ctl(argc, argv, "ND", -1, "0", NULL);
04214     if (ctl->nd < 0 || ctl->nd > ND)
04215         ERRMSG("Set 0 <= ND <= MAX!");
04216     for (id = 0; id < ctl->nd; id++)
04217         ctl->nu[id] = scan_ctl(argc, argv, "NU", id, "", NULL);
04218
04219     /* Spectral windows... */
04220     ctl->nw = (int) scan_ctl(argc, argv, "NW", -1, "1", NULL);
04221     if (ctl->nw < 0 || ctl->nw > NW)
04222         ERRMSG("Set 0 <= NW <= MAX!");

```



```

04223     for (id = 0; id < ctl->nd; id++)
04224         ctl->window[id] = (int) scan_ctl(argc, argv, "WINDOW", id, "0", NULL);
04225
04226     /* Emissivity look-up tables... */
04227     scan_ctl(argc, argv, "TBLBASE", -1, "-", ctl->tblbase);
04228
04229     /* Hydrostatic equilibrium... */
04230     ctl->hydZ = scan_ctl(argc, argv, "HYDZ", -1, "-999", NULL);
04231
04232     /* Continua... */
04233     ctl->ctm_co2 = (int) scan_ctl(argc, argv, "CTM_CO2", -1, "1", NULL);
04234     ctl->ctm_h2o = (int) scan_ctl(argc, argv, "CTM_H2O", -1, "1", NULL);
04235     ctl->ctm_n2 = (int) scan_ctl(argc, argv, "CTM_N2", -1, "1", NULL);
04236     ctl->ctm_o2 = (int) scan_ctl(argc, argv, "CTM_O2", -1, "1", NULL);
04237
04238     /* Ray-tracing... */
04239     ctl->refrac = (int) scan_ctl(argc, argv, "REFRAC", -1, "1", NULL);
04240     ctl->rayds = scan_ctl(argc, argv, "RAYDS", -1, "10", NULL);
04241     ctl->raydz = scan_ctl(argc, argv, "RAYDZ", -1, "0.5", NULL);
04242
04243     /* Field of view... */
04244     scan_ctl(argc, argv, "FOV", -1, "-", ctl->fov);
04245
04246     /* Retrieval interface... */
04247     ctl->retp_zmin = scan_ctl(argc, argv, "RETP_ZMIN", -1, "-999", NULL);
04248     ctl->retp_zmax = scan_ctl(argc, argv, "RETP_ZMAX", -1, "-999", NULL);
04249     ctl->rett_zmin = scan_ctl(argc, argv, "RETT_ZMIN", -1, "-999", NULL);
04250     ctl->rett_zmax = scan_ctl(argc, argv, "RETT_ZMAX", -1, "-999", NULL);
04251     for (ig = 0; ig < ctl->ng; ig++) {
04252         ctl->retq_zmin[ig] = scan_ctl(argc, argv, "RETQ_ZMIN", ig, "-999", NULL);
04253         ctl->retq_zmax[ig] = scan_ctl(argc, argv, "RETQ_ZMAX", ig, "-999", NULL);
04254     }
04255     for (iw = 0; iw < ctl->nw; iw++) {
04256         ctl->retk_zmin[iw] = scan_ctl(argc, argv, "RETK_ZMIN", iw, "-999", NULL);
04257         ctl->retk_zmax[iw] = scan_ctl(argc, argv, "RETK_ZMAX", iw, "-999", NULL);
04258     }
04259
04260     /* Output flags... */
04261     ctl->write_bbt = (int) scan_ctl(argc, argv, "WRITE_BBT", -1, "0", NULL);
04262     ctl->write_matrix =
04263         (int) scan_ctl(argc, argv, "WRITE_MATRIX", -1, "0", NULL);
04264 }

```

Here is the call graph for this function:



**5.23.2.36** void read\_matrix ( const char \* *dirname*, const char \* *filename*, gsl\_matrix \* *matrix* )

Read matrix.

Definition at line 4268 of file [jurassic.c](#).

```

04271     {
04272
04273     FILE *in;
04274
04275     char dum[LEN], file[LEN], line[LEN];
04276
04277     double value;
04278
04279     int i, j;
04280
04281     /* Set filename... */

```

```

04282     if (dirname != NULL)
04283         sprintf(file, "%s/%s", dirname, filename);
04284     else
04285         sprintf(file, "%s", filename);
04286
04287     /* Write info... */
04288     printf("Read matrix: %s\n", file);
04289
04290     /* Open file... */
04291     if (!(in = fopen(file, "r")))
04292         ERRMSG("Cannot open file!");
04293
04294     /* Read data... */
04295     gsl_matrix_set_zero(matrix);
04296     while (fgets(line, LEN, in))
04297         if (sscanf(line, "%d %s %s %s %s %s %d %s %s %s %s %s %lg",
04298             &i, dum, dum, dum, dum, dum,
04299             &j, dum, dum, dum, dum, dum, &value) == 13)
04300         gsl_matrix_set(matrix, (size_t) i, (size_t) j, value);
04301
04302     /* Close file... */
04303     fclose(in);
04304 }

```

### 5.23.2.37 void read\_obs ( const char \* *dirname*, const char \* *filename*, ctl\_t \* *ctl*, obs\_t \* *obs* )

Read observation data.

Definition at line 4308 of file [jurassic.c](#).

```

04312     {
04313
04314     FILE *in;
04315
04316     char file[LEN], line[LEN], *tok;
04317
04318     int id;
04319
04320     /* Init... */
04321     obs->nr = 0;
04322
04323     /* Set filename... */
04324     if (dirname != NULL)
04325         sprintf(file, "%s/%s", dirname, filename);
04326     else
04327         sprintf(file, "%s", filename);
04328
04329     /* Write info... */
04330     printf("Read observation data: %s\n", file);
04331
04332     /* Open file... */
04333     if (!(in = fopen(file, "r")))
04334         ERRMSG("Cannot open file!");
04335
04336     /* Read line... */
04337     while (fgets(line, LEN, in)) {
04338
04339         /* Read data... */
04340         TOK(line, tok, "%lg", obs->time[obs->nr]);
04341         TOK(NULL, tok, "%lg", obs->obsz[obs->nr]);
04342         TOK(NULL, tok, "%lg", obs->obslon[obs->nr]);
04343         TOK(NULL, tok, "%lg", obs->obslat[obs->nr]);
04344         TOK(NULL, tok, "%lg", obs->vpz[obs->nr]);
04345         TOK(NULL, tok, "%lg", obs->vplon[obs->nr]);
04346         TOK(NULL, tok, "%lg", obs->vplat[obs->nr]);
04347         TOK(NULL, tok, "%lg", obs->tpz[obs->nr]);
04348         TOK(NULL, tok, "%lg", obs->tplon[obs->nr]);
04349         TOK(NULL, tok, "%lg", obs->tplat[obs->nr]);
04350         for (id = 0; id < ctl->nd; id++)
04351             TOK(NULL, tok, "%lg", obs->rad[id][obs->nr]);
04352         for (id = 0; id < ctl->nd; id++)
04353             TOK(NULL, tok, "%lg", obs->tau[id][obs->nr]);
04354
04355         /* Increment counter... */
04356         if ((++obs->nr) > NR)
04357             ERRMSG("Too many rays!");
04358     }
04359
04360     /* Close file... */
04361     fclose(in);

```

```

04362
04363  /* Check number of points... */
04364  if (obs->nr < 1)
04365      ERRMSG("Could not read any data!");
04366  }

```

### 5.23.2.38 void read\_shape ( const char \* filename, double \* x, double \* y, int \* n )

Read shape function.

Definition at line 4370 of file [jurassic.c](#).

```

04374      {
04375
04376      FILE *in;
04377
04378      char line[LEN];
04379
04380      /* Write info... */
04381      printf("Read shape function: %s\n", filename);
04382
04383      /* Open file... */
04384      if (!(in = fopen(filename, "r")))
04385          ERRMSG("Cannot open file!");
04386
04387      /* Read data... */
04388      *n = 0;
04389      while (fgets(line, LEN, in))
04390          if (sscanf(line, "%lg %lg", &x[*n], &y[*n]) == 2)
04391              if (++(*n) > NSHAPE)
04392                  ERRMSG("Too many data points!");
04393
04394      /* Check number of points... */
04395      if (*n < 1)
04396          ERRMSG("Could not read any data!");
04397
04398      /* Close file... */
04399      fclose(in);
04400  }

```

### 5.23.2.39 double refractivity ( double p, double t )

Compute refractivity (return value is n - 1).

Definition at line 4404 of file [jurassic.c](#).

```

04406      {
04407
04408      /* Refractivity of air at 4 to 15 micron... */
04409      return 7.753e-05 * p / t;
04410  }

```

### 5.23.2.40 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 4414 of file [jurassic.c](#).

```

04420         {
04421
04422     FILE *in = NULL;
04423
04424     char dummy[LEN], fullname1[LEN], fullname2[LEN], line[LEN],
04425         msg[2 * LEN], rvarname[LEN], rval[LEN];
04426
04427     int contain = 0, i;
04428
04429     /* Open file... */
04430     if (argv[1][0] != '-')
04431         if (! (in = fopen(argv[1], "r")))
04432             ERRMSG("Cannot open file!");
04433
04434     /* Set full variable name... */
04435     if (arridx >= 0) {
04436         sprintf(fullname1, "%s[%d]", varname, arridx);
04437         sprintf(fullname2, "%s[*]", varname);
04438     } else {
04439         sprintf(fullname1, "%s", varname);
04440         sprintf(fullname2, "%s", varname);
04441     }
04442
04443     /* Read data... */
04444     if (in != NULL)
04445         while (fgets(line, LEN, in))
04446             if (sscanf(line, "%s %s %s", rvarname, dummy, rval) == 3)
04447                 if (strcasemp(rvarname, fullname1) == 0 ||
04448                     strcasemp(rvarname, fullname2) == 0) {
04449                     contain = 1;
04450                     break;
04451                 }
04452     for (i = 1; i < argc - 1; i++)
04453         if (strcasemp(argv[i], fullname1) == 0 ||
04454             strcasemp(argv[i], fullname2) == 0) {
04455             sprintf(rval, "%s", argv[i + 1]);
04456             contain = 1;
04457             break;
04458         }
04459
04460     /* Close file... */
04461     if (in != NULL)
04462         fclose(in);
04463
04464     /* Check for missing variables... */
04465     if (!contain) {
04466         if (strlen(defvalue) > 0)
04467             sprintf(rval, "%s", defvalue);
04468         else {
04469             sprintf(msg, "Missing variable %s!\n", fullname1);
04470             ERRMSG(msg);
04471         }
04472     }
04473
04474     /* Write info... */
04475     printf("%s = %s\n", fullname1, rval);
04476
04477     /* Return values... */
04478     if (value != NULL)
04479         sprintf(value, "%s", rval);
04480     return atof(rval);
04481 }

```

#### 5.23.2.41 void tangent\_point ( los\_t \* los, double \* tpz, double \* tplon, double \* tplat )

Find tangent point of a given LOS.

Definition at line 4485 of file [jurassic.c](#).

```

04489         {
04490
04491     double a, b, c, dummy, v[3], v0[3], v2[3], x, x1, x2, yy0, yy1, yy2;
04492
04493     size_t i, ip;
04494
04495     /* Find minimum altitude... */
04496     ip = gsl_stats_min_index(los->z, 1, (size_t) los->np);
04497
04498     /* Nadir or zenith... */
04499     if (ip <= 0 || ip >= (size_t) los->np - 1) {

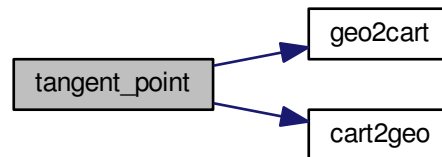
```

```

04500     *tpz = los->z[los->np - 1];
04501     *tplon = los->lon[los->np - 1];
04502     *tplat = los->lat[los->np - 1];
04503 }
04504
04505 /* Limb... */
04506 else {
04507
04508     /* Determine interpolating polynomial y=a*x^2+b*x+c... */
04509     yy0 = los->z[ip - 1];
04510     yy1 = los->z[ip];
04511     yy2 = los->z[ip + 1];
04512     x1 = sqrt(POW2(los->ds[ip]) - POW2(yy1 - yy0));
04513     x2 = x1 + sqrt(POW2(los->ds[ip + 1]) - POW2(yy2 - yy1));
04514     a = 1 / (x1 - x2) * (-(yy0 - yy1) / x1 + (yy0 - yy2) / x2);
04515     b = -(yy0 - yy1) / x1 - a * x1;
04516     c = yy0;
04517
04518     /* Get tangent point location... */
04519     x = -b / (2 * a);
04520     *tpz = a * x * x + b * x + c;
04521     geo2cart(los->z[ip - 1], los->lon[ip - 1], los->lat[ip - 1], v0);
04522     geo2cart(los->z[ip + 1], los->lon[ip + 1], los->lat[ip + 1], v2);
04523     for (i = 0; i < 3; i++)
04524         v[i] = LIN(0.0, v0[i], x2, v2[i], x);
04525     cart2geo(v, &dummy, tplon, tplat);
04526 }
04527 }

```

Here is the call graph for this function:



**5.23.2.42** 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 4531 of file [jurassic.c](#).

```

04539     {
04540
04541     struct tm t0, t1;
04542
04543     t0.tm_year = 100;
04544     t0.tm_mon = 0;
04545     t0.tm_mday = 1;
04546     t0.tm_hour = 0;
04547     t0.tm_min = 0;
04548     t0.tm_sec = 0;
04549
04550     t1.tm_year = year - 1900;
04551     t1.tm_mon = mon - 1;
04552     t1.tm_mday = day;
04553     t1.tm_hour = hour;
04554     t1.tm_min = min;
04555     t1.tm_sec = sec;
04556
04557     *jsec = (double) timegm(&t1) - (double) timegm(&t0) + remain;
04558 }

```

## 5.23.2.43 void timer ( const char \* name, const char \* file, const char \* func, int line, int mode )

Measure wall-clock time.

Definition at line 4562 of file [jurassic.c](#).

```

04567         {
04568
04569     static double w0[10];
04570
04571     static int l0[10], nt;
04572
04573     /* Start new timer... */
04574     if (mode == 1) {
04575         w0[nt] = omp_get_wtime();
04576         l0[nt] = line;
04577         if ((++nt) >= 10)
04578             ERRMSG("Too many timers!");
04579     }
04580
04581     /* Write elapsed time... */
04582     else {
04583
04584         /* Check timer index... */
04585         if (nt - 1 < 0)
04586             ERRMSG("Coding error!");
04587
04588         /* Write elapsed time... */
04589         printf("Timer '%s' (%s, %s, l%d-%d): %.3f sec\n",
04590             name, file, func, l0[nt - 1], line, omp_get_wtime() - w0[nt - 1]);
04591     }
04592
04593     /* Stop timer... */
04594     if (mode == 3)
04595         nt--;
04596 }

```

## 5.23.2.44 void write\_atm ( const char \* dirname, const char \* filename, ctl\_t \* ctl, atm\_t \* atm )

Write atmospheric data.

Definition at line 4600 of file [jurassic.c](#).

```

04604         {
04605
04606     FILE *out;
04607
04608     char file[LEN];
04609
04610     int ig, ip, iw, n = 6;
04611
04612     /* Set filename... */
04613     if (dirname != NULL)
04614         sprintf(file, "%s/%s", dirname, filename);
04615     else
04616         sprintf(file, "%s", filename);
04617
04618     /* Write info... */
04619     printf("Write atmospheric data: %s\n", file);
04620
04621     /* Create file... */
04622     if (!(out = fopen(file, "w")))
04623         ERRMSG("Cannot create file!");
04624
04625     /* Write header... */
04626     fprintf(out,
04627         "# $1 = time (seconds since 2000-01-01T00:00Z)\n"
04628         "# $2 = altitude [km]\n"
04629         "# $3 = longitude [deg]\n"
04630         "# $4 = latitude [deg]\n"
04631         "# $5 = pressure [hPa]\n" "# $6 = temperature [K]\n");
04632     for (ig = 0; ig < ctl->ng; ig++)
04633         fprintf(out, "# $%d = %s volume mixing ratio\n", ++n, ctl->emitter[ig]);
04634     for (iw = 0; iw < ctl->nw; iw++)
04635         fprintf(out, "# $%d = window %d: extinction [1/km]\n", ++n, iw);
04636 }

```

```

04637  /* Write data... */
04638  for (ip = 0; ip < atm->np; ip++) {
04639      if (ip == 0 || atm->lat[ip] != atm->lat[ip - 1]
04640          || atm->lon[ip] != atm->lon[ip - 1])
04641          fprintf(out, "\n");
04642      fprintf(out, "%.2f %g %g %g %g", atm->time[ip], atm->z[ip],
04643          atm->lon[ip], atm->lat[ip], atm->p[ip], atm->t[ip]);
04644      for (ig = 0; ig < ctl->ng; ig++)
04645          fprintf(out, " %g", atm->q[ig][ip]);
04646      for (iw = 0; iw < ctl->nw; iw++)
04647          fprintf(out, " %g", atm->k[iw][ip]);
04648      fprintf(out, "\n");
04649  }
04650
04651  /* Close file... */
04652  fclose(out);
04653 }

```

**5.23.2.45** `void write_matrix ( const char * dirname, const char * filename, ctl_t * ctl, gsl_matrix * matrix, atm_t * atm, obs_t * obs, const char * row_space, const char * col_space, const char * sort )`

Write matrix.

Definition at line 4657 of file [jurassic.c](#).

```

04666      {
04667
04668      FILE *out;
04669
04670      char file[LEN], quantity[LEN];
04671
04672      int *cida, *ciqa, *cipa, *cira, *rida, *riqa, *ripa, *rira;
04673
04674      size_t i, j, nc, nr;
04675
04676      /* Check output flag... */
04677      if (!ctl->write_matrix)
04678          return;
04679
04680      /* Allocate... */
04681      ALLOC(cida, int, M);
04682      ALLOC(ciqa, int,
04683          N);
04684      ALLOC(cipa, int,
04685          N);
04686      ALLOC(cira, int,
04687          M);
04688      ALLOC(rida, int,
04689          M);
04690      ALLOC(riqa, int,
04691          N);
04692      ALLOC(ripa, int,
04693          N);
04694      ALLOC(rira, int,
04695          M);
04696
04697      /* Set filename... */
04698      if (dirname != NULL)
04699          sprintf(file, "%s/%s", dirname, filename);
04700      else
04701          sprintf(file, "%s", filename);
04702
04703      /* Write info... */
04704      printf("Write matrix: %s\n", file);
04705
04706      /* Create file... */
04707      if (!(out = fopen(file, "w")))
04708          ERRMSG("Cannot create file!");
04709
04710      /* Write header (row space)... */
04711      if (row_space[0] == 'y') {
04712          fprintf(out,
04713              "# $1 = Row: index (measurement space)\n"
04714              "# $2 = Row: channel wavenumber [cm^-1]\n"
04715              "# $3 = Row: time (seconds since 2000-01-01T00:00Z)\n"
04716              "# $4 = Row: view point altitude [km]\n"
04717              "# $5 = Row: view point longitude [deg]\n"
04718              "# $6 = Row: view point latitude [deg]\n");
04719
04720

```

```

04721     /* Get number of rows... */
04722     nr = obs2y(ctl, obs, NULL, rida, rira);
04723
04724 } else {
04725
04726     fprintf(out,
04727         "# $1 = Row: index (state space)\n"
04728         "# $2 = Row: name of quantity\n"
04729         "# $3 = Row: time (seconds since 2000-01-01T00:00Z)\n"
04730         "# $4 = Row: altitude [km]\n"
04731         "# $5 = Row: longitude [deg]\n" "# $6 = Row: latitude [deg]\n");
04732
04733     /* Get number of rows... */
04734     nr = atm2x(ctl, atm, NULL, riq, ripa);
04735 }
04736
04737 /* Write header (column space)... */
04738 if (colspace[0] == 'y') {
04739
04740     fprintf(out,
04741         "# $7 = Col: index (measurement space)\n"
04742         "# $8 = Col: channel wavenumber [cm^-1]\n"
04743         "# $9 = Col: time (seconds since 2000-01-01T00:00Z)\n"
04744         "# $10 = Col: view point altitude [km]\n"
04745         "# $11 = Col: view point longitude [deg]\n"
04746         "# $12 = Col: view point latitude [deg]\n");
04747
04748     /* Get number of columns... */
04749     nc = obs2y(ctl, obs, NULL, cida, cira);
04750
04751 } else {
04752
04753     fprintf(out,
04754         "# $7 = Col: index (state space)\n"
04755         "# $8 = Col: name of quantity\n"
04756         "# $9 = Col: time (seconds since 2000-01-01T00:00Z)\n"
04757         "# $10 = Col: altitude [km]\n"
04758         "# $11 = Col: longitude [deg]\n" "# $12 = Col: latitude [deg]\n");
04759
04760     /* Get number of columns... */
04761     nc = atm2x(ctl, atm, NULL, cira, cipa);
04762 }
04763
04764 /* Write header entry... */
04765 fprintf(out, "# $13 = Matrix element\n\n");
04766
04767 /* Write matrix data... */
04768 i = j = 0;
04769 while (i < nr && j < nc) {
04770
04771     /* Write info about the row... */
04772     if (rowspan[0] == 'y')
04773         fprintf(out, "%d %g %.2f %g %g %g",
04774             (int) i, ctl->nu[rida[i]],
04775             obs->time[rira[i]], obs->vpz[rira[i]],
04776             obs->vplon[rira[i]], obs->vplat[rira[i]]);
04777     else {
04778         idx2name(ctl, rira[i], quantity);
04779         fprintf(out, "%d %s %.2f %g %g %g", (int) i, quantity,
04780             atm->time[riqa[i]], atm->z[riqa[i]],
04781             atm->lon[riqa[i]], atm->lat[riqa[i]]);
04782     }
04783
04784     /* Write info about the column... */
04785     if (colspace[0] == 'y')
04786         fprintf(out, " %d %g %.2f %g %g %g",
04787             (int) j, ctl->nu[cida[j]],
04788             obs->time[cira[j]], obs->vpz[cira[j]],
04789             obs->vplon[cira[j]], obs->vplat[cira[j]]);
04790     else {
04791         idx2name(ctl, cira[j], quantity);
04792         fprintf(out, " %d %s %.2f %g %g %g", (int) j, quantity,
04793             atm->time[cipa[j]], atm->z[cipa[j]],
04794             atm->lon[cipa[j]], atm->lat[cipa[j]]);
04795     }
04796
04797     /* Write matrix entry... */
04798     fprintf(out, " %g\n", gsl_matrix_get(matrix, i, j));
04799
04800     /* Set matrix indices... */
04801     if (sort[0] == 'r') {
04802         j++;
04803         if (j >= nc) {
04804             j = 0;
04805             i++;
04806             fprintf(out, "\n");
04807         }
04808     }

```

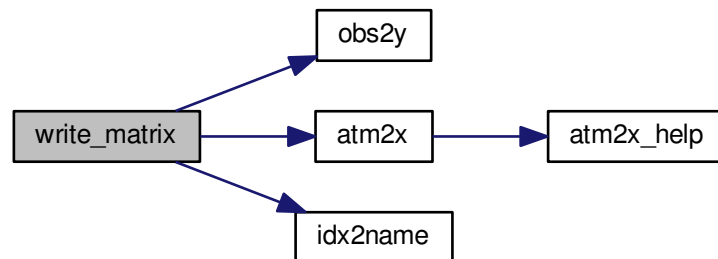


```

04808     } else {
04809         i++;
04810         if (i >= nr) {
04811             i = 0;
04812             j++;
04813             fprintf(out, "\n");
04814         }
04815     }
04816 }
04817
04818 /* Close file... */
04819 fclose(out);
04820
04821 /* Free... */
04822 free(cida);
04823 free(ciga);
04824 free(cipa);
04825 free(cira);
04826 free(rida);
04827 free(riqa);
04828 free(ripa);
04829 free(rira);
04830 }

```

Here is the call graph for this function:



**5.23.2.46** `void write_obs ( const char * dirname, const char * filename, ctl_t * ctl, obs_t * obs )`

Write observation data.

Definition at line [4834](#) of file [jurassic.c](#).

```

04838     {
04839
04840     FILE *out;
04841
04842     char file[LEN];
04843
04844     int id, ir, n = 10;
04845
04846     /* Set filename... */
04847     if (dirname != NULL)
04848         sprintf(file, "%s/%s", dirname, filename);
04849     else
04850         sprintf(file, "%s", filename);
04851
04852     /* Write info... */
04853     printf("Write observation data: %s\n", file);
04854
04855     /* Create file... */
04856     if (!(out = fopen(file, "w")))
04857         ERRMSG("Cannot create file!");
04858

```

```

04859  /* Write header... */
04860  fprintf(out,
04861          "# $1 = time (seconds since 2000-01-01T00:00Z)\n"
04862          "# $2 = observer altitude [km]\n"
04863          "# $3 = observer longitude [deg]\n"
04864          "# $4 = observer latitude [deg]\n"
04865          "# $5 = view point altitude [km]\n"
04866          "# $6 = view point longitude [deg]\n"
04867          "# $7 = view point latitude [deg]\n"
04868          "# $8 = tangent point altitude [km]\n"
04869          "# $9 = tangent point longitude [deg]\n"
04870          "# $10 = tangent point latitude [deg]\n");
04871  for (id = 0; id < ctl->nd; id++)
04872      fprintf(out, "# $d = channel %g: radiance [W/(m^2 sr cm^-1)]\n",
04873              ++n, ctl->nu[id]);
04874  for (id = 0; id < ctl->nd; id++)
04875      fprintf(out, "# $d = channel %g: transmittance\n", ++n, ctl->nu[id]);
04876
04877  /* Write data... */
04878  for (ir = 0; ir < obs->nr; ir++) {
04879      if (ir == 0 || obs->time[ir] != obs->time[ir - 1])
04880          fprintf(out, "\n");
04881      fprintf(out, "%.2f %g %g %g %g %g %g %g %g", obs->time[ir],
04882              obs->obsz[ir], obs->obslon[ir], obs->obslat[ir],
04883              obs->vpz[ir], obs->vplon[ir], obs->vplat[ir],
04884              obs->tpz[ir], obs->tplon[ir], obs->tplat[ir]);
04885      for (id = 0; id < ctl->nd; id++)
04886          fprintf(out, " %g", obs->rad[id][ir]);
04887      for (id = 0; id < ctl->nd; id++)
04888          fprintf(out, " %g", obs->tau[id][ir]);
04889      fprintf(out, "\n");
04890  }
04891
04892  /* Close file... */
04893  fclose(out);
04894 }

```

#### 5.23.2.47 void x2atm ( ctl\_t \* ctl, gsl\_vector \* x, atm\_t \* atm )

Decompose parameter vector or state vector.

Definition at line 4898 of file [jurassic.c](#).

```

04901      {
04902
04903      int ig, iw;
04904
04905      size_t n = 0;
04906
04907      /* Set pressure... */
04908      x2atm_help(atm, ctl->retp_zmin, ctl->retp_zmax, atm->
04909                p, x, &n);
04909
04910      /* Set temperature... */
04911      x2atm_help(atm, ctl->rett_zmin, ctl->rett_zmax, atm->
04912                t, x, &n);
04912
04913      /* Set volume mixing ratio... */
04914      for (ig = 0; ig < ctl->ng; ig++)
04915          x2atm_help(atm, ctl->retq_zmin[ig], ctl->retq_zmax[ig],
04916                    atm->q[ig], x, &n);
04917
04918      /* Set extinction... */
04919      for (iw = 0; iw < ctl->nw; iw++)
04920          x2atm_help(atm, ctl->retk_zmin[iw], ctl->retk_zmax[iw],
04921                    atm->k[iw], x, &n);
04922 }

```

Here is the call graph for this function:



### 5.23.2.48 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 4926 of file jurassic.c.

```

04932         {
04933
04934     int ip;
04935
04936     /* Extract state vector elements... */
04937     for (ip = 0; ip < atm->np; ip++)
04938         if (atm->z[ip] >= zmin && atm->z[ip] <= zmax) {
04939             value[ip] = gsl_vector_get(x, *n);
04940             (*n)++;
04941         }
04942 }
```

### 5.23.2.49 void y2obs ( ctl\_t \* ctl, gsl\_vector \* y, obs\_t \* obs )

Decompose measurement vector.

Definition at line 4946 of file jurassic.c.

```

04949         {
04950
04951     int id, ir;
04952
04953     size_t m = 0;
04954
04955     /* Decompose measurement vector... */
04956     for (ir = 0; ir < obs->nr; ir++)
04957         for (id = 0; id < ctl->nd; id++)
04958             if (gsl_finite(obs->rad[id][ir])) {
04959                 obs->rad[id][ir] = gsl_vector_get(y, m);
04960                 m++;
04961             }
04962 }
```

## 5.24 jurassic.h

```

00001 /*
00002  This file is part of JURASSIC.
00003
00004  JURASSIC is free software: you can redistribute it and/or modify
00005  it under the terms of the GNU General Public License as published by
00006  the Free Software Foundation, either version 3 of the License, or
00007  (at your option) any later version.
00008
00009  JURASSIC is distributed in the hope that it will be useful,
00010  but WITHOUT ANY WARRANTY; without even the implied warranty of
00011  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00012  GNU General Public License for more details.
00013
00014  You should have received a copy of the GNU General Public License
00015  along with JURASSIC. If not, see <http://www.gnu.org/licenses/>.
00016
00017  Copright (C) 2003-2015 Forschungszentrum Juelich GmbH
00018 */
00019
00034 #include <gsl/gsl_math.h>
00035 #include <gsl/gsl_blas.h>
00036 #include <gsl/gsl_linalg.h>
00037 #include <gsl/gsl_statistics.h>
00038 #include <math.h>
00039 #include <omp.h>
00040 #include <stdio.h>
00041 #include <stdlib.h>
00042 #include <string.h>
00043 #include <time.h>
00044
00045 /* -----
```

```

00046     Macros...
00047     ----- */
00048
00050 #define ALLOC(ptr, type, n)
00051     if ((ptr=malloc((size_t) (n)*sizeof(type)))==NULL)
00052         ERRMSG("Out of memory!");
00053
00055 #define DIST(a, b) sqrt(DIST2(a, b))
00056
00058 #define DIST2(a, b)
00059     ((a[0]-b[0])*(a[0]-b[0])+(a[1]-b[1])*(a[1]-b[1])+(a[2]-b[2])*(a[2]-b[2]))
00060
00062 #define DOTP(a, b) (a[0]*b[0]+a[1]*b[1]+a[2]*b[2])
00063
00065 #define ERRMSG(msg) {
00066     printf("\nError (%s, %s, l%d): %s\n\n",
00067         __FILE__, __func__, __LINE__, msg);
00068     exit(EXIT_FAILURE);
00069 }
00070
00072 #define EXP(x0, y0, x1, y1, x)
00073     ((y0>0 && (y1)>0)
00074      ? ((y0)*exp(log((y1)/(y0))/((x1)-(x0))*((x)-(x0))))
00075      : LIN(x0, y0, x1, y1, x))
00076
00078 #define LIN(x0, y0, x1, y1, x)
00079     ((y0)+((y1)-(y0))/((x1)-(x0))*((x)-(x0)))
00080
00082 #define NORM(a) sqrt(DOTP(a, a))
00083
00085 #define POW2(x) ((x)*(x))
00086
00088 #define POW3(x) ((x)*(x)*(x))
00089
00091 #define PRINT(format, var)
00092     printf("Print (%s, %s, l%d): %s= "format"\n",
00093         __FILE__, __func__, __LINE__, #var, var);
00094
00096 #define TIMER(name, mode)
00097     {timer(name, __FILE__, __func__, __LINE__, mode);}
00098
00100 #define TOK(line, tok, format, var) {
00101     if((tok)=strtok((line), " \t")) {
00102         if(sscanf(tok, format, &(var))!=1) continue;
00103     } else ERRMSG("Error while reading!");
00104 }
00105
00106 /* -----
00107     Constants...
00108     ----- */
00109
00111 #define TMIN 100.
00112
00114 #define TMAX 400.
00115
00117 #define C1 1.19104259e-8
00118
00120 #define C2 1.43877506
00121
00123 #define G0 9.80665
00124
00126 #define KB 1.3806504e-23
00127
00129 #define NA 6.02214199e23
00130
00132 #define H0 7.0
00133
00135 #define P0 1013.25
00136
00138 #define T0 273.15
00139
00141 #define RE 6367.421
00142
00144 #define RI 8.3144598
00145
00147 #define ME 5.976e24
00148
00149 /* -----
00150     Dimensions...
00151     ----- */
00152
00154 #define ND 50
00155
00157 #define NG 20
00158
00160 #define NP 1000
00161

```

```

00163 #define NR 1000
00164
00166 #define NW 5
00167
00169 #define LEN 5000
00170
00172 #define M (NR*ND)
00173
00175 #define N (NQ*NP)
00176
00178 #define NQ (2+NG+NW)
00179
00181 #define NLOS 1000
00182
00184 #define NSHAPE 10000
00185
00187 #define NFOV 5
00188
00190 #define TBLNP 41
00191
00193 #define TBLNT 30
00194
00196 #define TBLNU 320
00197
00199 #define TBLNS 1200
00200
00201 /* -----
00202     Quantity indices...
00203     ----- */
00204
00206 #define IDXP 0
00207
00209 #define IDXT 1
00210
00212 #define IDXQ(ig) (2+ig)
00213
00215 #define ID XK(iw) (2+ctl->ng+iw)
00216
00217 /* -----
00218     Structs...
00219     ----- */
00220
00222 typedef struct {
00223
00225     int np;
00226
00228     double time[NP];
00229
00231     double z[NP];
00232
00234     double lon[NP];
00235
00237     double lat[NP];
00238
00240     double p[NP];
00241
00243     double t[NP];
00244
00246     double q[NG][NP];
00247
00249     double k[NW][NP];
00250 } atm_t;
00251
00252
00254 typedef struct {
00255
00257     int ng;
00258
00260     char emitter[NG][LEN];
00261
00263     int nd;
00264
00266     int nw;
00267
00269     double nu[ND];
00270
00272     int window[ND];
00273
00275     char tblbase[LEN];
00276
00278     double hyd;
00279
00281     int ctm_co2;
00282
00284     int ctm_h2o;
00285
00287     int ctm_n2;

```

```
00288
00290     int   ctm_o2;
00291
00293     int   refrac;
00294
00296     double rayds;
00297
00299     double raydz;
00300
00302     char  fov[LEN];
00303
00305     double retp_zmin;
00306
00308     double retp_zmax;
00309
00311     double rett_zmin;
00312
00314     double rett_zmax;
00315
00317     double retq_zmin[NG];
00318
00320     double retq_zmax[NG];
00321
00323     double retk_zmin[NW];
00324
00326     double retk_zmax[NW];
00327
00329     int   write_bbt;
00330
00332     int   write_matrix;
00333
00334 }   ctl_t;
00335
00337 typedef struct {
00338
00340     int   np;
00341
00343     double z[NLOS];
00344
00346     double lon[NLOS];
00347
00349     double lat[NLOS];
00350
00352     double p[NLOS];
00353
00355     double t[NLOS];
00356
00358     double q[NG][NLOS];
00359
00361     double k[NW][NLOS];
00362
00364     double tsurf;
00365
00367     double ds[NLOS];
00368
00370     double u[NG][NLOS];
00371
00372 }   los_t;
00373
00375 typedef struct {
00376
00378     int   nr;
00379
00381     double time[NR];
00382
00384     double obsz[NR];
00385
00387     double obslon[NR];
00388
00390     double obslat[NR];
00391
00393     double vpz[NR];
00394
00396     double vplon[NR];
00397
00399     double vplat[NR];
00400
00402     double tpz[NR];
00403
00405     double tplon[NR];
00406
00408     double tplat[NR];
00409
00411     double tau[ND][NR];
00412
00414     double rad[ND][NR];
00415
```

```

00416 } obs_t;
00417
00419 typedef struct {
00420
00422     int np[NG][ND];
00423
00425     int nt[NG][ND][TBLNP];
00426
00428     int nu[NG][ND][TBLNP][TBLNT];
00429
00431     double p[NG][ND][TBLNP];
00432
00434     double t[NG][ND][TBLNP][TBLNT];
00435
00437     float u[NG][ND][TBLNP][TBLNT][TBLNU];
00438
00440     float eps[NG][ND][TBLNP][TBLNT][TBLNU];
00441
00443     double st[TBLNS];
00444
00446     double sr[ND][TBLNS];
00447 } tbl_t;
00448
00449
00450 /* -----
00451     Functions...
00452     ----- */
00453
00455 size_t atm2x(
00456     ctl_t * ctl,
00457     atm_t * atm,
00458     gsl_vector * x,
00459     int *iqa,
00460     int *ipa);
00461
00463 void atm2x_help(
00464     atm_t * atm,
00465     double zmin,
00466     double zmax,
00467     double *value,
00468     int val_iqa,
00469     gsl_vector * x,
00470     int *iqa,
00471     int *ipa,
00472     size_t * n);
00473
00475 double brightness(
00476     double rad,
00477     double nu);
00478
00480 void cart2geo(
00481     double *x,
00482     double *z,
00483     double *lon,
00484     double *lat);
00485
00487 void climatology(
00488     ctl_t * ctl,
00489     atm_t * atm_mean);
00490
00492 double ctmc02(
00493     double nu,
00494     double p,
00495     double t,
00496     double u);
00497
00499 double ctmh2o(
00500     double nu,
00501     double p,
00502     double t,
00503     double q,
00504     double u);
00505
00507 double ctmn2(
00508     double nu,
00509     double p,
00510     double t);
00511
00513 double ctmo2(
00514     double nu,
00515     double p,
00516     double t);
00517
00519 void copy_atm(
00520     ctl_t * ctl,
00521     atm_t * atm_dest,
00522     atm_t * atm_src,

```

```
00523     int init);
00524
00526 void copy_obs(
00527     ctl_t * ctl,
00528     obs_t * obs_dest,
00529     obs_t * obs_src,
00530     int init);
00531
00533 int find_emitter(
00534     ctl_t * ctl,
00535     const char *emitter);
00536
00538 void formod(
00539     ctl_t * ctl,
00540     atm_t * atm,
00541     obs_t * obs);
00542
00544 void formod_continua(
00545     ctl_t * ctl,
00546     los_t * los,
00547     int ip,
00548     double *beta);
00549
00551 void formod_fov(
00552     ctl_t * ctl,
00553     obs_t * obs);
00554
00556 void formod_pencil(
00557     ctl_t * ctl,
00558     atm_t * atm,
00559     obs_t * obs,
00560     int ir);
00561
00563 void formod_srcfunc(
00564     ctl_t * ctl,
00565     tbl_t * tbl,
00566     double t,
00567     double *src);
00568
00570 void geo2cart(
00571     double z,
00572     double lon,
00573     double lat,
00574     double *x);
00575
00577 void hydrostatic(
00578     ctl_t * ctl,
00579     atm_t * atm);
00580
00582 void idx2name(
00583     ctl_t * ctl,
00584     int idx,
00585     char *quantity);
00586
00588 void init_tbl(
00589     ctl_t * ctl,
00590     tbl_t * tbl);
00591
00593 void intpol_atm(
00594     ctl_t * ctl,
00595     atm_t * atm,
00596     double z,
00597     double *p,
00598     double *t,
00599     double *q,
00600     double *k);
00601
00603 void intpol_tbl(
00604     ctl_t * ctl,
00605     tbl_t * tbl,
00606     los_t * los,
00607     int ip,
00608     double tau_path[NG][ND],
00609     double tau_seg[ND]);
00610
00612 double intpol_tbl_eps(
00613     tbl_t * tbl,
00614     int ig,
00615     int id,
00616     int ip,
00617     int it,
00618     double u);
00619
00621 double intpol_tbl_u(
00622     tbl_t * tbl,
00623     int ig,
00624     int id,
```



```
00625     int ip,
00626     int it,
00627     double eps);
00628
00630 void jsec2time(
00631     double jsec,
00632     int *year,
00633     int *mon,
00634     int *day,
00635     int *hour,
00636     int *min,
00637     int *sec,
00638     double *remain);
00639
00641 void kernel(
00642     ctl_t * ctl,
00643     atm_t * atm,
00644     obs_t * obs,
00645     gsl_matrix * k);
00646
00648 int locate_irr(
00649     double **xx,
00650     int n,
00651     double x);
00652
00654 int locate_reg(
00655     double **xx,
00656     int n,
00657     double x);
00658
00660 int locate_tbl(
00661     float **xx,
00662     int n,
00663     double x);
00664
00666 size_t obs2y(
00667     ctl_t * ctl,
00668     obs_t * obs,
00669     gsl_vector * y,
00670     int *ida,
00671     int *ira);
00672
00674 double planck(
00675     double t,
00676     double nu);
00677
00679 void raytrace(
00680     ctl_t * ctl,
00681     atm_t * atm,
00682     obs_t * obs,
00683     los_t * los,
00684     int ir);
00685
00687 void read_atm(
00688     const char *dirname,
00689     const char *filename,
00690     ctl_t * ctl,
00691     atm_t * atm);
00692
00694 void read_ctl(
00695     int argc,
00696     char *argv[],
00697     ctl_t * ctl);
00698
00700 void read_matrix(
00701     const char *dirname,
00702     const char *filename,
00703     gsl_matrix * matrix);
00704
00706 void read_obs(
00707     const char *dirname,
00708     const char *filename,
00709     ctl_t * ctl,
00710     obs_t * obs);
00711
00713 void read_shape(
00714     const char *filename,
00715     double *x,
00716     double *y,
00717     int *n);
00718
00720 double refractivity(
00721     double p,
00722     double t);
00723
00725 double scan_ctl(
00726     int argc,
```

```

00727     char *argv[],
00728     const char *varname,
00729     int arridx,
00730     const char *defvalue,
00731     char *value);
00732
00733 void tangent_point(
00734     los_t * los,
00735     double *tpz,
00736     double *tplon,
00737     double *tplat);
00738
00739 void time2jsec(
00740     int year,
00741     int mon,
00742     int day,
00743     int hour,
00744     int min,
00745     int sec,
00746     double remain,
00747     double *jsec);
00748
00749 void timer(
00750     const char *name,
00751     const char *file,
00752     const char *func,
00753     int line,
00754     int mode);
00755
00756 void write_atm(
00757     const char *dirname,
00758     const char *filename,
00759     ctl_t * ctl,
00760     atm_t * atm);
00761
00762 void write_matrix(
00763     const char *dirname,
00764     const char *filename,
00765     ctl_t * ctl,
00766     gsl_matrix * matrix,
00767     atm_t * atm,
00768     obs_t * obs,
00769     const char *rowsep,
00770     const char *colsep,
00771     const char *sort);
00772
00773 void write_obs(
00774     const char *dirname,
00775     const char *filename,
00776     ctl_t * ctl,
00777     obs_t * obs);
00778
00779 void x2atm(
00780     ctl_t * ctl,
00781     gsl_vector * x,
00782     atm_t * atm);
00783
00784 void x2atm_help(
00785     atm_t * atm,
00786     double zmin,
00787     double zmax,
00788     double *value,
00789     gsl_vector * x,
00790     size_t * n);
00791
00792 void y2obs(
00793     ctl_t * ctl,
00794     gsl_vector * y,
00795     obs_t * obs);

```

## 5.25 libairs.c File Reference

### Functions

- void [add\\_att](#) (int ncid, int varid, const char \*unit, const char \*long\_name)  
*Add variable attributes to netCDF file.*
- 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 \*fclmag, 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.*

## 5.25.1 Function Documentation

### 5.25.1.1 void add\_att ( int ncid, int varid, const char \* unit, const char \* long\_name )

Add variable attributes to netCDF file.

Definition at line 5 of file `libairs.c`.

```
00009             {
00010
00011         /* Set long name... */
00012         NC(nc_put_att_text(ncid, varid, "long_name", strlen(long_name), long_name));
00013
00014         /* Set units... */
00015         NC(nc_put_att_text(ncid, varid, "units", strlen(unit), unit));
00016     }
```

### 5.25.1.2 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 20 of file `libairs.c`.

```
00028     {
00029
00030         /* Check if variable exists... */
00031         if (nc_inq_varid(ncid, varname, varid) != NC_NOERR) {
00032
00033             /* Define variable... */
00034             NC(nc_def_var(ncid, varname, type, ndims, dimid, varid));
00035
00036             /* Set long name... */
00037             NC(nc_put_att_text
00038                (ncid, *varid, "long_name", strlen(longname), longname));
00039
00040             /* Set units... */
00041             NC(nc_put_att_text(ncid, *varid, "units", strlen(unit), unit));
00042         }
00043     }
```

### 5.25.1.3 void background\_poly\_help ( double \* xx, double \* yy, int n, int dim )

Get background based on polynomial fits.

Definition at line 47 of file [libairs.c](#).

```

00051         {
00052
00053     gsl_multifit_linear_workspace *work;
00054     gsl_matrix *cov, *X;
00055     gsl_vector *c, *x, *y;
00056
00057     double chisq, xx2[WX > WY ? WX : WY], yy2[WX > WY ? WX : WY];
00058
00059     size_t i, i2, n2 = 0;
00060
00061     /* Check for nan... */
00062     for (i = 0; i < (size_t) n; i++)
00063         if (gsl_finite(yy[i])) {
00064             xx2[n2] = xx[i];
00065             yy2[n2] = yy[i];
00066             n2++;
00067         }
00068     if ((int) n2 < dim || n2 < 0.9 * n) {
00069         for (i = 0; i < (size_t) n; i++)
00070             yy[i] = GSL_NAN;
00071         return;
00072     }
00073
00074     /* Allocate... */
00075     work = gsl_multifit_linear_alloc((size_t) n2, (size_t) dim);
00076     cov = gsl_matrix_alloc((size_t) dim, (size_t) dim);
00077     X = gsl_matrix_alloc((size_t) n2, (size_t) dim);
00078     c = gsl_vector_alloc((size_t) dim);
00079     x = gsl_vector_alloc((size_t) n2);
00080     y = gsl_vector_alloc((size_t) n2);
00081
00082     /* Compute polynomial fit... */
00083     for (i = 0; i < (size_t) n2; i++) {
00084         gsl_vector_set(x, i, xx2[i]);
00085         gsl_vector_set(y, i, yy2[i]);
00086         for (i2 = 0; i2 < (size_t) dim; i2++)
00087             gsl_matrix_set(X, i, i2, pow(gsl_vector_get(x, i), (double) i2));
00088     }
00089     gsl_multifit_linear(X, y, c, cov, &chisq, work);
00090     for (i = 0; i < (size_t) n; i++)
00091         yy[i] = gsl_poly_eval(c->data, (int) dim, xx[i]);
00092
00093     /* Free... */
00094     gsl_multifit_linear_free(work);
00095     gsl_matrix_free(cov);
00096     gsl_matrix_free(X);
00097     gsl_vector_free(c);
00098     gsl_vector_free(x);
00099     gsl_vector_free(y);
00100 }

```

### 5.25.1.4 void background\_poly ( wave\_t \* wave, int dim\_x, int dim\_y )

Get background based on polynomial fits.

Definition at line 104 of file [libairs.c](#).

```

00107         {
00108
00109     double x[WX], x2[WY], y[WX], y2[WY];
00110
00111     int ix, iy;
00112
00113     /* Copy temperatures to background... */
00114     for (ix = 0; ix < wave->nx; ix++)
00115         for (iy = 0; iy < wave->ny; iy++) {
00116             wave->bg[ix][iy] = wave->temp[ix][iy];
00117             wave->pt[ix][iy] = 0;
00118         }
00119 }

```

```

00120  /* Check parameters... */
00121  if (dim_x <= 0 && dim_y <= 0)
00122      return;
00123
00124  /* Compute fit in x-direction... */
00125  if (dim_x > 0)
00126      for (iy = 0; iy < wave->ny; iy++) {
00127          for (ix = 0; ix < wave->nx; ix++) {
00128              x[ix] = (double) ix;
00129              y[ix] = wave->bg[ix][iy];
00130          }
00131          background_poly_help(x, y, wave->nx, dim_x);
00132          for (ix = 0; ix < wave->nx; ix++)
00133              wave->bg[ix][iy] = y[ix];
00134      }
00135
00136  /* Compute fit in y-direction... */
00137  if (dim_y > 0)
00138      for (ix = 0; ix < wave->nx; ix++) {
00139          for (iy = 0; iy < wave->ny; iy++) {
00140              x2[iy] = (int) iy;
00141              y2[iy] = wave->bg[ix][iy];
00142          }
00143          background_poly_help(x2, y2, wave->ny, dim_y);
00144          for (iy = 0; iy < wave->ny; iy++)
00145              wave->bg[ix][iy] = y2[iy];
00146      }
00147
00148  /* Recompute perturbations... */
00149  for (ix = 0; ix < wave->nx; ix++)
00150      for (iy = 0; iy < wave->ny; iy++)
00151          wave->pt[ix][iy] = wave->temp[ix][iy] - wave->bg[ix][iy];
00152 }

```

Here is the call graph for this function:



#### 5.25.1.5 void background\_smooth ( wave\_t \* wave, int npts\_x, int npts\_y )

Smooth background.

Definition at line 156 of file libairs.c.

```

00159      {
00160
00161          static double help[WX][WY], dmax = 2500.;
00162
00163          int dx, dy, i, j, ix, iy, n;
00164
00165          /* Check parameters... */
00166          if (npts_x <= 0 && npts_y <= 0)
00167              return;
00168
00169          /* Smooth background... */
00170          for (ix = 0; ix < wave->nx; ix++)
00171              for (iy = 0; iy < wave->ny; iy++) {
00172
00173                  /* Init... */
00174                  n = 0;
00175                  help[ix][iy] = 0;
00176
00177                  /* Set maximum range... */
00178                  dx = GSL_MIN(GSL_MIN(npts_x, ix), wave->nx - 1 - ix);

```

```

00179     dy = GSL_MIN(GSL_MIN(npts_y, iy), wave->ny - 1 - iy);
00180
00181     /* Average... */
00182     for (i = ix - dx; i <= ix + dx; i++)
00183         for (j = iy - dy; j <= iy + dy; j++)
00184             if (fabs(wave->x[ix] - wave->x[i]) < dmax &&
00185                 fabs(wave->y[iy] - wave->y[j]) < dmax) {
00186                 help[ix][iy] += wave->bg[i][j];
00187                 n++;
00188             }
00189
00190     /* Normalize... */
00191     if (n > 0)
00192         help[ix][iy] /= n;
00193     else
00194         help[ix][iy] = GSL_NAN;
00195 }
00196
00197 /* Recalculate perturbations... */
00198 for (ix = 0; ix < wave->nx; ix++)
00199     for (iy = 0; iy < wave->ny; iy++) {
00200         wave->bg[ix][iy] = help[ix][iy];
00201         wave->pt[ix][iy] = wave->temp[ix][iy] - wave->bg[ix][iy];
00202     }
00203 }

```

#### 5.25.1.6 void create\_background ( wave\_t \* wave )

Set background...

Definition at line 207 of file [libairs.c](#).

```

00208     {
00209
00210     int ix, iy;
00211
00212     /* Loop over grid points... */
00213     for (ix = 0; ix < wave->nx; ix++)
00214         for (iy = 0; iy < wave->ny; iy++) {
00215
00216             /* Set background for 4.3 micron BT measurements... */
00217             wave->bg[ix][iy] = 235.626 + 5.38165e-6 * gsl_pow_2(wave->x[ix]
00218                                     -
00219                                     0.5 * (wave->x[0] +
00220                                             wave->x
00221                                             [wave->nx -
00222                                             1]))
00223                 - 1.78519e-12 * gsl_pow_4(wave->x[ix] -
00224                                             0.5 * (wave->x[0] + wave->x[wave->nx - 1]));
00225
00226             /* Set temperature perturbation... */
00227             wave->pt[ix][iy] = 0;
00228
00229             /* Set temperature... */
00230             wave->temp[ix][iy] = wave->bg[ix][iy];
00231         }
00232 }

```

#### 5.25.1.7 void create\_noise ( wave\_t \* wave, double nedt )

Add noise to perturbations and temperatures...

Definition at line 236 of file [libairs.c](#).

```

00238     {
00239
00240     gsl_rng *r;
00241
00242     int ix, iy;
00243
00244     /* Initialize random number generator... */
00245     gsl_rng_env_setup();
00246     r = gsl_rng_alloc(gsl_rng_default);
00247     gsl_rng_set(r, (unsigned long int) time(NULL));

```

```

00248
00249  /* Add noise to temperature... */
00250  if (nedt > 0)
00251      for (ix = 0; ix < wave->nx; ix++)
00252          for (iy = 0; iy < wave->ny; iy++)
00253              wave->temp[ix][iy] += gsl_ran_gaussian(r, nedt);
00254
00255  /* Free... */
00256  gsl_rng_free(r);
00257 }

```

#### 5.25.1.8 void create\_wave ( wave\_t \* wave, double amp, double lx, double ly, double phi, double fwhm )

Add linear wave pattern...

Definition at line 261 of file [libairs.c](#).

```

00267      {
00268
00269      int ix, iy;
00270
00271      /* Loop over grid points... */
00272      for (ix = 0; ix < wave->nx; ix++)
00273          for (iy = 0; iy < wave->ny; iy++) {
00274
00275              /* Set wave perturbation... */
00276              wave->pt[ix][iy] = amp * cos((lx != 0 ? 2 * M_PI / lx : 0) * wave->x[ix]
00277                                          + (ly !=
00278                                              0 ? 2 * M_PI / ly : 0) * wave->y[iy]
00279                                              - phi * M_PI / 180.)
00280              * (fwhm > 0 ? exp(-0.5 * gsl_pow_2((wave->x[ix]) / (lx * fwhm) * 2.35)
00281                          -
00282                          0.5 * gsl_pow_2((wave->y[iy]) / (ly * fwhm) *
00283                          2.35)) : 1.0);
00284
00285              /* Add perturbation to temperature... */
00286              wave->temp[ix][iy] += wave->pt[ix][iy];
00287          }
00288 }

```

#### 5.25.1.9 void day2doy ( int year, int mon, int day, int \* doy )

Get day of year from date.

Definition at line 292 of file [libairs.c](#).

```

00296      {
00297
00298      int d0[12] = { 1, 32, 60, 91, 121, 152, 182, 213, 244, 274, 305, 335 };
00299      int d0l[12] = { 1, 32, 61, 92, 122, 153, 183, 214, 245, 275, 306, 336 };
00300
00301      /* Get day of year... */
00302      if (year % 400 == 0 || (year % 100 != 0 && year % 4 == 0))
00303          *doy = d0l[mon - 1] + day - 1;
00304      else
00305          *doy = d0[mon - 1] + day - 1;
00306 }

```



#### 5.25.1.10 void doy2day ( int year, int doy, int \* mon, int \* day )

Get date from day of year.

Definition at line 310 of file [libairs.c](#).

```

00314         {
00315
00316     int d0[12] = { 1, 32, 60, 91, 121, 152, 182, 213, 244, 274, 305, 335 };
00317     int d0l[12] = { 1, 32, 61, 92, 122, 153, 183, 214, 245, 275, 306, 336 };
00318     int i;
00319
00320     /* Get month and day... */
00321     if (year % 400 == 0 || (year % 100 != 0 && year % 4 == 0)) {
00322         for (i = 11; i >= 0; i--)
00323             if (d0l[i] <= doy)
00324                 break;
00325         *mon = i + 1;
00326         *day = doy - d0l[i] + 1;
00327     } else {
00328         for (i = 11; i >= 0; i--)
00329             if (d0[i] <= doy)
00330                 break;
00331         *mon = i + 1;
00332         *day = doy - d0[i] + 1;
00333     }
00334 }

```

#### 5.25.1.11 void fft\_help ( double \* fcReal, double \* fclmag, int n )

Calculate 1-D FFT...

Definition at line 338 of file [libairs.c](#).

```

00341         {
00342
00343     gsl_fft_complex_wavetable *wavetable;
00344     gsl_fft_complex_workspace *workspace;
00345
00346     double data[2 * PMAX];
00347
00348     int i;
00349
00350     /* Check size... */
00351     if (n > PMAX)
00352         ERRMSG("Too many data points!");
00353
00354     /* Allocate... */
00355     wavetable = gsl_fft_complex_wavetable_alloc((size_t) n);
00356     workspace = gsl_fft_complex_workspace_alloc((size_t) n);
00357
00358     /* Set data (real, complex)... */
00359     for (i = 0; i < n; i++) {
00360         data[2 * i] = fcReal[i];
00361         data[2 * i + 1] = fcImag[i];
00362     }
00363
00364     /* Calculate FFT... */
00365     gsl_fft_complex_forward(data, 1, (size_t) n, wavetable, workspace);
00366
00367     /* Copy data... */
00368     for (i = 0; i < n; i++) {
00369         fcReal[i] = data[2 * i];
00370         fcImag[i] = data[2 * i + 1];
00371     }
00372
00373     /* Free... */
00374     gsl_fft_complex_wavetable_free(wavetable);
00375     gsl_fft_complex_workspace_free(workspace);
00376 }

```

**5.25.1.12** void `fft ( wave_t * wave, double * Amax, double * phimax, double * lhmax, double * alphamax, double * betamax, char * filename )`

Calculate 2-D FFT...

Definition at line 380 of file [libairs.c](#).

```

00387         {
00388
00389     static double A[PMAX][PMAX], phi[PMAX][PMAX], kx[PMAX], ky[PMAX],
00390         kxmax, kymax, cutReal[PMAX], cutImag[PMAX],
00391         boxImag[PMAX][PMAX], boxReal[PMAX][PMAX];
00392
00393     FILE *out;
00394
00395     int i, i2, imin, imax, j, j2, jmin, jmax, nx, ny;
00396
00397     /* Find box... */
00398     imin = jmin = 9999;
00399     imax = jmax = -9999;
00400     for (i = 0; i < wave->nx; i++)
00401         for (j = 0; j < wave->ny; j++)
00402             if (gsl_finite(wave->var[i][j])) {
00403                 imin = GSL_MIN(imin, i);
00404                 imax = GSL_MAX(imax, i);
00405                 jmin = GSL_MIN(jmin, j);
00406                 jmax = GSL_MAX(jmax, j);
00407             }
00408     nx = imax - imin + 1;
00409     ny = jmax - jmin + 1;
00410
00411     /* Copy data... */
00412     for (i = imin; i <= imax; i++)
00413         for (j = jmin; j <= jmax; j++) {
00414             if (gsl_finite(wave->pt[i][j]))
00415                 boxReal[i - imin][j - jmin] = wave->pt[i][j];
00416             else
00417                 boxReal[i - imin][j - jmin] = 0.0;
00418             boxImag[i - imin][j - jmin] = 0.0;
00419         }
00420
00421     /* FFT of the rows... */
00422     for (i = 0; i < nx; i++) {
00423         for (j = 0; j < ny; j++) {
00424             cutReal[j] = boxReal[i][j];
00425             cutImag[j] = boxImag[i][j];
00426         }
00427         fft_help(cutReal, cutImag, ny);
00428         for (j = 0; j < ny; j++) {
00429             boxReal[i][j] = cutReal[j];
00430             boxImag[i][j] = cutImag[j];
00431         }
00432     }
00433
00434     /* FFT of the columns... */
00435     for (j = 0; j < ny; j++) {
00436         for (i = 0; i < nx; i++) {
00437             cutReal[i] = boxReal[i][j];
00438             cutImag[i] = boxImag[i][j];
00439         }
00440         fft_help(cutReal, cutImag, nx);
00441         for (i = 0; i < nx; i++) {
00442             boxReal[i][j] = cutReal[i];
00443             boxImag[i][j] = cutImag[i];
00444         }
00445     }
00446
00447     /* Get frequencies, amplitude, and phase... */
00448     for (i = 0; i < nx; i++)
00449         kx[i] = 2. * M_PI * ((i < nx / 2) ? (double) i : -(double) (nx - i))
00450         / (nx * fabs(wave->x[imax] - wave->x[imin]) / (nx - 1.0));
00451     for (j = 0; j < ny; j++)
00452         ky[j] = 2. * M_PI * ((j < ny / 2) ? (double) j : -(double) (ny - j))
00453         / (ny * fabs(wave->y[jmax] - wave->y[jmin]) / (ny - 1.0));
00454     for (i = 0; i < nx; i++)
00455         for (j = 0; j < ny; j++) {
00456             A[i][j]
00457                 = (i == 0 && j == 0 ? 1.0 : 2.0) / (nx * ny)
00458                 * sqrt(gsl_pow_2(boxReal[i][j]) + gsl_pow_2(boxImag[i][j]));
00459             phi[i][j]
00460                 = 180. / M_PI * atan2(boxImag[i][j], boxReal[i][j]);
00461         }
00462

```

```

00463  /* Check frequencies... */
00464  for (i = 0; i < nx; i++)
00465      for (j = 0; j < ny; j++)
00466          if (kx[i] == 0 || ky[j] == 0) {
00467              A[i][j] = GSL_NAN;
00468              phi[i][j] = GSL_NAN;
00469          }
00470
00471  /* Find maximum... */
00472  *Amax = 0;
00473  for (i = 0; i < nx; i++)
00474      for (j = 0; j < ny / 2; j++)
00475          if (gsl_finite(A[i][j]) && A[i][j] > *Amax) {
00476              *Amax = A[i][j];
00477              *phimax = phi[i][j];
00478              kxmax = kx[i];
00479              kymax = ky[j];
00480              imax = i;
00481              jmax = j;
00482          }
00483
00484  /* Get horizontal wavelength... */
00485  *lhmax = 2 * M_PI / sqrt(gsl_pow_2(kxmax) + gsl_pow_2(kymax));
00486
00487  /* Get propagation direction in xy-plane... */
00488  *alphamax = 90. - 180. / M_PI * atan2(kxmax, kymax);
00489
00490  /* Get propagation direction in lon,lat-plane... */
00491  *betamax = *alphamax
00492  +
00493  180. / M_PI *
00494  atan2(wave->lat[wave->nx / 2 >
00495          0 ? wave->nx / 2 - 1 : wave->nx / 2][wave->ny / 2]
00496      - wave->lat[wave->nx / 2 <
00497          wave->nx - 1 ? wave->nx / 2 +
00498          1 : wave->nx / 2][wave->ny / 2],
00499      wave->lon[wave->nx / 2 >
00500          0 ? wave->nx / 2 - 1 : wave->nx / 2][wave->ny / 2]
00501      - wave->lon[wave->nx / 2 <
00502          wave->nx - 1 ? wave->nx / 2 +
00503          1 : wave->nx / 2][wave->ny / 2]);
00504
00505  /* Save FFT data... */
00506  if (filename != NULL) {
00507
00508      /* Write info... */
00509      printf("Write FFT data: %s\n", filename);
00510
00511      /* Create file... */
00512      if (!(out = fopen(filename, "w")))
00513          ERRMSG("Cannot create file!");
00514
00515      /* Write header... */
00516      fprintf(out,
00517          "# $1 = altitude [km]\n"
00518          "# $2 = wavelength in x-direction [km]\n"
00519          "# $3 = wavelength in y-direction [km]\n"
00520          "# $4 = wavenumber in x-direction [1/km]\n"
00521          "# $5 = wavenumber in y-direction [1/km]\n"
00522          "# $6 = amplitude [K]\n" "# $7 = phase [rad]\n");
00523
00524      /* Write data... */
00525      for (i = nx - 1; i > 0; i--) {
00526          fprintf(out, "\n");
00527          for (j = ny / 2; j > 0; j--) {
00528              i2 = (i == nx / 2 ? 0 : i);
00529              j2 = (j == ny / 2 ? 0 : j);
00530              fprintf(out, "%g %g %g %g %g %g\n", wave->z,
00531                  (kx[i2] != 0 ? 2 * M_PI / kx[i2] : 0),
00532                  (ky[j2] != 0 ? 2 * M_PI / ky[j2] : 0),
00533                  kx[i2], ky[j2], A[i2][j2], phi[i2][j2]);
00534          }
00535      }
00536
00537      /* Close file... */
00538      fclose(out);
00539  }
00540 }

```

Here is the call graph for this function:



#### 5.25.1.13 void gauss ( wave\_t \* wave, double fwhm )

Apply Gaussian filter to perturbations...

Definition at line 544 of file [libairs.c](#).

```

00546         {
00547
00548     static double d2, help[WX][WY], sigma2, w, wsum;
00549
00550     int ix, ix2, iy, iy2;
00551
00552     /* Check parameters... */
00553     if (fwhm <= 0)
00554         return;
00555
00556     /* Compute sigma^2... */
00557     sigma2 = gsl_pow_2(fwhm / 2.3548);
00558
00559     /* Loop over data points... */
00560     for (ix = 0; ix < wave->nx; ix++)
00561         for (iy = 0; iy < wave->ny; iy++) {
00562
00563         /* Init... */
00564         wsum = 0;
00565         help[ix][iy] = 0;
00566
00567         /* Average... */
00568         for (ix2 = 0; ix2 < wave->nx; ix2++)
00569             for (iy2 = 0; iy2 < wave->ny; iy2++) {
00570                 d2 = gsl_pow_2(wave->x[ix] - wave->x[ix2])
00571                     + gsl_pow_2(wave->y[iy] - wave->y[iy2]);
00572                 if (d2 <= 9 * sigma2) {
00573                     w = exp(-d2 / (2 * sigma2));
00574                     wsum += w;
00575                     help[ix][iy] += w * wave->pt[ix2][iy2];
00576                 }
00577             }
00578
00579         /* Normalize... */
00580         wave->pt[ix][iy] = help[ix][iy] / wsum;
00581     }
00582 }
  
```

#### 5.25.1.14 void hamming ( wave\_t \* wave, int nit )

Apply Hamming filter to perturbations...

Definition at line 586 of file [libairs.c](#).

```

00588         {
00589
00590     static double help[WX][WY];
00591
00592     int iter, ix, iy;
00593
  
```

```

00594  /* Iterations... */
00595  for (iter = 0; iter < niter; iter++) {
00596
00597      /* Filter in x direction... */
00598      for (ix = 0; ix < wave->nx; ix++)
00599          for (iy = 0; iy < wave->ny; iy++)
00600              help[ix][iy]
00601                  = 0.23 * wave->pt[ix > 0 ? ix - 1 : ix][iy]
00602                    + 0.54 * wave->pt[ix][iy]
00603                    + 0.23 * wave->pt[ix < wave->nx - 1 ? ix + 1 : ix][iy];
00604
00605      /* Filter in y direction... */
00606      for (ix = 0; ix < wave->nx; ix++)
00607          for (iy = 0; iy < wave->ny; iy++)
00608              wave->pt[ix][iy]
00609                  = 0.23 * help[ix][iy > 0 ? iy - 1 : iy]
00610                    + 0.54 * help[ix][iy]
00611                    + 0.23 * help[ix][iy < wave->ny - 1 ? iy + 1 : iy];
00612  }
00613 }

```

#### 5.25.1.15 void intpol\_x ( wave\_t \* wave, int n )

Interpolate to regular grid in x-direction.

Definition at line 617 of file [libairs.c](#).

```

00619  {
00620
00621      gsl_interp_accel *acc;
00622      gsl_spline *spline;
00623
00624      double dummy, x[WX], xc[WX][3], xc2[WX][3], y[WX];
00625
00626      int i, ic, ix, iy;
00627
00628      /* Check parameters... */
00629      if (n <= 0)
00630          return;
00631      if (n > WX)
00632          ERRMSG("Too many data points!");
00633
00634      /* Set new x-coordinates... */
00635      for (i = 0; i < n; i++)
00636          x[i] = LIN(0.0, wave->x[0], n - 1.0, wave->x[wave->nx - 1], i);
00637
00638      /* Allocate... */
00639      acc = gsl_interp_accel_alloc();
00640      spline = gsl_spline_alloc(gsl_interp_cspline, (size_t) wave->nx);
00641
00642      /* Loop over scans... */
00643      for (iy = 0; iy < wave->ny; iy++) {
00644
00645          /* Interpolate Cartesian coordinates... */
00646          for (ix = 0; ix < wave->nx; ix++)
00647              geo2cart(0, wave->lon[ix][iy], wave->lat[ix][iy], xc[ix]);
00648          for (ic = 0; ic < 3; ic++) {
00649              for (ix = 0; ix < wave->nx; ix++)
00650                  y[ix] = xc[ix][ic];
00651              gsl_spline_init(spline, wave->x, y, (size_t) wave->nx);
00652              for (i = 0; i < n; i++)
00653                  xc2[i][ic] = gsl_spline_eval(spline, x[i], acc);
00654          }
00655          for (i = 0; i < n; i++)
00656              cart2geo(xc2[i], &dummy, &wave->lon[i][iy], &wave->lat[i][iy]);
00657
00658          /* Interpolate temperature... */
00659          for (ix = 0; ix < wave->nx; ix++)
00660              y[ix] = wave->temp[ix][iy];
00661          gsl_spline_init(spline, wave->x, y, (size_t) wave->nx);
00662          for (i = 0; i < n; i++)
00663              wave->temp[i][iy] = gsl_spline_eval(spline, x[i], acc);
00664
00665          /* Interpolate background... */
00666          for (ix = 0; ix < wave->nx; ix++)
00667              y[ix] = wave->bg[ix][iy];
00668          gsl_spline_init(spline, wave->x, y, (size_t) wave->nx);
00669          for (i = 0; i < n; i++)
00670              wave->bg[i][iy] = gsl_spline_eval(spline, x[i], acc);
00671

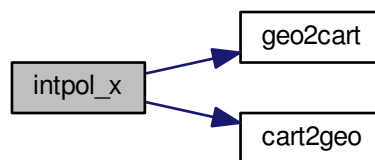
```

```

00672     /* Interpolate perturbations... */
00673     for (ix = 0; ix < wave->nx; ix++)
00674         y[ix] = wave->pt[ix][iy];
00675     gsl_spline_init(spline, wave->x, y, (size_t) wave->nx);
00676     for (i = 0; i < n; i++)
00677         wave->pt[i][iy] = gsl_spline_eval(spline, x[i], acc);
00678
00679     /* Interpolate variance... */
00680     for (ix = 0; ix < wave->nx; ix++)
00681         y[ix] = wave->var[ix][iy];
00682     gsl_spline_init(spline, wave->x, y, (size_t) wave->nx);
00683     for (i = 0; i < n; i++)
00684         wave->var[i][iy] = gsl_spline_eval(spline, x[i], acc);
00685 }
00686
00687 /* Free... */
00688 gsl_spline_free(spline);
00689 gsl_interp_accel_free(acc);
00690
00691 /* Set new x-coordinates... */
00692 for (i = 0; i < n; i++)
00693     wave->x[i] = x[i];
00694 wave->nx = n;
00695 }

```

Here is the call graph for this function:



#### 5.25.1.16 void median ( wave\_t \* wave, int dx )

Apply median filter to perturbations...

Definition at line 699 of file `libairs.c`.

```

00701     {
00702
00703     static double data[WX * WY], help[WX][WY];
00704
00705     int ix, ix2, iy, iy2;
00706
00707     size_t n;
00708
00709     /* Check parameters... */
00710     if (dx <= 0)
00711         return;
00712
00713     /* Loop over data points... */
00714     for (ix = 0; ix < wave->nx; ix++)
00715         for (iy = 0; iy < wave->ny; iy++) {
00716
00717             /* Init... */
00718             n = 0;
00719
00720             /* Get data... */
00721             for (ix2 = GSL_MAX(ix - dx, 0); ix2 < GSL_MIN(ix + dx, wave->nx - 1);
00722                  ix2++)
00723                 for (iy2 = GSL_MAX(iy - dx, 0); iy2 < GSL_MIN(iy + dx, wave->ny - 1);
00724                      iy2++) {
00725                     data[n] = wave->pt[ix2][iy2];

```

```

00726         n++;
00727     }
00728
00729     /* Normalize... */
00730     gsl_sort(data, 1, n);
00731     help[ix][iy] = gsl_stats_median_from_sorted_data(data, 1, n);
00732 }
00733
00734 /* Loop over data points... */
00735 for (ix = 0; ix < wave->nx; ix++)
00736     for (iy = 0; iy < wave->ny; iy++)
00737         wave->pt[ix][iy] = help[ix][iy];
00738 }

```

#### 5.25.1.17 void merge\_y( wave\_t \* wave1, wave\_t \* wave2 )

Merge wave structs in y-direction.

Definition at line 742 of file [libairs.c](#).

```

00744     {
00745
00746         double y;
00747
00748         int ix, iy;
00749
00750         /* Check data... */
00751         if (wave1->nx != wave2->nx)
00752             ERRMSG("Across-track sizes do not match!");
00753         if (wave1->ny + wave2->ny > WY)
00754             ERRMSG("Too many data points!");
00755
00756         /* Get offset in y direction... */
00757         y =
00758             wave1->y[wave1->ny - 1] + (wave1->y[wave1->ny - 1] -
00759                                     wave1->y[0]) / (wave1->ny - 1);
00760
00761         /* Merge data... */
00762         for (ix = 0; ix < wave2->nx; ix++)
00763             for (iy = 0; iy < wave2->ny; iy++) {
00764                 wave1->y[wave1->ny + iy] = y + wave2->y[iy];
00765                 wave1->lon[ix][wave1->ny + iy] = wave2->lon[ix][iy];
00766                 wave1->lat[ix][wave1->ny + iy] = wave2->lat[ix][iy];
00767                 wave1->temp[ix][wave1->ny + iy] = wave2->temp[ix][iy];
00768                 wave1->bg[ix][wave1->ny + iy] = wave2->bg[ix][iy];
00769                 wave1->pt[ix][wave1->ny + iy] = wave2->pt[ix][iy];
00770                 wave1->var[ix][wave1->ny + iy] = wave2->var[ix][iy];
00771             }
00772
00773         /* Increment counter... */
00774         wave1->ny += wave2->ny;
00775     }

```

#### 5.25.1.18 void noise( wave\_t \* wave, double \* mu, double \* sig )

Estimate noise.

Definition at line 779 of file [libairs.c](#).

```

00782     {
00783
00784         int ix, ix2, iy, iy2, n = 0, okay;
00785
00786         /* Init... */
00787         *mu = 0;
00788         *sig = 0;
00789
00790         /* Estimate noise (Immerkaer, 1996)... */
00791         for (ix = 1; ix < wave->nx - 1; ix++)
00792             for (iy = 1; iy < wave->ny - 1; iy++) {
00793
00794                 /* Check data... */
00795                 okay = 1;
00796                 for (ix2 = ix - 1; ix2 <= ix + 1; ix2++)

```

```

00797         for (iy2 = iy - 1; iy2 <= iy + 1; iy2++)
00798             if (!gsl_finite(wave->temp[ix2][iy2]))
00799                 okay = 0;
00800         if (!okay)
00801             continue;
00802
00803         /* Get mean noise... */
00804         n++;
00805         *mu += wave->temp[ix][iy];
00806         *sig += gsl_pow_2(+4. / 6. * wave->temp[ix][iy]
00807             - 2. / 6. * (wave->temp[ix - 1][iy]
00808                 + wave->temp[ix + 1][iy]
00809                 + wave->temp[ix][iy - 1]
00810                 + wave->temp[ix][iy + 1])
00811             + 1. / 6. * (wave->temp[ix - 1][iy - 1]
00812                 + wave->temp[ix + 1][iy - 1]
00813                 + wave->temp[ix - 1][iy + 1]
00814                 + wave->temp[ix + 1][iy + 1]));
00815     }
00816
00817     /* Normalize... */
00818     *mu /= (double) n;
00819     *sig = sqrt(*sig / (double) n);
00820 }

```

**5.25.1.19 void period ( wave\_t \* wave, double \* Amax, double \* phimax, double \* lymax, double \* alphamax, double \* betamax, char \* filename )**

Compute periodogram.

Definition at line 824 of file `libairs.c`.

```

00831     {
00832
00833         FILE *out;
00834
00835         static double kx[PMAX], ky[PMAX], kx_ny, ky_ny, kxmax, kymax, A[PMAX][PMAX],
00836             phi[PMAX][PMAX], cx[PMAX][WX], cy[PMAX][WY], sx[PMAX][WX], sy[PMAX][WY],
00837             a, b, c, lx, ly, lxymax = 1000, dlxy = 10;
00838
00839         int i, imin, imax, j, jmin, jmax, l, lmax = 0, m, mmax = 0;
00840
00841         /* Compute wavenumbers and periodogram coefficients... */
00842         for (lx = -lxymax; lx <= lxymax; lx += dlxy) {
00843             kx[lmax] = (lx != 0 ? 2 * M_PI / lx : 0);
00844             for (i = 0; i < wave->nx; i++) {
00845                 cx[lmax][i] = cos(kx[lmax] * wave->x[i]);
00846                 sx[lmax][i] = sin(kx[lmax] * wave->x[i]);
00847             }
00848             if ((++lmax) > PMAX)
00849                 ERRMSG("Too many wavenumbers for periodogram!");
00850         }
00851         for (ly = 0; ly <= lxymax; ly += dlxy) {
00852             ky[mmax] = (ly != 0 ? 2 * M_PI / ly : 0);
00853             for (j = 0; j < wave->ny; j++) {
00854                 cy[mmax][j] = cos(ky[mmax] * wave->y[j]);
00855                 sy[mmax][j] = sin(ky[mmax] * wave->y[j]);
00856             }
00857             if ((++mmax) > PMAX)
00858                 ERRMSG("Too many wavenumbers for periodogram!");
00859         }
00860
00861         /* Find area... */
00862         imin = jmin = 9999;
00863         imax = jmax = -9999;
00864         for (i = 0; i < wave->nx; i++)
00865             for (j = 0; j < wave->ny; j++)
00866                 if (gsl_finite(wave->var[i][j])) {
00867                     imin = GSL_MIN(imin, i);
00868                     imax = GSL_MAX(imax, i);
00869                     jmin = GSL_MIN(jmin, j);
00870                     jmax = GSL_MAX(jmax, j);
00871                 }
00872
00873         /* Get Nyquist frequencies... */
00874         kx_ny =
00875             M_PI / fabs((wave->x[imax] - wave->x[imin]) /
00876                 ((double) imax - (double) imin));
00877         ky_ny =
00878             M_PI / fabs((wave->y[jmax] - wave->y[jmin]) /

```



```

00879         ((double) jmax - (double) jmin));
00880
00881     /* Loop over wavelengths... */
00882     for (l = 0; l < lmax; l++)
00883         for (m = 0; m < mmax; m++) {
00884
00885             /* Check frequencies... */
00886             if (kx[l] == 0 || fabs(kx[l]) > kx_ny ||
00887                 ky[m] == 0 || fabs(ky[m]) > ky_ny) {
00888                 A[l][m] = GSL_NAN;
00889                 phi[l][m] = GSL_NAN;
00890                 continue;
00891             }
00892
00893             /* Compute periodogram... */
00894             a = b = c = 0;
00895             for (i = imin; i <= imax; i++)
00896                 for (j = jmin; j <= jmax; j++)
00897                     if (gsl_finite(wave->var[i][j])) {
00898                         a += wave->pt[i][j] * (cx[l][i] * cy[m][j] - sx[l][i] * sy[m][j]);
00899                         b += wave->pt[i][j] * (sx[l][i] * cy[m][j] + cx[l][i] * sy[m][j]);
00900                         c++;
00901                     }
00902             a *= 2. / c;
00903             b *= 2. / c;
00904
00905             /* Get amplitude and phase... */
00906             A[l][m] = sqrt(gsl_pow_2(a) + gsl_pow_2(b));
00907             phi[l][m] = atan2(b, a) * 180. / M_PI;
00908         }
00909
00910     /* Find maximum... */
00911     *Amax = 0;
00912     for (l = 0; l < lmax; l++)
00913         for (m = 0; m < mmax; m++)
00914             if (gsl_finite(A[l][m]) && A[l][m] > *Amax) {
00915                 *Amax = A[l][m];
00916                 *phimax = phi[l][m];
00917                 kxmax = kx[l];
00918                 kymax = ky[m];
00919                 imax = i;
00920                 jmax = j;
00921             }
00922
00923     /* Get horizontal wavelength... */
00924     *lhmax = 2 * M_PI / sqrt(gsl_pow_2(kxmax) + gsl_pow_2(kymax));
00925
00926     /* Get propagation direction in xy-plane... */
00927     *alphamax = 90. - 180. / M_PI * atan2(kxmax, kymax);
00928
00929     /* Get propagation direction in lon,lat-plane... */
00930     *betamax = *alphamax
00931         +
00932         180. / M_PI *
00933         atan2(wave->lat[wave->nx / 2 >
00934             0 ? wave->nx / 2 - 1 : wave->nx / 2][wave->ny / 2]
00935             - wave->lat[wave->nx / 2 <
00936                 wave->nx - 1 ? wave->nx / 2 +
00937                 1 : wave->nx / 2][wave->ny / 2],
00938             wave->lon[wave->nx / 2 >
00939                 0 ? wave->nx / 2 - 1 : wave->nx / 2][wave->ny / 2]
00940             - wave->lon[wave->nx / 2 <
00941                 wave->nx - 1 ? wave->nx / 2 +
00942                 1 : wave->nx / 2][wave->ny / 2]);
00943
00944     /* Save periodogram data... */
00945     if (filename != NULL) {
00946
00947         /* Write info... */
00948         printf("Write periodogram data: %s\n", filename);
00949
00950         /* Create file... */
00951         if (!(out = fopen(filename, "w")))
00952             ERRMSG("Cannot create file!");
00953
00954         /* Write header... */
00955         fprintf(out,
00956             "# $1 = altitude [km]\n"
00957             "# $2 = wavelength in x-direction [km]\n"
00958             "# $3 = wavelength in y-direction [km]\n"
00959             "# $4 = wavenumber in x-direction [1/km]\n"
00960             "# $5 = wavenumber in y-direction [1/km]\n"
00961             "# $6 = amplitude [K]\n" "# $7 = phase [rad]\n");
00962
00963         /* Write data... */
00964         for (l = 0; l < lmax; l++) {
00965             fprintf(out, "\n");

```

```

00966         for (m = 0; m < mmax; m++)
00967             fprintf(out, "%g %g %g %g %g %g %g\n", wave->z,
00968                     (kx[l] != 0 ? 2 * M_PI / kx[l] : 0),
00969                     (ky[m] != 0 ? 2 * M_PI / ky[m] : 0),
00970                     kx[l], ky[m], A[l][m], phi[l][m]);
00971     }
00972
00973     /* Close file... */
00974     fclose(out);
00975 }
00976 }

```

#### 5.25.1.20 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 980 of file [libairs.c](#).

```

00986     {
00987
00988         double x0[3], x1[3];
00989
00990         int itrack, ixtrack;
00991
00992         /* Check ranges... */
00993         track0 = GSL_MIN(GSL_MAX(track0, 0), pert->ntrack - 1);
00994         track1 = GSL_MIN(GSL_MAX(track1, 0), pert->ntrack - 1);
00995         xtrack0 = GSL_MIN(GSL_MAX(xtrack0, 0), pert->nxtrack - 1);
00996         xtrack1 = GSL_MIN(GSL_MAX(xtrack1, 0), pert->nxtrack - 1);
00997
00998         /* Set size... */
00999         wave->nx = xtrack1 - xtrack0 + 1;
01000         if (wave->nx > WX)
01001             ERRMSG("Too many across-track values!");
01002         wave->ny = track1 - track0 + 1;
01003         if (wave->ny > WY)
01004             ERRMSG("Too many along-track values!");
01005
01006         /* Loop over footprints... */
01007         for (itrack = track0; itrack <= track1; itrack++)
01008             for (ixtrack = xtrack0; ixtrack <= xtrack1; ixtrack++) {
01009
01010                 /* Get distances... */
01011                 if (itrack == track0) {
01012                     wave->x[0] = 0;
01013                     if (ixtrack > xtrack0) {
01014                         geo2cart(0, pert->lon[itrack][ixtrack - 1],
01015                                 pert->lat[itrack][ixtrack - 1], x0);
01016                         geo2cart(0, pert->lon[itrack][ixtrack],
01017                                 pert->lat[itrack][ixtrack], x1);
01018                         wave->x[ixtrack - xtrack0] =
01019                             wave->x[ixtrack - xtrack0 - 1] + DIST(x0, x1);
01020                     }
01021                 }
01022                 if (ixtrack == xtrack0) {
01023                     wave->y[0] = 0;
01024                     if (itrack > track0) {
01025                         geo2cart(0, pert->lon[itrack - 1][ixtrack],
01026                                 pert->lat[itrack - 1][ixtrack], x0);
01027                         geo2cart(0, pert->lon[itrack][ixtrack],
01028                                 pert->lat[itrack][ixtrack], x1);
01029                     }
01030                     wave->y[ixtrack - track0] =
01031                         wave->y[ixtrack - track0 - 1] + DIST(x0, x1);
01032                 }
01033
01034                 /* Save geolocation... */
01035                 wave->time = pert->time[(track0 + track1) / 2][(xtrack0 + xtrack1) / 2];
01036                 wave->z = 0;
01037                 wave->lon[ixtrack - xtrack0][itrack - track0] =
01038                     pert->lon[itrack][ixtrack];
01039                 wave->lat[ixtrack - xtrack0][itrack - track0] =
01040                     pert->lat[itrack][ixtrack];
01041
01042                 /* Save temperature data... */
01043                 wave->temp[ixtrack - xtrack0][itrack - track0]
01044                     = pert->bt[itrack][ixtrack];
01045                 wave->bg[ixtrack - xtrack0][itrack - track0]
01046                     = pert->bt[itrack][ixtrack] - pert->pt[itrack][ixtrack];
01047                 wave->pt[ixtrack - xtrack0][itrack - track0]

```

```

01048         = pert->pt[itrack][ixtrack];
01049         wave->var[ixtrack - xtrack0][itrack - track0]
01050         = pert->var[itrack][ixtrack];
01051     }
01052 }

```

Here is the call graph for this function:



#### 5.25.1.21 void read\_l1 ( char \* filename, airs\_l1\_t \* l1 )

Read AIRS Level-1 data.

Definition at line 1056 of file libairs.c.

```

01058     {
01059
01060     int ncid, varid;
01061
01062     /* Open netCDF file... */
01063     printf("Read AIRS Level-1 file: %s\n", filename);
01064     NC(nc_open(filename, NC_NOWRITE, &ncid));
01065
01066     /* Read data... */
01067     NC(nc_inq_varid(ncid, "l1_time", &varid));
01068     NC(nc_get_var_double(ncid, varid, l1->time[0]));
01069     NC(nc_inq_varid(ncid, "l1_lon", &varid));
01070     NC(nc_get_var_double(ncid, varid, l1->lon[0]));
01071     NC(nc_inq_varid(ncid, "l1_lat", &varid));
01072     NC(nc_get_var_double(ncid, varid, l1->lat[0]));
01073     NC(nc_inq_varid(ncid, "l1_sat_z", &varid));
01074     NC(nc_get_var_double(ncid, varid, l1->sat_z));
01075     NC(nc_inq_varid(ncid, "l1_sat_lon", &varid));
01076     NC(nc_get_var_double(ncid, varid, l1->sat_lon));
01077     NC(nc_inq_varid(ncid, "l1_sat_lat", &varid));
01078     NC(nc_get_var_double(ncid, varid, l1->sat_lat));
01079     NC(nc_inq_varid(ncid, "l1_nu", &varid));
01080     NC(nc_get_var_double(ncid, varid, l1->nu));
01081     NC(nc_inq_varid(ncid, "l1_rad", &varid));
01082     NC(nc_get_var_float(ncid, varid, l1->rad[0][0]));
01083
01084     /* Close file... */
01085     NC(nc_close(ncid));
01086 }

```

#### 5.25.1.22 void read\_l2 ( char \* filename, airs\_l2\_t \* l2 )

Read AIRS Level-2 data.

Definition at line 1090 of file libairs.c.

```

01092         {
01093
01094     int ncid, varid;
01095
01096     /* Open netCDF file... */
01097     printf("Read AIRS Level-2 file: %s\n", filename);
01098     NC(nc_open(filename, NC_NOWRITE, &ncid));
01099
01100     /* Read data... */
01101     NC(nc_inq_varid(ncid, "l2_time", &varid));
01102     NC(nc_get_var_double(ncid, varid, l2->time[0]));
01103     NC(nc_inq_varid(ncid, "l2_z", &varid));
01104     NC(nc_get_var_double(ncid, varid, l2->z[0][0]));
01105     NC(nc_inq_varid(ncid, "l2_lon", &varid));
01106     NC(nc_get_var_double(ncid, varid, l2->lon[0]));
01107     NC(nc_inq_varid(ncid, "l2_lat", &varid));
01108     NC(nc_get_var_double(ncid, varid, l2->lat[0]));
01109     NC(nc_inq_varid(ncid, "l2_press", &varid));
01110     NC(nc_get_var_double(ncid, varid, l2->p));
01111     NC(nc_inq_varid(ncid, "l2_temp", &varid));
01112     NC(nc_get_var_double(ncid, varid, l2->t[0][0]));
01113
01114     /* Close file... */
01115     NC(nc_close(ncid));
01116 }

```

#### 5.25.1.23 void read\_pert( char \* filename, char \* pertname, pert\_t \* pert )

Read radiance perturbation data.

Definition at line 1120 of file libairs.c.

```

01123         {
01124
01125     static char varname[LEN];
01126
01127     static int dimid[2], ncid, varid;
01128
01129     static size_t itrack, ntrack, nxtrack, start[2] = { 0, 0 }, count[2] = {
01130     1, 1};
01131
01132     /* Write info... */
01133     printf("Read perturbation data: %s\n", filename);
01134
01135     /* Open netCDF file... */
01136     NC(nc_open(filename, NC_NOWRITE, &ncid));
01137
01138     /* Get dimensions... */
01139     NC(nc_inq_dimid(ncid, "NTRACK", &dimid[0]));
01140     NC(nc_inq_dimid(ncid, "NXTRACK", &dimid[1]));
01141     NC(nc_inq_dimlen(ncid, dimid[0], &ntrack));
01142     NC(nc_inq_dimlen(ncid, dimid[1], &nxtrack));
01143     if (nxtrack > PERT_NXTRACK)
01144         ERRMSG("Too many tracks!");
01145     if (ntrack > PERT_NTRACK)
01146         ERRMSG("Too many scans!");
01147     pert->ntrack = (int) ntrack;
01148     pert->nxtrack = (int) nxtrack;
01149     count[1] = nxtrack;
01150
01151     /* Read data... */
01152     NC(nc_inq_varid(ncid, "time", &varid));
01153     for (itrack = 0; itrack < ntrack; itrack++) {
01154         start[0] = itrack;
01155         NC(nc_get_vara_double(ncid, varid, start, count, pert->time[itrack]));
01156     }
01157
01158     NC(nc_inq_varid(ncid, "lon", &varid));
01159     for (itrack = 0; itrack < ntrack; itrack++) {
01160         start[0] = itrack;
01161         NC(nc_get_vara_double(ncid, varid, start, count, pert->lon[itrack]));
01162     }
01163
01164     NC(nc_inq_varid(ncid, "lat", &varid));
01165     for (itrack = 0; itrack < ntrack; itrack++) {
01166         start[0] = itrack;
01167         NC(nc_get_vara_double(ncid, varid, start, count, pert->lat[itrack]));
01168     }
01169
01170     NC(nc_inq_varid(ncid, "bt_8mu", &varid));

```

```

01171     for (itrack = 0; itrack < ntrack; itrack++) {
01172         start[0] = itrack;
01173         NC(nc_get_vara_double(ncid, varid, start, count, pert->dc[itrack]));
01174     }
01175
01176     sprintf(varname, "bt_%s", pertname);
01177     NC(nc_inq_varid(ncid, varname, &varid));
01178     for (itrack = 0; itrack < ntrack; itrack++) {
01179         start[0] = itrack;
01180         NC(nc_get_vara_double(ncid, varid, start, count, pert->bt[itrack]));
01181     }
01182
01183     sprintf(varname, "bt_%s_pt", pertname);
01184     NC(nc_inq_varid(ncid, varname, &varid));
01185     for (itrack = 0; itrack < ntrack; itrack++) {
01186         start[0] = itrack;
01187         NC(nc_get_vara_double(ncid, varid, start, count, pert->pt[itrack]));
01188     }
01189
01190     sprintf(varname, "bt_%s_var", pertname);
01191     NC(nc_inq_varid(ncid, varname, &varid));
01192     for (itrack = 0; itrack < ntrack; itrack++) {
01193         start[0] = itrack;
01194         NC(nc_get_vara_double(ncid, varid, start, count, pert->var[itrack]));
01195     }
01196
01197     /* Close file... */
01198     NC(nc_close(ncid));
01199 }

```

#### 5.25.1.24 void read\_retr ( char \* filename, ret\_t \* ret )

Read AIRS retrieval data.

Definition at line 1203 of file libairs.c.

```

01205     {
01206
01207         static double help[NDS * NPG];
01208
01209         int dimid, ids = 0, ip, ncid, varid;
01210
01211         size_t itrack, ixtrack, nds, np, ntrack, nxtrack;
01212
01213         /* Write info... */
01214         printf("Read retrieval data: %s\n", filename);
01215
01216         /* Open netCDF file... */
01217         NC(nc_open(filename, NC_NOWRITE, &ncid));
01218
01219         /* Read new retrieval file format... */
01220         if (nc_inq_dimid(ncid, "L1_NTRACK", &dimid) == NC_NOERR) {
01221
01222             /* Get dimensions... */
01223             NC(nc_inq_dimid(ncid, "RET_NP", &dimid));
01224             NC(nc_inq_dimlen(ncid, dimid, &np));
01225             ret->np = (int) np;
01226             if (ret->np > NPG)
01227                 ERRMSG("Too many data points!");
01228
01229             NC(nc_inq_dimid(ncid, "L1_NTRACK", &dimid));
01230             NC(nc_inq_dimlen(ncid, dimid, &ntrack));
01231             NC(nc_inq_dimid(ncid, "L1_NXTRACK", &dimid));
01232             NC(nc_inq_dimlen(ncid, dimid, &nxtrack));
01233             ret->nds = (int) (ntrack * nxtrack);
01234             if (ret->nds > NDS)
01235                 ERRMSG("Too many data sets!");
01236
01237             /* Read time... */
01238             NC(nc_inq_varid(ncid, "l1_time", &varid));
01239             NC(nc_get_var_double(ncid, varid, help));
01240             ids = 0;
01241             for (itrack = 0; itrack < ntrack; itrack++)
01242                 for (ixtrack = 0; ixtrack < nxtrack; ixtrack++) {
01243                     for (ip = 0; ip < ret->np; ip++)
01244                         ret->time[ids][ip] = help[ids];
01245                     ids++;
01246                 }
01247
01248             /* Read altitudes... */

```

```

01249     NC(nc_inq_varid(ncid, "ret_z", &varid));
01250     NC(nc_get_var_double(ncid, varid, help));
01251     ids = 0;
01252     for (itrack = 0; itrack < ntrack; itrack++)
01253         for (ixtrack = 0; ixtrack < nxtrack; ixtrack++) {
01254             for (ip = 0; ip < ret->np; ip++)
01255                 ret->z[ids][ip] = help[ip];
01256             ids++;
01257         }
01258
01259     /* Read longitudes... */
01260     NC(nc_inq_varid(ncid, "ll_lon", &varid));
01261     NC(nc_get_var_double(ncid, varid, help));
01262     ids = 0;
01263     for (itrack = 0; itrack < ntrack; itrack++)
01264         for (ixtrack = 0; ixtrack < nxtrack; ixtrack++) {
01265             for (ip = 0; ip < ret->np; ip++)
01266                 ret->lon[ids][ip] = help[ids];
01267             ids++;
01268         }
01269
01270     /* Read latitudes... */
01271     NC(nc_inq_varid(ncid, "ll_lat", &varid));
01272     NC(nc_get_var_double(ncid, varid, help));
01273     ids = 0;
01274     for (itrack = 0; itrack < ntrack; itrack++)
01275         for (ixtrack = 0; ixtrack < nxtrack; ixtrack++) {
01276             for (ip = 0; ip < ret->np; ip++)
01277                 ret->lat[ids][ip] = help[ids];
01278             ids++;
01279         }
01280
01281     /* Read temperatures... */
01282     NC(nc_inq_varid(ncid, "ret_temp", &varid));
01283     NC(nc_get_var_double(ncid, varid, help));
01284     ids = 0;
01285     for (itrack = 0; itrack < ntrack; itrack++)
01286         for (ixtrack = 0; ixtrack < nxtrack; ixtrack++) {
01287             for (ip = 0; ip < ret->np; ip++)
01288                 ret->t[ids][ip] =
01289                     help[(itrack * nxtrack + ixtrack) * (size_t) np + (size_t) ip];
01290             ids++;
01291         }
01292 }
01293
01294 /* Read old retrieval file format... */
01295 if (nc_inq_dimid(ncid, "np", &dimid) == NC_NOERR) {
01296
01297     /* Get dimensions... */
01298     NC(nc_inq_dimid(ncid, "np", &dimid));
01299     NC(nc_inq_dimlen(ncid, dimid, &np));
01300     ret->np = (int) np;
01301     if (ret->np > NPG)
01302         ERRMSG("Too many data points!");
01303
01304     NC(nc_inq_dimid(ncid, "nds", &dimid));
01305     NC(nc_inq_dimlen(ncid, dimid, &nds));
01306     ret->nds = (int) nds;
01307     if (ret->nds > NDS)
01308         ERRMSG("Too many data sets!");
01309
01310     /* Read data... */
01311     NC(nc_inq_varid(ncid, "time", &varid));
01312     NC(nc_get_var_double(ncid, varid, help));
01313     read_retr_help(help, ret->nds, ret->np, ret->time);
01314
01315     NC(nc_inq_varid(ncid, "z", &varid));
01316     NC(nc_get_var_double(ncid, varid, help));
01317     read_retr_help(help, ret->nds, ret->np, ret->z);
01318
01319     NC(nc_inq_varid(ncid, "lon", &varid));
01320     NC(nc_get_var_double(ncid, varid, help));
01321     read_retr_help(help, ret->nds, ret->np, ret->lon);
01322
01323     NC(nc_inq_varid(ncid, "lat", &varid));
01324     NC(nc_get_var_double(ncid, varid, help));
01325     read_retr_help(help, ret->nds, ret->np, ret->lat);
01326
01327     NC(nc_inq_varid(ncid, "press", &varid));
01328     NC(nc_get_var_double(ncid, varid, help));
01329     read_retr_help(help, ret->nds, ret->np, ret->p);
01330
01331     NC(nc_inq_varid(ncid, "temp", &varid));
01332     NC(nc_get_var_double(ncid, varid, help));
01333     read_retr_help(help, ret->nds, ret->np, ret->t);
01334
01335     NC(nc_inq_varid(ncid, "temp_apr", &varid));

```

```

01336     NC(nc_get_var_double(ncid, varid, help));
01337     read_retr_help(help, ret->nds, ret->np, ret->t_apr);
01338
01339     NC(nc_inq_varid(ncid, "temp_total", &varid));
01340     NC(nc_get_var_double(ncid, varid, help));
01341     read_retr_help(help, ret->nds, ret->np, ret->t_tot);
01342
01343     NC(nc_inq_varid(ncid, "temp_noise", &varid));
01344     NC(nc_get_var_double(ncid, varid, help));
01345     read_retr_help(help, ret->nds, ret->np, ret->t_noise);
01346
01347     NC(nc_inq_varid(ncid, "temp_formod", &varid));
01348     NC(nc_get_var_double(ncid, varid, help));
01349     read_retr_help(help, ret->nds, ret->np, ret->t_fm);
01350
01351     NC(nc_inq_varid(ncid, "temp_cont", &varid));
01352     NC(nc_get_var_double(ncid, varid, help));
01353     read_retr_help(help, ret->nds, ret->np, ret->t_cont);
01354
01355     NC(nc_inq_varid(ncid, "temp_res", &varid));
01356     NC(nc_get_var_double(ncid, varid, help));
01357     read_retr_help(help, ret->nds, ret->np, ret->t_res);
01358
01359     NC(nc_inq_varid(ncid, "chisq", &varid));
01360     NC(nc_get_var_double(ncid, varid, ret->chisq));
01361 }
01362
01363 /* Close file... */
01364 NC(nc_close(ncid));
01365 }

```

Here is the call graph for this function:



#### 5.25.1.25 void read\_retr\_help ( double \* help, int nds, int np, double mat[NDS][NPG] )

Convert array.

Definition at line 1369 of file libairs.c.

```

01373     {
01374
01375     int ids, ip, n = 0;
01376
01377     for (ip = 0; ip < np; ip++)
01378         for (ids = 0; ids < nds; ids++)
01379             mat[ids][ip] = help[n++];
01380 }

```

#### 5.25.1.26 void read\_wave ( char \* filename, wave\_t \* wave )

Read wave analysis data.

Definition at line 1384 of file libairs.c.

```

01386         {
01387
01388     FILE *in;
01389
01390     char line[LEN];
01391
01392     double rtime, rz, rlon, rlat, rx, ry, ryold = -1e10, rtemp, rbg, rpt, rvar;
01393
01394     /* Init... */
01395     wave->nx = 0;
01396     wave->ny = 0;
01397
01398     /* Write info... */
01399     printf("Read wave data: %s\n", filename);
01400
01401     /* Open file... */
01402     if (!(in = fopen(filename, "r")))
01403         ERRMSG("Cannot open file!");
01404
01405     /* Read data... */
01406     while (fgets(line, LEN, in))
01407         if (sscanf(line, "%lg %lg %lg %lg %lg %lg %lg %lg %lg %lg", &rtime,
01408             &rz, &rlon, &rlat, &rx, &ry, &rtemp, &rbg, &rpt,
01409             &rvar) == 10) {
01410
01411         /* Set index... */
01412         if (ry != ryold) {
01413             if ((++wave->ny >= WY))
01414                 ERRMSG("Too many y-values!");
01415             wave->nx = 0;
01416         } else if ((++wave->nx >= WX))
01417             ERRMSG("Too many x-values!");
01418         ryold = ry;
01419
01420         /* Save data... */
01421         wave->time = rtime;
01422         wave->z = rz;
01423         wave->lon[wave->nx][wave->ny] = rlon;
01424         wave->lat[wave->nx][wave->ny] = rlat;
01425         wave->x[wave->nx] = rx;
01426         wave->y[wave->ny] = ry;
01427         wave->temp[wave->nx][wave->ny] = rtemp;
01428         wave->bg[wave->nx][wave->ny] = rbg;
01429         wave->pt[wave->nx][wave->ny] = rpt;
01430         wave->var[wave->nx][wave->ny] = rvar;
01431     }
01432
01433     /* Increment counters... */
01434     wave->nx++;
01435     wave->ny++;
01436
01437     /* Close file... */
01438     fclose(in);
01439 }

```

#### 5.25.1.27 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 1443 of file `libairs.c`.

```

01447         {
01448
01449     double x0[3], x1[3];
01450
01451     int ichan[AIRS_RAD_CHANNEL], id, track, xtrack;
01452
01453     /* Get channel numbers... */
01454     for (id = 0; id < nd; id++) {
01455         for (ichan[id] = 0; ichan[id] < AIRS_RAD_CHANNEL; ichan[id]++)
01456             if (fabs(gran->nominal_freq[ichan[id]] - nu[id]) < 0.1)
01457                 break;
01458         if (ichan[id] >= AIRS_RAD_CHANNEL)
01459             ERRMSG("Could not find channel!");
01460     }
01461
01462     /* Set size... */
01463     wave->nx = AIRS_RAD_GEOXTRACK;
01464     wave->ny = AIRS_RAD_GEOTRACK;
01465     if (wave->nx > WX || wave->ny > WY)

```

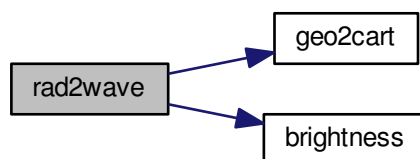


```

01466     ERRMSG("Wave struct too small!");
01467
01468     /* Set Cartesian coordinates... */
01469     geo2cart(0, gran->Longitude[0][0], gran->Latitude[0][0], x0);
01470     for (xtrack = 0; xtrack < AIRS_RAD_GEOXTRACK; xtrack++) {
01471         geo2cart(0, gran->Longitude[0][xtrack], gran->Latitude[0][xtrack], x1);
01472         wave->x[xtrack] = DIST(x0, x1);
01473     }
01474     for (track = 0; track < AIRS_RAD_GEOTRACK; track++) {
01475         geo2cart(0, gran->Longitude[track][0], gran->Latitude[track][0], x1);
01476         wave->y[track] = DIST(x0, x1);
01477     }
01478
01479     /* Set geolocation... */
01480     wave->time =
01481         gran->Time[AIRS_RAD_GEOTRACK / 2][AIRS_RAD_GEOXTRACK / 2] - 220838400;
01482     wave->z = 0;
01483     for (track = 0; track < AIRS_RAD_GEOTRACK; track++) {
01484         for (xtrack = 0; xtrack < AIRS_RAD_GEOXTRACK; xtrack++) {
01485             wave->lon[xtrack][track] = gran->Longitude[track][xtrack];
01486             wave->lat[xtrack][track] = gran->Latitude[track][xtrack];
01487         }
01488     }
01489
01490     /* Set brightness temperature... */
01491     for (track = 0; track < AIRS_RAD_GEOTRACK; track++) {
01492         for (xtrack = 0; xtrack < AIRS_RAD_GEOXTRACK; xtrack++) {
01493             wave->temp[xtrack][track] = 0;
01494             wave->bg[xtrack][track] = 0;
01495             wave->pt[xtrack][track] = 0;
01496             wave->var[xtrack][track] = 0;
01497             for (id = 0; id < nd; id++) {
01498                 if ((gran->state[track][xtrack] != 0)
01499                     || (gran->ExcludedChans[ichan[id]] > 2)
01500                     || (gran->CalChanSummary[ichan[id]] & 8)
01501                     || (gran->CalChanSummary[ichan[id]] & (32 + 64))
01502                     || (gran->CalFlag[track][ichan[id]] & 16))
01503                 wave->temp[xtrack][track] = GSL_NAN;
01504             }
01505             else
01506                 wave->temp[xtrack][track]
01507                     += brightness(gran->radiances[track][xtrack][ichan[id]] * 1e-3,
01508                                 gran->nominal_freq[ichan[id]]) / nd;
01509         }
01510     }
01511 }

```

Here is the call graph for this function:



#### 5.25.1.28 void ret2wave ( ret\_t \* ret, wave\_t \* wave, int dataset, int ip )

Convert AIRS retrieval results to wave analysis struct.

Definition at line 1513 of file libairs.c.

```

01517     {
01518
01519         double x0[3], x1[3];
01520
01521         int ids, ix, iy;

```

```

01522
01523  /* Initialize... */
01524  wave->nx = 90;
01525  if (wave->nx > WX)
01526      ERRMSG("Too many across-track values!");
01527  wave->ny = 135;
01528  if (wave->ny > WY)
01529      ERRMSG("Too many along-track values!");
01530  if (ip < 0 || ip >= ret->np)
01531      ERRMSG("Altitude index out of range!");
01532
01533  /* Loop over data sets and data points... */
01534  for (ids = 0; ids < ret->nds; ids++) {
01535
01536      /* Get horizontal indices... */
01537      ix = ids % 90;
01538      iy = ids / 90;
01539
01540      /* Get distances... */
01541      if (iy == 0) {
01542          geo2cart(0.0, ret->lon[0][0], ret->lat[0][0], x0);
01543          geo2cart(0.0, ret->lon[ids][ip], ret->lat[ids][ip], x1);
01544          wave->x[ix] = DIST(x0, x1);
01545      }
01546      if (ix == 0) {
01547          geo2cart(0.0, ret->lon[0][0], ret->lat[0][0], x0);
01548          geo2cart(0.0, ret->lon[ids][ip], ret->lat[ids][ip], x1);
01549          wave->y[iy] = DIST(x0, x1);
01550      }
01551
01552      /* Save geolocation... */
01553      wave->time = ret->time[0][0];
01554      if (ix == 0 && iy == 0)
01555          wave->z = ret->z[ids][ip];
01556      wave->lon[ix][iy] = ret->lon[ids][ip];
01557      wave->lat[ix][iy] = ret->lat[ids][ip];
01558
01559      /* Save temperature... */
01560      if (dataset == 1)
01561          wave->temp[ix][iy] = ret->t[ids][ip];
01562      else if (dataset == 2)
01563          wave->temp[ix][iy] = ret->t_apr[ids][ip];
01564  }
01565 }

```

Here is the call graph for this function:



#### 5.25.1.29 double sza ( double sec, double lon, double lat )

Calculate solar zenith angle.

Definition at line 1569 of file libairs.c.

```

01572      {
01573
01574      double D, dec, e, g, GMST, h, L, LST, q, ra;
01575
01576      /* Number of days and fraction with respect to 2000-01-01T12:00Z... */
01577      D = sec / 86400 - 0.5;
01578
01579      /* Geocentric apparent ecliptic longitude [rad]... */
01580      g = (357.529 + 0.98560028 * D) * M_PI / 180;

```

```

01581 q = 280.459 + 0.98564736 * D;
01582 L = (q + 1.915 * sin(g) + 0.020 * sin(2 * g)) * M_PI / 180;
01583
01584 /* Mean obliquity of the ecliptic [rad]... */
01585 e = (23.439 - 0.00000036 * D) * M_PI / 180;
01586
01587 /* Declination [rad]... */
01588 dec = asin(sin(e) * sin(L));
01589
01590 /* Right ascension [rad]... */
01591 ra = atan2(cos(e) * sin(L), cos(L));
01592
01593 /* Greenwich Mean Sidereal Time [h]... */
01594 GMST = 18.697374558 + 24.06570982441908 * D;
01595
01596 /* Local Sidereal Time [h]... */
01597 LST = GMST + lon / 15;
01598
01599 /* Hour angle [rad]... */
01600 h = LST / 12 * M_PI - ra;
01601
01602 /* Convert latitude... */
01603 lat *= M_PI / 180;
01604
01605 /* Return solar zenith angle [deg]... */
01606 return acos(sin(lat) * sin(dec) +
01607             cos(lat) * cos(dec) * cos(h)) * 180 / M_PI;
01608 }

```

### 5.25.1.30 void variance ( wave\_t \* wave, double dh )

Compute local variance.

Definition at line 1612 of file [libairs.c](#).

```

01614 {
01615     double dh2, mu, help;
01616     int dx, dy, ix, ix2, iy, iy2, n;
01617
01618     /* Check parameters... */
01619     if (dh <= 0)
01620         return;
01621
01622     /* Compute squared radius... */
01623     dh2 = gsl_pow_2(dh);
01624
01625     /* Get sampling distances... */
01626     dx =
01627         (int) (dh / fabs(wave->x[wave->nx - 1] - wave->x[0]) * (wave->nx - 1.0) +
01628             1);
01629     dy =
01630         (int) (dh / fabs(wave->y[wave->ny - 1] - wave->y[0]) * (wave->ny - 1.0) +
01631             1);
01632
01633     /* Loop over data points... */
01634     for (ix = 0; ix < wave->nx; ix++)
01635         for (iy = 0; iy < wave->ny; iy++) {
01636             /* Init... */
01637             mu = help = 0;
01638             n = 0;
01639
01640             /* Get data... */
01641             for (ix2 = GSL_MAX(ix - dx, 0); ix2 <= GSL_MIN(ix + dx, wave->nx - 1);
01642                 ix2++)
01643                 for (iy2 = GSL_MAX(iy - dy, 0); iy2 <= GSL_MIN(iy + dy, wave->ny - 1);
01644                     iy2++)
01645                     if ((gsl_pow_2(wave->x[ix] - wave->x[ix2])
01646                         + gsl_pow_2(wave->y[iy] - wave->y[iy2])) <= dh2)
01647                         if (gsl_finite(wave->pt[ix2][iy2])) {
01648                             mu += wave->pt[ix2][iy2];
01649                             help += gsl_pow_2(wave->pt[ix2][iy2]);
01650                             n++;
01651                         }
01652
01653             /* Compute local variance... */
01654             if (n > 1)
01655                 wave->var[ix][iy] = help / n - gsl_pow_2(mu / n);
01656             else
01657                 wave->var[ix][iy] = GSL_NAN;
01658         }
01659     }
01660 }
01661
01662 }

```

## 5.25.1.31 void write\_l1 ( char \* filename, airs\_l1\_t \* l1 )

Write AIRS Level-1 data.

Definition at line 1666 of file libairs.c.

```

01668         {
01669
01670     int dimid[10], ncid, time_id, lon_id, lat_id,
01671         sat_z_id, sat_lon_id, sat_lat_id, nu_id, rad_id;
01672
01673     /* Open or create netCDF file... */
01674     printf("Write AIRS Level-1 file: %s\n", filename);
01675     if (nc_open(filename, NC_WRITE, &ncid) != NC_NOERR) {
01676         NC(nc_create(filename, NC_CLOBBER, &ncid));
01677     } else {
01678         NC(nc_redef(ncid));
01679     }
01680
01681     /* Set dimensions... */
01682     if (nc_inq_dimid(ncid, "L1_NTRACK", &dimid[0]) != NC_NOERR)
01683         NC(nc_def_dim(ncid, "L1_NTRACK", L1_NTRACK, &dimid[0]));
01684     if (nc_inq_dimid(ncid, "L1_NXTRACK", &dimid[1]) != NC_NOERR)
01685         NC(nc_def_dim(ncid, "L1_NXTRACK", L1_NXTRACK, &dimid[1]));
01686     if (nc_inq_dimid(ncid, "L1_NCHAN", &dimid[2]) != NC_NOERR)
01687         NC(nc_def_dim(ncid, "L1_NCHAN", L1_NCHAN, &dimid[2]));
01688
01689     /* Add variables... */
01690     add_var(ncid, "l1_time", "s", "time (seconds since 2000-01-01T00:00Z)",
01691         NC_DOUBLE, dimid, &time_id, 2);
01692     add_var(ncid, "l1_lon", "deg", "longitude", NC_DOUBLE, dimid, &lon_id, 2);
01693     add_var(ncid, "l1_lat", "deg", "latitude", NC_DOUBLE, dimid, &lat_id, 2);
01694     add_var(ncid, "l1_sat_z", "km", "satellite altitude",
01695         NC_DOUBLE, dimid, &sat_z_id, 1);
01696     add_var(ncid, "l1_sat_lon", "deg", "satellite longitude",
01697         NC_DOUBLE, dimid, &sat_lon_id, 1);
01698     add_var(ncid, "l1_sat_lat", "deg", "satellite latitude",
01699         NC_DOUBLE, dimid, &sat_lat_id, 1);
01700     add_var(ncid, "l1_nu", "cm^-1", "channel wavenumber",
01701         NC_DOUBLE, &dimid[2], &nu_id, 1);
01702     add_var(ncid, "l1_rad", "W/(m^2 sr cm^-1)", "channel radiance",
01703         NC_FLOAT, dimid, &rad_id, 3);
01704
01705     /* Leave define mode... */
01706     NC(nc_enddef(ncid));
01707
01708     /* Write data... */
01709     NC(nc_put_var_double(ncid, time_id, l1->time[0]));
01710     NC(nc_put_var_double(ncid, lon_id, l1->lon[0]));
01711     NC(nc_put_var_double(ncid, lat_id, l1->lat[0]));
01712     NC(nc_put_var_double(ncid, sat_z_id, l1->sat_z));
01713     NC(nc_put_var_double(ncid, sat_lon_id, l1->sat_lon));
01714     NC(nc_put_var_double(ncid, sat_lat_id, l1->sat_lat));
01715     NC(nc_put_var_double(ncid, nu_id, l1->nu));
01716     NC(nc_put_var_float(ncid, rad_id, l1->rad[0][0]));
01717
01718     /* Close file... */
01719     NC(nc_close(ncid));
01720 }

```

Here is the call graph for this function:



### 5.25.1.32 void write\_l2 ( char \* filename, airs\_l2\_t \* l2 )

Write AIRS Level-2 data.

Definition at line 1724 of file [libairs.c](#).

```

01726         {
01727
01728     int dimid[10], ncid, time_id, z_id, lon_id, lat_id, p_id, t_id;
01729
01730     /* Create netCDF file... */
01731     printf("Write AIRS Level-2 file: %s\n", filename);
01732     if (nc_open(filename, NC_WRITE, &ncid) != NC_NOERR) {
01733         NC(nc_create(filename, NC_CLOBBER, &ncid));
01734     } else {
01735         NC(nc_redef(ncid));
01736     }
01737
01738     /* Set dimensions... */
01739     if (nc_inq_dimid(ncid, "L2_NTRACK", &dimid[0]) != NC_NOERR)
01740         NC(nc_def_dim(ncid, "L2_NTRACK", L2_NTRACK, &dimid[0]));
01741     if (nc_inq_dimid(ncid, "L2_NXTRACK", &dimid[1]) != NC_NOERR)
01742         NC(nc_def_dim(ncid, "L2_NXTRACK", L2_NXTRACK, &dimid[1]));
01743     if (nc_inq_dimid(ncid, "L2_NLAY", &dimid[2]) != NC_NOERR)
01744         NC(nc_def_dim(ncid, "L2_NLAY", L2_NLAY, &dimid[2]));
01745
01746     /* Add variables... */
01747     add_var(ncid, "l2_time", "s", "time (seconds since 2000-01-01T00:00Z)",
01748         NC_DOUBLE, dimid, &time_id, 2);
01749     add_var(ncid, "l2_z", "km", "altitude", NC_DOUBLE, dimid, &z_id, 3);
01750     add_var(ncid, "l2_lon", "deg", "longitude", NC_DOUBLE, dimid, &lon_id, 2);
01751     add_var(ncid, "l2_lat", "deg", "latitude", NC_DOUBLE, dimid, &lat_id, 2);
01752     add_var(ncid, "l2_press", "hPa", "pressure",
01753         NC_DOUBLE, &dimid[2], &p_id, 1);
01754     add_var(ncid, "l2_temp", "K", "temperature", NC_DOUBLE, dimid, &t_id, 3);
01755
01756     /* Leave define mode... */
01757     NC(nc_enddef(ncid));
01758
01759     /* Write data... */
01760     NC(nc_put_var_double(ncid, time_id, l2->time[0]));
01761     NC(nc_put_var_double(ncid, z_id, l2->z[0][0]));
01762     NC(nc_put_var_double(ncid, lon_id, l2->lon[0]));
01763     NC(nc_put_var_double(ncid, lat_id, l2->lat[0]));
01764     NC(nc_put_var_double(ncid, p_id, l2->p));
01765     NC(nc_put_var_double(ncid, t_id, l2->t[0][0]));
01766
01767     /* Close file... */
01768     NC(nc_close(ncid));
01769 }

```

Here is the call graph for this function:



### 5.25.1.33 void write\_wave ( char \* filename, wave\_t \* wave )

Write wave analysis data.

Definition at line 1773 of file [libairs.c](#).

```

01775         {
01776
01777     FILE *out;
01778
01779     int i, j;
01780
01781     /* Write info... */
01782     printf("Write wave data: %s\n", filename);
01783
01784     /* Create file... */
01785     if (!(out = fopen(filename, "w")))
01786         ERRMSG("Cannot create file!");
01787
01788     /* Write header... */
01789     fprintf(out,
01790         "# $1 = time (seconds since 2000-01-01T00:00Z)\n"
01791         "# $2 = altitude [km]\n"
01792         "# $3 = longitude [deg]\n"
01793         "# $4 = latitude [deg]\n"
01794         "# $5 = across-track distance [km]\n"
01795         "# $6 = along-track distance [km]\n"
01796         "# $7 = temperature [K]\n"
01797         "# $8 = background [K]\n"
01798         "# $9 = perturbation [K]\n" "# $10 = variance [K^2]\n");
01799
01800     /* Write data... */
01801     for (j = 0; j < wave->ny; j++) {
01802         fprintf(out, "\n");
01803         for (i = 0; i < wave->nx; i++)
01804             fprintf(out, "%.2f %g %g %g %g %g %g %g %g\n",
01805                 wave->time, wave->z, wave->lon[i][j], wave->lat[i][j],
01806                 wave->x[i], wave->y[j], wave->temp[i][j], wave->bq[i][j],
01807                 wave->pt[i][j], wave->var[i][j]);
01808     }
01809
01810     /* Close file... */
01811     fclose(out);
01812 }

```

## 5.26 libairs.c

```

00001 #include "libairs.h"
00002
00003 /*****
00004
00005 void add_att(
00006     int ncid,
00007     int varid,
00008     const char *unit,
00009     const char *long_name) {
00010
00011     /* Set long name... */
00012     NC(nc_put_att_text(ncid, varid, "long_name", strlen(long_name), long_name));
00013
00014     /* Set units... */
00015     NC(nc_put_att_text(ncid, varid, "units", strlen(unit), unit));
00016 }
00017
00018 /*****
00019
00020 void add_var(
00021     int ncid,
00022     const char *varname,
00023     const char *unit,
00024     const char *longname,
00025     int type,
00026     int dimid[],
00027     int *varid,
00028     int ndims) {
00029
00030     /* Check if variable exists... */
00031     if (nc_inq_varid(ncid, varname, varid) != NC_NOERR) {
00032
00033         /* Define variable... */
00034         NC(nc_def_var(ncid, varname, type, ndims, dimid, varid));
00035
00036         /* Set long name... */
00037         NC(nc_put_att_text
00038             (ncid, *varid, "long_name", strlen(longname), longname));
00039
00040         /* Set units... */
00041         NC(nc_put_att_text(ncid, *varid, "units", strlen(unit), unit));
00042     }

```

```

00043 }
00044
00045 /*****
00046
00047 void background_poly_help(
00048     double *xx,
00049     double *yy,
00050     int n,
00051     int dim) {
00052
00053     gsl_multifit_linear_workspace *work;
00054     gsl_matrix *cov, *X;
00055     gsl_vector *c, *x, *y;
00056
00057     double chisq, xx2[WX > WY ? WX : WY], yy2[WX > WY ? WX : WY];
00058
00059     size_t i, i2, n2 = 0;
00060
00061     /* Check for nan... */
00062     for (i = 0; i < (size_t) n; i++)
00063         if (gsl_finite(yy[i])) {
00064             xx2[n2] = xx[i];
00065             yy2[n2] = yy[i];
00066             n2++;
00067         }
00068     if ((int) n2 < dim || n2 < 0.9 * n) {
00069         for (i = 0; i < (size_t) n; i++)
00070             yy[i] = GSL_NAN;
00071         return;
00072     }
00073
00074     /* Allocate... */
00075     work = gsl_multifit_linear_alloc((size_t) n2, (size_t) dim);
00076     cov = gsl_matrix_alloc((size_t) dim, (size_t) dim);
00077     X = gsl_matrix_alloc((size_t) n2, (size_t) dim);
00078     c = gsl_vector_alloc((size_t) dim);
00079     x = gsl_vector_alloc((size_t) n2);
00080     y = gsl_vector_alloc((size_t) n2);
00081
00082     /* Compute polynomial fit... */
00083     for (i = 0; i < (size_t) n2; i++) {
00084         gsl_vector_set(x, i, xx2[i]);
00085         gsl_vector_set(y, i, yy2[i]);
00086         for (i2 = 0; i2 < (size_t) dim; i2++)
00087             gsl_matrix_set(X, i, i2, pow(gsl_vector_get(x, i), (double) i2));
00088     }
00089     gsl_multifit_linear(X, y, c, cov, &chisq, work);
00090     for (i = 0; i < (size_t) n; i++)
00091         yy[i] = gsl_poly_eval(c->data, (int) dim, xx[i]);
00092
00093     /* Free... */
00094     gsl_multifit_linear_free(work);
00095     gsl_matrix_free(cov);
00096     gsl_matrix_free(X);
00097     gsl_vector_free(c);
00098     gsl_vector_free(x);
00099     gsl_vector_free(y);
00100 }
00101
00102 /*****
00103
00104 void background_poly(
00105     wave_t * wave,
00106     int dim_x,
00107     int dim_y) {
00108
00109     double x[WX], x2[WY], y[WX], y2[WY];
00110
00111     int ix, iy;
00112
00113     /* Copy temperatures to background... */
00114     for (ix = 0; ix < wave->nx; ix++)
00115         for (iy = 0; iy < wave->ny; iy++) {
00116             wave->bg[ix][iy] = wave->temp[ix][iy];
00117             wave->pt[ix][iy] = 0;
00118         }
00119
00120     /* Check parameters... */
00121     if (dim_x <= 0 && dim_y <= 0)
00122         return;
00123
00124     /* Compute fit in x-direction... */
00125     if (dim_x > 0)
00126         for (iy = 0; iy < wave->ny; iy++) {
00127             for (ix = 0; ix < wave->nx; ix++) {
00128                 x[ix] = (double) ix;
00129                 y[ix] = wave->bg[ix][iy];

```

```

00130     }
00131     background_poly_help(x, y, wave->nx, dim_x);
00132     for (ix = 0; ix < wave->nx; ix++)
00133         wave->bg[ix][iy] = y[ix];
00134     }
00135
00136     /* Compute fit in y-direction... */
00137     if (dim_y > 0)
00138         for (ix = 0; ix < wave->nx; ix++) {
00139             for (iy = 0; iy < wave->ny; iy++) {
00140                 x2[iy] = (int) iy;
00141                 y2[iy] = wave->bg[ix][iy];
00142             }
00143             background_poly_help(x2, y2, wave->ny, dim_y);
00144             for (iy = 0; iy < wave->ny; iy++)
00145                 wave->bg[ix][iy] = y2[iy];
00146         }
00147
00148     /* Recompute perturbations... */
00149     for (ix = 0; ix < wave->nx; ix++)
00150         for (iy = 0; iy < wave->ny; iy++)
00151             wave->pt[ix][iy] = wave->temp[ix][iy] - wave->bg[ix][iy];
00152 }
00153
00154 /*****
00155 void background_smooth(
00156     wave_t * wave,
00157     int npts_x,
00158     int npts_y) {
00159
00160     static double help[WX][WY], dmax = 2500.;
00161
00162     int dx, dy, i, j, ix, iy, n;
00163
00164     /* Check parameters... */
00165     if (npts_x <= 0 && npts_y <= 0)
00166         return;
00167
00168     /* Smooth background... */
00169     for (ix = 0; ix < wave->nx; ix++)
00170         for (iy = 0; iy < wave->ny; iy++) {
00171
00172             /* Init... */
00173             n = 0;
00174             help[ix][iy] = 0;
00175
00176             /* Set maximum range... */
00177             dx = GSL_MIN(GSL_MIN(npts_x, ix), wave->nx - 1 - ix);
00178             dy = GSL_MIN(GSL_MIN(npts_y, iy), wave->ny - 1 - iy);
00179
00180             /* Average... */
00181             for (i = ix - dx; i <= ix + dx; i++)
00182                 for (j = iy - dy; j <= iy + dy; j++)
00183                     if (fabs(wave->x[ix] - wave->x[i]) < dmax &&
00184                         fabs(wave->y[iy] - wave->y[j]) < dmax) {
00185                         help[ix][iy] += wave->bg[i][j];
00186                         n++;
00187                     }
00188
00189             /* Normalize... */
00190             if (n > 0)
00191                 help[ix][iy] /= n;
00192             else
00193                 help[ix][iy] = GSL_NAN;
00194         }
00195
00196     /* Recalculate perturbations... */
00197     for (ix = 0; ix < wave->nx; ix++)
00198         for (iy = 0; iy < wave->ny; iy++) {
00199             wave->bg[ix][iy] = help[ix][iy];
00200             wave->pt[ix][iy] = wave->temp[ix][iy] - wave->bg[ix][iy];
00201         }
00202 }
00203
00204 /*****
00205 void create_background(
00206     wave_t * wave) {
00207
00208     int ix, iy;
00209
00210     /* Loop over grid points... */
00211     for (ix = 0; ix < wave->nx; ix++)
00212         for (iy = 0; iy < wave->ny; iy++) {
00213
00214             /* Set background for 4.3 micron BT measurements... */

```



```

00217     wave->bg[ix][iy] = 235.626 + 5.38165e-6 * gsl_pow_2(wave->x[ix]
00218                                     -
00219                                     0.5 * (wave->x[0] +
00220                                     wave->x
00221                                     [wave->nx -
00222                                     1]))
00223     - 1.78519e-12 * gsl_pow_4(wave->x[ix] -
00224                               0.5 * (wave->x[0] + wave->x[wave->nx - 1]));
00225
00226     /* Set temperature perturbation... */
00227     wave->pt[ix][iy] = 0;
00228
00229     /* Set temperature... */
00230     wave->temp[ix][iy] = wave->bg[ix][iy];
00231 }
00232 }
00233
00234 /*****
00235
00236 void create_noise(
00237     wave_t * wave,
00238     double nedt) {
00239
00240     gsl_rng *r;
00241
00242     int ix, iy;
00243
00244     /* Initialize random number generator... */
00245     gsl_rng_env_setup();
00246     r = gsl_rng_alloc(gsl_rng_default);
00247     gsl_rng_set(r, (unsigned long int) time(NULL));
00248
00249     /* Add noise to temperature... */
00250     if (nedt > 0)
00251         for (ix = 0; ix < wave->nx; ix++)
00252             for (iy = 0; iy < wave->ny; iy++)
00253                 wave->temp[ix][iy] += gsl_rng_gaussian(r, nedt);
00254
00255     /* Free... */
00256     gsl_rng_free(r);
00257 }
00258
00259 /*****
00260
00261 void create_wave(
00262     wave_t * wave,
00263     double amp,
00264     double lx,
00265     double ly,
00266     double phi,
00267     double fwhm) {
00268
00269     int ix, iy;
00270
00271     /* Loop over grid points... */
00272     for (ix = 0; ix < wave->nx; ix++)
00273         for (iy = 0; iy < wave->ny; iy++) {
00274
00275             /* Set wave perturbation... */
00276             wave->pt[ix][iy] = amp * cos((lx != 0 ? 2 * M_PI / lx : 0) * wave->x[ix]
00277                                     + (ly !=
00278                                     0 ? 2 * M_PI / ly : 0) * wave->y[iy]
00279                                     - phi * M_PI / 180.)
00280             * (fwhm > 0 ? exp(-0.5 * gsl_pow_2((wave->x[ix]) / (lx * fwhm) * 2.35)
00281                                     -
00282                                     0.5 * gsl_pow_2((wave->y[iy]) / (ly * fwhm) *
00283                                     2.35)) : 1.0);
00284
00285             /* Add perturbation to temperature... */
00286             wave->temp[ix][iy] += wave->pt[ix][iy];
00287         }
00288     }
00289
00290 /*****
00291
00292 void day2doy(
00293     int year,
00294     int mon,
00295     int day,
00296     int *doy) {
00297
00298     int d0[12] = { 1, 32, 60, 91, 121, 152, 182, 213, 244, 274, 305, 335 };
00299     int d0l[12] = { 1, 32, 61, 92, 122, 153, 183, 214, 245, 275, 306, 336 };
00300
00301     /* Get day of year... */
00302     if (year % 400 == 0 || (year % 100 != 0 && year % 4 == 0))
00303         *doy = d0l[mon - 1] + day - 1;

```

```

00304     else
00305         *doy = d0[mon - 1] + day - 1;
00306 }
00307
00308 /*****
00309
00310 void doy2day(
00311     int year,
00312     int doy,
00313     int *mon,
00314     int *day) {
00315
00316     int d0[12] = { 1, 32, 60, 91, 121, 152, 182, 213, 244, 274, 305, 335 };
00317     int d0l[12] = { 1, 32, 61, 92, 122, 153, 183, 214, 245, 275, 306, 336 };
00318     int i;
00319
00320     /* Get month and day... */
00321     if (year % 400 == 0 || (year % 100 != 0 && year % 4 == 0)) {
00322         for (i = 11; i >= 0; i--)
00323             if (d0l[i] <= doy)
00324                 break;
00325         *mon = i + 1;
00326         *day = doy - d0l[i] + 1;
00327     } else {
00328         for (i = 11; i >= 0; i--)
00329             if (d0[i] <= doy)
00330                 break;
00331         *mon = i + 1;
00332         *day = doy - d0[i] + 1;
00333     }
00334 }
00335
00336 /*****
00337
00338 void fft_help(
00339     double *fcReal,
00340     double *fcImag,
00341     int n) {
00342
00343     gsl_fft_complex_wavetable *wavetable;
00344     gsl_fft_complex_workspace *workspace;
00345
00346     double data[2 * PMAX];
00347
00348     int i;
00349
00350     /* Check size... */
00351     if (n > PMAX)
00352         ERRMSG("Too many data points!");
00353
00354     /* Allocate... */
00355     wavetable = gsl_fft_complex_wavetable_alloc((size_t) n);
00356     workspace = gsl_fft_complex_workspace_alloc((size_t) n);
00357
00358     /* Set data (real, complex)... */
00359     for (i = 0; i < n; i++) {
00360         data[2 * i] = fcReal[i];
00361         data[2 * i + 1] = fcImag[i];
00362     }
00363
00364     /* Calculate FFT... */
00365     gsl_fft_complex_forward(data, 1, (size_t) n, wavetable, workspace);
00366
00367     /* Copy data... */
00368     for (i = 0; i < n; i++) {
00369         fcReal[i] = data[2 * i];
00370         fcImag[i] = data[2 * i + 1];
00371     }
00372
00373     /* Free... */
00374     gsl_fft_complex_wavetable_free(wavetable);
00375     gsl_fft_complex_workspace_free(workspace);
00376 }
00377
00378 /*****
00379
00380 void fft(
00381     wave_t * wave,
00382     double *Amax,
00383     double *phimax,
00384     double *lhmax,
00385     double *alphamax,
00386     double *betamax,
00387     char *filename) {
00388
00389     static double A[PMAX][PMAX], phi[PMAX][PMAX], kx[PMAX], ky[PMAX],
00390         kxmax, kymax, cutReal[PMAX], cutImag[PMAX],

```

```

00391     boxImag[PMAX][PMAX], boxReal[PMAX][PMAX];
00392
00393     FILE *out;
00394
00395     int i, i2, imin, imax, j, j2, jmin, jmax, nx, ny;
00396
00397     /* Find box... */
00398     imin = jmin = 9999;
00399     imax = jmax = -9999;
00400     for (i = 0; i < wave->nx; i++)
00401         for (j = 0; j < wave->ny; j++)
00402             if (gsl_finite(wave->var[i][j])) {
00403                 imin = GSL_MIN(imin, i);
00404                 imax = GSL_MAX(imax, i);
00405                 jmin = GSL_MIN(jmin, j);
00406                 jmax = GSL_MAX(jmax, j);
00407             }
00408     nx = imax - imin + 1;
00409     ny = jmax - jmin + 1;
00410
00411     /* Copy data... */
00412     for (i = imin; i <= imax; i++)
00413         for (j = jmin; j <= jmax; j++) {
00414             if (gsl_finite(wave->pt[i][j]))
00415                 boxReal[i - imin][j - jmin] = wave->pt[i][j];
00416             else
00417                 boxReal[i - imin][j - jmin] = 0.0;
00418             boxImag[i - imin][j - jmin] = 0.0;
00419         }
00420
00421     /* FFT of the rows... */
00422     for (i = 0; i < nx; i++) {
00423         for (j = 0; j < ny; j++) {
00424             cutReal[j] = boxReal[i][j];
00425             cutImag[j] = boxImag[i][j];
00426         }
00427         fft_help(cutReal, cutImag, ny);
00428         for (j = 0; j < ny; j++) {
00429             boxReal[i][j] = cutReal[j];
00430             boxImag[i][j] = cutImag[j];
00431         }
00432     }
00433
00434     /* FFT of the columns... */
00435     for (j = 0; j < ny; j++) {
00436         for (i = 0; i < nx; i++) {
00437             cutReal[i] = boxReal[i][j];
00438             cutImag[i] = boxImag[i][j];
00439         }
00440         fft_help(cutReal, cutImag, nx);
00441         for (i = 0; i < nx; i++) {
00442             boxReal[i][j] = cutReal[i];
00443             boxImag[i][j] = cutImag[i];
00444         }
00445     }
00446
00447     /* Get frequencies, amplitude, and phase... */
00448     for (i = 0; i < nx; i++)
00449         kx[i] = 2. * M_PI * ((i < nx / 2) ? (double) i : -(double) (nx - i))
00450             / (nx * fabs(wave->x[imax] - wave->x[imin]) / (nx - 1.0));
00451     for (j = 0; j < ny; j++)
00452         ky[j] = 2. * M_PI * ((j < ny / 2) ? (double) j : -(double) (ny - j))
00453             / (ny * fabs(wave->y[jmax] - wave->y[jmin]) / (ny - 1.0));
00454     for (i = 0; i < nx; i++)
00455         for (j = 0; j < ny; j++) {
00456             A[i][j]
00457                 = (i == 0 && j == 0 ? 1.0 : 2.0) / (nx * ny)
00458                 * sqrt(gsl_pow_2(boxReal[i][j]) + gsl_pow_2(boxImag[i][j]));
00459             phi[i][j]
00460                 = 180. / M_PI * atan2(boxImag[i][j], boxReal[i][j]);
00461         }
00462
00463     /* Check frequencies... */
00464     for (i = 0; i < nx; i++)
00465         for (j = 0; j < ny; j++)
00466             if (kx[i] == 0 || ky[j] == 0) {
00467                 A[i][j] = GSL_NAN;
00468                 phi[i][j] = GSL_NAN;
00469             }
00470
00471     /* Find maximum... */
00472     *Amax = 0;
00473     for (i = 0; i < nx; i++)
00474         for (j = 0; j < ny / 2; j++)
00475             if (gsl_finite(A[i][j]) && A[i][j] > *Amax) {
00476                 *Amax = A[i][j];
00477                 *phimax = phi[i][j];

```

```

00478         kxmax = kx[i];
00479         kymax = ky[j];
00480         imax = i;
00481         jmax = j;
00482     }
00483
00484     /* Get horizontal wavelength... */
00485     *lhmax = 2 * M_PI / sqrt(gsl_pow_2(kxmax) + gsl_pow_2(kymax));
00486
00487     /* Get propagation direction in xy-plane... */
00488     *alphamax = 90. - 180. / M_PI * atan2(kxmax, kymax);
00489
00490     /* Get propagation direction in lon,lat-plane... */
00491     *betamax = *alphamax
00492     +
00493     180. / M_PI *
00494     atan2(wave->lat[wave->nx / 2 >
00495           0 ? wave->nx / 2 - 1 : wave->nx / 2][wave->ny / 2]
00496         - wave->lat[wave->nx / 2 <
00497           wave->nx - 1 ? wave->nx / 2 +
00498           1 : wave->nx / 2][wave->ny / 2],
00499         wave->lon[wave->nx / 2 >
00500           0 ? wave->nx / 2 - 1 : wave->nx / 2][wave->ny / 2]
00501         - wave->lon[wave->nx / 2 <
00502           wave->nx - 1 ? wave->nx / 2 +
00503           1 : wave->nx / 2][wave->ny / 2]);
00504
00505     /* Save FFT data... */
00506     if (filename != NULL) {
00507
00508         /* Write info... */
00509         printf("Write FFT data: %s\n", filename);
00510
00511         /* Create file... */
00512         if (!(out = fopen(filename, "w")))
00513             ERRMSG("Cannot create file!");
00514
00515         /* Write header... */
00516         fprintf(out,
00517             "# $1 = altitude [km]\n"
00518             "# $2 = wavelength in x-direction [km]\n"
00519             "# $3 = wavelength in y-direction [km]\n"
00520             "# $4 = wavenumber in x-direction [1/km]\n"
00521             "# $5 = wavenumber in y-direction [1/km]\n"
00522             "# $6 = amplitude [K]\n" "# $7 = phase [rad]\n");
00523
00524         /* Write data... */
00525         for (i = nx - 1; i > 0; i--) {
00526             fprintf(out, "\n");
00527             for (j = ny / 2; j > 0; j--) {
00528                 i2 = (i == nx / 2 ? 0 : i);
00529                 j2 = (j == ny / 2 ? 0 : j);
00530                 fprintf(out, "%g %g %g %g %g %g\n", wave->z,
00531                     (kx[i2] != 0 ? 2 * M_PI / kx[i2] : 0),
00532                     (ky[j2] != 0 ? 2 * M_PI / ky[j2] : 0),
00533                     kx[i2], ky[j2], A[i2][j2], phi[i2][j2]);
00534             }
00535         }
00536
00537         /* Close file... */
00538         fclose(out);
00539     }
00540 }
00541
00542 /*****
00543
00544 void gauss(
00545     wave_t * wave,
00546     double fwhm) {
00547
00548     static double d2, help[WX][WY], sigma2, w, wsum;
00549
00550     int ix, ix2, iy, iy2;
00551
00552     /* Check parameters... */
00553     if (fwhm <= 0)
00554         return;
00555
00556     /* Compute sigma^2... */
00557     sigma2 = gsl_pow_2(fwhm / 2.3548);
00558
00559     /* Loop over data points... */
00560     for (ix = 0; ix < wave->nx; ix++)
00561         for (iy = 0; iy < wave->ny; iy++) {
00562
00563             /* Init... */
00564             wsum = 0;

```

```

00565     help[ix][iy] = 0;
00566
00567     /* Average... */
00568     for (ix2 = 0; ix2 < wave->nx; ix2++)
00569         for (iy2 = 0; iy2 < wave->ny; iy2++) {
00570             d2 = gsl_pow_2(wave->x[ix] - wave->x[ix2])
00571                 + gsl_pow_2(wave->y[iy] - wave->y[iy2]);
00572             if (d2 <= 9 * sigma2) {
00573                 w = exp(-d2 / (2 * sigma2));
00574                 wsum += w;
00575                 help[ix][iy] += w * wave->pt[ix2][iy2];
00576             }
00577         }
00578
00579     /* Normalize... */
00580     wave->pt[ix][iy] = help[ix][iy] / wsum;
00581 }
00582 }
00583
00584 /*****
00585
00586 void hamming(
00587     wave_t * wave,
00588     int niter) {
00589
00590     static double help[WX][WY];
00591
00592     int iter, ix, iy;
00593
00594     /* Iterations... */
00595     for (iter = 0; iter < niter; iter++) {
00596
00597         /* Filter in x direction... */
00598         for (ix = 0; ix < wave->nx; ix++)
00599             for (iy = 0; iy < wave->ny; iy++)
00600                 help[ix][iy]
00601                     = 0.23 * wave->pt[ix > 0 ? ix - 1 : ix][iy]
00602                       + 0.54 * wave->pt[ix][iy]
00603                       + 0.23 * wave->pt[ix < wave->nx - 1 ? ix + 1 : ix][iy];
00604
00605         /* Filter in y direction... */
00606         for (ix = 0; ix < wave->nx; ix++)
00607             for (iy = 0; iy < wave->ny; iy++)
00608                 wave->pt[ix][iy]
00609                     = 0.23 * help[ix][iy > 0 ? iy - 1 : iy]
00610                       + 0.54 * help[ix][iy]
00611                       + 0.23 * help[ix][iy < wave->ny - 1 ? iy + 1 : iy];
00612     }
00613 }
00614
00615 /*****
00616
00617 void intpol_x(
00618     wave_t * wave,
00619     int n) {
00620
00621     gsl_interp_accel *acc;
00622     gsl_spline *spline;
00623
00624     double dummy, x[WX], xc[WX][3], xc2[WX][3], y[WX];
00625
00626     int i, ic, ix, iy;
00627
00628     /* Check parameters... */
00629     if (n <= 0)
00630         return;
00631     if (n > WX)
00632         ERRMSG("Too many data points!");
00633
00634     /* Set new x-coordinates... */
00635     for (i = 0; i < n; i++)
00636         x[i] = LIN(0.0, wave->x[0], n - 1.0, wave->x[wave->nx - 1], i);
00637
00638     /* Allocate... */
00639     acc = gsl_interp_accel_alloc();
00640     spline = gsl_spline_alloc(gsl_interp_cspline, (size_t) wave->nx);
00641
00642     /* Loop over scans... */
00643     for (iy = 0; iy < wave->ny; iy++) {
00644
00645         /* Interpolate Cartesian coordinates... */
00646         for (ix = 0; ix < wave->nx; ix++)
00647             geo2cart(0, wave->lon[ix][iy], wave->lat[ix][iy], xc[ix]);
00648         for (ic = 0; ic < 3; ic++) {
00649             for (ix = 0; ix < wave->nx; ix++)
00650                 y[ix] = xc[ix][ic];
00651             gsl_spline_init(spline, wave->x, y, (size_t) wave->nx);

```

```

00652     for (i = 0; i < n; i++)
00653         xc2[i][ic] = gsl_spline_eval(spline, x[i], acc);
00654 }
00655 for (i = 0; i < n; i++)
00656     cart2geo(xc2[i], &dummy, &wave->lon[i][iy], &wave->lat[i][iy]);
00657
00658 /* Interpolate temperature... */
00659 for (ix = 0; ix < wave->nx; ix++)
00660     y[ix] = wave->temp[ix][iy];
00661 gsl_spline_init(spline, wave->x, y, (size_t) wave->nx);
00662 for (i = 0; i < n; i++)
00663     wave->temp[i][iy] = gsl_spline_eval(spline, x[i], acc);
00664
00665 /* Interpolate background... */
00666 for (ix = 0; ix < wave->nx; ix++)
00667     y[ix] = wave->bg[ix][iy];
00668 gsl_spline_init(spline, wave->x, y, (size_t) wave->nx);
00669 for (i = 0; i < n; i++)
00670     wave->bg[i][iy] = gsl_spline_eval(spline, x[i], acc);
00671
00672 /* Interpolate perturbations... */
00673 for (ix = 0; ix < wave->nx; ix++)
00674     y[ix] = wave->pt[ix][iy];
00675 gsl_spline_init(spline, wave->x, y, (size_t) wave->nx);
00676 for (i = 0; i < n; i++)
00677     wave->pt[i][iy] = gsl_spline_eval(spline, x[i], acc);
00678
00679 /* Interpolate variance... */
00680 for (ix = 0; ix < wave->nx; ix++)
00681     y[ix] = wave->var[ix][iy];
00682 gsl_spline_init(spline, wave->x, y, (size_t) wave->nx);
00683 for (i = 0; i < n; i++)
00684     wave->var[i][iy] = gsl_spline_eval(spline, x[i], acc);
00685 }
00686
00687 /* Free... */
00688 gsl_spline_free(spline);
00689 gsl_interp_accel_free(acc);
00690
00691 /* Set new x-coordinates... */
00692 for (i = 0; i < n; i++)
00693     wave->x[i] = x[i];
00694 wave->nx = n;
00695 }
00696
00697 /*****
00698 void median(
00699     wave_t * wave,
00700     int dx) {
00701
00702     static double data[WX * WY], help[WX][WY];
00703
00704     int ix, ix2, iy, iy2;
00705
00706     size_t n;
00707
00708     /* Check parameters... */
00709     if (dx <= 0)
00710         return;
00711
00712     /* Loop over data points... */
00713     for (ix = 0; ix < wave->nx; ix++)
00714         for (iy = 0; iy < wave->ny; iy++) {
00715
00716             /* Init... */
00717             n = 0;
00718
00719             /* Get data... */
00720             for (ix2 = GSL_MAX(ix - dx, 0); ix2 < GSL_MIN(ix + dx, wave->nx - 1);
00721                  ix2++)
00722                 for (iy2 = GSL_MAX(iy - dx, 0); iy2 < GSL_MIN(iy + dx, wave->ny - 1);
00723                      iy2++) {
00724                     data[n] = wave->pt[ix2][iy2];
00725                     n++;
00726                 }
00727
00728             /* Normalize... */
00729             gsl_sort(data, 1, n);
00730             help[ix][iy] = gsl_stats_median_from_sorted_data(data, 1, n);
00731         }
00732
00733     /* Loop over data points... */
00734     for (ix = 0; ix < wave->nx; ix++)
00735         for (iy = 0; iy < wave->ny; iy++)
00736             wave->pt[ix][iy] = help[ix][iy];
00737 }
00738

```

```

00739
00740 /*****
00741
00742 void merge_y(
00743     wave_t * wave1,
00744     wave_t * wave2) {
00745
00746     double y;
00747
00748     int ix, iy;
00749
00750     /* Check data... */
00751     if (wave1->nx != wave2->nx)
00752         ERRMSG("Across-track sizes do not match!");
00753     if (wave1->ny + wave2->ny > WY)
00754         ERRMSG("Too many data points!");
00755
00756     /* Get offset in y direction... */
00757     y =
00758         wave1->y[wave1->ny - 1] + (wave1->y[wave1->ny - 1] -
00759                                     wave1->y[0]) / (wave1->ny - 1);
00760
00761     /* Merge data... */
00762     for (ix = 0; ix < wave2->nx; ix++)
00763         for (iy = 0; iy < wave2->ny; iy++) {
00764             wave1->y[wave1->ny + iy] = y + wave2->y[iy];
00765             wave1->lon[ix][wave1->ny + iy] = wave2->lon[ix][iy];
00766             wave1->lat[ix][wave1->ny + iy] = wave2->lat[ix][iy];
00767             wave1->temp[ix][wave1->ny + iy] = wave2->temp[ix][iy];
00768             wave1->bg[ix][wave1->ny + iy] = wave2->bg[ix][iy];
00769             wave1->pt[ix][wave1->ny + iy] = wave2->pt[ix][iy];
00770             wave1->var[ix][wave1->ny + iy] = wave2->var[ix][iy];
00771         }
00772
00773     /* Increment counter... */
00774     wave1->ny += wave2->ny;
00775 }
00776
00777 /*****
00778
00779 void noise(
00780     wave_t * wave,
00781     double *mu,
00782     double *sig) {
00783
00784     int ix, ix2, iy, iy2, n = 0, okay;
00785
00786     /* Init... */
00787     *mu = 0;
00788     *sig = 0;
00789
00790     /* Estimate noise (Immerkaer, 1996)... */
00791     for (ix = 1; ix < wave->nx - 1; ix++)
00792         for (iy = 1; iy < wave->ny - 1; iy++) {
00793
00794             /* Check data... */
00795             okay = 1;
00796             for (ix2 = ix - 1; ix2 <= ix + 1; ix2++)
00797                 for (iy2 = iy - 1; iy2 <= iy + 1; iy2++)
00798                     if (!gsl_finite(wave->temp[ix2][iy2]))
00799                         okay = 0;
00800             if (!okay)
00801                 continue;
00802
00803             /* Get mean noise... */
00804             n++;
00805             *mu += wave->temp[ix][iy];
00806             *sig += gsl_pow_2(+4. / 6. * wave->temp[ix][iy]
00807                             - 2. / 6. * (wave->temp[ix - 1][iy]
00808                                         + wave->temp[ix + 1][iy]
00809                                         + wave->temp[ix][iy - 1]
00810                                         + wave->temp[ix][iy + 1])
00811                             + 1. / 6. * (wave->temp[ix - 1][iy - 1]
00812                                         + wave->temp[ix + 1][iy - 1]
00813                                         + wave->temp[ix - 1][iy + 1]
00814                                         + wave->temp[ix + 1][iy + 1]));
00815         }
00816
00817     /* Normalize... */
00818     *mu /= (double) n;
00819     *sig = sqrt(*sig / (double) n);
00820 }
00821
00822 /*****
00823
00824 void period(
00825     wave_t * wave,

```

```

00826 double *Amax,
00827 double *phimax,
00828 double *lhmax,
00829 double *alphamax,
00830 double *betamax,
00831 char *filename) {
00832
00833 FILE *out;
00834
00835 static double kx[PMAX], ky[PMAX], kx_ny, ky_ny, kxmax, kymax, A[PMAX][PMAX],
00836 phi[PMAX][PMAX], cx[PMAX][WX], cy[PMAX][WY], sx[PMAX][WX], sy[PMAX][WY],
00837 a, b, c, lx, ly, lymax = 1000, dlxy = 10;
00838
00839 int i, imin, imax, j, jmin, jmax, l, lmax = 0, m, mmax = 0;
00840
00841 /* Compute wavenumbers and periodogram coefficients... */
00842 for (lx = -lymax; lx <= lymax; lx += dlxy) {
00843     kx[lmax] = (lx != 0 ? 2 * M_PI / lx : 0);
00844     for (i = 0; i < wave->nx; i++) {
00845         cx[lmax][i] = cos(kx[lmax] * wave->x[i]);
00846         sx[lmax][i] = sin(kx[lmax] * wave->x[i]);
00847     }
00848     if ((++lmax) > PMAX)
00849         ERRMSG("Too many wavenumbers for periodogram!");
00850 }
00851 for (ly = 0; ly <= lymax; ly += dlxy) {
00852     ky[mmax] = (ly != 0 ? 2 * M_PI / ly : 0);
00853     for (j = 0; j < wave->ny; j++) {
00854         cy[mmax][j] = cos(ky[mmax] * wave->y[j]);
00855         sy[mmax][j] = sin(ky[mmax] * wave->y[j]);
00856     }
00857     if ((++mmax) > PMAX)
00858         ERRMSG("Too many wavenumbers for periodogram!");
00859 }
00860
00861 /* Find area... */
00862 imin = jmin = 9999;
00863 imax = jmax = -9999;
00864 for (i = 0; i < wave->nx; i++)
00865     for (j = 0; j < wave->ny; j++)
00866         if (gsl_finite(wave->var[i][j])) {
00867             imin = GSL_MIN(imin, i);
00868             imax = GSL_MAX(imax, i);
00869             jmin = GSL_MIN(jmin, j);
00870             jmax = GSL_MAX(jmax, j);
00871         }
00872
00873 /* Get Nyquist frequencies... */
00874 kx_ny =
00875     M_PI / fabs((wave->x[imax] - wave->x[imin]) /
00876         ((double) imax - (double) imin));
00877 ky_ny =
00878     M_PI / fabs((wave->y[jmax] - wave->y[jmin]) /
00879         ((double) jmax - (double) jmin));
00880
00881 /* Loop over wavelengths... */
00882 for (l = 0; l < lmax; l++)
00883     for (m = 0; m < mmax; m++) {
00884
00885         /* Check frequencies... */
00886         if (kx[l] == 0 || fabs(kx[l]) > kx_ny ||
00887             ky[m] == 0 || fabs(ky[m]) > ky_ny) {
00888             A[l][m] = GSL_NAN;
00889             phi[l][m] = GSL_NAN;
00890             continue;
00891         }
00892
00893         /* Compute periodogram... */
00894         a = b = c = 0;
00895         for (i = imin; i <= imax; i++)
00896             for (j = jmin; j <= jmax; j++)
00897                 if (gsl_finite(wave->var[i][j])) {
00898                     a += wave->pt[i][j] * (cx[l][i] * cy[m][j] - sx[l][i] * sy[m][j]);
00899                     b += wave->pt[i][j] * (sx[l][i] * cy[m][j] + cx[l][i] * sy[m][j]);
00900                     c++;
00901                 }
00902         a *= 2. / c;
00903         b *= 2. / c;
00904
00905         /* Get amplitude and phase... */
00906         A[l][m] = sqrt(gsl_pow_2(a) + gsl_pow_2(b));
00907         phi[l][m] = atan2(b, a) * 180. / M_PI;
00908     }
00909
00910 /* Find maximum... */
00911 *Amax = 0;
00912 for (l = 0; l < lmax; l++)

```



```

00913     for (m = 0; m < mmax; m++)
00914     if (gsl_finite(A[l][m]) && A[l][m] > *Amax) {
00915         *Amax = A[l][m];
00916         *phimax = phi[l][m];
00917         kxmax = kx[l];
00918         kymax = ky[m];
00919         imax = i;
00920         jmax = j;
00921     }
00922
00923     /* Get horizontal wavelength... */
00924     *lhmax = 2 * M_PI / sqrt(gsl_pow_2(kxmax) + gsl_pow_2(kymax));
00925
00926     /* Get propagation direction in xy-plane... */
00927     *alphamax = 90. - 180. / M_PI * atan2(kxmax, kymax);
00928
00929     /* Get propagation direction in lon,lat-plane... */
00930     *betamax = *alphamax
00931     +
00932     180. / M_PI *
00933     atan2(wave->lat[wave->nx / 2 >
00934         0 ? wave->nx / 2 - 1 : wave->nx / 2][wave->ny / 2]
00935         - wave->lat[wave->nx / 2 <
00936             wave->nx - 1 ? wave->nx / 2 +
00937             1 : wave->nx / 2][wave->ny / 2],
00938         wave->lon[wave->nx / 2 >
00939             0 ? wave->nx / 2 - 1 : wave->nx / 2][wave->ny / 2]
00940         - wave->lon[wave->nx / 2 <
00941             wave->nx - 1 ? wave->nx / 2 +
00942             1 : wave->nx / 2][wave->ny / 2]);
00943
00944     /* Save periodogram data... */
00945     if (filename != NULL) {
00946
00947         /* Write info... */
00948         printf("Write periodogram data: %s\n", filename);
00949
00950         /* Create file... */
00951         if (!(out = fopen(filename, "w")))
00952             ERRMSG("Cannot create file!");
00953
00954         /* Write header... */
00955         fprintf(out,
00956             "# $1 = altitude [km]\n"
00957             "# $2 = wavelength in x-direction [km]\n"
00958             "# $3 = wavelength in y-direction [km]\n"
00959             "# $4 = wavenumber in x-direction [1/km]\n"
00960             "# $5 = wavenumber in y-direction [1/km]\n"
00961             "# $6 = amplitude [K]\n" "# $7 = phase [rad]\n");
00962
00963         /* Write data... */
00964         for (l = 0; l < lmax; l++) {
00965             fprintf(out, "\n");
00966             for (m = 0; m < mmax; m++)
00967                 fprintf(out, "%g %g %g %g %g %g %g\n", wave->z,
00968                     (kx[l] != 0 ? 2 * M_PI / kx[l] : 0),
00969                     (ky[l] != 0 ? 2 * M_PI / ky[l] : 0),
00970                     kx[l], ky[l], A[l][m], phi[l][m]);
00971         }
00972
00973         /* Close file... */
00974         fclose(out);
00975     }
00976 }
00977
00978 /*****
00979
00980 void pert2wave(
00981     pert_t * pert,
00982     wave_t * wave,
00983     int track0,
00984     int track1,
00985     int xtrack0,
00986     int xtrack1) {
00987
00988     double x0[3], x1[3];
00989
00990     int itrack, ixtrack;
00991
00992     /* Check ranges... */
00993     track0 = GSL_MIN(GSL_MAX(track0, 0), pert->ntrack - 1);
00994     track1 = GSL_MIN(GSL_MAX(track1, 0), pert->ntrack - 1);
00995     xtrack0 = GSL_MIN(GSL_MAX(xtrack0, 0), pert->nxtrack - 1);
00996     xtrack1 = GSL_MIN(GSL_MAX(xtrack1, 0), pert->nxtrack - 1);
00997
00998     /* Set size... */
00999     wave->nx = xtrack1 - xtrack0 + 1;

```

```

01000     if (wave->nx > WX)
01001         ERRMSG("Too many across-track values!");
01002     wave->ny = track1 - track0 + 1;
01003     if (wave->ny > WY)
01004         ERRMSG("Too many along-track values!");
01005
01006     /* Loop over footprints... */
01007     for (itrack = track0; itrack <= track1; itrack++)
01008         for (ixtrack = xtrack0; ixtrack <= xtrack1; ixtrack++) {
01009
01010             /* Get distances... */
01011             if (itrack == track0) {
01012                 wave->x[0] = 0;
01013                 if (ixtrack > xtrack0) {
01014                     geo2cart(0, pert->lon[itrack][ixtrack - 1],
01015                             pert->lat[itrack][ixtrack - 1], x0);
01016                     geo2cart(0, pert->lon[itrack][ixtrack],
01017                             pert->lat[itrack][ixtrack], x1);
01018                     wave->x[ixtrack - xtrack0] =
01019                         wave->x[ixtrack - xtrack0 - 1] + DIST(x0, x1);
01020                 }
01021             }
01022             if (ixtrack == xtrack0) {
01023                 wave->y[0] = 0;
01024                 if (itrack > track0) {
01025                     geo2cart(0, pert->lon[itrack - 1][ixtrack],
01026                             pert->lat[itrack - 1][ixtrack], x0);
01027                     geo2cart(0, pert->lon[itrack][ixtrack],
01028                             pert->lat[itrack][ixtrack], x1);
01029                     wave->y[ixtrack - track0] =
01030                         wave->y[ixtrack - track0 - 1] + DIST(x0, x1);
01031                 }
01032             }
01033
01034             /* Save geolocation... */
01035             wave->time = pert->time[(track0 + track1) / 2][(xtrack0 + xtrack1) / 2];
01036             wave->z = 0;
01037             wave->lon[ixtrack - xtrack0][itrack - track0] =
01038                 pert->lon[itrack][ixtrack];
01039             wave->lat[ixtrack - xtrack0][itrack - track0] =
01040                 pert->lat[itrack][ixtrack];
01041
01042             /* Save temperature data... */
01043             wave->temp[ixtrack - xtrack0][itrack - track0]
01044                 = pert->bt[itrack][ixtrack];
01045             wave->bg[ixtrack - xtrack0][itrack - track0]
01046                 = pert->bt[itrack][ixtrack] - pert->pt[itrack][ixtrack];
01047             wave->pt[ixtrack - xtrack0][itrack - track0]
01048                 = pert->pt[itrack][ixtrack];
01049             wave->var[ixtrack - xtrack0][itrack - track0]
01050                 = pert->var[itrack][ixtrack];
01051         }
01052 }
01053
01054 /*****
01055 void read_l1(
01056     char *filename,
01057     airs_l1_t * l1) {
01058
01059     int ncid, varid;
01060
01061     /* Open netCDF file... */
01062     printf("Read AIRS Level-1 file: %s\n", filename);
01063     NC(nc_open(filename, NC_NOWRITE, &ncid));
01064
01065     /* Read data... */
01066     NC(nc_inq_varid(ncid, "l1_time", &varid));
01067     NC(nc_get_var_double(ncid, varid, l1->time[0]));
01068     NC(nc_inq_varid(ncid, "l1_lon", &varid));
01069     NC(nc_get_var_double(ncid, varid, l1->lon[0]));
01070     NC(nc_inq_varid(ncid, "l1_lat", &varid));
01071     NC(nc_get_var_double(ncid, varid, l1->lat[0]));
01072     NC(nc_inq_varid(ncid, "l1_sat_z", &varid));
01073     NC(nc_get_var_double(ncid, varid, l1->sat_z));
01074     NC(nc_inq_varid(ncid, "l1_sat_lon", &varid));
01075     NC(nc_get_var_double(ncid, varid, l1->sat_lon));
01076     NC(nc_inq_varid(ncid, "l1_sat_lat", &varid));
01077     NC(nc_get_var_double(ncid, varid, l1->sat_lat));
01078     NC(nc_inq_varid(ncid, "l1_nu", &varid));
01079     NC(nc_get_var_double(ncid, varid, l1->nu));
01080     NC(nc_inq_varid(ncid, "l1_rad", &varid));
01081     NC(nc_get_var_float(ncid, varid, l1->rad[0][0]));
01082
01083     /* Close file... */
01084     NC(nc_close(ncid));
01085 }
01086 */

```

```

01087
01088 /*****
01089
01090 void read_l2(
01091     char *filename,
01092     airs_l2_t * l2) {
01093
01094     int ncid, varid;
01095
01096     /* Open netCDF file... */
01097     printf("Read AIRS Level-2 file: %s\n", filename);
01098     NC(nc_open(filename, NC_NOWRITE, &ncid));
01099
01100     /* Read data... */
01101     NC(nc_inq_varid(ncid, "l2_time", &varid));
01102     NC(nc_get_var_double(ncid, varid, l2->time[0]));
01103     NC(nc_inq_varid(ncid, "l2_z", &varid));
01104     NC(nc_get_var_double(ncid, varid, l2->z[0][0]));
01105     NC(nc_inq_varid(ncid, "l2_lon", &varid));
01106     NC(nc_get_var_double(ncid, varid, l2->lon[0]));
01107     NC(nc_inq_varid(ncid, "l2_lat", &varid));
01108     NC(nc_get_var_double(ncid, varid, l2->lat[0]));
01109     NC(nc_inq_varid(ncid, "l2_press", &varid));
01110     NC(nc_get_var_double(ncid, varid, l2->p));
01111     NC(nc_inq_varid(ncid, "l2_temp", &varid));
01112     NC(nc_get_var_double(ncid, varid, l2->t[0][0]));
01113
01114     /* Close file... */
01115     NC(nc_close(ncid));
01116 }
01117
01118 /*****
01119
01120 void read_pert(
01121     char *filename,
01122     char *pertname,
01123     pert_t * pert) {
01124
01125     static char varname[LEN];
01126
01127     static int dimid[2], ncid, varid;
01128
01129     static size_t itrack, ntrack, nxtrack, start[2] = { 0, 0 }, count[2] = {
01130         1, 1};
01131
01132     /* Write info... */
01133     printf("Read perturbation data: %s\n", filename);
01134
01135     /* Open netCDF file... */
01136     NC(nc_open(filename, NC_NOWRITE, &ncid));
01137
01138     /* Get dimensions... */
01139     NC(nc_inq_dimid(ncid, "NTRACK", &dimid[0]));
01140     NC(nc_inq_dimid(ncid, "NXTRACK", &dimid[1]));
01141     NC(nc_inq_dimlen(ncid, dimid[0], &ntrack));
01142     NC(nc_inq_dimlen(ncid, dimid[1], &nxtrack));
01143     if (nxtrack > PERT_NXTRACK)
01144         ERRMSG("Too many tracks!");
01145     if (ntrack > PERT_NTRACK)
01146         ERRMSG("Too many scans!");
01147     pert->ntrack = (int) ntrack;
01148     pert->nxtrack = (int) nxtrack;
01149     count[1] = nxtrack;
01150
01151     /* Read data... */
01152     NC(nc_inq_varid(ncid, "time", &varid));
01153     for (itrack = 0; itrack < ntrack; itrack++) {
01154         start[0] = itrack;
01155         NC(nc_get_vara_double(ncid, varid, start, count, pert->time[itrack]));
01156     }
01157
01158     NC(nc_inq_varid(ncid, "lon", &varid));
01159     for (itrack = 0; itrack < ntrack; itrack++) {
01160         start[0] = itrack;
01161         NC(nc_get_vara_double(ncid, varid, start, count, pert->lon[itrack]));
01162     }
01163
01164     NC(nc_inq_varid(ncid, "lat", &varid));
01165     for (itrack = 0; itrack < ntrack; itrack++) {
01166         start[0] = itrack;
01167         NC(nc_get_vara_double(ncid, varid, start, count, pert->lat[itrack]));
01168     }
01169
01170     NC(nc_inq_varid(ncid, "bt_8mu", &varid));
01171     for (itrack = 0; itrack < ntrack; itrack++) {
01172         start[0] = itrack;
01173         NC(nc_get_vara_double(ncid, varid, start, count, pert->dc[itrack]));

```

```

01174     }
01175
01176     sprintf(varname, "bt_%s", pertname);
01177     NC(nc_inq_varid(ncid, varname, &varid));
01178     for (itrack = 0; itrack < ntrack; itrack++) {
01179         start[0] = itrack;
01180         NC(nc_get_vara_double(ncid, varid, start, count, pert->bt[itrack]));
01181     }
01182
01183     sprintf(varname, "bt_%s_pt", pertname);
01184     NC(nc_inq_varid(ncid, varname, &varid));
01185     for (itrack = 0; itrack < ntrack; itrack++) {
01186         start[0] = itrack;
01187         NC(nc_get_vara_double(ncid, varid, start, count, pert->pt[itrack]));
01188     }
01189
01190     sprintf(varname, "bt_%s_var", pertname);
01191     NC(nc_inq_varid(ncid, varname, &varid));
01192     for (itrack = 0; itrack < ntrack; itrack++) {
01193         start[0] = itrack;
01194         NC(nc_get_vara_double(ncid, varid, start, count, pert->var[itrack]));
01195     }
01196
01197     /* Close file... */
01198     NC(nc_close(ncid));
01199 }
01200
01201 /*****
01202
01203 void read_retr(
01204     char *filename,
01205     ret_t *ret) {
01206
01207     static double help[NDS * NPG];
01208
01209     int dimid, ids = 0, ip, ncid, varid;
01210
01211     size_t itrack, ixtrack, nds, np, ntrack, nxtrack;
01212
01213     /* Write info... */
01214     printf("Read retrieval data: %s\n", filename);
01215
01216     /* Open netCDF file... */
01217     NC(nc_open(filename, NC_NOWRITE, &ncid));
01218
01219     /* Read new retrieval file format... */
01220     if (nc_inq_dimid(ncid, "L1_NTRACK", &dimid) == NC_NOERR) {
01221
01222         /* Get dimensions... */
01223         NC(nc_inq_dimid(ncid, "RET_NP", &dimid));
01224         NC(nc_inq_dimlen(ncid, dimid, &np));
01225         ret->np = (int) np;
01226         if (ret->np > NPG)
01227             ERRMSG("Too many data points!");
01228
01229         NC(nc_inq_dimid(ncid, "L1_NTRACK", &dimid));
01230         NC(nc_inq_dimlen(ncid, dimid, &ntrack));
01231         NC(nc_inq_dimid(ncid, "L1_NXTRACK", &dimid));
01232         NC(nc_inq_dimlen(ncid, dimid, &nxtrack));
01233         ret->nds = (int) (ntrack * nxtrack);
01234         if (ret->nds > NDS)
01235             ERRMSG("Too many data sets!");
01236
01237         /* Read time... */
01238         NC(nc_inq_varid(ncid, "l1_time", &varid));
01239         NC(nc_get_var_double(ncid, varid, help));
01240         ids = 0;
01241         for (itrack = 0; itrack < ntrack; itrack++)
01242             for (ixtrack = 0; ixtrack < nxtrack; ixtrack++) {
01243                 for (ip = 0; ip < ret->np; ip++)
01244                     ret->time[ids][ip] = help[ids];
01245                 ids++;
01246             }
01247
01248         /* Read altitudes... */
01249         NC(nc_inq_varid(ncid, "ret_z", &varid));
01250         NC(nc_get_var_double(ncid, varid, help));
01251         ids = 0;
01252         for (itrack = 0; itrack < ntrack; itrack++)
01253             for (ixtrack = 0; ixtrack < nxtrack; ixtrack++) {
01254                 for (ip = 0; ip < ret->np; ip++)
01255                     ret->z[ids][ip] = help[ip];
01256                 ids++;
01257             }
01258
01259         /* Read longitudes... */
01260         NC(nc_inq_varid(ncid, "l1_lon", &varid));

```

```

01261     NC(nc_get_var_double(ncid, varid, help));
01262     ids = 0;
01263     for (itrack = 0; itrack < ntrack; itrack++)
01264         for (ixtrack = 0; ixtrack < nxtrack; ixtrack++) {
01265             for (ip = 0; ip < ret->np; ip++)
01266                 ret->lon[ids][ip] = help[ids];
01267             ids++;
01268         }
01269
01270     /* Read latitudes... */
01271     NC(nc_inq_varid(ncid, "ll_lat", &varid));
01272     NC(nc_get_var_double(ncid, varid, help));
01273     ids = 0;
01274     for (itrack = 0; itrack < ntrack; itrack++)
01275         for (ixtrack = 0; ixtrack < nxtrack; ixtrack++) {
01276             for (ip = 0; ip < ret->np; ip++)
01277                 ret->lat[ids][ip] = help[ids];
01278             ids++;
01279         }
01280
01281     /* Read temperatures... */
01282     NC(nc_inq_varid(ncid, "ret_temp", &varid));
01283     NC(nc_get_var_double(ncid, varid, help));
01284     ids = 0;
01285     for (itrack = 0; itrack < ntrack; itrack++)
01286         for (ixtrack = 0; ixtrack < nxtrack; ixtrack++) {
01287             for (ip = 0; ip < ret->np; ip++)
01288                 ret->t[ids][ip] =
01289                     help[(itrack * nxtrack + ixtrack) * (size_t) np + (size_t) ip];
01290             ids++;
01291         }
01292 }
01293
01294 /* Read old retrieval file format... */
01295 if (nc_inq_dimid(ncid, "np", &dimid) == NC_NOERR) {
01296
01297     /* Get dimensions... */
01298     NC(nc_inq_dimid(ncid, "np", &dimid));
01299     NC(nc_inq_dimlen(ncid, dimid, &np));
01300     ret->np = (int) np;
01301     if (ret->np > NPG)
01302         ERRMSG("Too many data points!");
01303
01304     NC(nc_inq_dimid(ncid, "nds", &dimid));
01305     NC(nc_inq_dimlen(ncid, dimid, &nds));
01306     ret->nds = (int) nds;
01307     if (ret->nds > NDS)
01308         ERRMSG("Too many data sets!");
01309
01310     /* Read data... */
01311     NC(nc_inq_varid(ncid, "time", &varid));
01312     NC(nc_get_var_double(ncid, varid, help));
01313     read_retr_help(help, ret->nds, ret->np, ret->time);
01314
01315     NC(nc_inq_varid(ncid, "z", &varid));
01316     NC(nc_get_var_double(ncid, varid, help));
01317     read_retr_help(help, ret->nds, ret->np, ret->z);
01318
01319     NC(nc_inq_varid(ncid, "lon", &varid));
01320     NC(nc_get_var_double(ncid, varid, help));
01321     read_retr_help(help, ret->nds, ret->np, ret->lon);
01322
01323     NC(nc_inq_varid(ncid, "lat", &varid));
01324     NC(nc_get_var_double(ncid, varid, help));
01325     read_retr_help(help, ret->nds, ret->np, ret->lat);
01326
01327     NC(nc_inq_varid(ncid, "press", &varid));
01328     NC(nc_get_var_double(ncid, varid, help));
01329     read_retr_help(help, ret->nds, ret->np, ret->p);
01330
01331     NC(nc_inq_varid(ncid, "temp", &varid));
01332     NC(nc_get_var_double(ncid, varid, help));
01333     read_retr_help(help, ret->nds, ret->np, ret->t);
01334
01335     NC(nc_inq_varid(ncid, "temp_apr", &varid));
01336     NC(nc_get_var_double(ncid, varid, help));
01337     read_retr_help(help, ret->nds, ret->np, ret->t_apr);
01338
01339     NC(nc_inq_varid(ncid, "temp_total", &varid));
01340     NC(nc_get_var_double(ncid, varid, help));
01341     read_retr_help(help, ret->nds, ret->np, ret->t_tot);
01342
01343     NC(nc_inq_varid(ncid, "temp_noise", &varid));
01344     NC(nc_get_var_double(ncid, varid, help));
01345     read_retr_help(help, ret->nds, ret->np, ret->t_noise);
01346
01347     NC(nc_inq_varid(ncid, "temp_formod", &varid));

```

```

01348     NC(nc_get_var_double(ncid, varid, help));
01349     read_retr_help(help, ret->nds, ret->np, ret->t_fm);
01350
01351     NC(nc_inq_varid(ncid, "temp_cont", &varid));
01352     NC(nc_get_var_double(ncid, varid, help));
01353     read_retr_help(help, ret->nds, ret->np, ret->t_cont);
01354
01355     NC(nc_inq_varid(ncid, "temp_res", &varid));
01356     NC(nc_get_var_double(ncid, varid, help));
01357     read_retr_help(help, ret->nds, ret->np, ret->t_res);
01358
01359     NC(nc_inq_varid(ncid, "chisq", &varid));
01360     NC(nc_get_var_double(ncid, varid, ret->chisq));
01361 }
01362
01363 /* Close file... */
01364 NC(nc_close(ncid));
01365 }
01366
01367 /*****
01368
01369 void read_retr_help(
01370     double *help,
01371     int nds,
01372     int np,
01373     double mat[NDS][NPG]) {
01374
01375     int ids, ip, n = 0;
01376
01377     for (ip = 0; ip < np; ip++)
01378         for (ids = 0; ids < nds; ids++)
01379             mat[ids][ip] = help[n++];
01380 }
01381
01382 /*****
01383
01384 void read_wave(
01385     char *filename,
01386     wave_t * wave) {
01387
01388     FILE *in;
01389
01390     char line[LEN];
01391
01392     double rtime, rz, rlon, rlat, rx, ry, ryold = -1e10, rtemp, rbg, rpt, rvar;
01393
01394     /* Init... */
01395     wave->nx = 0;
01396     wave->ny = 0;
01397
01398     /* Write info... */
01399     printf("Read wave data: %s\n", filename);
01400
01401     /* Open file... */
01402     if (!(in = fopen(filename, "r")))
01403         ERRMSG("Cannot open file!");
01404
01405     /* Read data... */
01406     while (fgets(line, LEN, in))
01407         if (sscanf(line, "%lg %lg %lg %lg %lg %lg %lg %lg %lg", &rtime,
01408             &rz, &rlon, &rlat, &rx, &ry, &rtemp, &rbg, &rpt,
01409             &rvar) == 10) {
01410
01411         /* Set index... */
01412         if (ry != ryold) {
01413             if ((++wave->ny >= WY))
01414                 ERRMSG("Too many y-values!");
01415             wave->nx = 0;
01416         } else if ((++wave->nx >= WX))
01417             ERRMSG("Too many x-values!");
01418         ryold = ry;
01419
01420         /* Save data... */
01421         wave->time = rtime;
01422         wave->z = rz;
01423         wave->lon[wave->nx][wave->ny] = rlon;
01424         wave->lat[wave->nx][wave->ny] = rlat;
01425         wave->x[wave->nx] = rx;
01426         wave->y[wave->ny] = ry;
01427         wave->temp[wave->nx][wave->ny] = rtemp;
01428         wave->bg[wave->nx][wave->ny] = rbg;
01429         wave->pt[wave->nx][wave->ny] = rpt;
01430         wave->var[wave->nx][wave->ny] = rvar;
01431     }
01432
01433     /* Increment counters... */
01434     wave->nx++;

```

```

01435     wave->ny++;
01436
01437     /* Close file... */
01438     fclose(in);
01439 }
01440
01441 /*****
01442
01443 void rad2wave(
01444     airs_rad_gran_t * gran,
01445     double *nu,
01446     int nd,
01447     wave_t * wave) {
01448
01449     double x0[3], x1[3];
01450
01451     int ichan[AIRS_RAD_CHANNEL], id, track, xtrack;
01452
01453     /* Get channel numbers... */
01454     for (id = 0; id < nd; id++) {
01455         for (ichan[id] = 0; ichan[id] < AIRS_RAD_CHANNEL; ichan[id]++)
01456             if (fabs(gran->nominal_freq[ichan[id]] - nu[id]) < 0.1)
01457                 break;
01458         if (ichan[id] >= AIRS_RAD_CHANNEL)
01459             ERRMSG("Could not find channel!");
01460     }
01461
01462     /* Set size... */
01463     wave->nx = AIRS_RAD_GEOXTRACK;
01464     wave->ny = AIRS_RAD_GEOTRACK;
01465     if (wave->nx > WX || wave->ny > WY)
01466         ERRMSG("Wave struct too small!");
01467
01468     /* Set Cartesian coordinates... */
01469     geo2cart(0, gran->Longitude[0][0], gran->Latitude[0][0], x0);
01470     for (xtrack = 0; xtrack < AIRS_RAD_GEOXTRACK; xtrack++) {
01471         geo2cart(0, gran->Longitude[0][xtrack], gran->Latitude[0][xtrack], x1);
01472         wave->x[xtrack] = DIST(x0, x1);
01473     }
01474     for (track = 0; track < AIRS_RAD_GEOTRACK; track++) {
01475         geo2cart(0, gran->Longitude[track][0], gran->Latitude[track][0], x1);
01476         wave->y[track] = DIST(x0, x1);
01477     }
01478
01479     /* Set geolocation... */
01480     wave->time =
01481         gran->Time[AIRS_RAD_GEOTRACK / 2][AIRS_RAD_GEOXTRACK / 2] - 220838400;
01482     wave->z = 0;
01483     for (track = 0; track < AIRS_RAD_GEOTRACK; track++)
01484         for (xtrack = 0; xtrack < AIRS_RAD_GEOXTRACK; xtrack++) {
01485             wave->lon[xtrack][track] = gran->Longitude[track][xtrack];
01486             wave->lat[xtrack][track] = gran->Latitude[track][xtrack];
01487         }
01488
01489     /* Set brightness temperature... */
01490     for (track = 0; track < AIRS_RAD_GEOTRACK; track++)
01491         for (xtrack = 0; xtrack < AIRS_RAD_GEOXTRACK; xtrack++) {
01492             wave->temp[xtrack][track] = 0;
01493             wave->bg[xtrack][track] = 0;
01494             wave->pt[xtrack][track] = 0;
01495             wave->var[xtrack][track] = 0;
01496             for (id = 0; id < nd; id++) {
01497                 if ((gran->state[track][xtrack] != 0)
01498                     || (gran->ExcludedChans[ichan[id]] > 2)
01499                     || (gran->CalChanSummary[ichan[id]] & 8)
01500                     || (gran->CalChanSummary[ichan[id]] & (32 + 64))
01501                     || (gran->CalFlag[track][ichan[id]] & 16))
01502                     wave->temp[xtrack][track] = GSL_NAN;
01503             }
01504             else
01505                 wave->temp[xtrack][track]
01506                     += brightness(gran->radiances[track][xtrack][ichan[id]] * 1e-3,
01507                                 gran->nominal_freq[ichan[id]]) / nd;
01508         }
01509 }
01510
01511 /*****
01512
01513 void ret2wave(
01514     ret_t * ret,
01515     wave_t * wave,
01516     int dataset,
01517     int ip) {
01518
01519     double x0[3], x1[3];
01520
01521     int ids, ix, iy;

```

```

01522
01523 /* Initialize... */
01524 wave->nx = 90;
01525 if (wave->nx > WX)
01526     ERRMSG("Too many across-track values!");
01527 wave->ny = 135;
01528 if (wave->ny > WY)
01529     ERRMSG("Too many along-track values!");
01530 if (ip < 0 || ip >= ret->np)
01531     ERRMSG("Altitude index out of range!");
01532
01533 /* Loop over data sets and data points... */
01534 for (ids = 0; ids < ret->nds; ids++) {
01535
01536     /* Get horizontal indices... */
01537     ix = ids % 90;
01538     iy = ids / 90;
01539
01540     /* Get distances... */
01541     if (iy == 0) {
01542         geo2cart(0.0, ret->lon[0][0], ret->lat[0][0], x0);
01543         geo2cart(0.0, ret->lon[ids][ip], ret->lat[ids][ip], x1);
01544         wave->x[ix] = DIST(x0, x1);
01545     }
01546     if (ix == 0) {
01547         geo2cart(0.0, ret->lon[0][0], ret->lat[0][0], x0);
01548         geo2cart(0.0, ret->lon[ids][ip], ret->lat[ids][ip], x1);
01549         wave->y[iy] = DIST(x0, x1);
01550     }
01551
01552     /* Save geolocation... */
01553     wave->time = ret->time[0][0];
01554     if (ix == 0 && iy == 0)
01555         wave->z = ret->z[ids][ip];
01556     wave->lon[ix][iy] = ret->lon[ids][ip];
01557     wave->lat[ix][iy] = ret->lat[ids][ip];
01558
01559     /* Save temperature... */
01560     if (dataset == 1)
01561         wave->temp[ix][iy] = ret->t[ids][ip];
01562     else if (dataset == 2)
01563         wave->temp[ix][iy] = ret->t_apr[ids][ip];
01564 }
01565 }
01566
01567 /*****
01568
01569 double sza(
01570     double sec,
01571     double lon,
01572     double lat) {
01573
01574     double D, dec, e, g, GMST, h, L, LST, q, ra;
01575
01576     /* Number of days and fraction with respect to 2000-01-01T12:00Z... */
01577     D = sec / 86400 - 0.5;
01578
01579     /* Geocentric apparent ecliptic longitude [rad]... */
01580     g = (357.529 + 0.98560028 * D) * M_PI / 180;
01581     q = 280.459 + 0.98564736 * D;
01582     L = (q + 1.915 * sin(g) + 0.020 * sin(2 * g)) * M_PI / 180;
01583
01584     /* Mean obliquity of the ecliptic [rad]... */
01585     e = (23.439 - 0.00000036 * D) * M_PI / 180;
01586
01587     /* Declination [rad]... */
01588     dec = asin(sin(e) * sin(L));
01589
01590     /* Right ascension [rad]... */
01591     ra = atan2(cos(e) * sin(L), cos(L));
01592
01593     /* Greenwich Mean Sidereal Time [h]... */
01594     GMST = 18.697374558 + 24.06570982441908 * D;
01595
01596     /* Local Sidereal Time [h]... */
01597     LST = GMST + lon / 15;
01598
01599     /* Hour angle [rad]... */
01600     h = LST / 12 * M_PI - ra;
01601
01602     /* Convert latitude... */
01603     lat *= M_PI / 180;
01604
01605     /* Return solar zenith angle [deg]... */
01606     return acos(sin(lat) * sin(dec) +
01607         cos(lat) * cos(dec) * cos(h)) * 180 / M_PI;
01608 }

```



```

01609
01610 /*****
01611
01612 void variance(
01613     wave_t * wave,
01614     double dh) {
01615
01616     double dh2, mu, help;
01617
01618     int dx, dy, ix, ix2, iy, iy2, n;
01619
01620     /* Check parameters... */
01621     if (dh <= 0)
01622         return;
01623
01624     /* Compute squared radius... */
01625     dh2 = gsl_pow_2(dh);
01626
01627     /* Get sampling distances... */
01628     dx =
01629         (int) (dh / fabs(wave->x[wave->nx - 1] - wave->x[0]) * (wave->nx - 1.0) +
01630             1);
01631     dy =
01632         (int) (dh / fabs(wave->y[wave->ny - 1] - wave->y[0]) * (wave->ny - 1.0) +
01633             1);
01634
01635     /* Loop over data points... */
01636     for (ix = 0; ix < wave->nx; ix++)
01637         for (iy = 0; iy < wave->ny; iy++) {
01638
01639             /* Init... */
01640             mu = help = 0;
01641             n = 0;
01642
01643             /* Get data... */
01644             for (ix2 = GSL_MAX(ix - dx, 0); ix2 <= GSL_MIN(ix + dx, wave->nx - 1);
01645                 ix2++)
01646                 for (iy2 = GSL_MAX(iy - dy, 0); iy2 <= GSL_MIN(iy + dy, wave->ny - 1);
01647                     iy2++)
01648                     if ((gsl_pow_2(wave->x[ix] - wave->x[ix2])
01649                         + gsl_pow_2(wave->y[iy] - wave->y[iy2])) <= dh2)
01650                         if (gsl_finite(wave->pt[ix2][iy2])) {
01651                             mu += wave->pt[ix2][iy2];
01652                             help += gsl_pow_2(wave->pt[ix2][iy2]);
01653                             n++;
01654                         }
01655
01656             /* Compute local variance... */
01657             if (n > 1)
01658                 wave->var[ix][iy] = help / n - gsl_pow_2(mu / n);
01659             else
01660                 wave->var[ix][iy] = GSL_NAN;
01661         }
01662 }
01663
01664 /*****
01665
01666 void write_ll(
01667     char *filename,
01668     airs_ll_t * ll) {
01669
01670     int dimid[10], ncid, time_id, lon_id, lat_id,
01671         sat_z_id, sat_lon_id, sat_lat_id, nu_id, rad_id;
01672
01673     /* Open or create netCDF file... */
01674     printf("Write AIRS Level-1 file: %s\n", filename);
01675     if (nc_open(filename, NC_WRITE, &ncid) != NC_NOERR) {
01676         NC(nc_create(filename, NC_CLOBBER, &ncid));
01677     } else {
01678         NC(nc_redef(ncid));
01679     }
01680
01681     /* Set dimensions... */
01682     if (nc_inq_dimid(ncid, "L1_NTRACK", &dimid[0]) != NC_NOERR)
01683         NC(nc_def_dim(ncid, "L1_NTRACK", L1_NTRACK, &dimid[0]));
01684     if (nc_inq_dimid(ncid, "L1_NXTRACK", &dimid[1]) != NC_NOERR)
01685         NC(nc_def_dim(ncid, "L1_NXTRACK", L1_NXTRACK, &dimid[1]));
01686     if (nc_inq_dimid(ncid, "L1_NCHAN", &dimid[2]) != NC_NOERR)
01687         NC(nc_def_dim(ncid, "L1_NCHAN", L1_NCHAN, &dimid[2]));
01688
01689     /* Add variables... */
01690     add_var(ncid, "ll_time", "s", "time (seconds since 2000-01-01T00:00Z)",
01691         NC_DOUBLE, dimid, &time_id, 2);
01692     add_var(ncid, "ll_lon", "deg", "longitude", NC_DOUBLE, dimid, &lon_id, 2);
01693     add_var(ncid, "ll_lat", "deg", "latitude", NC_DOUBLE, dimid, &lat_id, 2);
01694     add_var(ncid, "ll_sat_z", "km", "satellite altitude",
01695         NC_DOUBLE, dimid, &sat_z_id, 1);

```

```

01696     add_var(ncid, "l1_sat_lon", "deg", "satellite longitude",
01697             NC_DOUBLE, dimid, &sat_lon_id, 1);
01698     add_var(ncid, "l1_sat_lat", "deg", "satellite latitude",
01699             NC_DOUBLE, dimid, &sat_lat_id, 1);
01700     add_var(ncid, "l1_nu", "cm^-1", "channel wavenumber",
01701             NC_DOUBLE, &dimid[2], &nu_id, 1);
01702     add_var(ncid, "l1_rad", "W/(m^2 sr cm^-1)", "channel radiance",
01703             NC_FLOAT, dimid, &rad_id, 3);
01704
01705     /* Leave define mode... */
01706     NC(nc_enddef(ncid));
01707
01708     /* Write data... */
01709     NC(nc_put_var_double(ncid, time_id, l1->time[0]));
01710     NC(nc_put_var_double(ncid, lon_id, l1->lon[0]));
01711     NC(nc_put_var_double(ncid, lat_id, l1->lat[0]));
01712     NC(nc_put_var_double(ncid, sat_z_id, l1->sat_z));
01713     NC(nc_put_var_double(ncid, sat_lon_id, l1->sat_lon));
01714     NC(nc_put_var_double(ncid, sat_lat_id, l1->sat_lat));
01715     NC(nc_put_var_double(ncid, nu_id, l1->nu));
01716     NC(nc_put_var_float(ncid, rad_id, l1->rad[0][0]));
01717
01718     /* Close file... */
01719     NC(nc_close(ncid));
01720 }
01721
01722 /*****
01723
01724 void write_l2(
01725     char *filename,
01726     airs_l2_t * l2) {
01727
01728     int dimid[10], ncid, time_id, z_id, lon_id, lat_id, p_id, t_id;
01729
01730     /* Create netCDF file... */
01731     printf("Write AIRS Level-2 file: %s\n", filename);
01732     if (nc_open(filename, NC_WRITE, &ncid) != NC_NOERR) {
01733         NC(nc_create(filename, NC_CLOBBER, &ncid));
01734     } else {
01735         NC(nc_redef(ncid));
01736     }
01737
01738     /* Set dimensions... */
01739     if (nc_inq_dimid(ncid, "L2_NTRACK", &dimid[0]) != NC_NOERR)
01740         NC(nc_def_dim(ncid, "L2_NTRACK", L2_NTRACK, &dimid[0]));
01741     if (nc_inq_dimid(ncid, "L2_NXTRACK", &dimid[1]) != NC_NOERR)
01742         NC(nc_def_dim(ncid, "L2_NXTRACK", L2_NXTRACK, &dimid[1]));
01743     if (nc_inq_dimid(ncid, "L2_NLAY", &dimid[2]) != NC_NOERR)
01744         NC(nc_def_dim(ncid, "L2_NLAY", L2_NLAY, &dimid[2]));
01745
01746     /* Add variables... */
01747     add_var(ncid, "l2_time", "s", "time (seconds since 2000-01-01T00:00Z)",
01748             NC_DOUBLE, dimid, &time_id, 2);
01749     add_var(ncid, "l2_z", "km", "altitude", NC_DOUBLE, dimid, &z_id, 3);
01750     add_var(ncid, "l2_lon", "deg", "longitude", NC_DOUBLE, dimid, &lon_id, 2);
01751     add_var(ncid, "l2_lat", "deg", "latitude", NC_DOUBLE, dimid, &lat_id, 2);
01752     add_var(ncid, "l2_press", "hPa", "pressure",
01753             NC_DOUBLE, &dimid[2], &p_id, 1);
01754     add_var(ncid, "l2_temp", "K", "temperature", NC_DOUBLE, dimid, &t_id, 3);
01755
01756     /* Leave define mode... */
01757     NC(nc_enddef(ncid));
01758
01759     /* Write data... */
01760     NC(nc_put_var_double(ncid, time_id, l2->time[0]));
01761     NC(nc_put_var_double(ncid, z_id, l2->z[0][0]));
01762     NC(nc_put_var_double(ncid, lon_id, l2->lon[0]));
01763     NC(nc_put_var_double(ncid, lat_id, l2->lat[0]));
01764     NC(nc_put_var_double(ncid, p_id, l2->p));
01765     NC(nc_put_var_double(ncid, t_id, l2->t[0][0]));
01766
01767     /* Close file... */
01768     NC(nc_close(ncid));
01769 }
01770
01771 /*****
01772
01773 void write_wave(
01774     char *filename,
01775     wave_t * wave) {
01776
01777     FILE *out;
01778
01779     int i, j;
01780
01781     /* Write info... */
01782     printf("Write wave data: %s\n", filename);

```

```

01783
01784  /* Create file... */
01785  if (!(out = fopen(filename, "w")))
01786      ERRMSG("Cannot create file!");
01787
01788  /* Write header... */
01789  fprintf(out,
01790          "# $1 = time (seconds since 2000-01-01T00:00Z)\n"
01791          "# $2 = altitude [km]\n"
01792          "# $3 = longitude [deg]\n"
01793          "# $4 = latitude [deg]\n"
01794          "# $5 = across-track distance [km]\n"
01795          "# $6 = along-track distance [km]\n"
01796          "# $7 = temperature [K]\n"
01797          "# $8 = background [K]\n"
01798          "# $9 = perturbation [K]\n" "# $10 = variance [K^2]\n");
01799
01800  /* Write data... */
01801  for (j = 0; j < wave->ny; j++) {
01802      fprintf(out, "\n");
01803      for (i = 0; i < wave->nx; i++)
01804          fprintf(out, "%.2f %g %g %g %g %g %g %g %g\n",
01805                  wave->time, wave->z, wave->lon[i][j], wave->lat[i][j],
01806                  wave->x[i], wave->y[j], wave->temp[i][j], wave->bg[i][j],
01807                  wave->pt[i][j], wave->var[i][j]);
01808  }
01809
01810  /* Close file... */
01811  fclose(out);
01812 }

```

## 5.27 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\\_att](#) (int ncid, int varid, const char \*unit, const char \*long\_name)  
*Add variable attributes to netCDF file.*
- 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 \*fclmag, 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.*

### 5.27.1 Function Documentation

#### 5.27.1.1 void `add_att` ( int *ncid*, int *varid*, const char \* *unit*, const char \* *long\_name* )

Add variable attributes to netCDF file.

Definition at line 5 of file `libairs.c`.

```
00009             {
00010
00011             /* Set long name... */
00012             NC(nc_put_att_text(ncid, varid, "long_name", strlen(long_name), long_name));
00013
00014             /* Set units... */
00015             NC(nc_put_att_text(ncid, varid, "units", strlen(unit), unit));
00016 }
```

#### 5.27.1.2 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.

Add variable to netCDF file.

Definition at line 20 of file `libairs.c`.

```
00028             {
00029
00030             /* Check if variable exists... */
00031             if (nc_inq_varid(ncid, varname, varid) != NC_NOERR) {
00032
00033             /* Define variable... */
00034             NC(nc_def_var(ncid, varname, type, ndims, dimid, varid));
00035
00036             /* Set long name... */
00037             NC(nc_put_att_text
00038             (ncid, *varid, "long_name", strlen(longname), longname));
00039
00040             /* Set units... */
00041             NC(nc_put_att_text(ncid, *varid, "units", strlen(unit), unit));
00042             }
00043 }
```

## 5.27.1.3 void background\_poly ( wave\_t \* wave, int dim\_x, int dim\_y )

Get background based on polynomial fits.

Definition at line 104 of file [libairs.c](#).

```

00107         {
00108
00109         double x[WX], x2[WY], y[WX], y2[WY];
00110
00111         int ix, iy;
00112
00113         /* Copy temperatures to background... */
00114         for (ix = 0; ix < wave->nx; ix++)
00115             for (iy = 0; iy < wave->ny; iy++) {
00116                 wave->bg[ix][iy] = wave->temp[ix][iy];
00117                 wave->pt[ix][iy] = 0;
00118             }
00119
00120         /* Check parameters... */
00121         if (dim_x <= 0 && dim_y <= 0)
00122             return;
00123
00124         /* Compute fit in x-direction... */
00125         if (dim_x > 0)
00126             for (iy = 0; iy < wave->ny; iy++) {
00127                 for (ix = 0; ix < wave->nx; ix++) {
00128                     x[ix] = (double) ix;
00129                     y[ix] = wave->bg[ix][iy];
00130                 }
00131                 background_poly_help(x, y, wave->nx, dim_x);
00132                 for (ix = 0; ix < wave->nx; ix++)
00133                     wave->bg[ix][iy] = y[ix];
00134             }
00135
00136         /* Compute fit in y-direction... */
00137         if (dim_y > 0)
00138             for (ix = 0; ix < wave->nx; ix++) {
00139                 for (iy = 0; iy < wave->ny; iy++) {
00140                     x2[iy] = (int) iy;
00141                     y2[iy] = wave->bg[ix][iy];
00142                 }
00143                 background_poly_help(x2, y2, wave->ny, dim_y);
00144                 for (iy = 0; iy < wave->ny; iy++)
00145                     wave->bg[ix][iy] = y2[iy];
00146             }
00147
00148         /* Recompute perturbations... */
00149         for (ix = 0; ix < wave->nx; ix++)
00150             for (iy = 0; iy < wave->ny; iy++)
00151                 wave->pt[ix][iy] = wave->temp[ix][iy] - wave->bg[ix][iy];
00152     }

```

Here is the call graph for this function:



## 5.27.1.4 void background\_poly\_help ( double \* xx, double \* yy, int n, int dim )

Get background based on polynomial fits.

Definition at line 47 of file [libairs.c](#).

```

00051         {
00052
00053     gsl_multifit_linear_workspace *work;
00054     gsl_matrix *cov, *X;
00055     gsl_vector *c, *x, *y;
00056
00057     double chisq, xx2[WX > WY ? WX : WY], yy2[WX > WY ? WX : WY];
00058
00059     size_t i, i2, n2 = 0;
00060
00061     /* Check for nan... */
00062     for (i = 0; i < (size_t) n; i++)
00063         if (gsl_finite(yy[i])) {
00064             xx2[n2] = xx[i];
00065             yy2[n2] = yy[i];
00066             n2++;
00067         }
00068     if ((int) n2 < dim || n2 < 0.9 * n) {
00069         for (i = 0; i < (size_t) n; i++)
00070             yy[i] = GSL_NAN;
00071         return;
00072     }
00073
00074     /* Allocate... */
00075     work = gsl_multifit_linear_alloc((size_t) n2, (size_t) dim);
00076     cov = gsl_matrix_alloc((size_t) dim, (size_t) dim);
00077     X = gsl_matrix_alloc((size_t) n2, (size_t) dim);
00078     c = gsl_vector_alloc((size_t) dim);
00079     x = gsl_vector_alloc((size_t) n2);
00080     y = gsl_vector_alloc((size_t) n2);
00081
00082     /* Compute polynomial fit... */
00083     for (i = 0; i < (size_t) n2; i++) {
00084         gsl_vector_set(x, i, xx2[i]);
00085         gsl_vector_set(y, i, yy2[i]);
00086         for (i2 = 0; i2 < (size_t) dim; i2++)
00087             gsl_matrix_set(X, i, i2, pow(gsl_vector_get(x, i), (double) i2));
00088     }
00089     gsl_multifit_linear(X, y, c, cov, &chisq, work);
00090     for (i = 0; i < (size_t) n; i++)
00091         yy[i] = gsl_poly_eval(c->data, (int) dim, xx[i]);
00092
00093     /* Free... */
00094     gsl_multifit_linear_free(work);
00095     gsl_matrix_free(cov);
00096     gsl_matrix_free(X);
00097     gsl_vector_free(c);
00098     gsl_vector_free(x);
00099     gsl_vector_free(y);
00100 }

```

#### 5.27.1.5 void background\_smooth ( wave\_t \* wave, int npts\_x, int npts\_y )

Smooth background.

Definition at line 156 of file libairs.c.

```

00159         {
00160
00161     static double help[WX][WY], dmax = 2500.;
00162
00163     int dx, dy, i, j, ix, iy, n;
00164
00165     /* Check parameters... */
00166     if (npts_x <= 0 && npts_y <= 0)
00167         return;
00168
00169     /* Smooth background... */
00170     for (ix = 0; ix < wave->nx; ix++)
00171         for (iy = 0; iy < wave->ny; iy++) {
00172
00173         /* Init... */
00174         n = 0;
00175         help[ix][iy] = 0;
00176
00177         /* Set maximum range... */
00178         dx = GSL_MIN(GSL_MIN(npts_x, ix), wave->nx - 1 - ix);
00179         dy = GSL_MIN(GSL_MIN(npts_y, iy), wave->ny - 1 - iy);
00180
00181         /* Average... */

```

```

00182     for (i = ix - dx; i <= ix + dx; i++)
00183     for (j = iy - dy; j <= iy + dy; j++)
00184         if (fabs(wave->x[ix] - wave->x[i]) < dmax &&
00185             fabs(wave->y[iy] - wave->y[j]) < dmax) {
00186             help[ix][iy] += wave->bg[i][j];
00187             n++;
00188         }
00189
00190     /* Normalize... */
00191     if (n > 0)
00192         help[ix][iy] /= n;
00193     else
00194         help[ix][iy] = GSL_NAN;
00195 }
00196
00197 /* Recalculate perturbations... */
00198 for (ix = 0; ix < wave->nx; ix++)
00199     for (iy = 0; iy < wave->ny; iy++) {
00200         wave->bg[ix][iy] = help[ix][iy];
00201         wave->pt[ix][iy] = wave->temp[ix][iy] - wave->bg[ix][iy];
00202     }
00203 }

```

#### 5.27.1.6 void create\_background ( wave\_t \* wave )

Set background...

Definition at line 207 of file [libairs.c](#).

```

00208     {
00209
00210     int ix, iy;
00211
00212     /* Loop over grid points... */
00213     for (ix = 0; ix < wave->nx; ix++)
00214         for (iy = 0; iy < wave->ny; iy++) {
00215
00216             /* Set background for 4.3 micron BT measurements... */
00217             wave->bg[ix][iy] = 235.626 + 5.38165e-6 * gsl_pow_2(wave->x[ix]
00218                                     -
00219                                     0.5 * (wave->x[0] +
00220                                             wave->x
00221                                             [wave->nx -
00222                                             1]))
00223             - 1.78519e-12 * gsl_pow_4(wave->x[ix] -
00224                                     0.5 * (wave->x[0] + wave->x[wave->nx - 1]));
00225
00226             /* Set temperature perturbation... */
00227             wave->pt[ix][iy] = 0;
00228
00229             /* Set temperature... */
00230             wave->temp[ix][iy] = wave->bg[ix][iy];
00231         }
00232 }

```

#### 5.27.1.7 void create\_noise ( wave\_t \* wave, double nedt )

Add noise to perturbations and temperatures...

Definition at line 236 of file [libairs.c](#).

```

00238     {
00239
00240     gsl_rng *r;
00241
00242     int ix, iy;
00243
00244     /* Initialize random number generator... */
00245     gsl_rng_env_setup();
00246     r = gsl_rng_alloc(gsl_rng_default);
00247     gsl_rng_set(r, (unsigned long int) time(NULL));
00248
00249     /* Add noise to temperature... */
00250     if (nedt > 0)
00251         for (ix = 0; ix < wave->nx; ix++)
00252             for (iy = 0; iy < wave->ny; iy++)
00253                 wave->temp[ix][iy] += gsl_ran_gaussian(r, nedt);
00254
00255     /* Free... */
00256     gsl_rng_free(r);
00257 }

```



### 5.27.1.8 void create\_wave ( wave\_t \* wave, double amp, double lx, double ly, double phi, double fwhm )

Add linear wave pattern...

Definition at line 261 of file [libairs.c](#).

```

00267         {
00268
00269     int ix, iy;
00270
00271     /* Loop over grid points... */
00272     for (ix = 0; ix < wave->nx; ix++)
00273         for (iy = 0; iy < wave->ny; iy++) {
00274
00275         /* Set wave perturbation... */
00276         wave->pt[ix][iy] = amp * cos((lx != 0 ? 2 * M_PI / lx : 0) * wave->x[ix]
00277                                     + (ly !=
00278                                     0 ? 2 * M_PI / ly : 0) * wave->y[iy]
00279                                     - phi * M_PI / 180.)
00280         * (fwhm > 0 ? exp(-0.5 * gsl_pow_2((wave->x[ix]) / (lx * fwhm) * 2.35)
00281                     -
00282                     0.5 * gsl_pow_2((wave->y[iy]) / (ly * fwhm) *
00283                     2.35)) : 1.0);
00284
00285         /* Add perturbation to temperature... */
00286         wave->temp[ix][iy] += wave->pt[ix][iy];
00287     }
00288 }
```

### 5.27.1.9 void day2doy ( int year, int mon, int day, int \* doy )

Get day of year from date.

Definition at line 292 of file [libairs.c](#).

```

00296         {
00297
00298     int d0[12] = { 1, 32, 60, 91, 121, 152, 182, 213, 244, 274, 305, 335 };
00299     int d0l[12] = { 1, 32, 61, 92, 122, 153, 183, 214, 245, 275, 306, 336 };
00300
00301     /* Get day of year... */
00302     if (year % 400 == 0 || (year % 100 != 0 && year % 4 == 0))
00303         *doy = d0l[mon - 1] + day - 1;
00304     else
00305         *doy = d0[mon - 1] + day - 1;
00306 }
```

### 5.27.1.10 void doy2day ( int year, int doy, int \* mon, int \* day )

Get date from day of year.

Definition at line 310 of file [libairs.c](#).

```

00314         {
00315
00316     int d0[12] = { 1, 32, 60, 91, 121, 152, 182, 213, 244, 274, 305, 335 };
00317     int d0l[12] = { 1, 32, 61, 92, 122, 153, 183, 214, 245, 275, 306, 336 };
00318     int i;
00319
00320     /* Get month and day... */
00321     if (year % 400 == 0 || (year % 100 != 0 && year % 4 == 0)) {
00322         for (i = 11; i >= 0; i--)
00323             if (d0l[i] <= doy)
00324                 break;
00325         *mon = i + 1;
00326         *day = doy - d0l[i] + 1;
00327     } else {
00328         for (i = 11; i >= 0; i--)
00329             if (d0[i] <= doy)
00330                 break;
00331         *mon = i + 1;
00332         *day = doy - d0[i] + 1;
00333     }
00334 }
```

5.27.1.11 void `fft_help` ( double \* *fcReal*, double \* *fcImag*, int *n* )

Calculate 1-D FFT...

Definition at line 338 of file [libairs.c](#).

```

00341         {
00342
00343     gsl_fft_complex_wavetable *wavetable;
00344     gsl_fft_complex_workspace *workspace;
00345
00346     double data[2 * PMAX];
00347
00348     int i;
00349
00350     /* Check size... */
00351     if (n > PMAX)
00352         ERRMSG("Too many data points!");
00353
00354     /* Allocate... */
00355     wavetable = gsl_fft_complex_wavetable_alloc((size_t) n);
00356     workspace = gsl_fft_complex_workspace_alloc((size_t) n);
00357
00358     /* Set data (real, complex)... */
00359     for (i = 0; i < n; i++) {
00360         data[2 * i] = fcReal[i];
00361         data[2 * i + 1] = fcImag[i];
00362     }
00363
00364     /* Calculate FFT... */
00365     gsl_fft_complex_forward(data, 1, (size_t) n, wavetable, workspace);
00366
00367     /* Copy data... */
00368     for (i = 0; i < n; i++) {
00369         fcReal[i] = data[2 * i];
00370         fcImag[i] = data[2 * i + 1];
00371     }
00372
00373     /* Free... */
00374     gsl_fft_complex_wavetable_free(wavetable);
00375     gsl_fft_complex_workspace_free(workspace);
00376 }

```

5.27.1.12 void `fft` ( wave\_t \* *wave*, double \* *Amax*, double \* *phimax*, double \* *lhmax*, double \* *alphamax*, double \* *betamax*, char \* *filename* )

Calculate 2-D FFT...

Definition at line 380 of file [libairs.c](#).

```

00387         {
00388
00389     static double A[PMAX][PMAX], phi[PMAX][PMAX], kx[PMAX], ky[PMAX],
00390         kxmax, kymax, cutReal[PMAX], cutImag[PMAX],
00391         boxImag[PMAX][PMAX], boxReal[PMAX][PMAX];
00392
00393     FILE *out;
00394
00395     int i, i2, imin, imax, j, j2, jmin, jmax, nx, ny;
00396
00397     /* Find box... */
00398     imin = jmin = 9999;
00399     imax = jmax = -9999;
00400     for (i = 0; i < wave->nx; i++)
00401         for (j = 0; j < wave->ny; j++)
00402             if (gsl_finite(wave->var[i][j])) {
00403                 imin = GSL_MIN(imin, i);
00404                 imax = GSL_MAX(imax, i);
00405                 jmin = GSL_MIN(jmin, j);
00406                 jmax = GSL_MAX(jmax, j);
00407             }
00408     nx = imax - imin + 1;
00409     ny = jmax - jmin + 1;
00410
00411     /* Copy data... */
00412     for (i = imin; i <= imax; i++)

```

```

00413     for (j = jmin; j <= jmax; j++) {
00414         if (gsl_finite(wave->pt[i][j]))
00415             boxReal[i - imin][j - jmin] = wave->pt[i][j];
00416         else
00417             boxReal[i - imin][j - jmin] = 0.0;
00418         boxImag[i - imin][j - jmin] = 0.0;
00419     }
00420
00421     /* FFT of the rows... */
00422     for (i = 0; i < nx; i++) {
00423         for (j = 0; j < ny; j++) {
00424             cutReal[j] = boxReal[i][j];
00425             cutImag[j] = boxImag[i][j];
00426         }
00427         fft_help(cutReal, cutImag, ny);
00428         for (j = 0; j < ny; j++) {
00429             boxReal[i][j] = cutReal[j];
00430             boxImag[i][j] = cutImag[j];
00431         }
00432     }
00433
00434     /* FFT of the columns... */
00435     for (j = 0; j < ny; j++) {
00436         for (i = 0; i < nx; i++) {
00437             cutReal[i] = boxReal[i][j];
00438             cutImag[i] = boxImag[i][j];
00439         }
00440         fft_help(cutReal, cutImag, nx);
00441         for (i = 0; i < nx; i++) {
00442             boxReal[i][j] = cutReal[i];
00443             boxImag[i][j] = cutImag[i];
00444         }
00445     }
00446
00447     /* Get frequencies, amplitude, and phase... */
00448     for (i = 0; i < nx; i++)
00449         kx[i] = 2. * M_PI * ((i < nx / 2) ? (double) i : -(double) (nx - i))
00450         / (nx * fabs(wave->x[imax] - wave->x[imin]) / (nx - 1.0));
00451     for (j = 0; j < ny; j++)
00452         ky[j] = 2. * M_PI * ((j < ny / 2) ? (double) j : -(double) (ny - j))
00453         / (ny * fabs(wave->y[jmax] - wave->y[jmin]) / (ny - 1.0));
00454     for (i = 0; i < nx; i++)
00455         for (j = 0; j < ny; j++) {
00456             A[i][j]
00457             = (i == 0 && j == 0 ? 1.0 : 2.0) / (nx * ny)
00458             * sqrt(gsl_pow_2(boxReal[i][j]) + gsl_pow_2(boxImag[i][j]));
00459             phi[i][j]
00460             = 180. / M_PI * atan2(boxImag[i][j], boxReal[i][j]);
00461         }
00462
00463     /* Check frequencies... */
00464     for (i = 0; i < nx; i++)
00465         for (j = 0; j < ny; j++)
00466             if (kx[i] == 0 || ky[j] == 0) {
00467                 A[i][j] = GSL_NAN;
00468                 phi[i][j] = GSL_NAN;
00469             }
00470
00471     /* Find maximum... */
00472     *Amax = 0;
00473     for (i = 0; i < nx; i++)
00474         for (j = 0; j < ny / 2; j++)
00475             if (gsl_finite(A[i][j]) && A[i][j] > *Amax) {
00476                 *Amax = A[i][j];
00477                 *phimax = phi[i][j];
00478                 kxmax = kx[i];
00479                 kymax = ky[j];
00480                 imax = i;
00481                 jmax = j;
00482             }
00483
00484     /* Get horizontal wavelength... */
00485     *lhmax = 2 * M_PI / sqrt(gsl_pow_2(kxmax) + gsl_pow_2(kymax));
00486
00487     /* Get propagation direction in xy-plane... */
00488     *alphamax = 90. - 180. / M_PI * atan2(kxmax, kymax);
00489
00490     /* Get propagation direction in lon,lat-plane... */
00491     *betamax = *alphamax
00492     +
00493     180. / M_PI *
00494     atan2(wave->lat[wave->nz / 2 >
00495           0 ? wave->nz / 2 - 1 : wave->nz / 2][wave->ny / 2]
00496         - wave->lat[wave->nz / 2 <
00497           wave->nz - 1 ? wave->nz / 2 +
00498           1 : wave->nz / 2][wave->ny / 2],
00499         wave->lon[wave->nz / 2 >

```

```

00500          0 ? wave->nx / 2 - 1 : wave->nx / 2][wave->ny / 2]
00501      - wave->lon[wave->nx / 2 <
00502          wave->nx - 1 ? wave->nx / 2 +
00503          1 : wave->nx / 2][wave->ny / 2]);
00504
00505  /* Save FFT data... */
00506  if (filename != NULL) {
00507
00508      /* Write info... */
00509      printf("Write FFT data: %s\n", filename);
00510
00511      /* Create file... */
00512      if (!(out = fopen(filename, "w")))
00513          ERRMSG("Cannot create file!");
00514
00515      /* Write header... */
00516      fprintf(out,
00517          "# $1 = altitude [km]\n"
00518          "# $2 = wavelength in x-direction [km]\n"
00519          "# $3 = wavelength in y-direction [km]\n"
00520          "# $4 = wavenumber in x-direction [1/km]\n"
00521          "# $5 = wavenumber in y-direction [1/km]\n"
00522          "# $6 = amplitude [K]\n" "# $7 = phase [rad]\n");
00523
00524      /* Write data... */
00525      for (i = nx - 1; i > 0; i--) {
00526          fprintf(out, "\n");
00527          for (j = ny / 2; j > 0; j--) {
00528              i2 = (i == nx / 2 ? 0 : i);
00529              j2 = (j == ny / 2 ? 0 : j);
00530              fprintf(out, "%g %g %g %g %g %g\n", wave->z,
00531                  (kx[i2] != 0 ? 2 * M_PI / kx[i2] : 0),
00532                  (ky[j2] != 0 ? 2 * M_PI / ky[j2] : 0),
00533                  kx[i2], ky[j2], A[i2][j2], phi[i2][j2]);
00534          }
00535      }
00536
00537      /* Close file... */
00538      fclose(out);
00539  }
00540 }

```

Here is the call graph for this function:



#### 5.27.1.13 void gauss ( wave\_t \* wave, double fwhm )

Apply Gaussian filter to perturbations...

Definition at line 544 of file [libairs.c](#).

```

00546      {
00547
00548      static double d2, help[WX][WY], sigma2, w, wsum;
00549
00550      int ix, ix2, iy, iy2;
00551
00552      /* Check parameters... */
00553      if (fwhm <= 0)
00554          return;
00555
00556      /* Compute sigma^2... */
00557      sigma2 = gsl_pow_2(fwhm / 2.3548);

```

```

00558
00559  /* Loop over data points... */
00560  for (ix = 0; ix < wave->nx; ix++)
00561      for (iy = 0; iy < wave->ny; iy++) {
00562
00563          /* Init... */
00564          wsum = 0;
00565          help[ix][iy] = 0;
00566
00567          /* Average... */
00568          for (ix2 = 0; ix2 < wave->nx; ix2++)
00569              for (iy2 = 0; iy2 < wave->ny; iy2++) {
00570                  d2 = gsl_pow_2(wave->x[ix] - wave->x[ix2])
00571                      + gsl_pow_2(wave->y[iy] - wave->y[iy2]);
00572                  if (d2 <= 9 * sigma2) {
00573                      w = exp(-d2 / (2 * sigma2));
00574                      wsum += w;
00575                      help[ix][iy] += w * wave->pt[ix2][iy2];
00576                  }
00577              }
00578
00579          /* Normalize... */
00580          wave->pt[ix][iy] = help[ix][iy] / wsum;
00581      }
00582 }

```

#### 5.27.1.14 void hamming ( wave\_t \* wave, int nit )

Apply Hamming filter to perturbations...

Definition at line 586 of file [libairs.c](#).

```

00588      {
00589
00590          static double help[WX][WY];
00591
00592          int iter, ix, iy;
00593
00594          /* Iterations... */
00595          for (iter = 0; iter < niter; iter++) {
00596
00597              /* Filter in x direction... */
00598              for (ix = 0; ix < wave->nx; ix++)
00599                  for (iy = 0; iy < wave->ny; iy++)
00600                      help[ix][iy]
00601                          = 0.23 * wave->pt[ix > 0 ? ix - 1 : ix][iy]
00602                              + 0.54 * wave->pt[ix][iy]
00603                                  + 0.23 * wave->pt[ix < wave->nx - 1 ? ix + 1 : ix][iy];
00604
00605              /* Filter in y direction... */
00606              for (ix = 0; ix < wave->nx; ix++)
00607                  for (iy = 0; iy < wave->ny; iy++)
00608                      wave->pt[ix][iy]
00609                          = 0.23 * help[ix][iy > 0 ? iy - 1 : iy]
00610                              + 0.54 * help[ix][iy]
00611                                  + 0.23 * help[ix][iy < wave->ny - 1 ? iy + 1 : iy];
00612          }
00613 }

```

#### 5.27.1.15 void intpol\_x ( wave\_t \* wave, int n )

Interpolate to regular grid in x-direction.

Definition at line 617 of file [libairs.c](#).

```

00619      {
00620
00621          gsl_interp_accel *acc;
00622          gsl_spline *spline;
00623
00624          double dummy, x[WX], xc[WX][3], xc2[WX][3], y[WX];
00625
00626          int i, ic, ix, iy;
00627

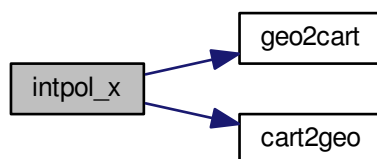
```

```

00628  /* Check parameters... */
00629  if (n <= 0)
00630      return;
00631  if (n > WX)
00632      ERRMSG("Too many data points!");
00633
00634  /* Set new x-coordinates... */
00635  for (i = 0; i < n; i++)
00636      x[i] = LIN(0.0, wave->x[0], n - 1.0, wave->x[wave->nx - 1], i);
00637
00638  /* Allocate... */
00639  acc = gsl_interp_accel_alloc();
00640  spline = gsl_spline_alloc(gsl_interp_cspline, (size_t) wave->nx);
00641
00642  /* Loop over scans... */
00643  for (iy = 0; iy < wave->ny; iy++) {
00644
00645      /* Interpolate Cartesian coordinates... */
00646      for (ix = 0; ix < wave->nx; ix++)
00647          geo2cart(0, wave->lon[ix][iy], wave->lat[ix][iy], xc[ix]);
00648      for (ic = 0; ic < 3; ic++) {
00649          for (ix = 0; ix < wave->nx; ix++)
00650              y[ix] = xc[ix][ic];
00651          gsl_spline_init(spline, wave->x, y, (size_t) wave->nx);
00652          for (i = 0; i < n; i++)
00653              xc2[i][ic] = gsl_spline_eval(spline, x[i], acc);
00654      }
00655      for (i = 0; i < n; i++)
00656          cart2geo(xc2[i], &dummy, &wave->lon[i][iy], &wave->lat[i][iy]);
00657
00658      /* Interpolate temperature... */
00659      for (ix = 0; ix < wave->nx; ix++)
00660          y[ix] = wave->temp[ix][iy];
00661      gsl_spline_init(spline, wave->x, y, (size_t) wave->nx);
00662      for (i = 0; i < n; i++)
00663          wave->temp[i][iy] = gsl_spline_eval(spline, x[i], acc);
00664
00665      /* Interpolate background... */
00666      for (ix = 0; ix < wave->nx; ix++)
00667          y[ix] = wave->bg[ix][iy];
00668      gsl_spline_init(spline, wave->x, y, (size_t) wave->nx);
00669      for (i = 0; i < n; i++)
00670          wave->bg[i][iy] = gsl_spline_eval(spline, x[i], acc);
00671
00672      /* Interpolate perturbations... */
00673      for (ix = 0; ix < wave->nx; ix++)
00674          y[ix] = wave->pt[ix][iy];
00675      gsl_spline_init(spline, wave->x, y, (size_t) wave->nx);
00676      for (i = 0; i < n; i++)
00677          wave->pt[i][iy] = gsl_spline_eval(spline, x[i], acc);
00678
00679      /* Interpolate variance... */
00680      for (ix = 0; ix < wave->nx; ix++)
00681          y[ix] = wave->var[ix][iy];
00682      gsl_spline_init(spline, wave->x, y, (size_t) wave->nx);
00683      for (i = 0; i < n; i++)
00684          wave->var[i][iy] = gsl_spline_eval(spline, x[i], acc);
00685  }
00686
00687  /* Free... */
00688  gsl_spline_free(spline);
00689  gsl_interp_accel_free(acc);
00690
00691  /* Set new x-coordinates... */
00692  for (i = 0; i < n; i++)
00693      wave->x[i] = x[i];
00694  wave->nx = n;
00695 }

```

Here is the call graph for this function:



#### 5.27.1.16 void median ( wave\_t \* wave, int dx )

Apply median filter to perturbations...

Definition at line 699 of file [libairs.c](#).

```

00701     {
00702
00703     static double data[WX * WY], help[WX][WY];
00704
00705     int ix, ix2, iy, iy2;
00706
00707     size_t n;
00708
00709     /* Check parameters... */
00710     if (dx <= 0)
00711         return;
00712
00713     /* Loop over data points... */
00714     for (ix = 0; ix < wave->nx; ix++)
00715         for (iy = 0; iy < wave->ny; iy++) {
00716
00717         /* Init... */
00718         n = 0;
00719
00720         /* Get data... */
00721         for (ix2 = GSL_MAX(ix - dx, 0); ix2 < GSL_MIN(ix + dx, wave->nx - 1);
00722              ix2++)
00723             for (iy2 = GSL_MAX(iy - dx, 0); iy2 < GSL_MIN(iy + dx, wave->ny - 1);
00724                  iy2++) {
00725                 data[n] = wave->pt[ix2][iy2];
00726                 n++;
00727             }
00728
00729         /* Normalize... */
00730         gsl_sort(data, 1, n);
00731         help[ix][iy] = gsl_stats_median_from_sorted_data(data, 1, n);
00732     }
00733
00734     /* Loop over data points... */
00735     for (ix = 0; ix < wave->nx; ix++)
00736         for (iy = 0; iy < wave->ny; iy++)
00737             wave->pt[ix][iy] = help[ix][iy];
00738 }
  
```

#### 5.27.1.17 void merge\_y ( wave\_t \* wave1, wave\_t \* wave2 )

Merge wave structs in y-direction.

Definition at line 742 of file [libairs.c](#).

```

00744         {
00745
00746     double y;
00747
00748     int ix, iy;
00749
00750     /* Check data... */
00751     if (wave1->nx != wave2->nx)
00752         ERRMSG("Across-track sizes do not match!");
00753     if (wave1->ny + wave2->ny > WY)
00754         ERRMSG("Too many data points!");
00755
00756     /* Get offset in y direction... */
00757     y =
00758         wave1->y[wave1->ny - 1] + (wave1->y[wave1->ny - 1] -
00759                                 wave1->y[0]) / (wave1->ny - 1);
00760
00761     /* Merge data... */
00762     for (ix = 0; ix < wave2->nx; ix++)
00763         for (iy = 0; iy < wave2->ny; iy++) {
00764             wave1->y[wave1->ny + iy] = y + wave2->y[iy];
00765             wave1->lon[ix][wave1->ny + iy] = wave2->lon[ix][iy];
00766             wave1->lat[ix][wave1->ny + iy] = wave2->lat[ix][iy];
00767             wave1->temp[ix][wave1->ny + iy] = wave2->temp[ix][iy];
00768             wave1->bg[ix][wave1->ny + iy] = wave2->bg[ix][iy];
00769             wave1->pt[ix][wave1->ny + iy] = wave2->pt[ix][iy];
00770             wave1->var[ix][wave1->ny + iy] = wave2->var[ix][iy];
00771         }
00772
00773     /* Increment counter... */
00774     wave1->ny += wave2->ny;
00775 }

```

#### 5.27.1.18 void noise ( wave\_t\* wave, double \* mu, double \* sig )

Estimate noise.

Definition at line 779 of file [libairs.c](#).

```

00782         {
00783
00784     int ix, ix2, iy, iy2, n = 0, okay;
00785
00786     /* Init... */
00787     *mu = 0;
00788     *sig = 0;
00789
00790     /* Estimate noise (Immerkaer, 1996)... */
00791     for (ix = 1; ix < wave->nx - 1; ix++)
00792         for (iy = 1; iy < wave->ny - 1; iy++) {
00793
00794             /* Check data... */
00795             okay = 1;
00796             for (ix2 = ix - 1; ix2 <= ix + 1; ix2++)
00797                 for (iy2 = iy - 1; iy2 <= iy + 1; iy2++)
00798                     if (!gsl_finite(wave->temp[ix2][iy2]))
00799                         okay = 0;
00800             if (!okay)
00801                 continue;
00802
00803             /* Get mean noise... */
00804             n++;
00805             *mu += wave->temp[ix][iy];
00806             *sig += gsl_pow_2(+4. / 6. * wave->temp[ix][iy]
00807                             - 2. / 6. * (wave->temp[ix - 1][iy]
00808                                         + wave->temp[ix + 1][iy]
00809                                         + wave->temp[ix][iy - 1]
00810                                         + wave->temp[ix][iy + 1])
00811                             + 1. / 6. * (wave->temp[ix - 1][iy - 1]
00812                                         + wave->temp[ix + 1][iy - 1]
00813                                         + wave->temp[ix - 1][iy + 1]
00814                                         + wave->temp[ix + 1][iy + 1]));
00815         }
00816
00817     /* Normalize... */
00818     *mu /= (double) n;
00819     *sig = sqrt(*sig / (double) n);
00820 }

```



**5.27.1.19 void period ( wave\_t \* wave, double \* Amax, double \* phimax, double \* lhmax, double \* alphamax, double \* betamax, char \* filename )**

Compute periodogram.

Definition at line 824 of file [libairs.c](#).

```

00831         {
00832
00833     FILE *out;
00834
00835     static double kx[PMAX], ky[PMAX], kx_ny, ky_ny, kxmax, kymax, A[PMAX][PMAX],
00836         phi[PMAX][PMAX], cx[PMAX][WX], cy[PMAX][WY], sx[PMAX][WX], sy[PMAX][WY],
00837         a, b, c, lx, ly, lymax = 1000, dlyx = 10;
00838
00839     int i, imin, imax, j, jmin, jmax, l, lmax = 0, m, mmax = 0;
00840
00841     /* Compute wavenumbers and periodogram coefficients... */
00842     for (lx = -lymax; lx <= lymax; lx += dlyx) {
00843         kx[lmax] = (lx != 0 ? 2 * M_PI / lx : 0);
00844         for (i = 0; i < wave->nx; i++) {
00845             cx[lmax][i] = cos(kx[lmax] * wave->x[i]);
00846             sx[lmax][i] = sin(kx[lmax] * wave->x[i]);
00847         }
00848         if ((++lmax) > PMAX)
00849             ERRMSG("Too many wavenumbers for periodogram!");
00850     }
00851     for (ly = 0; ly <= lymax; ly += dlyx) {
00852         ky[mmax] = (ly != 0 ? 2 * M_PI / ly : 0);
00853         for (j = 0; j < wave->ny; j++) {
00854             cy[mmax][j] = cos(ky[mmax] * wave->y[j]);
00855             sy[mmax][j] = sin(ky[mmax] * wave->y[j]);
00856         }
00857         if ((++mmax) > PMAX)
00858             ERRMSG("Too many wavenumbers for periodogram!");
00859     }
00860
00861     /* Find area... */
00862     imin = jmin = 9999;
00863     imax = jmax = -9999;
00864     for (i = 0; i < wave->nx; i++)
00865         for (j = 0; j < wave->ny; j++)
00866             if (gsl_finite(wave->var[i][j])) {
00867                 imin = GSL_MIN(imin, i);
00868                 imax = GSL_MAX(imax, i);
00869                 jmin = GSL_MIN(jmin, j);
00870                 jmax = GSL_MAX(jmax, j);
00871             }
00872
00873     /* Get Nyquist frequencies... */
00874     kx_ny =
00875         M_PI / fabs((wave->x[imax] - wave->x[jmin]) /
00876             ((double) imax - (double) imin));
00877     ky_ny =
00878         M_PI / fabs((wave->y[jmax] - wave->y[jmin]) /
00879             ((double) jmax - (double) jmin));
00880
00881     /* Loop over wavelengths... */
00882     for (l = 0; l < lmax; l++)
00883         for (m = 0; m < mmax; m++) {
00884
00885             /* Check frequencies... */
00886             if (kx[l] == 0 || fabs(kx[l]) > kx_ny ||
00887                 ky[m] == 0 || fabs(ky[m]) > ky_ny) {
00888                 A[l][m] = GSL_NAN;
00889                 phi[l][m] = GSL_NAN;
00890                 continue;
00891             }
00892
00893             /* Compute periodogram... */
00894             a = b = c = 0;
00895             for (i = imin; i <= imax; i++)
00896                 for (j = jmin; j <= jmax; j++)
00897                     if (gsl_finite(wave->var[i][j])) {
00898                         a += wave->pt[i][j] * (cx[l][i] * cy[m][j] - sx[l][i] * sy[m][j]);
00899                         b += wave->pt[i][j] * (sx[l][i] * cy[m][j] + cx[l][i] * sy[m][j]);
00900                         c++;
00901                     }
00902             a *= 2. / c;
00903             b *= 2. / c;
00904
00905             /* Get amplitude and phase... */
00906             A[l][m] = sqrt(gsl_pow_2(a) + gsl_pow_2(b));

```

```

00907     phi[l][m] = atan2(b, a) * 180. / M_PI;
00908 }
00909
00910 /* Find maximum... */
00911 *Amax = 0;
00912 for (l = 0; l < lmax; l++)
00913     for (m = 0; m < mmax; m++)
00914         if (gsl_finite(A[l][m]) && A[l][m] > *Amax) {
00915             *Amax = A[l][m];
00916             *phimax = phi[l][m];
00917             kxmax = kx[l];
00918             kymax = ky[m];
00919             imax = i;
00920             jmax = j;
00921         }
00922
00923 /* Get horizontal wavelength... */
00924 *lhmax = 2 * M_PI / sqrt(gsl_pow_2(kxmax) + gsl_pow_2(kymax));
00925
00926 /* Get propagation direction in xy-plane... */
00927 *alphamax = 90. - 180. / M_PI * atan2(kxmax, kymax);
00928
00929 /* Get propagation direction in lon,lat-plane... */
00930 *betamax = *alphamax
00931 +
00932     180. / M_PI *
00933     atan2(wave->lat[wave->nx / 2 >
00934             0 ? wave->nx / 2 - 1 : wave->nx / 2][wave->ny / 2]
00935           - wave->lat[wave->nx / 2 <
00936             wave->nx - 1 ? wave->nx / 2 +
00937               1 : wave->nx / 2][wave->ny / 2],
00938           wave->lon[wave->nx / 2 >
00939             0 ? wave->nx / 2 - 1 : wave->nx / 2][wave->ny / 2]
00940           - wave->lon[wave->nx / 2 <
00941             wave->nx - 1 ? wave->nx / 2 +
00942               1 : wave->nx / 2][wave->ny / 2]);
00943
00944 /* Save periodogram data... */
00945 if (filename != NULL) {
00946
00947     /* Write info... */
00948     printf("Write periodogram data: %s\n", filename);
00949
00950     /* Create file... */
00951     if (!(out = fopen(filename, "w")))
00952         ERRMSG("Cannot create file!");
00953
00954     /* Write header... */
00955     fprintf(out,
00956             "# $1 = altitude [km]\n"
00957             "# $2 = wavelength in x-direction [km]\n"
00958             "# $3 = wavelength in y-direction [km]\n"
00959             "# $4 = wavenumber in x-direction [1/km]\n"
00960             "# $5 = wavenumber in y-direction [1/km]\n"
00961             "# $6 = amplitude [K]\n" "# $7 = phase [rad]\n");
00962
00963     /* Write data... */
00964     for (l = 0; l < lmax; l++) {
00965         fprintf(out, "\n");
00966         for (m = 0; m < mmax; m++)
00967             fprintf(out, "%g %g %g %g %g %g %g\n", wave->z,
00968                 (kx[l] != 0 ? 2 * M_PI / kx[l] : 0),
00969                 (ky[m] != 0 ? 2 * M_PI / ky[m] : 0),
00970                 kx[l], ky[m], A[l][m], phi[l][m]);
00971     }
00972
00973     /* Close file... */
00974     fclose(out);
00975 }
00976 }

```

#### 5.27.1.20 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 980 of file [libairs.c](#).

```

00986     {
00987
00988     double x0[3], x1[3];

```

```

00989
00990     int itrack, ixtrack;
00991
00992     /* Check ranges... */
00993     track0 = GSL_MIN(GSL_MAX(track0, 0), pert->ntrack - 1);
00994     track1 = GSL_MIN(GSL_MAX(track1, 0), pert->ntrack - 1);
00995     xtrack0 = GSL_MIN(GSL_MAX(xtrack0, 0), pert->nxtrack - 1);
00996     xtrack1 = GSL_MIN(GSL_MAX(xtrack1, 0), pert->nxtrack - 1);
00997
00998     /* Set size... */
00999     wave->nx = xtrack1 - xtrack0 + 1;
01000     if (wave->nx > WX)
01001         ERRMSG("Too many across-track values!");
01002     wave->ny = track1 - track0 + 1;
01003     if (wave->ny > WY)
01004         ERRMSG("Too many along-track values!");
01005
01006     /* Loop over footprints... */
01007     for (itrack = track0; itrack <= track1; itrack++)
01008         for (ixtrack = xtrack0; ixtrack <= xtrack1; ixtrack++) {
01009
01010         /* Get distances... */
01011         if (itrack == track0) {
01012             wave->x[0] = 0;
01013             if (ixtrack > xtrack0) {
01014                 geo2cart(0, pert->lon[itrack][ixtrack - 1],
01015                         pert->lat[itrack][ixtrack - 1], x0);
01016                 geo2cart(0, pert->lon[itrack][ixtrack],
01017                         pert->lat[itrack][ixtrack], x1);
01018                 wave->x[ixtrack - xtrack0] =
01019                     wave->x[ixtrack - xtrack0 - 1] + DIST(x0, x1);
01020             }
01021         }
01022         if (ixtrack == xtrack0) {
01023             wave->y[0] = 0;
01024             if (itrack > track0) {
01025                 geo2cart(0, pert->lon[itrack - 1][ixtrack],
01026                         pert->lat[itrack - 1][ixtrack], x0);
01027                 geo2cart(0, pert->lon[itrack][ixtrack],
01028                         pert->lat[itrack][ixtrack], x1);
01029                 wave->y[itrack - track0] =
01030                     wave->y[itrack - track0 - 1] + DIST(x0, x1);
01031             }
01032         }
01033
01034         /* Save geolocation... */
01035         wave->time = pert->time[(track0 + track1) / 2][(xtrack0 + xtrack1) / 2];
01036         wave->z = 0;
01037         wave->lon[ixtrack - xtrack0][itrack - track0] =
01038             pert->lon[itrack][ixtrack];
01039         wave->lat[ixtrack - xtrack0][itrack - track0] =
01040             pert->lat[itrack][ixtrack];
01041
01042         /* Save temperature data... */
01043         wave->temp[ixtrack - xtrack0][itrack - track0]
01044             = pert->bt[itrack][ixtrack];
01045         wave->bg[ixtrack - xtrack0][itrack - track0]
01046             = pert->bt[itrack][ixtrack] - pert->pt[itrack][ixtrack];
01047         wave->pt[ixtrack - xtrack0][itrack - track0]
01048             = pert->pt[itrack][ixtrack];
01049         wave->var[ixtrack - xtrack0][itrack - track0]
01050             = pert->var[itrack][ixtrack];
01051     }
01052 }

```

Here is the call graph for this function:



### 5.27.1.21 void read\_l1 ( char \* filename, airs\_l1\_t \* l1 )

Read AIRS Level-1 data.

Definition at line 1056 of file [libairs.c](#).

```
01058         {
01059
01060     int ncid, varid;
01061
01062     /* Open netCDF file... */
01063     printf("Read AIRS Level-1 file: %s\n", filename);
01064     NC(nc_open(filename, NC_NOWRITE, &ncid));
01065
01066     /* Read data... */
01067     NC(nc_inq_varid(ncid, "l1_time", &varid));
01068     NC(nc_get_var_double(ncid, varid, l1->time[0]));
01069     NC(nc_inq_varid(ncid, "l1_lon", &varid));
01070     NC(nc_get_var_double(ncid, varid, l1->lon[0]));
01071     NC(nc_inq_varid(ncid, "l1_lat", &varid));
01072     NC(nc_get_var_double(ncid, varid, l1->lat[0]));
01073     NC(nc_inq_varid(ncid, "l1_sat_z", &varid));
01074     NC(nc_get_var_double(ncid, varid, l1->sat_z));
01075     NC(nc_inq_varid(ncid, "l1_sat_lon", &varid));
01076     NC(nc_get_var_double(ncid, varid, l1->sat_lon));
01077     NC(nc_inq_varid(ncid, "l1_sat_lat", &varid));
01078     NC(nc_get_var_double(ncid, varid, l1->sat_lat));
01079     NC(nc_inq_varid(ncid, "l1_nu", &varid));
01080     NC(nc_get_var_double(ncid, varid, l1->nu));
01081     NC(nc_inq_varid(ncid, "l1_rad", &varid));
01082     NC(nc_get_var_float(ncid, varid, l1->rad[0][0]));
01083
01084     /* Close file... */
01085     NC(nc_close(ncid));
01086 }
```

### 5.27.1.22 void read\_l2 ( char \* filename, airs\_l2\_t \* l2 )

Read AIRS Level-2 data.

Definition at line 1090 of file [libairs.c](#).

```
01092         {
01093
01094     int ncid, varid;
01095
01096     /* Open netCDF file... */
01097     printf("Read AIRS Level-2 file: %s\n", filename);
01098     NC(nc_open(filename, NC_NOWRITE, &ncid));
01099
01100     /* Read data... */
01101     NC(nc_inq_varid(ncid, "l2_time", &varid));
01102     NC(nc_get_var_double(ncid, varid, l2->time[0]));
01103     NC(nc_inq_varid(ncid, "l2_z", &varid));
01104     NC(nc_get_var_double(ncid, varid, l2->z[0][0]));
01105     NC(nc_inq_varid(ncid, "l2_lon", &varid));
01106     NC(nc_get_var_double(ncid, varid, l2->lon[0]));
01107     NC(nc_inq_varid(ncid, "l2_lat", &varid));
01108     NC(nc_get_var_double(ncid, varid, l2->lat[0]));
01109     NC(nc_inq_varid(ncid, "l2_press", &varid));
01110     NC(nc_get_var_double(ncid, varid, l2->p));
01111     NC(nc_inq_varid(ncid, "l2_temp", &varid));
01112     NC(nc_get_var_double(ncid, varid, l2->t[0][0]));
01113
01114     /* Close file... */
01115     NC(nc_close(ncid));
01116 }
```

### 5.27.1.23 void read\_pert ( char \* filename, char \* pertname, pert\_t \* pert )

Read radiance perturbation data.

Definition at line 1120 of file libairs.c.

```

01123         {
01124
01125     static char varname[LEN];
01126
01127     static int dimid[2], ncid, varid;
01128
01129     static size_t itrack, ntrack, nxtrack, start[2] = { 0, 0 }, count[2] = {
01130     1, 1};
01131
01132     /* Write info... */
01133     printf("Read perturbation data: %s\n", filename);
01134
01135     /* Open netCDF file... */
01136     NC(nc_open(filename, NC_NOWRITE, &ncid));
01137
01138     /* Get dimensions... */
01139     NC(nc_inq_dimid(ncid, "NTRACK", &dimid[0]));
01140     NC(nc_inq_dimid(ncid, "NXTRACK", &dimid[1]));
01141     NC(nc_inq_dimlen(ncid, dimid[0], &ntrack));
01142     NC(nc_inq_dimlen(ncid, dimid[1], &nxtrack));
01143     if (nxtrack > PERT_NXTRACK)
01144         ERRMSG("Too many tracks!");
01145     if (ntrack > PERT_NTRACK)
01146         ERRMSG("Too many scans!");
01147     pert->ntrack = (int) ntrack;
01148     pert->nxtrack = (int) nxtrack;
01149     count[1] = nxtrack;
01150
01151     /* Read data... */
01152     NC(nc_inq_varid(ncid, "time", &varid));
01153     for (itrack = 0; itrack < ntrack; itrack++) {
01154         start[0] = itrack;
01155         NC(nc_get_vara_double(ncid, varid, start, count, pert->time[itrack]));
01156     }
01157
01158     NC(nc_inq_varid(ncid, "lon", &varid));
01159     for (itrack = 0; itrack < ntrack; itrack++) {
01160         start[0] = itrack;
01161         NC(nc_get_vara_double(ncid, varid, start, count, pert->lon[itrack]));
01162     }
01163
01164     NC(nc_inq_varid(ncid, "lat", &varid));
01165     for (itrack = 0; itrack < ntrack; itrack++) {
01166         start[0] = itrack;
01167         NC(nc_get_vara_double(ncid, varid, start, count, pert->lat[itrack]));
01168     }
01169
01170     NC(nc_inq_varid(ncid, "bt_8mu", &varid));
01171     for (itrack = 0; itrack < ntrack; itrack++) {
01172         start[0] = itrack;
01173         NC(nc_get_vara_double(ncid, varid, start, count, pert->dc[itrack]));
01174     }
01175
01176     sprintf(varname, "bt_%s", pertname);
01177     NC(nc_inq_varid(ncid, varname, &varid));
01178     for (itrack = 0; itrack < ntrack; itrack++) {
01179         start[0] = itrack;
01180         NC(nc_get_vara_double(ncid, varid, start, count, pert->bt[itrack]));
01181     }
01182
01183     sprintf(varname, "bt_%s_pt", pertname);
01184     NC(nc_inq_varid(ncid, varname, &varid));
01185     for (itrack = 0; itrack < ntrack; itrack++) {
01186         start[0] = itrack;
01187         NC(nc_get_vara_double(ncid, varid, start, count, pert->pt[itrack]));
01188     }
01189
01190     sprintf(varname, "bt_%s_var", pertname);
01191     NC(nc_inq_varid(ncid, varname, &varid));
01192     for (itrack = 0; itrack < ntrack; itrack++) {
01193         start[0] = itrack;
01194         NC(nc_get_vara_double(ncid, varid, start, count, pert->var[itrack]));
01195     }
01196
01197     /* Close file... */
01198     NC(nc_close(ncid));
01199 }

```

## 5.27.1.24 void read\_retr ( char \* filename, ret\_t \* ret )

Read AIRS retrieval data.

Definition at line 1203 of file [libairs.c](#).

```

01205         {
01206
01207     static double help[NDS * NPG];
01208
01209     int dimid, ids = 0, ip, ncid, varid;
01210
01211     size_t itrack, ixtrack, nds, np, ntrack, nxtrack;
01212
01213     /* Write info... */
01214     printf("Read retrieval data: %s\n", filename);
01215
01216     /* Open netCDF file... */
01217     NC(nc_open(filename, NC_NOWRITE, &ncid));
01218
01219     /* Read new retrieval file format... */
01220     if (nc_inq_dimid(ncid, "L1_NTRACK", &dimid) == NC_NOERR) {
01221
01222         /* Get dimensions... */
01223         NC(nc_inq_dimid(ncid, "RET_NP", &dimid));
01224         NC(nc_inq_dimlen(ncid, dimid, &np));
01225         ret->np = (int) np;
01226         if (ret->np > NPG)
01227             ERRMSG("Too many data points!");
01228
01229         NC(nc_inq_dimid(ncid, "L1_NTRACK", &dimid));
01230         NC(nc_inq_dimlen(ncid, dimid, &ntrack));
01231         NC(nc_inq_dimid(ncid, "L1_NXTRACK", &dimid));
01232         NC(nc_inq_dimlen(ncid, dimid, &nxtrack));
01233         ret->nds = (int) (ntrack * nxtrack);
01234         if (ret->nds > NDS)
01235             ERRMSG("Too many data sets!");
01236
01237         /* Read time... */
01238         NC(nc_inq_varid(ncid, "l1_time", &varid));
01239         NC(nc_get_var_double(ncid, varid, help));
01240         ids = 0;
01241         for (itrack = 0; itrack < ntrack; itrack++)
01242             for (ixtrack = 0; ixtrack < nxtrack; ixtrack++) {
01243                 for (ip = 0; ip < ret->np; ip++)
01244                     ret->time[ids][ip] = help[ids];
01245                 ids++;
01246             }
01247
01248         /* Read altitudes... */
01249         NC(nc_inq_varid(ncid, "ret_z", &varid));
01250         NC(nc_get_var_double(ncid, varid, help));
01251         ids = 0;
01252         for (itrack = 0; itrack < ntrack; itrack++)
01253             for (ixtrack = 0; ixtrack < nxtrack; ixtrack++) {
01254                 for (ip = 0; ip < ret->np; ip++)
01255                     ret->z[ids][ip] = help[ip];
01256                 ids++;
01257             }
01258
01259         /* Read longitudes... */
01260         NC(nc_inq_varid(ncid, "l1_lon", &varid));
01261         NC(nc_get_var_double(ncid, varid, help));
01262         ids = 0;
01263         for (itrack = 0; itrack < ntrack; itrack++)
01264             for (ixtrack = 0; ixtrack < nxtrack; ixtrack++) {
01265                 for (ip = 0; ip < ret->np; ip++)
01266                     ret->lon[ids][ip] = help[ids];
01267                 ids++;
01268             }
01269
01270         /* Read latitudes... */
01271         NC(nc_inq_varid(ncid, "l1_lat", &varid));
01272         NC(nc_get_var_double(ncid, varid, help));
01273         ids = 0;
01274         for (itrack = 0; itrack < ntrack; itrack++)
01275             for (ixtrack = 0; ixtrack < nxtrack; ixtrack++) {
01276                 for (ip = 0; ip < ret->np; ip++)
01277                     ret->lat[ids][ip] = help[ids];
01278                 ids++;
01279             }
01280
01281         /* Read temperatures... */

```

```

01282     NC(nc_inq_varid(ncid, "ret_temp", &varid));
01283     NC(nc_get_var_double(ncid, varid, help));
01284     ids = 0;
01285     for (itrack = 0; itrack < ntrack; itrack++)
01286         for (ixtrack = 0; ixtrack < nxtrack; ixtrack++) {
01287             for (ip = 0; ip < ret->np; ip++)
01288                 ret->t[ids][ip] =
01289                     help[(itrack * nxtrack + ixtrack) * (size_t) np + (size_t) ip];
01290             ids++;
01291         }
01292     }
01293
01294     /* Read old retrieval file format... */
01295     if (nc_inq_dimid(ncid, "np", &dimid) == NC_NOERR) {
01296
01297         /* Get dimensions... */
01298         NC(nc_inq_dimid(ncid, "np", &dimid));
01299         NC(nc_inq_dimlen(ncid, dimid, &np));
01300         ret->np = (int) np;
01301         if (ret->np > NPG)
01302             ERRMSG("Too many data points!");
01303
01304         NC(nc_inq_dimid(ncid, "nds", &dimid));
01305         NC(nc_inq_dimlen(ncid, dimid, &nds));
01306         ret->nds = (int) nds;
01307         if (ret->nds > NDS)
01308             ERRMSG("Too many data sets!");
01309
01310         /* Read data... */
01311         NC(nc_inq_varid(ncid, "time", &varid));
01312         NC(nc_get_var_double(ncid, varid, help));
01313         read_retr_help(help, ret->nds, ret->np, ret->time);
01314
01315         NC(nc_inq_varid(ncid, "z", &varid));
01316         NC(nc_get_var_double(ncid, varid, help));
01317         read_retr_help(help, ret->nds, ret->np, ret->z);
01318
01319         NC(nc_inq_varid(ncid, "lon", &varid));
01320         NC(nc_get_var_double(ncid, varid, help));
01321         read_retr_help(help, ret->nds, ret->np, ret->lon);
01322
01323         NC(nc_inq_varid(ncid, "lat", &varid));
01324         NC(nc_get_var_double(ncid, varid, help));
01325         read_retr_help(help, ret->nds, ret->np, ret->lat);
01326
01327         NC(nc_inq_varid(ncid, "press", &varid));
01328         NC(nc_get_var_double(ncid, varid, help));
01329         read_retr_help(help, ret->nds, ret->np, ret->p);
01330
01331         NC(nc_inq_varid(ncid, "temp", &varid));
01332         NC(nc_get_var_double(ncid, varid, help));
01333         read_retr_help(help, ret->nds, ret->np, ret->t);
01334
01335         NC(nc_inq_varid(ncid, "temp_apr", &varid));
01336         NC(nc_get_var_double(ncid, varid, help));
01337         read_retr_help(help, ret->nds, ret->np, ret->t_apr);
01338
01339         NC(nc_inq_varid(ncid, "temp_total", &varid));
01340         NC(nc_get_var_double(ncid, varid, help));
01341         read_retr_help(help, ret->nds, ret->np, ret->t_tot);
01342
01343         NC(nc_inq_varid(ncid, "temp_noise", &varid));
01344         NC(nc_get_var_double(ncid, varid, help));
01345         read_retr_help(help, ret->nds, ret->np, ret->t_noise);
01346
01347         NC(nc_inq_varid(ncid, "temp_formod", &varid));
01348         NC(nc_get_var_double(ncid, varid, help));
01349         read_retr_help(help, ret->nds, ret->np, ret->t_fm);
01350
01351         NC(nc_inq_varid(ncid, "temp_cont", &varid));
01352         NC(nc_get_var_double(ncid, varid, help));
01353         read_retr_help(help, ret->nds, ret->np, ret->t_cont);
01354
01355         NC(nc_inq_varid(ncid, "temp_res", &varid));
01356         NC(nc_get_var_double(ncid, varid, help));
01357         read_retr_help(help, ret->nds, ret->np, ret->t_res);
01358
01359         NC(nc_inq_varid(ncid, "chisq", &varid));
01360         NC(nc_get_var_double(ncid, varid, ret->chisq));
01361     }
01362
01363     /* Close file... */
01364     NC(nc_close(ncid));
01365 }

```

Here is the call graph for this function:



#### 5.27.1.25 void read\_retr\_help ( double \* help, int nds, int np, double mat[NDS][NPG] )

Convert array.

Definition at line 1369 of file [libairs.c](#).

```

01373             {
01374
01375     int ids, ip, n = 0;
01376
01377     for (ip = 0; ip < np; ip++)
01378         for (ids = 0; ids < nds; ids++)
01379             mat[ids][ip] = help[n++];
01380 }
  
```

#### 5.27.1.26 void read\_wave ( char \* filename, wave\_t \* wave )

Read wave analysis data.

Definition at line 1384 of file [libairs.c](#).

```

01386             {
01387
01388     FILE *in;
01389
01390     char line[LEN];
01391
01392     double rtime, rz, rlon, rlat, rx, ry, ryold = -1e10, rtemp, rbg, rpt, rvar;
01393
01394     /* Init... */
01395     wave->nx = 0;
01396     wave->ny = 0;
01397
01398     /* Write info... */
01399     printf("Read wave data: %s\n", filename);
01400
01401     /* Open file... */
01402     if (!(in = fopen(filename, "r")))
01403         ERRMSG("Cannot open file!");
01404
01405     /* Read data... */
01406     while (fgets(line, LEN, in))
01407         if (sscanf(line, "%lg %lg %lg %lg %lg %lg %lg %lg %lg", &rtime,
01408                     &rz, &rlon, &rlat, &rx, &ry, &rtemp, &rbg, &rpt,
01409                     &rvar) == 10) {
01410
01411         /* Set index... */
01412         if (ry != ryold) {
01413             if ((++wave->ny >= WY))
01414                 ERRMSG("Too many y-values!");
01415             wave->nx = 0;
01416         } else if ((++wave->nx >= WX))
01417             ERRMSG("Too many x-values!");
01418         ryold = ry;
01419
01420         /* Save data... */
  
```



```

01421     wave->time = rtime;
01422     wave->z = rz;
01423     wave->lon[wave->nx][wave->ny] = rlon;
01424     wave->lat[wave->nx][wave->ny] = rlat;
01425     wave->x[wave->nx] = rx;
01426     wave->y[wave->ny] = ry;
01427     wave->temp[wave->nx][wave->ny] = rtemp;
01428     wave->bg[wave->nx][wave->ny] = rbg;
01429     wave->pt[wave->nx][wave->ny] = rpt;
01430     wave->var[wave->nx][wave->ny] = rvar;
01431 }
01432
01433 /* Increment counters... */
01434 wave->nx++;
01435 wave->ny++;
01436
01437 /* Close file... */
01438 fclose(in);
01439 }

```

**5.27.1.27** 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 1443 of file libairs.c.

```

01447     {
01448
01449     double x0[3], x1[3];
01450
01451     int ichan[AIRS_RAD_CHANNEL], id, track, xtrack;
01452
01453     /* Get channel numbers... */
01454     for (id = 0; id < nd; id++) {
01455         for (ichan[id] = 0; ichan[id] < AIRS_RAD_CHANNEL; ichan[id]++)
01456             if (fabs(gran->nominal_freq[ichan[id]] - nu[id]) < 0.1)
01457                 break;
01458         if (ichan[id] >= AIRS_RAD_CHANNEL)
01459             ERRMSG("Could not find channel!");
01460     }
01461
01462     /* Set size... */
01463     wave->nx = AIRS_RAD_GEOXTRACK;
01464     wave->ny = AIRS_RAD_GEOTRACK;
01465     if (wave->nx > WX || wave->ny > WY)
01466         ERRMSG("Wave struct too small!");
01467
01468     /* Set Cartesian coordinates... */
01469     geo2cart(0, gran->Longitude[0][0], gran->Latitude[0][0], x0);
01470     for (xtrack = 0; xtrack < AIRS_RAD_GEOXTRACK; xtrack++) {
01471         geo2cart(0, gran->Longitude[0][xtrack], gran->Latitude[0][xtrack], x1);
01472         wave->x[xtrack] = DIST(x0, x1);
01473     }
01474     for (track = 0; track < AIRS_RAD_GEOTRACK; track++) {
01475         geo2cart(0, gran->Longitude[track][0], gran->Latitude[track][0], x1);
01476         wave->y[track] = DIST(x0, x1);
01477     }
01478
01479     /* Set geolocation... */
01480     wave->time =
01481         gran->Time[AIRS_RAD_GEOTRACK / 2][AIRS_RAD_GEOXTRACK / 2] - 220838400;
01482     wave->z = 0;
01483     for (track = 0; track < AIRS_RAD_GEOTRACK; track++)
01484         for (xtrack = 0; xtrack < AIRS_RAD_GEOXTRACK; xtrack++) {
01485             wave->lon[xtrack][track] = gran->Longitude[track][xtrack];
01486             wave->lat[xtrack][track] = gran->Latitude[track][xtrack];
01487         }
01488
01489     /* Set brightness temperature... */
01490     for (track = 0; track < AIRS_RAD_GEOTRACK; track++)
01491         for (xtrack = 0; xtrack < AIRS_RAD_GEOXTRACK; xtrack++) {
01492             wave->temp[xtrack][track] = 0;
01493             wave->bg[xtrack][track] = 0;
01494             wave->pt[xtrack][track] = 0;
01495             wave->var[xtrack][track] = 0;
01496             for (id = 0; id < nd; id++) {
01497                 if ((gran->state[track][xtrack] != 0)
01498                     || (gran->ExcludedChans[ichan[id]] > 2)
01499                     || (gran->CalChanSummary[ichan[id]] & 8)
01500                     || (gran->CalChanSummary[ichan[id]] & (32 + 64)))

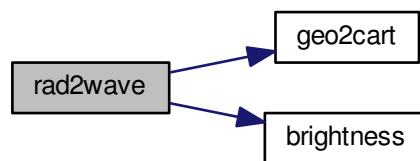
```

```

01501         || (gran->CalFlag[track][ichan[id]] & 16))
01502         wave->temp[xtrack][track] = GSL_NAN;
01503     else
01504         wave->temp[xtrack][track]
01505         += brightness(gran->radiances[track][xtrack][ichan[id]] * 1e-3,
01506                     gran->nominal_freq[ichan[id]]) / nd;
01507     }
01508 }
01509 }

```

Here is the call graph for this function:



#### 5.27.1.28 void ret2wave ( ret\_t \* ret, wave\_t \* wave, int dataset, int ip )

Convert AIRS retrieval results to wave analysis struct.

Definition at line 1513 of file [libairs.c](#).

```

01517     {
01518
01519     double x0[3], x1[3];
01520
01521     int ids, ix, iy;
01522
01523     /* Initialize... */
01524     wave->nx = 90;
01525     if (wave->nx > WX)
01526         ERRMSG("Too many across-track values!");
01527     wave->ny = 135;
01528     if (wave->ny > WY)
01529         ERRMSG("Too many along-track values!");
01530     if (ip < 0 || ip >= ret->np)
01531         ERRMSG("Altitude index out of range!");
01532
01533     /* Loop over data sets and data points... */
01534     for (ids = 0; ids < ret->nds; ids++) {
01535
01536         /* Get horizontal indices... */
01537         ix = ids % 90;
01538         iy = ids / 90;
01539
01540         /* Get distances... */
01541         if (iy == 0) {
01542             geo2cart(0.0, ret->lon[0][0], ret->lat[0][0], x0);
01543             geo2cart(0.0, ret->lon[ids][ip], ret->lat[ids][ip], x1);
01544             wave->x[ix] = DIST(x0, x1);
01545         }
01546         if (ix == 0) {
01547             geo2cart(0.0, ret->lon[0][0], ret->lat[0][0], x0);
01548             geo2cart(0.0, ret->lon[ids][ip], ret->lat[ids][ip], x1);
01549             wave->y[iy] = DIST(x0, x1);
01550         }
01551
01552         /* Save geolocation... */
01553         wave->time = ret->time[0][0];
01554         if (ix == 0 && iy == 0)
01555             wave->z = ret->z[ids][ip];
01556         wave->lon[ix][iy] = ret->lon[ids][ip];
01557     }
01558 }

```

```

01557     wave->lat[ix][iy] = ret->lat[ids][ip];
01558
01559     /* Save temperature... */
01560     if (dataset == 1)
01561         wave->temp[ix][iy] = ret->t[ids][ip];
01562     else if (dataset == 2)
01563         wave->temp[ix][iy] = ret->t_apr[ids][ip];
01564     }
01565 }

```

Here is the call graph for this function:



#### 5.27.1.29 double sza ( double sec, double lon, double lat )

Calculate solar zenith angle.

Definition at line 1569 of file libairs.c.

```

01572     {
01573
01574     double D, dec, e, g, GMST, h, L, LST, q, ra;
01575
01576     /* Number of days and fraction with respect to 2000-01-01T12:00Z... */
01577     D = sec / 86400 - 0.5;
01578
01579     /* Geocentric apparent ecliptic longitude [rad]... */
01580     g = (357.529 + 0.98560028 * D) * M_PI / 180;
01581     q = 280.459 + 0.98564736 * D;
01582     L = (q + 1.915 * sin(g) + 0.020 * sin(2 * g)) * M_PI / 180;
01583
01584     /* Mean obliquity of the ecliptic [rad]... */
01585     e = (23.439 - 0.00000036 * D) * M_PI / 180;
01586
01587     /* Declination [rad]... */
01588     dec = asin(sin(e) * sin(L));
01589
01590     /* Right ascension [rad]... */
01591     ra = atan2(cos(e) * sin(L), cos(L));
01592
01593     /* Greenwich Mean Sidereal Time [h]... */
01594     GMST = 18.697374558 + 24.06570982441908 * D;
01595
01596     /* Local Sidereal Time [h]... */
01597     LST = GMST + lon / 15;
01598
01599     /* Hour angle [rad]... */
01600     h = LST / 12 * M_PI - ra;
01601
01602     /* Convert latitude... */
01603     lat *= M_PI / 180;
01604
01605     /* Return solar zenith angle [deg]... */
01606     return acos(sin(lat) * sin(dec) +
01607                cos(lat) * cos(dec) * cos(h)) * 180 / M_PI;
01608 }

```

## 5.27.1.30 void variance ( wave\_t \* wave, double dh )

Compute local variance.

Definition at line 1612 of file libairs.c.

```

01614         {
01615
01616         double dh2, mu, help;
01617
01618         int dx, dy, ix, ix2, iy, iy2, n;
01619
01620         /* Check parameters... */
01621         if (dh <= 0)
01622             return;
01623
01624         /* Compute squared radius... */
01625         dh2 = gsl_pow_2(dh);
01626
01627         /* Get sampling distances... */
01628         dx =
01629             (int) (dh / fabs(wave->x[wave->nx - 1] - wave->x[0]) * (wave->nx - 1.0) +
01630                 1);
01631         dy =
01632             (int) (dh / fabs(wave->y[wave->ny - 1] - wave->y[0]) * (wave->ny - 1.0) +
01633                 1);
01634
01635         /* Loop over data points... */
01636         for (ix = 0; ix < wave->nx; ix++)
01637             for (iy = 0; iy < wave->ny; iy++) {
01638
01639                 /* Init... */
01640                 mu = help = 0;
01641                 n = 0;
01642
01643                 /* Get data... */
01644                 for (ix2 = GSL_MAX(ix - dx, 0); ix2 <= GSL_MIN(ix + dx, wave->nx - 1);
01645                     ix2++)
01646                     for (iy2 = GSL_MAX(iy - dy, 0); iy2 <= GSL_MIN(iy + dy, wave->ny - 1);
01647                         iy2++)
01648                         if ((gsl_pow_2(wave->x[ix] - wave->x[ix2])
01649                             + gsl_pow_2(wave->y[iy] - wave->y[iy2])) <= dh2)
01650                             if (gsl_finite(wave->pt[ix2][iy2])) {
01651                                 mu += wave->pt[ix2][iy2];
01652                                 help += gsl_pow_2(wave->pt[ix2][iy2]);
01653                                 n++;
01654                             }
01655
01656                 /* Compute local variance... */
01657                 if (n > 1)
01658                     wave->var[ix][iy] = help / n - gsl_pow_2(mu / n);
01659                 else
01660                     wave->var[ix][iy] = GSL_NAN;
01661             }
01662     }

```

## 5.27.1.31 void write\_l1 ( char \* filename, airs\_l1\_t \* l1 )

Write AIRS Level-1 data.

Definition at line 1666 of file libairs.c.

```

01668         {
01669
01670         int dimid[10], ncid, time_id, lon_id, lat_id,
01671             sat_z_id, sat_lon_id, sat_lat_id, nu_id, rad_id;
01672
01673         /* Open or create netCDF file... */
01674         printf("Write AIRS Level-1 file: %s\n", filename);
01675         if (nc_open(filename, NC_WRITE, &ncid) != NC_NOERR) {
01676             NC(nc_create(filename, NC_CLOBBER, &ncid));
01677         } else {
01678             NC(nc_redef(ncid));
01679         }
01680
01681         /* Set dimensions... */

```

```

01682  if (nc_inq_dimid(ncid, "L1_NTRACK", &dimid[0]) != NC_NOERR)
01683      NC(nc_def_dim(ncid, "L1_NTRACK", L1_NTRACK, &dimid[0]));
01684  if (nc_inq_dimid(ncid, "L1_NXTRACK", &dimid[1]) != NC_NOERR)
01685      NC(nc_def_dim(ncid, "L1_NXTRACK", L1_NXTRACK, &dimid[1]));
01686  if (nc_inq_dimid(ncid, "L1_NCHAN", &dimid[2]) != NC_NOERR)
01687      NC(nc_def_dim(ncid, "L1_NCHAN", L1_NCHAN, &dimid[2]));
01688
01689  /* Add variables... */
01690  add_var(ncid, "l1_time", "s", "time (seconds since 2000-01-01T00:00Z)",
01691          NC_DOUBLE, dimid, &time_id, 2);
01692  add_var(ncid, "l1_lon", "deg", "longitude", NC_DOUBLE, dimid, &lon_id, 2);
01693  add_var(ncid, "l1_lat", "deg", "latitude", NC_DOUBLE, dimid, &lat_id, 2);
01694  add_var(ncid, "l1_sat_z", "km", "satellite altitude",
01695          NC_DOUBLE, dimid, &sat_z_id, 1);
01696  add_var(ncid, "l1_sat_lon", "deg", "satellite longitude",
01697          NC_DOUBLE, dimid, &sat_lon_id, 1);
01698  add_var(ncid, "l1_sat_lat", "deg", "satellite latitude",
01699          NC_DOUBLE, dimid, &sat_lat_id, 1);
01700  add_var(ncid, "l1_nu", "cm^-1", "channel wavenumber",
01701          NC_DOUBLE, &dimid[2], &nu_id, 1);
01702  add_var(ncid, "l1_rad", "W/(m^2 sr cm^-1)", "channel radiance",
01703          NC_FLOAT, dimid, &rad_id, 3);
01704
01705  /* Leave define mode... */
01706  NC(nc_enddef(ncid));
01707
01708  /* Write data... */
01709  NC(nc_put_var_double(ncid, time_id, l1->time[0]));
01710  NC(nc_put_var_double(ncid, lon_id, l1->lon[0]));
01711  NC(nc_put_var_double(ncid, lat_id, l1->lat[0]));
01712  NC(nc_put_var_double(ncid, sat_z_id, l1->sat_z));
01713  NC(nc_put_var_double(ncid, sat_lon_id, l1->sat_lon));
01714  NC(nc_put_var_double(ncid, sat_lat_id, l1->sat_lat));
01715  NC(nc_put_var_double(ncid, nu_id, l1->nu));
01716  NC(nc_put_var_float(ncid, rad_id, l1->rad[0][0]));
01717
01718  /* Close file... */
01719  NC(nc_close(ncid));
01720 }

```

Here is the call graph for this function:



#### 5.27.1.32 void write\_l2 ( char \* filename, airs\_l2\_t \* l2 )

Write AIRS Level-2 data.

Definition at line 1724 of file libairs.c.

```

01726      {
01727
01728  int dimid[10], ncid, time_id, z_id, lon_id, lat_id, p_id, t_id;
01729
01730  /* Create netCDF file... */
01731  printf("Write AIRS Level-2 file: %s\n", filename);
01732  if (nc_open(filename, NC_WRITE, &ncid) != NC_NOERR) {
01733      NC(nc_create(filename, NC_CLOBBER, &ncid));
01734  } else {
01735      NC(nc_redef(ncid));
01736  }
01737
01738  /* Set dimensions... */
01739  if (nc_inq_dimid(ncid, "L2_NTRACK", &dimid[0]) != NC_NOERR)

```

```

01740     NC(nc_def_dim(ncid, "L2_NTRACK", L2_NTRACK, &dimid[0]));
01741     if (nc_inq_dimid(ncid, "L2_NXTRACK", &dimid[1]) != NC_NOERR)
01742         NC(nc_def_dim(ncid, "L2_NXTRACK", L2_NXTRACK, &dimid[1]));
01743     if (nc_inq_dimid(ncid, "L2_NLAY", &dimid[2]) != NC_NOERR)
01744         NC(nc_def_dim(ncid, "L2_NLAY", L2_NLAY, &dimid[2]));
01745
01746     /* Add variables... */
01747     add_var(ncid, "l2_time", "s", "time (seconds since 2000-01-01T00:00Z)",
01748             NC_DOUBLE, dimid, &time_id, 2);
01749     add_var(ncid, "l2_z", "km", "altitude", NC_DOUBLE, dimid, &z_id, 3);
01750     add_var(ncid, "l2_lon", "deg", "longitude", NC_DOUBLE, dimid, &lon_id, 2);
01751     add_var(ncid, "l2_lat", "deg", "latitude", NC_DOUBLE, dimid, &lat_id, 2);
01752     add_var(ncid, "l2_press", "hPa", "pressure",
01753             NC_DOUBLE, &dimid[2], &p_id, 1);
01754     add_var(ncid, "l2_temp", "K", "temperature", NC_DOUBLE, dimid, &t_id, 3);
01755
01756     /* Leave define mode... */
01757     NC(nc_enddef(ncid));
01758
01759     /* Write data... */
01760     NC(nc_put_var_double(ncid, time_id, l2->time[0]));
01761     NC(nc_put_var_double(ncid, z_id, l2->z[0][0]));
01762     NC(nc_put_var_double(ncid, lon_id, l2->lon[0]));
01763     NC(nc_put_var_double(ncid, lat_id, l2->lat[0]));
01764     NC(nc_put_var_double(ncid, p_id, l2->p));
01765     NC(nc_put_var_double(ncid, t_id, l2->t[0][0]));
01766
01767     /* Close file... */
01768     NC(nc_close(ncid));
01769 }

```

Here is the call graph for this function:



#### 5.27.1.33 void write\_wave ( char \* filename, wave\_t \* wave )

Write wave analysis data.

Definition at line 1773 of file libairs.c.

```

01775     {
01776
01777     FILE *out;
01778
01779     int i, j;
01780
01781     /* Write info... */
01782     printf("Write wave data: %s\n", filename);
01783
01784     /* Create file... */
01785     if (!(out = fopen(filename, "w")))
01786         ERRMSG("Cannot create file!");
01787
01788     /* Write header... */
01789     fprintf(out,
01790            "# $1 = time (seconds since 2000-01-01T00:00Z)\n"
01791            "# $2 = altitude [km]\n"
01792            "# $3 = longitude [deg]\n"
01793            "# $4 = latitude [deg]\n"
01794            "# $5 = across-track distance [km]\n"
01795            "# $6 = along-track distance [km]\n"
01796            "# $7 = temperature [K]\n"
01797            "# $8 = background [K]\n"

```

```

01798         "# $9 = perturbation [K]\n" "# $10 = variance [K^2]\n");
01799
01800     /* Write data... */
01801     for (j = 0; j < wave->ny; j++) {
01802         fprintf(out, "\n");
01803         for (i = 0; i < wave->nx; i++)
01804             fprintf(out, "%.2f %g %g %g %g %g %g %g %g\n",
01805                     wave->time, wave->z, wave->lon[i][j], wave->lat[i][j],
01806                     wave->x[i], wave->y[j], wave->temp[i][j], wave->bg[i][j],
01807                     wave->pt[i][j], wave->var[i][j]);
01808     }
01809
01810     /* Close file... */
01811     fclose(out);
01812 }

```

## 5.28 libairs.h

```

00001 #include <netcdf.h>
00002 #include <gsl/gsl_randist.h>
00003 #include <gsl/gsl_fft_complex.h>
00004 #include <gsl/gsl_multifit.h>
00005 #include <gsl/gsl_poly.h>
00006 #include <gsl/gsl_sort.h>
00007 #include <gsl/gsl_spline.h>
00008 #include <airs_rad_typ.h>
00009 #include <airs_rad_struct.h>
00010 #include <airs_ret_typ.h>
00011 #include <airs_ret_struct.h>
00012 #include "jurassic.h"
00013
00014 /* -----
00015     Dimensions...
00016     ----- */
00017
00019 #define NDS 13000
00020
00022 #define NPG 30
00023
00025 #define L1_NCHAN 34
00026
00028 #define L1_NTRACK 135
00029
00031 #define L1_NXTRACK 90
00032
00034 #define L2_NLAY 27
00035
00037 #define L2_NTRACK 45
00038
00040 #define L2_NXTRACK 30
00041
00043 #define PERT_NTRACK 132000
00044
00046 #define PERT_NXTRACK 360
00047
00049 #define WX 300
00050
00052 #define WY 33000
00053
00055 #define PMAX 512
00056
00057 /* -----
00058     Macros...
00059     ----- */
00060
00062 #define NC(cmd) {
00063     if ((cmd) != NC_NOERR)
00064         ERRMSG(nc_strerror(cmd));
00065 }
00066
00067 /* -----
00068     Structs...
00069     ----- */
00070
00072 typedef struct {
00073     double time[L1_NTRACK][L1_NXTRACK];
00074
00075     double lon[L1_NTRACK][L1_NXTRACK];
00076
00078     double lat[L1_NTRACK][L1_NXTRACK];
00079
00081     double sat_z[L1_NTRACK];

```

```

00085
00087     double sat_lon[L1_NTRACK];
00088
00090     double sat_lat[L1_NTRACK];
00091
00093     double nu[L1_NCHAN];
00094
00096     float rad[L1_NTRACK][L1_NXTRACK][L1_NCHAN];
00097
00098 } airs_l1_t;
00099
00101 typedef struct {
00102
00104     double time[L2_NTRACK][L2_NXTRACK];
00105
00107     double z[L2_NTRACK][L2_NXTRACK][L2_NLAY];
00108
00110     double lon[L2_NTRACK][L2_NXTRACK];
00111
00113     double lat[L2_NTRACK][L2_NXTRACK];
00114
00116     double p[L2_NLAY];
00117
00119     double t[L2_NTRACK][L2_NXTRACK][L2_NLAY];
00120
00121 } airs_l2_t;
00122
00124 typedef struct {
00125
00127     int ntrack;
00128
00130     int nxtrack;
00131
00133     double time[PERT_NTRACK][PERT_NXTRACK];
00134
00136     double lon[PERT_NTRACK][PERT_NXTRACK];
00137
00139     double lat[PERT_NTRACK][PERT_NXTRACK];
00140
00142     double dc[PERT_NTRACK][PERT_NXTRACK];
00143
00145     double bt[PERT_NTRACK][PERT_NXTRACK];
00146
00148     double pt[PERT_NTRACK][PERT_NXTRACK];
00149
00151     double var[PERT_NTRACK][PERT_NXTRACK];
00152
00153 } pert_t;
00154
00156 typedef struct {
00157
00159     int nds;
00160
00162     int np;
00163
00165     double time[NDS][NPG];
00166
00168     double z[NDS][NPG];
00169
00171     double lon[NDS][NPG];
00172
00174     double lat[NDS][NPG];
00175
00177     double p[NDS][NPG];
00178
00180     double t[NDS][NPG];
00181
00183     double t_apr[NDS][NPG];
00184
00186     double t_tot[NDS][NPG];
00187
00189     double t_noise[NDS][NPG];
00190
00192     double t_fm[NDS][NPG];
00193
00195     double t_cont[NDS][NPG];
00196
00198     double t_res[NDS][NPG];
00199
00201     double chisq[NDS];
00202
00203 } ret_t;
00204
00206 typedef struct {
00207
00209     int nx;
00210

```



```

00212     int ny;
00213
00215     double time;
00216
00218     double z;
00219
00221     double lon[WX][WY];
00222
00224     double lat[WX][WY];
00225
00227     double x[WX];
00228
00230     double y[WY];
00231
00233     double temp[WX][WY];
00234
00236     double bg[WX][WY];
00237
00239     double pt[WX][WY];
00240
00242     double var[WX][WY];
00243
00244 } wave_t;
00245
00246 /* -----
00247     Functions...
00248     ----- */
00249
00251 void add_att(
00252     int ncid,
00253     int varid,
00254     const char *unit,
00255     const char *long_name);
00256
00258 void add_var(
00259     int ncid,
00260     const char *varname,
00261     const char *unit,
00262     const char *longname,
00263     int type,
00264     int dimid[],
00265     int *varid,
00266     int ndims);
00267
00269 void background_poly(
00270     wave_t * wave,
00271     int dim_x,
00272     int dim_y);
00273
00275 void background_poly_help(
00276     double *xx,
00277     double *yy,
00278     int n,
00279     int dim);
00280
00282 void background_smooth(
00283     wave_t * wave,
00284     int npts_x,
00285     int npts_y);
00286
00288 void create_background(
00289     wave_t * wave);
00290
00292 void create_noise(
00293     wave_t * wave,
00294     double nedt);
00295
00297 void create_wave(
00298     wave_t * wave,
00299     double amp,
00300     double lx,
00301     double ly,
00302     double phi,
00303     double fwhm);
00304
00306 void day2doy(
00307     int year,
00308     int mon,
00309     int day,
00310     int *doy);
00311
00313 void doy2day(
00314     int year,
00315     int doy,
00316     int *mon,
00317     int *day);
00318

```

```
00320 void fft_help(  
00321     double *fcReal,  
00322     double *fcImag,  
00323     int n);  
00324  
00326 void fft(  
00327     wave_t * wave,  
00328     double *Amax,  
00329     double *phimax,  
00330     double *lhmax,  
00331     double *alphamax,  
00332     double *betamax,  
00333     char *filename);  
00334  
00336 void gauss(  
00337     wave_t * wave,  
00338     double fwhm);  
00339  
00341 void hamming(  
00342     wave_t * wave,  
00343     int nit);  
00344  
00346 void intpol_x(  
00347     wave_t * wave,  
00348     int n);  
00349  
00351 void median(  
00352     wave_t * wave,  
00353     int dx);  
00354  
00356 void merge_y(  
00357     wave_t * wave1,  
00358     wave_t * wave2);  
00359  
00361 void noise(  
00362     wave_t * wave,  
00363     double *mu,  
00364     double *sig);  
00365  
00367 void period(  
00368     wave_t * wave,  
00369     double *Amax,  
00370     double *phimax,  
00371     double *lhmax,  
00372     double *alphamax,  
00373     double *betamax,  
00374     char *filename);  
00375  
00377 void pert2wave(  
00378     pert_t * pert,  
00379     wave_t * wave,  
00380     int track0,  
00381     int track1,  
00382     int xtrack0,  
00383     int xtrack1);  
00384  
00386 void read_l1(  
00387     char *filename,  
00388     airs_l1_t * l1);  
00389  
00391 void read_l2(  
00392     char *filename,  
00393     airs_l2_t * l2);  
00394  
00396 void read_pert(  
00397     char *filename,  
00398     char *pertname,  
00399     pert_t * pert);  
00400  
00402 void read_retr(  
00403     char *filename,  
00404     ret_t * ret);  
00405  
00407 void read_retr_help(  
00408     double *help,  
00409     int nds,  
00410     int np,  
00411     double mat[NDS][NPG]);  
00412  
00414 void read_wave(  
00415     char *filename,  
00416     wave_t * wave);  
00417  
00419 void rad2wave(  
00420     airs_rad_gran_t * airs_rad_gran,  
00421     double *nu,  
00422     int nd,
```

```

00423     wave_t * wave);
00424
00426 void ret2wave(
00427     ret_t * ret,
00428     wave_t * wave,
00429     int dataset,
00430     int ip);
00431
00433 double sza(
00434     double sec,
00435     double lon,
00436     double lat);
00437
00439 void variance(
00440     wave_t * wave,
00441     double dh);
00442
00444 void write_l1(
00445     char *filename,
00446     airs_l1_t * l1);
00447
00449 void write_l2(
00450     char *filename,
00451     airs_l2_t * l2);
00452
00454 void write_wave(
00455     char *filename,
00456     wave_t * wave);

```

## 5.29 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[])

### 5.29.1 Function Documentation

#### 5.29.1.1 double fill\_array ( double var[PERT\_NTRACK][PERT\_NXTRACK], int ntrack, int itrack, int ixtrack )

Definition at line [209](#) of file [map\\_pert.c](#).

```

00213         {
00214
00215     double d1 = 0, d2 = 0, v1 = 0, v2 = 0;
00216
00217     int i;
00218
00219     /* Find nearest neighbours... */
00220     for (i = itrack + 1; i < ntrack; i++)
00221         if (gsl_finite(var[i][ixtrack])) {
00222             d1 = fabs(i - itrack);
00223             v1 = var[i][ixtrack];
00224             break;
00225         }
00226     for (i = itrack - 1; i >= 0; i--)
00227         if (gsl_finite(var[i][ixtrack])) {
00228             d2 = fabs(i - itrack);
00229             v2 = var[i][ixtrack];
00230             break;
00231         }
00232
00233     /* Interpolate... */
00234     if (d1 + d2 > 0)
00235         return (d2 * v1 + d1 * v2) / (d1 + d2);
00236     else
00237         return GSL_NAN;
00238 }

```

## 5.29.1.2 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, sza0, sza1, sza2 = 0;
00029
00030     int asc, bg_poly_x, bg_poly_y, bg_smooth_x, bg_smooth_y, ham_iter,
00031         itrack, itrackx, 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     sza0 = scan_ctl(argc, argv, "SZA0", -1, "-1e100", NULL);
00055     sza1 = scan_ctl(argc, argv, "SZA1", -1, "1e100", NULL);
00056     dt230 = scan_ctl(argc, argv, "DT230", -1, "0.16", NULL);
00057     nu = scan_ctl(argc, argv, "NU", -1, "2345.0", NULL);
00058     fill = (int) scan_ctl(argc, argv, "FILL", -1, "0", NULL);
00059
00060     /* Allocate... */
00061     ALLOC(pert, pert_t, 1);
00062     ALLOC(pert2, pert_t, 1);
00063
00064     /* Read perturbation data... */
00065     read_pert(argv[2], pertname, pert);
00066
00067     /* Recalculate background and perturbations... */
00068     if (bg_poly_x > 0 || bg_poly_y > 0 ||
00069         bg_smooth_x > 0 || bg_smooth_y > 0 ||
00070         gauss_fwhm > 0 || ham_iter > 0 || med_dx > 0 || var_dh > 0) {
00071
00072         /* Convert to wave analysis struct... */
00073         pert2wave(pert, &wave, 0, pert->ntrack - 1, 0, pert->ntrack - 1);
00074
00075         /* Estimate background... */
00076         background_poly(&wave, bg_poly_x, bg_poly_y);
00077         background_smooth(&wave, bg_smooth_x, bg_smooth_y);
00078
00079         /* Gaussian filter... */
00080         gauss(&wave, gauss_fwhm);
00081
00082         /* Hamming filter... */
00083         hamming(&wave, ham_iter);
00084
00085         /* Median filter... */
00086         median(&wave, med_dx);
00087
00088         /* Compute variance... */
00089         variance(&wave, var_dh);
00090
00091         /* Copy data... */
00092         for (ix = 0; ix < wave.nx; ix++)
00093             for (iy = 0; iy < wave.ny; iy++) {
00094                 pert->pt[ix][iy] = wave.pt[ix][iy];
00095                 pert->var[ix][iy] = wave.var[ix][iy];
00096             }
00097     }
00098
00099     /* Fill data gaps... */

```

```

00100     if (fill)
00101     for (itrack = 0; itrack < pert->ntrack; itrack++)
00102     for (ixtrack = 0; ixtrack < pert->nxtrack; ixtrack++) {
00103         if (!gsl_finite(pert->dc[itrack][ixtrack]))
00104             pert->dc[itrack][ixtrack]
00105             = fill_array(pert->dc, pert->ntrack, itrack, ixtrack);
00106         if (!gsl_finite(pert->bt[itrack][ixtrack]))
00107             pert->bt[itrack][ixtrack]
00108             = fill_array(pert->bt, pert->ntrack, itrack, ixtrack);
00109         if (!gsl_finite(pert->pt[itrack][ixtrack]))
00110             pert->pt[itrack][ixtrack]
00111             = fill_array(pert->pt, pert->ntrack, itrack, ixtrack);
00112         if (!gsl_finite(pert->var[itrack][ixtrack]))
00113             pert->var[itrack][ixtrack]
00114             = fill_array(pert->var, pert->ntrack, itrack, ixtrack);
00115     }
00116
00117     /* Interpolate to fine grid... */
00118     memcpy(pert2, pert, sizeof(pert_t));
00119
00120     /* Create output file... */
00121     printf("Write perturbation data: %s\n", argv[3]);
00122     if (!(out = fopen(argv[3], "w")))
00123         ERRMSG("Cannot create file!");
00124
00125     /* Write header... */
00126     fprintf(out,
00127         "## $1 = time (seconds since 01-JAN-2000, 00:00 UTC)\n"
00128         "## $2 = along-track index\n"
00129         "## $3 = longitude [deg]\n"
00130         "## $4 = latitude [deg]\n"
00131         "## $5 = 8mu brightness temperature [K]\n"
00132         "## $6 = %s brightness temperature [K]\n"
00133         "## $7 = %s brightness temperature perturbation [K]\n"
00134         "## $8 = %s brightness temperature variance [K^2]\n",
00135         pertname, pertname, pertname);
00136
00137     /* Write data... */
00138     for (itrack = 0; itrack < pert->ntrack; itrack++) {
00139
00140         /* Count orbits... */
00141         if (itrack > 0)
00142             if (pert->lat[itrack - 1][pert->nxtrack / 2] <= orblat
00143                 && pert->lat[itrack][pert->nxtrack / 2] >= orblat)
00144                 orb++;
00145
00146         /* Write output... */
00147         fprintf(out, "\n");
00148
00149         /* Check for data gaps... */
00150         if (itrack > 0 && pert->time[itrack][pert->nxtrack / 2]
00151             - pert->time[itrack - 1][pert->nxtrack / 2] >= 10)
00152             fprintf(out, "\n");
00153
00154         /* Loop over scan... */
00155         for (ixtrack = 0; ixtrack < pert->nxtrack; ixtrack++) {
00156
00157             /* Check data... */
00158             if (pert->lon[itrack][ixtrack] < -180
00159                 || pert->lon[itrack][ixtrack] > 180
00160                 || pert->lat[itrack][ixtrack] < -90
00161                 || pert->lat[itrack][ixtrack] > 90)
00162                 continue;
00163
00164             /* Get ascending/descending flag... */
00165             asc = (pert->lat[itrack] > 0 ? itrack : itrack + 1)[pert->nxtrack / 2]
00166                 > pert->lat[itrack]
00167                 ? 0 : itrack - 1 : itrack][pert->nxtrack / 2]);
00168
00169             /* Calculate solar zenith angle... */
00170             if (sza0 >= -1e10 && sza0 <= 1e10 && sza1 >= -1e10 && sza1 <= 1e10)
00171                 sza2 = sza(pert->time[itrack][ixtrack], pert->lon[itrack][ixtrack],
00172                     pert->lat[itrack][ixtrack]);
00173
00174             /* Estimate noise... */
00175             if (dt230 > 0) {
00176                 nesr = planck(t230 + dt230, nu) - planck(t230, nu);
00177                 tbg = pert->bt[itrack][ixtrack] - pert->pt[itrack][ixtrack];
00178                 nedt = brightness(planck(tbg, nu) + nesr, nu) - tbg;
00179             }
00180
00181             /* Write data... */
00182             if (orbit < 0 || orb == orbit)
00183                 if (set[0] == 'f' || (set[0] == 'a' && asc)
00184                     || (set[0] == 'd' && !asc))
00185                     if (pert->time[itrack][ixtrack] >= t0
00186                         && pert->time[itrack][ixtrack] <= t1

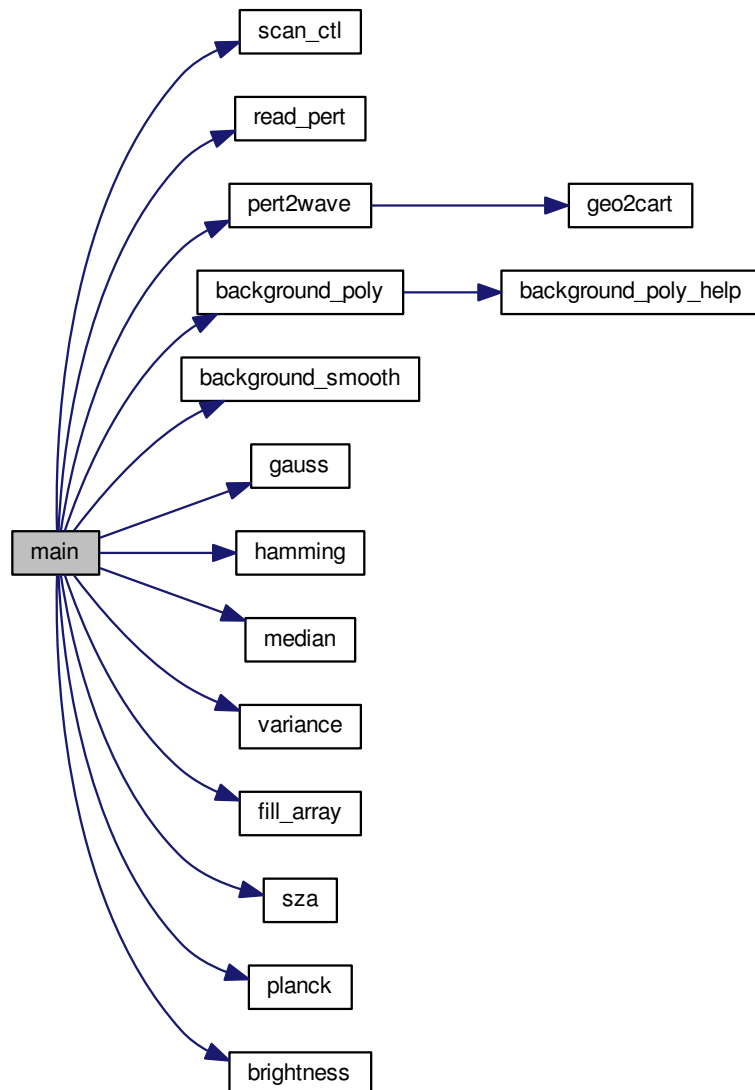
```

```

00187         && sza2 >= sza0 && sza2 <= sza1)
00188     fprintf(out, "%.2f %d %g %g %g %g %g %g\n",
00189         pert->time[itrack][ixtrack], itrack,
00190         pert->lon[itrack][ixtrack], pert->lat[itrack][ixtrack],
00191         pert->dc[itrack][ixtrack], pert->bt[itrack][ixtrack],
00192         pert->pt[itrack][ixtrack],
00193         pert->var[itrack][ixtrack] - gsl_pow_2(nedt));
00194     }
00195 }
00196
00197 /* Close file... */
00198 fclose(out);
00199
00200 /* Free... */
00201 free(pert);
00202 free(pert2);
00203
00204 return EXIT_SUCCESS;
00205 }

```

Here is the call graph for this function:



## 5.30 map\_pert.c

```

00001 #include "libairs.h"
00002
00003 /* -----
00004    Functions...
00005 ----- */
00006
00007 /* Fill data gaps in perturbation data. */
00008 double fill_array(
00009     double var[PERT_NTRACK][PERT_NXTRACK],
00010     int ntrack,
00011     int itrack,
00012     int ixtrack);
00013
00014 /* -----
00015    Main...
00016 ----- */
00017
00018 int main(
00019     int argc,
00020     char *argv[]) {
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, sza0, sza1, sza2 = 0;
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     sza0 = scan_ctl(argc, argv, "SZA0", -1, "-1e100", NULL);
00055     sza1 = scan_ctl(argc, argv, "SZA1", -1, "1e100", NULL);
00056     dt230 = scan_ctl(argc, argv, "DT230", -1, "0.16", NULL);
00057     nu = scan_ctl(argc, argv, "NU", -1, "2345.0", NULL);
00058     fill = (int) scan_ctl(argc, argv, "FILL", -1, "0", NULL);
00059
00060     /* Allocate... */
00061     ALLOC(pert, pert_t, 1);
00062     ALLOC(pert2, pert_t, 1);
00063
00064     /* Read perturbation data... */
00065     read_pert(argv[2], pertname, pert);
00066
00067     /* Recalculate background and perturbations... */
00068     if (bg_poly_x > 0 || bg_poly_y > 0 ||
00069         bg_smooth_x > 0 || bg_smooth_y > 0 ||
00070         gauss_fwhm > 0 || ham_iter > 0 || med_dx > 0 || var_dh > 0) {
00071
00072         /* Convert to wave analysis struct... */
00073         pert2wave(pert, &wave, 0, pert->ntrack - 1, 0, pert->nxtrack - 1);
00074
00075         /* Estimate background... */
00076         background_poly(&wave, bg_poly_x, bg_poly_y);
00077         background_smooth(&wave, bg_smooth_x, bg_smooth_y);
00078
00079         /* Gaussian filter... */
00080         gauss(&wave, gauss_fwhm);
00081
00082         /* Hamming filter... */
00083         hamming(&wave, ham_iter);
00084

```

```

00085     /* Median filter... */
00086     median(&wave, med_dx);
00087
00088     /* Compute variance... */
00089     variance(&wave, var_dh);
00090
00091     /* Copy data... */
00092     for (ix = 0; ix < wave.nx; ix++)
00093         for (iy = 0; iy < wave.ny; iy++) {
00094             pert->pt[iy][ix] = wave.pt[ix][iy];
00095             pert->var[iy][ix] = wave.var[ix][iy];
00096         }
00097 }
00098
00099 /* Fill data gaps... */
00100 if (fill)
00101     for (itrack = 0; itrack < pert->ntrack; itrack++)
00102         for (ixtrack = 0; ixtrack < pert->nxtrack; ixtrack++) {
00103             if (!gsl_finite(pert->dc[itrack][ixtrack]))
00104                 pert->dc[itrack][ixtrack]
00105                     = fill_array(pert->dc, pert->ntrack, itrack, ixtrack);
00106             if (!gsl_finite(pert->bt[itrack][ixtrack]))
00107                 pert->bt[itrack][ixtrack]
00108                     = fill_array(pert->bt, pert->ntrack, itrack, ixtrack);
00109             if (!gsl_finite(pert->pt[itrack][ixtrack]))
00110                 pert->pt[itrack][ixtrack]
00111                     = fill_array(pert->pt, pert->ntrack, itrack, ixtrack);
00112             if (!gsl_finite(pert->var[itrack][ixtrack]))
00113                 pert->var[itrack][ixtrack]
00114                     = fill_array(pert->var, pert->ntrack, itrack, ixtrack);
00115         }
00116
00117 /* Interpolate to fine grid... */
00118 memcpy(pert2, pert, sizeof(pert_t));
00119
00120 /* Create output file... */
00121 printf("Write perturbation data: %s\n", argv[3]);
00122 if (!(out = fopen(argv[3], "w")))
00123     ERRMSG("Cannot create file!");
00124
00125 /* Write header... */
00126 fprintf(out,
00127     "# $1 = time (seconds since 01-JAN-2000, 00:00 UTC)\n"
00128     "# $2 = along-track index\n"
00129     "# $3 = longitude [deg]\n"
00130     "# $4 = latitude [deg]\n"
00131     "# $5 = 8mu brightness temperature [K]\n"
00132     "# $6 = %s brightness temperature [K]\n"
00133     "# $7 = %s brightness temperature perturbation [K]\n"
00134     "# $8 = %s brightness temperature variance [K^2]\n",
00135     pertname, pertname);
00136
00137 /* Write data... */
00138 for (itrack = 0; itrack < pert->ntrack; itrack++) {
00139
00140     /* Count orbits... */
00141     if (itrack > 0)
00142         if (pert->lat[itrack - 1][pert->nxtrack / 2] <= orblat
00143             && pert->lat[itrack][pert->nxtrack / 2] >= orblat)
00144             orb++;
00145
00146     /* Write output... */
00147     fprintf(out, "\n");
00148
00149     /* Check for data gaps... */
00150     if (itrack > 0 && pert->time[itrack][pert->nxtrack / 2]
00151         - pert->time[itrack - 1][pert->nxtrack / 2] >= 10)
00152         fprintf(out, "\n");
00153
00154     /* Loop over scan... */
00155     for (ixtrack = 0; ixtrack < pert->nxtrack; ixtrack++) {
00156
00157         /* Check data... */
00158         if (pert->lon[itrack][ixtrack] < -180
00159             || pert->lon[itrack][ixtrack] > 180
00160             || pert->lat[itrack][ixtrack] < -90
00161             || pert->lat[itrack][ixtrack] > 90)
00162             continue;
00163
00164         /* Get ascending/descending flag... */
00165         asc = (pert->lat[itrack > 0 ? itrack : itrack + 1][pert->nxtrack / 2]
00166             > pert->lat[itrack >
00167                 0 ? itrack - 1 : itrack][pert->nxtrack / 2]);
00168
00169         /* Calculate solar zenith angle... */
00170         if (sza0 >= -1e10 && sza0 <= 1e10 && sza1 >= -1e10 && sza1 <= 1e10)
00171             sza2 = sza(pert->time[itrack][ixtrack], pert->lon[itrack][ixtrack],

```



```

00172         pert->lat[itrack][ixtrack]);
00173
00174     /* Estimate noise... */
00175     if (dt230 > 0) {
00176         nesr = planck(t230 + dt230, nu) - planck(t230, nu);
00177         tbg = pert->bt[itrack][ixtrack] - pert->pt[itrack][ixtrack];
00178         nedt = brightness(planck(tbg, nu) + nesr, nu) - tbg;
00179     }
00180
00181     /* Write data... */
00182     if (orbit < 0 || orb == orbit)
00183         if (set[0] == 'f' || (set[0] == 'a' && asc)
00184             || (set[0] == 'd' && !asc))
00185             if (pert->time[itrack][ixtrack] >= t0
00186                 && pert->time[itrack][ixtrack] <= t1
00187                 && sza2 >= sza0 && sza2 <= sza1)
00188                 fprintf(out, "%.2f %d %g %g %g %g %g %g\n",
00189                     pert->time[itrack][ixtrack], itrack,
00190                     pert->lon[itrack][ixtrack], pert->lat[itrack][ixtrack],
00191                     pert->dc[itrack][ixtrack], pert->bt[itrack][ixtrack],
00192                     pert->pt[itrack][ixtrack],
00193                     pert->var[itrack][ixtrack] - gsl_pow_2(nedt));
00194     }
00195 }
00196
00197 /* Close file... */
00198 fclose(out);
00199
00200 /* Free... */
00201 free(pert);
00202 free(pert2);
00203
00204 return EXIT_SUCCESS;
00205 }
00206
00207 /*****
00208
00209 double fill_array(
00210     double var[PERT_NTRACK][PERT_NXTRACK],
00211     int ntrack,
00212     int itrack,
00213     int ixtrack) {
00214
00215     double d1 = 0, d2 = 0, v1 = 0, v2 = 0;
00216
00217     int i;
00218
00219     /* Find nearest neighbours... */
00220     for (i = itrack + 1; i < ntrack; i++)
00221         if (gsl_finite(var[i][ixtrack])) {
00222             d1 = fabs(i - itrack);
00223             v1 = var[i][ixtrack];
00224             break;
00225         }
00226     for (i = itrack - 1; i >= 0; i--)
00227         if (gsl_finite(var[i][ixtrack])) {
00228             d2 = fabs(i - itrack);
00229             v2 = var[i][ixtrack];
00230             break;
00231         }
00232
00233     /* Interpolate... */
00234     if (d1 + d2 > 0)
00235         return (d2 * v1 + d1 * v2) / (d1 + d2);
00236     else
00237         return GSL_NAN;
00238 }

```

## 5.31 map\_rad.c File Reference

### Functions

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

#### 5.31.1 Function Documentation

##### 5.31.1.1 int main ( int argc, char \* argv[] )

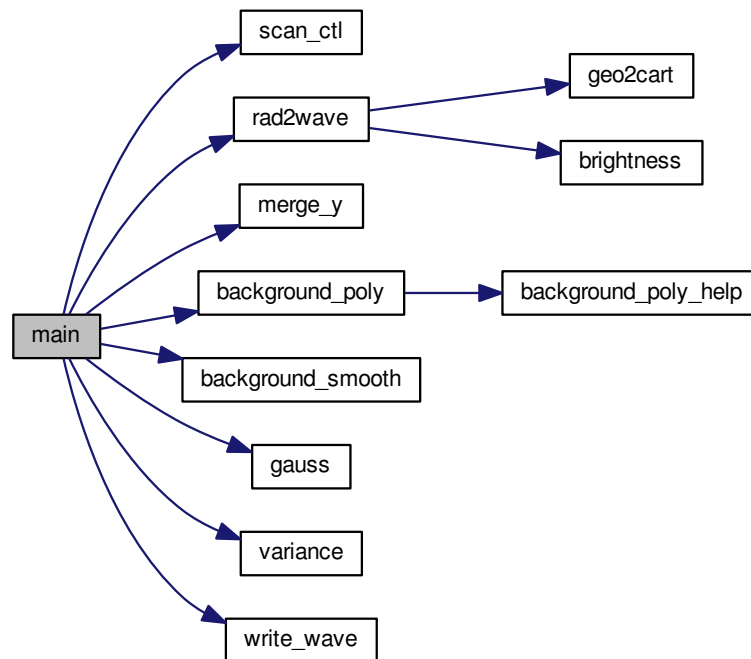
Definition at line 3 of file [map\\_rad.c](#).

```

00005         {
00006
00007     static airs_rad_gran_t airs_rad_gran;
00008     static wave_t wave, wave2;
00009
00010     double gauss_fwhm, nu, var_dh;
00011
00012     int bg_poly_x, bg_poly_y, bg_smooth_x, bg_smooth_y;
00013
00014     /* Check arguments... */
00015     if (argc < 6)
00016         ERRMSG("Give parameters: <ctl> <l1b_file1> <l1b_file2> <nu> <wave.tab>");
00017
00018     /* Get control parameters... */
00019     bg_poly_x = (int) scan_ctl(argc, argv, "BG_POLY_X", -1, "5", NULL);
00020     bg_poly_y = (int) scan_ctl(argc, argv, "BG_POLY_Y", -1, "0", NULL);
00021     bg_smooth_x = (int) scan_ctl(argc, argv, "BG_SMOOTH_X", -1, "0", NULL);
00022     bg_smooth_y = (int) scan_ctl(argc, argv, "BG_SMOOTH_Y", -1, "0", NULL);
00023     gauss_fwhm = scan_ctl(argc, argv, "GAUSS_FWHM", -1, "0", NULL);
00024     var_dh = scan_ctl(argc, argv, "VAR_DH", -1, "0", NULL);
00025
00026     /* Get channel.. */
00027     nu = atof(argv[4]);
00028
00029     /* Read AIRS data... */
00030     printf("Read AIRS Level-1B data file: %s\n", argv[2]);
00031     airs_rad_rdr(argv[2], &airs_rad_gran);
00032
00033     /* Convert radiance data to wave struct... */
00034     rad2wave(&airs_rad_gran, &nu, 1, &wave);
00035
00036     /* Check if second file is available... */
00037     if (argv[3][0] != '-') {
00038
00039         /* Read AIRS data... */
00040         printf("Read AIRS Level-1B data file: %s\n", argv[3]);
00041         airs_rad_rdr(argv[3], &airs_rad_gran);
00042
00043         /* Convert radiance data to wave struct... */
00044         rad2wave(&airs_rad_gran, &nu, 1, &wave2);
00045
00046         /* Merge with first file... */
00047         merge_y(&wave, &wave2);
00048     }
00049
00050     /* Compute background... */
00051     background_poly(&wave, bg_poly_x, bg_poly_y);
00052     background_smooth(&wave, bg_smooth_x, bg_smooth_y);
00053
00054     /* Gaussian filter... */
00055     gauss(&wave, gauss_fwhm);
00056
00057     /* Compute variance... */
00058     variance(&wave, var_dh);
00059
00060     /* Write files... */
00061     write_wave(argv[5], &wave);
00062
00063     return EXIT_SUCCESS;
00064 }

```

Here is the call graph for this function:



### 5.32 map\_rad.c

```

00001 #include "libairs.h"
00002
00003 int main(
00004     int argc,
00005     char *argv[]) {
00006
00007     static airs_rad_gran_t airs_rad_gran;
00008     static wave_t wave, wave2;
00009
00010     double gauss_fwhm, nu, var_dh;
00011
00012     int bg_poly_x, bg_poly_y, bg_smooth_x, bg_smooth_y;
00013
00014     /* Check arguments... */
00015     if (argc < 6)
00016         ERRMSG("Give parameters: <ctl> <l1b_file1> <l1b_file2> <nu> <wave.tab>");
00017
00018     /* Get control parameters... */
00019     bg_poly_x = (int) scan_ctl(argc, argv, "BG_POLY_X", -1, "5", NULL);
00020     bg_poly_y = (int) scan_ctl(argc, argv, "BG_POLY_Y", -1, "0", NULL);
00021     bg_smooth_x = (int) scan_ctl(argc, argv, "BG_SMOOTH_X", -1, "0", NULL);
00022     bg_smooth_y = (int) scan_ctl(argc, argv, "BG_SMOOTH_Y", -1, "0", NULL);
00023     gauss_fwhm = scan_ctl(argc, argv, "GAUSS_FWHM", -1, "0", NULL);
00024     var_dh = scan_ctl(argc, argv, "VAR_DH", -1, "0", NULL);
00025
00026     /* Get channel... */
00027     nu = atof(argv[4]);
00028
00029     /* Read AIRS data... */
00030     printf("Read AIRS Level-1B data file: %s\n", argv[2]);
00031     airs_rad_rdr(argv[2], &airs_rad_gran);
00032
00033     /* Convert radiance data to wave struct... */
00034     rad2wave(&airs_rad_gran, &nu, 1, &wave);
00035
00036     /* Check if second file is available... */

```

```

00037  if (argv[3][0] != '-') {
00038
00039      /* Read AIRS data... */
00040      printf("Read AIRS Level-1B data file: %s\n", argv[3]);
00041      airs_rad_rdr(argv[3], &airs_rad_gran);
00042
00043      /* Convert radiance data to wave struct... */
00044      rad2wave(&airs_rad_gran, &nu, 1, &wave2);
00045
00046      /* Merge with first file... */
00047      merge_y(&wave, &wave2);
00048  }
00049
00050      /* Compute background... */
00051      background_poly(&wave, bg_poly_x, bg_poly_y);
00052      background_smooth(&wave, bg_smooth_x, bg_smooth_y);
00053
00054      /* Gaussian filter... */
00055      gauss(&wave, gauss_fwhm);
00056
00057      /* Compute variance... */
00058      variance(&wave, var_dh);
00059
00060      /* Write files... */
00061      write_wave(argv[5], &wave);
00062
00063      return EXIT_SUCCESS;
00064 }

```

## 5.33 map\_ret.c File Reference

### Functions

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

#### 5.33.1 Function Documentation

##### 5.33.1.1 int main ( int argc, char \* argv[] )

Definition at line 3 of file [map\\_ret.c](#).

```

00005      {
00006
00007      static ret_t ret;
00008      static wave_t wave;
00009
00010      static double tbg[NDS], tabg[NDS], z0;
00011
00012      FILE *out;
00013
00014      char set[LEN];
00015
00016      int asc, bg_poly_x, bg_poly_y, bg_smooth_x, bg_smooth_y, ids, ip, ix, iy;
00017
00018      /* Check arguments... */
00019      if (argc < 4)
00020          ERRMSG("Give parameters: <ctl> <airs.nc> <map.tab>");
00021
00022      /* Get control parameters... */
00023      scan_ctl(argc, argv, "SET", -1, "full", set);
00024      z0 = scan_ctl(argc, argv, "Z0", -1, "", NULL);
00025      bg_poly_x = (int) scan_ctl(argc, argv, "BG_POLY_X", -1, "5", NULL);
00026      bg_poly_y = (int) scan_ctl(argc, argv, "BG_POLY_Y", -1, "0", NULL);
00027      bg_smooth_x = (int) scan_ctl(argc, argv, "BG_SMOOTH_X", -1, "0", NULL);
00028      bg_smooth_y = (int) scan_ctl(argc, argv, "BG_SMOOTH_Y", -1, "0", NULL);
00029
00030      /* Read AIRS data... */
00031      read_retr(argv[2], &ret);
00032
00033      /* Get altitude index... */
00034      for (ip = 0; ip <= ret.np; ip++) {
00035          if (ip == ret.np)
00036              ERRMSG("Altitude level not found!");
00037          if (fabs(ret.z[0][ip] - z0) < 0.1)

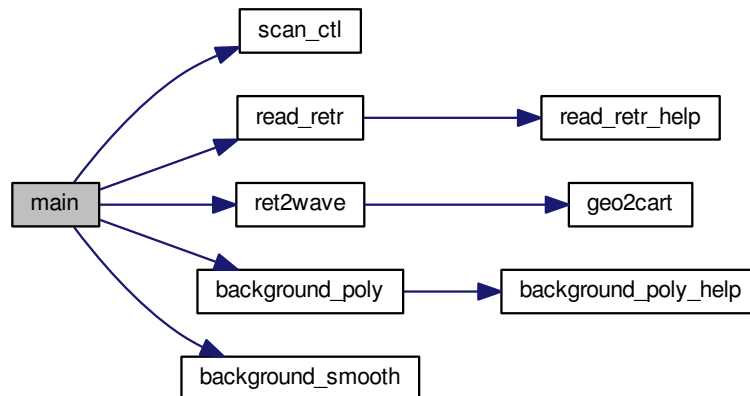
```

```

00038         break;
00039     }
00040
00041     /* Compute background... */
00042     ret2wave(&ret, &wave, 1, ip);
00043     background_poly(&wave, bg_poly_x, bg_poly_y);
00044     background_smooth(&wave, bg_smooth_x, bg_smooth_y);
00045     for (ix = 0; ix < wave.nx; ix++)
00046         for (iy = 0; iy < wave.ny; iy++)
00047             tbg[iy * 90 + ix] = wave.bg[ix][iy];
00048     ret2wave(&ret, &wave, 2, ip);
00049     background_poly(&wave, bg_poly_x, bg_poly_y);
00050     background_smooth(&wave, bg_smooth_x, bg_smooth_y);
00051     for (ix = 0; ix < wave.nx; ix++)
00052         for (iy = 0; iy < wave.ny; iy++)
00053             tabg[iy * 90 + ix] = wave.bg[ix][iy];
00054
00055     /* Create output file... */
00056     printf("Write AIRS map data: %s\n", argv[3]);
00057     if (!(out = fopen(argv[3], "w")))
00058         ERRMSG("Cannot create file!");
00059
00060     /* Write header... */
00061     fprintf(out,
00062         "## $1 = time (seconds since 01-JAN-2000, 00:00 UTC)\n"
00063         "## $2 = altitude [km]\n"
00064         "## $3 = longitude [deg]\n"
00065         "## $4 = latitude [deg]\n"
00066         "## $5 = pressure [hPa]\n"
00067         "## $6 = temperature (retrieved) [K]\n"
00068         "## $7 = temperature (retrieved) perturbation [K]\n"
00069         "## $8 = temperature (a priori) [K]\n"
00070         "## $9 = temperature (a priori) perturbation [K]\n");
00071     fprintf(out,
00072         "## $10 = temperature (total error) [K]\n"
00073         "## $11 = temperature (noise error) [K]\n"
00074         "## $12 = temperature (forward model error) [K]\n"
00075         "## $13 = temperature (measurement content)\n"
00076         "## $14 = temperature (resolution)\n" "## $15 = normalized chi^2\n");
00077
00078     /* Write data... */
00079     for (ids = 0; ids < ret.nds; ids++) {
00080
00081         /* Write new line... */
00082         if (ids % 90 == 0)
00083             fprintf(out, "\n");
00084
00085         /* Check data... */
00086         if (ret.lon[ids][ip] < -180 || ret.lon[ids][ip] > 180
00087             || ret.lat[ids][ip] < -90 || ret.lat[ids][ip] > 90
00088             || ret.t[ids][ip] < 100 || ret.t[ids][ip] > 400)
00089             continue;
00090
00091         /* Get ascending/descending flag... */
00092         asc = (ret.lat[ids] > 90 ? ids : ids + 90)[0]
00093             > ret.lat[ids] > 90 ? ids - 90 : ids[0];
00094
00095         /* Write data... */
00096         if (set[0] == 'f' || (set[0] == 'a' && asc) || (set[0] == 'd' && !asc))
00097             fprintf(out, "%.2f %g %g %g %g %g %g %g %g %g %g %g %g\n",
00098                 ret.time[ids][ip], ret.z[ids][ip],
00099                 ret.lon[ids][ip], ret.lat[ids][ip],
00100                 ret.p[ids][ip], ret.t[ids][ip], ret.t[ids][ip] - tbg[ids],
00101                 ret.t_apr[ids][ip], ret.t_apr[ids][ip] - tabg[ids],
00102                 ret.t_tot[ids][ip], ret.t_noise[ids][ip], ret.t_fm[ids][ip],
00103                 ret.t_cont[ids][ip], ret.t_res[ids][ip], ret.chisq[ids]);
00104     }
00105
00106     /* Close file... */
00107     fclose(out);
00108
00109     return EXIT_SUCCESS;
00110 }

```

Here is the call graph for this function:



## 5.34 map\_ret.c

```

00001 #include "libairs.h"
00002
00003 int main(
00004     int argc,
00005     char *argv[]) {
00006
00007     static ret_t ret;
00008     static wave_t wave;
00009
00010     static double tbg[NDS], tabg[NDS], z0;
00011
00012     FILE *out;
00013
00014     char set[LEN];
00015
00016     int asc, bg_poly_x, bg_poly_y, bg_smooth_x, bg_smooth_y, ids, ip, ix, iy;
00017
00018     /* Check arguments... */
00019     if (argc < 4)
00020         ERRMSG("Give parameters: <ctl> <airs.nc> <map.tab>");
00021
00022     /* Get control parameters... */
00023     scan_ctl(argc, argv, "SET", -1, "full", set);
00024     z0 = scan_ctl(argc, argv, "Z0", -1, "", NULL);
00025     bg_poly_x = (int) scan_ctl(argc, argv, "BG_POLY_X", -1, "5", NULL);
00026     bg_poly_y = (int) scan_ctl(argc, argv, "BG_POLY_Y", -1, "0", NULL);
00027     bg_smooth_x = (int) scan_ctl(argc, argv, "BG_SMOOTH_X", -1, "0", NULL);
00028     bg_smooth_y = (int) scan_ctl(argc, argv, "BG_SMOOTH_Y", -1, "0", NULL);
00029
00030     /* Read AIRS data... */
00031     read_retr(argv[2], &ret);
00032
00033     /* Get altitude index... */
00034     for (ip = 0; ip <= ret.np; ip++) {
00035         if (ip == ret.np)
00036             ERRMSG("Altitude level not found!");
00037         if (fabs(ret.z[0][ip] - z0) < 0.1)
00038             break;
00039     }
00040
00041     /* Compute background... */
00042     ret2wave(&ret, &wave, 1, ip);
00043     background_poly(&wave, bg_poly_x, bg_poly_y);
00044     background_smooth(&wave, bg_smooth_x, bg_smooth_y);
00045     for (ix = 0; ix < wave.nx; ix++)
00046         for (iy = 0; iy < wave.ny; iy++)
00047             tbg[iy * 90 + ix] = wave.bg[ix][iy];
00048     ret2wave(&ret, &wave, 2, ip);
00049     background_poly(&wave, bg_poly_x, bg_poly_y);

```

```

00050 background_smooth(&wave, bg_smooth_x, bg_smooth_y);
00051 for (ix = 0; ix < wave.nx; ix++)
00052     for (iy = 0; iy < wave.ny; iy++)
00053         tabg[iy * 90 + ix] = wave.bg[ix][iy];
00054
00055 /* Create output file... */
00056 printf("Write AIRS map data: %s\n", argv[3]);
00057 if (!(out = fopen(argv[3], "w")))
00058     ERRMSG("Cannot create file!");
00059
00060 /* Write header... */
00061 fprintf(out,
00062     "# $1 = time (seconds since 01-JAN-2000, 00:00 UTC)\n"
00063     "# $2 = altitude [km]\n"
00064     "# $3 = longitude [deg]\n"
00065     "# $4 = latitude [deg]\n"
00066     "# $5 = pressure [hPa]\n"
00067     "# $6 = temperature (retrieved) [K]\n"
00068     "# $7 = temperature (retrieved) perturbation [K]\n"
00069     "# $8 = temperature (a priori) [K]\n"
00070     "# $9 = temperature (a priori) perturbation [K]\n");
00071 fprintf(out,
00072     "# $10 = temperature (total error) [K]\n"
00073     "# $11 = temperature (noise error) [K]\n"
00074     "# $12 = temperature (forward model error) [K]\n"
00075     "# $13 = temperature (measurement content)\n"
00076     "# $14 = temperature (resolution)\n" "# $15 = normalized chi^2\n");
00077
00078 /* Write data... */
00079 for (ids = 0; ids < ret.nds; ids++) {
00080
00081     /* Write new line... */
00082     if (ids % 90 == 0)
00083         fprintf(out, "\n");
00084
00085     /* Check data... */
00086     if (ret.lon[ids][ip] < -180 || ret.lon[ids][ip] > 180
00087         || ret.lat[ids][ip] < -90 || ret.lat[ids][ip] > 90
00088         || ret.t[ids][ip] < 100 || ret.t[ids][ip] > 400)
00089         continue;
00090
00091     /* Get ascending/descending flag... */
00092     asc = (ret.lat[ids > 90 ? ids : ids + 90][0]
00093         > ret.lat[ids > 90 ? ids - 90 : ids][0]);
00094
00095     /* Write data... */
00096     if (set[0] == 'f' || (set[0] == 'a' && asc) || (set[0] == 'd' && !asc))
00097         fprintf(out, "%.2f %g %g %g %g %g %g %g %g %g %g %g %g\n",
00098             ret.time[ids][ip], ret.z[ids][ip],
00099             ret.lon[ids][ip], ret.lat[ids][ip],
00100             ret.p[ids][ip], ret.t[ids][ip], ret.t[ids][ip] - tbg[ids],
00101             ret.t_apr[ids][ip], ret.t_apr[ids][ip] - tabg[ids],
00102             ret.t_tot[ids][ip], ret.t_noise[ids][ip], ret.t_fm[ids][ip],
00103             ret.t_cont[ids][ip], ret.t_res[ids][ip], ret.chisq[ids]);
00104 }
00105
00106 /* Close file... */
00107 fclose(out);
00108
00109 return EXIT_SUCCESS;
00110 }

```

## 5.35 noise\_pert.c File Reference

### Functions

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

#### 5.35.1 Function Documentation

##### 5.35.1.1 int main ( int argc, char \* argv[ ] )

Definition at line 3 of file [noise\\_pert.c](#).

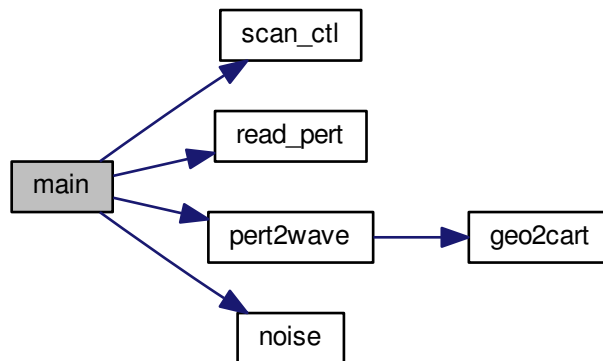
```

00005         {
00006
00007     static pert_t *pert;
00008     static wave_t wave;
00009
00010     FILE *out;
00011
00012     char pertname[LEN];
00013
00014     double maxvar, mu, nedt = -1e99, nedt_old;
00015
00016     int bsize, itrack;
00017
00018     /* Check arguments... */
00019     if (argc < 4)
00020         ERRMSG("Give parameters: <ctl> <pert.nc> <noise.tab>");
00021
00022     /* Read control parameters... */
00023     scan_ctl(argc, argv, "PERTNAME", -1, "4mu", pertname);
00024     bsize = (int) scan_ctl(argc, argv, "BSIZE", -1, "-999", NULL);
00025     maxvar = (int) scan_ctl(argc, argv, "MAXVAR", -1, "-999", NULL);
00026
00027     /* Allocate... */
00028     ALLOC(pert, pert_t, 1);
00029
00030     /* Read perturbation data... */
00031     read_pert(argv[2], pertname, pert);
00032
00033     /* Set block size... */
00034     if (bsize < 0)
00035         bsize = pert->ntrack;
00036
00037     /* Create file... */
00038     printf("Write noise data: %s\n", argv[3]);
00039     if (!(out = fopen(argv[3], "w")))
00040         ERRMSG("Cannot create file!");
00041
00042     /* Write header... */
00043     fprintf(out,
00044         "# $1 = longitude [deg]\n"
00045         "# $2 = latitude [deg]\n"
00046         "# $3 = mean brightness temperature [K]\n"
00047         "# $4 = noise estimate [K]\n\n");
00048
00049     /* Loop over granules... */
00050     for (itrack = 0; itrack < pert->ntrack; itrack += bsize) {
00051
00052         /* Convert retrieval data to wave struct... */
00053         pert2wave(pert, &wave, itrack, itrack + bsize,
00054             pert->ntrack / 2 - bsize / 2, pert->ntrack / 2 + bsize / 2);
00055
00056         /* Estimate noise... */
00057         nedt_old = nedt;
00058         noise(&wave, &mu, &nedt);
00059
00060         /* Write output... */
00061         if (maxvar <= 0
00062             || fabs(200 * (nedt - nedt_old) / (nedt + nedt_old)) < maxvar)
00063             fprintf(out, "%g %g %g %g\n", wave.lon[wave.nx / 2][wave.ny / 2],
00064                 wave.lat[wave.nx / 2][wave.ny / 2], mu, nedt);
00065     }
00066
00067     /* Close file... */
00068     fclose(out);
00069
00070     /* Free... */
00071     free(pert);
00072
00073     return EXIT_SUCCESS;
00074 }

```



Here is the call graph for this function:



### 5.36 noise\_pert.c

```

00001 #include "libairs.h"
00002
00003 int main(
00004     int argc,
00005     char *argv[]) {
00006
00007     static pert_t *pert;
00008     static wave_t wave;
00009
00010     FILE *out;
00011
00012     char pertname[LEN];
00013
00014     double maxvar, mu, nedt = -1e99, nedt_old;
00015
00016     int bsize, itrack;
00017
00018     /* Check arguments... */
00019     if (argc < 4)
00020         ERRMSG("Give parameters: <ctl> <pert.nc> <noise.tab>");
00021
00022     /* Read control parameters... */
00023     scan_ctl(argc, argv, "PERTNAME", -1, "4mu", pertname);
00024     bsize = (int) scan_ctl(argc, argv, "BSIZE", -1, "-999", NULL);
00025     maxvar = (int) scan_ctl(argc, argv, "MAXVAR", -1, "-999", NULL);
00026
00027     /* Allocate... */
00028     ALLOC(pert, pert_t, 1);
00029
00030     /* Read perturbation data... */
00031     read_pert(argv[2], pertname, pert);
00032
00033     /* Set block size... */
00034     if (bsize < 0)
00035         bsize = pert->ntrack;
00036
00037     /* Create file... */
00038     printf("Write noise data: %s\n", argv[3]);
00039     if (!(out = fopen(argv[3], "w")))
00040         ERRMSG("Cannot create file!");
00041
00042     /* Write header... */
00043     fprintf(out,
00044         "# $1 = longitude [deg]\n"
00045         "# $2 = latitude [deg]\n"
00046         "# $3 = mean brightness temperature [K]\n"
00047         "# $4 = noise estimate [K]\n\n");
00048
00049     /* Loop over granules... */

```

```

00050     for (itrack = 0; itrack < pert->ntrack; itrack += bsize) {
00051
00052         /* Convert retrieval data to wave struct... */
00053         pert2wave(pert, &wave, itrack, itrack + bsize,
00054                 pert->ntrack / 2 - bsize / 2, pert->ntrack / 2 + bsize / 2);
00055
00056         /* Estimate noise... */
00057         nedt_old = nedt;
00058         noise(&wave, &mu, &nedt);
00059
00060         /* Write output... */
00061         if (maxvar <= 0
00062             || fabs(200 * (nedt - nedt_old) / (nedt + nedt_old)) < maxvar)
00063             fprintf(out, "%g %g %g %g\n", wave.lon[wave.nx / 2][wave.ny / 2],
00064                     wave.lat[wave.nx / 2][wave.ny / 2], mu, nedt);
00065     }
00066
00067     /* Close file... */
00068     fclose(out);
00069
00070     /* Free... */
00071     free(pert);
00072
00073     return EXIT_SUCCESS;
00074 }

```

## 5.37 noise\_ret.c File Reference

### Functions

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

### 5.37.1 Function Documentation

#### 5.37.1.1 int main ( int argc, char \* argv[] )

Definition at line 3 of file [noise\\_ret.c](#).

```

00005     {
00006
00007     static ret_t ret;
00008     static wave_t wave, wave2;
00009
00010     FILE *out;
00011
00012     double mu, mu2, nedt, nedt2;
00013
00014     int ip;
00015
00016     /* Check arguments... */
00017     if (argc < 4)
00018         ERRMSG("Give parameters: <ctl> <airs.nc> <noise.tab>");
00019
00020     /* Read AIRS data... */
00021     read_retr(argv[2], &ret);
00022
00023     /* Create file... */
00024     printf("Write noise data: %s\n", argv[3]);
00025     if (!(out = fopen(argv[3], "w")))
00026         ERRMSG("Cannot create file!");
00027
00028     /* Write header... */
00029     fprintf(out,
00030             "# $1 = altitude [km]\n"
00031             "# $2 = longitude [deg]\n"
00032             "# $3 = latitude [deg]\n"
00033             "# $4 = mean temperature (retrieval) [K]\n"
00034             "# $5 = noise estimate (retrieval) [K]\n"
00035             "# $6 = mean temperature (a priori) [K]\n"
00036             "# $7 = noise estimate (a priori) [K]\n\n");
00037
00038     /* Loop over altitudes... */
00039     for (ip = 0; ip < ret.np; ip++) {
00040

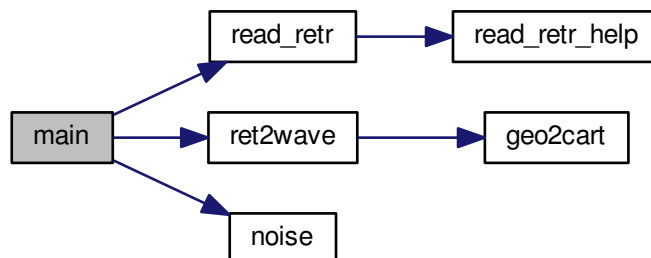
```

```

00041      /* Convert retrieval data to wave struct... */
00042      ret2wave(&ret, &wave, 1, ip);
00043      ret2wave(&ret, &wave2, 2, ip);
00044
00045      /* Estimate noise... */
00046      noise(&wave, &mu, &nedt);
00047      noise(&wave2, &mu2, &nedt2);
00048
00049      /* Estimate noise... */
00050      fprintf(out, "%g %g %g %g %g %g %g\n",
00051              wave.z,
00052              wave.lon[wave.nx / 2][wave.ny / 2],
00053              wave.lat[wave.nx / 2][wave.ny / 2], mu, nedt, mu2, nedt2);
00054  }
00055
00056  /* Close file... */
00057  fclose(out);
00058
00059  return EXIT_SUCCESS;
00060 }

```

Here is the call graph for this function:



### 5.38 noise\_ret.c

```

00001 #include "libairs.h"
00002
00003 int main(
00004     int argc,
00005     char *argv[]) {
00006
00007     static ret_t ret;
00008     static wave_t wave, wave2;
00009
00010     FILE *out;
00011
00012     double mu, mu2, nedt, nedt2;
00013
00014     int ip;
00015
00016     /* Check arguments... */
00017     if (argc < 4)
00018         ERRMSG("Give parameters: <ctl> <airs.nc> <noise.tab>");
00019
00020     /* Read AIRS data... */
00021     read_retr(argv[2], &ret);
00022
00023     /* Create file... */
00024     printf("Write noise data: %s\n", argv[3]);
00025     if (!(out = fopen(argv[3], "w")))
00026         ERRMSG("Cannot create file!");
00027
00028     /* Write header... */
00029     fprintf(out,
00030             "# $1 = altitude [km]\n"
00031             "# $2 = longitude [deg]\n"

```

```

00032         "# $3 = latitude [deg]\n"
00033         "# $4 = mean temperature (retrieval) [K]\n"
00034         "# $5 = noise estimate (retrieval) [K]\n"
00035         "# $6 = mean temperature (a priori) [K]\n"
00036         "# $7 = noise estimate (a priori) [K]\n\n");
00037
00038     /* Loop over altitudes... */
00039     for (ip = 0; ip < ret.np; ip++) {
00040
00041         /* Convert retrieval data to wave struct... */
00042         ret2wave(&ret, &wave, 1, ip);
00043         ret2wave(&ret, &wave2, 2, ip);
00044
00045         /* Estimate noise... */
00046         noise(&wave, &mu, &nedt);
00047         noise(&wave2, &mu2, &nedt2);
00048
00049         /* Estimate noise... */
00050         fprintf(out, "%g %g %g %g %g %g %g\n",
00051             wave.z,
00052             wave.lon[wave.nx / 2][wave.ny / 2],
00053             wave.lat[wave.nx / 2][wave.ny / 2], mu, nedt, mu2, nedt2);
00054     }
00055
00056     /* Close file... */
00057     fclose(out);
00058
00059     return EXIT_SUCCESS;
00060 }

```

## 5.39 optimize\_btd.c File Reference

### Functions

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

#### 5.39.1 Function Documentation

##### 5.39.1.1 int main ( int argc, char \* argv[] )

Definition at line 7 of file [optimize\\_btd.c](#).

```

00009         {
00010
00011         static airs_rad_gran_t airs_rad_gran;
00012
00013         static FILE *out;
00014
00015         static double bt[AIRS_RAD_CHANNEL], bt2, dbt, lat0, lat1, lon0, lon1,
00016             mean[AIRS_RAD_CHANNEL][AIRS_RAD_CHANNEL],
00017             max[AIRS_RAD_CHANNEL][AIRS_RAD_CHANNEL],
00018             var[AIRS_RAD_CHANNEL][AIRS_RAD_CHANNEL];
00019
00020         static int bg_chan0, bg_chan1, sig_chan0, sig_chan1, iarg, iavg, ichan,
00021             ichan2, n[AIRS_RAD_CHANNEL][AIRS_RAD_CHANNEL], navg, track, xtrack;
00022
00023         /* Check arguments... */
00024         if (argc < 12)
00025             ERRMSG("Give parameters: <opt.tab> <sig_chan0> <sig_chan1>"
00026                 " <bg_chan0> <bg_chan1> <lon0> <lon1> <lat0> <lat1> <navg>"
00027                 " <l1b_file1> [<l1b_file2> ...]");
00028
00029         /* Get parameters... */
00030         sig_chan0 = GSL_MIN(GSL_MAX(atoi(argv[2]), 0), AIRS_RAD_CHANNEL - 1);
00031         sig_chan1 = GSL_MIN(GSL_MAX(atoi(argv[3]), 0), AIRS_RAD_CHANNEL - 1);
00032         bg_chan0 = GSL_MIN(GSL_MAX(atoi(argv[4]), 0), AIRS_RAD_CHANNEL - 1);
00033         bg_chan1 = GSL_MIN(GSL_MAX(atoi(argv[5]), 0), AIRS_RAD_CHANNEL - 1);
00034         lon0 = atof(argv[6]);
00035         lon1 = atof(argv[7]);
00036         lat0 = atof(argv[8]);
00037         lat1 = atof(argv[9]);
00038         navg = atoi(argv[10]);
00039
00040         /* Loop over HDF files... */

```

```

00041 for (iarg = 1; iarg < argc; iarg++) {
00042
00043     /* Read AIRS data... */
00044     printf("Read AIRS Level-1B data file: %s\n", argv[iarg]);
00045     airs_rad_rdr(argv[iarg], &airs_rad_gran);
00046
00047     /* Loop over footprints... */
00048     for (track = 0; track < AIRS_RAD_GEOTRACK; track++)
00049         for (xtrack = 0; xtrack < AIRS_RAD_GEOXTRACK; xtrack++)
00050             if (airs_rad_gran.Longitude[track][xtrack] >= lon0 &&
00051                 airs_rad_gran.Longitude[track][xtrack] <= lon1 &&
00052                 airs_rad_gran.Latitude[track][xtrack] >= lat0 &&
00053                 airs_rad_gran.Latitude[track][xtrack] <= lat1) {
00054
00055         /* Get brightness temperature... */
00056         for (ichan = 0; ichan < AIRS_RAD_CHANNEL; ichan++)
00057             if ((airs_rad_gran.state[track][xtrack] != 0)
00058                 || (airs_rad_gran.ExcludedChans[ichan] > 2)
00059                 || (airs_rad_gran.CalChanSummary[ichan] & 8)
00060                 || (airs_rad_gran.CalChanSummary[ichan] & (32 + 64))
00061                 || (airs_rad_gran.CalFlag[track][ichan] & 16))
00062             bt[ichan] = GSL_NAN;
00063         else
00064             bt[ichan]
00065             = brightness(airs_rad_gran.radiances[track][xtrack][ichan]
00066                         * 0.001, airs_rad_gran.nominal_freq[ichan]);
00067
00068     /* Average channels... */
00069     for (ichan = 0; ichan < AIRS_RAD_CHANNEL - navg; ichan++) {
00070         bt2 = 0;
00071         for (iavg = 0; iavg < navg; iavg++)
00072             bt2 += bt[ichan + iavg];
00073         bt[ichan] = bt2 / navg;
00074     }
00075
00076     /* Get statistics... */
00077     for (ichan = sig_chan0; ichan <= sig_chan1; ichan++)
00078         for (ichan2 = bg_chan0; ichan2 <= bg_chan1; ichan2++)
00079             if (gsl_finite(bt[ichan]) && gsl_finite(bt[ichan2])) {
00080
00081                 /* Get brightness temperature difference... */
00082                 dbt = (bt[ichan2] - bt[ichan]);
00083                 if (fabs(dbt) > 100)
00084                     continue;
00085
00086                 /* Check filter... */
00087                 if (n[ichan][ichan2] <= 0)
00088                     max[ichan][ichan2] = dbt;
00089                 else
00090                     max[ichan][ichan2] = GSL_MAX(max[ichan][ichan2], dbt);
00091                 mean[ichan][ichan2] += dbt;
00092                 var[ichan][ichan2] += gsl_pow_2(dbt);
00093                 n[ichan][ichan2]++;
00094             }
00095     }
00096 }
00097
00098 /* Normalize... */
00099 for (ichan = sig_chan0; ichan <= sig_chan1; ichan++)
00100     for (ichan2 = bg_chan0; ichan2 <= bg_chan1; ichan2++) {
00101         if (n[ichan][ichan2] > 0) {
00102             mean[ichan][ichan2] /= n[ichan][ichan2];
00103             var[ichan][ichan2] = sqrt(var[ichan][ichan2] / n[ichan][ichan2]
00104                                     - gsl_pow_2(mean[ichan][ichan2]));
00105         } else
00106             mean[ichan][ichan2] = var[ichan][ichan2] = max[ichan][ichan2] =
00107                 GSL_NAN;
00108     }
00109
00110 /* Write info... */
00111 printf("Write optimization data: %s\n", argv[1]);
00112
00113 /* Create file... */
00114 if (!(out = fopen(argv[1], "w")))
00115     ERRMSG("Cannot create file!");
00116
00117 /* Write header... */
00118 fprintf(out,
00119         "# $1 = signal channel\n"
00120         "# $2 = signal wavenumber [cm^-1]\n"
00121         "# $3 = background channel\n"
00122         "# $4 = background wavenumber [cm^-1]\n"
00123         "# $5 = BTD(bg-sig) mean [K]\n"
00124         "# $6 = BTD(bg-sig) standard deviation [K]\n"
00125         "# $7 = BTD(bg-sig) maximum [K]\n"
00126         "# $8 = effective SNR (= max/RMS)\n"
00127         "# $9 = number of footprints\n");

```

```

00128
00129  /* Write info... */
00130  for (ichan = sig_chan0; ichan <= sig_chan1; ichan++) {
00131      fprintf(out, "\n");
00132      for (ichan2 = bg_chan0; ichan2 <= bg_chan1; ichan2++)
00133          fprintf(out, "%d %.3f %d %.3f %g %g %g %g %d\n",
00134                  ichan, airs_rad_gran.nominal_freq[ichan],
00135                  ichan2, airs_rad_gran.nominal_freq[ichan2],
00136                  mean[ichan][ichan2], var[ichan][ichan2], max[ichan][ichan2],
00137                  max[ichan][ichan2] / sqrt(gsl_pow_2(var[ichan][ichan2])
00138                                          + gsl_pow_2(mean[ichan][ichan2])),
00139                  n[ichan][ichan2]);
00140  }
00141
00142  /* Close file... */
00143  fclose(out);
00144
00145  return EXIT_SUCCESS;
00146 }

```

Here is the call graph for this function:



## 5.40 optimize\_btd.c

```

00001 #include "libairs.h"
00002
00003 /* -----
00004  Main...
00005  ----- */
00006
00007 int main(
00008     int argc,
00009     char *argv[]) {
00010
00011     static airs_rad_gran_t airs_rad_gran;
00012
00013     static FILE *out;
00014
00015     static double bt[AIRS_RAD_CHANNEL], bt2, dbt, lat0, lat1, lon0, lon1,
00016                 mean[AIRS_RAD_CHANNEL][AIRS_RAD_CHANNEL],
00017                 max[AIRS_RAD_CHANNEL][AIRS_RAD_CHANNEL],
00018                 var[AIRS_RAD_CHANNEL][AIRS_RAD_CHANNEL];
00019
00020     static int bg_chan0, bg_chan1, sig_chan0, sig_chan1, iarg, iavg, ichan,
00021               ichan2, n[AIRS_RAD_CHANNEL][AIRS_RAD_CHANNEL], navg, track, xtrack;
00022
00023     /* Check arguments... */
00024     if (argc < 12)
00025         ERRMSG("Give parameters: <opt.tab> <sig_chan0> <sig_chan1>"
00026               " <bg_chan0> <bg_chan1> <lon0> <lon1> <lat0> <lat1> <navg>"
00027               " <lib_file1> [<lib_file2> ...]");
00028
00029     /* Get parameters... */
00030     sig_chan0 = GSL_MIN(GSL_MAX(atoi(argv[2]), 0), AIRS_RAD_CHANNEL - 1);
00031     sig_chan1 = GSL_MIN(GSL_MAX(atoi(argv[3]), 0), AIRS_RAD_CHANNEL - 1);
00032     bg_chan0 = GSL_MIN(GSL_MAX(atoi(argv[4]), 0), AIRS_RAD_CHANNEL - 1);
00033     bg_chan1 = GSL_MIN(GSL_MAX(atoi(argv[5]), 0), AIRS_RAD_CHANNEL - 1);
00034     lon0 = atof(argv[6]);
00035     lon1 = atof(argv[7]);
00036     lat0 = atof(argv[8]);
00037     lat1 = atof(argv[9]);
00038     navg = atoi(argv[10]);
00039
00040     /* Loop over HDF files... */
00041     for (iarg = 11; iarg < argc; iarg++) {

```

```

00042
00043 /* Read AIRS data... */
00044 printf("Read AIRS Level-1B data file: %s\n", argv[iarg]);
00045 airs_rad_rdr(argv[iarg], &airs_rad_gran);
00046
00047 /* Loop over footprints... */
00048 for (track = 0; track < AIRS_RAD_GEOTRACK; track++)
00049     for (xtrack = 0; xtrack < AIRS_RAD_GEOXTRACK; xtrack++)
00050         if (airs_rad_gran.Longitude[track][xtrack] >= lon0 &&
00051             airs_rad_gran.Longitude[track][xtrack] <= lon1 &&
00052             airs_rad_gran.Latitude[track][xtrack] >= lat0 &&
00053             airs_rad_gran.Latitude[track][xtrack] <= lat1) {
00054
00055             /* Get brightness temperature... */
00056             for (ichan = 0; ichan < AIRS_RAD_CHANNEL; ichan++)
00057                 if ((airs_rad_gran.state[track][xtrack] != 0)
00058                     || (airs_rad_gran.ExcludedChans[ichan] > 2)
00059                     || (airs_rad_gran.CalChanSummary[ichan] & 8)
00060                     || (airs_rad_gran.CalChanSummary[ichan] & (32 + 64))
00061                     || (airs_rad_gran.CalFlag[track][ichan] & 16))
00062                 bt[ichan] = GSL_NAN;
00063             else
00064                 bt[ichan]
00065                     = brightness(airs_rad_gran.radiances[track][xtrack][ichan]
00066                                 * 0.001, airs_rad_gran.nominal_freq[ichan]);
00067
00068             /* Average channels... */
00069             for (ichan = 0; ichan < AIRS_RAD_CHANNEL - navg; ichan++) {
00070                 bt2 = 0;
00071                 for (iavg = 0; iavg < navg; iavg++)
00072                     bt2 += bt[ichan + iavg];
00073                 bt[ichan] = bt2 / navg;
00074             }
00075
00076             /* Get statistics... */
00077             for (ichan = sig_chan0; ichan <= sig_chan1; ichan++)
00078                 for (ichan2 = bg_chan0; ichan2 <= bg_chan1; ichan2++)
00079                     if (gsl_finite(bt[ichan]) && gsl_finite(bt[ichan2])) {
00080
00081                         /* Get brightness temperature difference... */
00082                         dbt = (bt[ichan2] - bt[ichan]);
00083                         if (fabs(dbt) > 100)
00084                             continue;
00085
00086                         /* Check filter... */
00087                         if (n[ichan][ichan2] <= 0)
00088                             max[ichan][ichan2] = dbt;
00089                         else
00090                             max[ichan][ichan2] = GSL_MAX(max[ichan][ichan2], dbt);
00091                         mean[ichan][ichan2] += dbt;
00092                         var[ichan][ichan2] += gsl_pow_2(dbt);
00093                         n[ichan][ichan2]++;
00094                     }
00095             }
00096         }
00097
00098 /* Normalize... */
00099 for (ichan = sig_chan0; ichan <= sig_chan1; ichan++)
00100     for (ichan2 = bg_chan0; ichan2 <= bg_chan1; ichan2++) {
00101         if (n[ichan][ichan2] > 0) {
00102             mean[ichan][ichan2] /= n[ichan][ichan2];
00103             var[ichan][ichan2] = sqrt(var[ichan][ichan2] / n[ichan][ichan2]
00104                                     - gsl_pow_2(mean[ichan][ichan2]));
00105         } else
00106             mean[ichan][ichan2] = var[ichan][ichan2] = max[ichan][ichan2] =
00107                 GSL_NAN;
00108     }
00109
00110 /* Write info... */
00111 printf("Write optimization data: %s\n", argv[1]);
00112
00113 /* Create file... */
00114 if (!(out = fopen(argv[1], "w")))
00115     ERRMSG("Cannot create file!");
00116
00117 /* Write header... */
00118 fprintf(out,
00119     "# $1 = signal channel\n"
00120     "# $2 = signal wavenumber [cm^-1]\n"
00121     "# $3 = background channel\n"
00122     "# $4 = background wavenumber [cm^-1]\n"
00123     "# $5 = BTD(bg-sig) mean [K]\n"
00124     "# $6 = BTD(bg-sig) standard deviation [K]\n"
00125     "# $7 = BTD(bg-sig) maximum [K]\n"
00126     "# $8 = effective SNR (= max/RMS)\n"
00127     "# $9 = number of footprints\n");
00128

```

```

00129  /* Write info... */
00130  for (ichan = sig_chan0; ichan <= sig_chan1; ichan++) {
00131      fprintf(out, "\n");
00132      for (ichan2 = bg_chan0; ichan2 <= bg_chan1; ichan2++)
00133          fprintf(out, "%d %.3f %d %.3f %g %g %g %g %d\n",
00134                  ichan, airs_rad_gran.nominal_freq[ichan],
00135                  ichan2, airs_rad_gran.nominal_freq[ichan2],
00136                  mean[ichan][ichan2], var[ichan][ichan2], max[ichan][ichan2],
00137                  max[ichan][ichan2] / sqrt(gsl_pow_2(var[ichan][ichan2])
00138                                           + gsl_pow_2(mean[ichan][ichan2])),
00139                  n[ichan][ichan2]);
00140      }
00141  }
00142  /* Close file... */
00143  fclose(out);
00144
00145  return EXIT_SUCCESS;
00146 }

```

## 5.41 orbit.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 [orbit.c](#).

```

00005      {
00006
00007      static airs_rad_gran_t airs_rad_gran;
00008
00009      FILE *out;
00010
00011      int i, track, xtrack;
00012
00013      /* Check arguments... */
00014      if (argc < 3)
00015          ERRMSG
00016          ("Give parameters: <orbit.tab> <airs_llb_file> [ <airs_llb_file2> ... ]");
00017
00018      /* Create file... */
00019      printf("Write orbit data: %s\n", argv[1]);
00020      if (!(out = fopen(argv[1], "w")))
00021          ERRMSG("Cannot create file!");
00022
00023      /* Write header... */
00024      fprintf(out,
00025              "# $1 = time (seconds since 01-JAN-2000, 00:00 UTC)\n"
00026              "# $2 = satellite longitude [deg]\n"
00027              "# $3 = satellite latitude [deg]\n"
00028              "# $4 = footprint longitude [deg]\n"
00029              "# $5 = footprint latitude [deg]\n");
00030
00031      /* Loop over files... */
00032      for (i = 2; i < argc; i++) {
00033
00034          /* Read AIRS data... */
00035          printf("Read AIRS Level-1B data file: %s\n", argv[i]);
00036          airs_rad_rdr(argv[i], &airs_rad_gran);
00037
00038          /* Write data... */
00039          for (track = 0; track < AIRS_RAD_GEOTRACK; track++) {
00040              fprintf(out, "\n");
00041              for (xtrack = 0; xtrack < AIRS_RAD_GEOXTRACK; xtrack++)
00042                  fprintf(out, "%.2f %g %g %g %g\n",
00043                          airs_rad_gran.Time[track][xtrack] - 220838400,
00044                          airs_rad_gran.sat_lon[track],
00045                          airs_rad_gran.sat_lat[track],
00046                          airs_rad_gran.Longitude[track][xtrack],
00047                          airs_rad_gran.Latitude[track][xtrack]);

```



```

00048     }
00049 }
00050
00051 /* Close file... */
00052 fclose(out);
00053
00054 return EXIT_SUCCESS;
00055 }

```

## 5.42 orbit.c

```

00001 #include "libairs.h"
00002
00003 int main(
00004     int argc,
00005     char *argv[]) {
00006
00007     static airs_rad_gran_t airs_rad_gran;
00008
00009     FILE *out;
00010
00011     int i, track, xtrack;
00012
00013     /* Check arguments... */
00014     if (argc < 3)
00015         ERRMSG
00016             ("Give parameters: <orbit.tab> <airs_llb_file> [ <airs_llb_file2> ... ]");
00017
00018     /* Create file... */
00019     printf("Write orbit data: %s\n", argv[1]);
00020     if (!(out = fopen(argv[1], "w")))
00021         ERRMSG("Cannot create file!");
00022
00023     /* Write header... */
00024     fprintf(out,
00025         "# $1 = time (seconds since 01-JAN-2000, 00:00 UTC)\n"
00026         "# $2 = satellite longitude [deg]\n"
00027         "# $3 = satellite latitude [deg]\n"
00028         "# $4 = footprint longitude [deg]\n"
00029         "# $5 = footprint latitude [deg]\n");
00030
00031     /* Loop over files... */
00032     for (i = 2; i < argc; i++) {
00033
00034         /* Read AIRS data... */
00035         printf("Read AIRS Level-1B data file: %s\n", argv[i]);
00036         airs_rad_rdr(argv[i], &airs_rad_gran);
00037
00038         /* Write data... */
00039         for (track = 0; track < AIRS_RAD_GEOTRACK; track++) {
00040             fprintf(out, "\n");
00041             for (xtrack = 0; xtrack < AIRS_RAD_GEOXTRACK; xtrack++)
00042                 fprintf(out, "%.2f %g %g %g %g\n",
00043                     airs_rad_gran.Time[track][xtrack] - 220838400,
00044                     airs_rad_gran.sat_lon[track],
00045                     airs_rad_gran.sat_lat[track],
00046                     airs_rad_gran.Longitude[track][xtrack],
00047                     airs_rad_gran.Latitude[track][xtrack]);
00048             }
00049         }
00050
00051     /* Close file... */
00052     fclose(out);
00053
00054     return EXIT_SUCCESS;
00055 }

```

## 5.43 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[])

## 5.43.1 Function Documentation

## 5.43.1.1 void write\_results ( FILE \* out, pert\_t \* pert, int track0, int xtrack0, int orb, double dmin, double obsz )

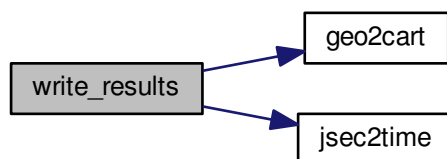
Definition at line 118 of file [overpass.c](#).

```

00125         {
00126
00127     double alpha, xf[3], xs[3], xsf[3], remain;
00128
00129     int asc, i, year, mon, day, hour, min, sec;
00130
00131     /* Calculate scan angle... */
00132     geo2cart(0, pert->lon[track0][xtrack0], pert->lat[track0][xtrack0], xf);
00133     geo2cart(0, pert->lon[track0][pert->nxtrack / 2],
00134             pert->lat[track0][pert->nxtrack / 2], xsf);
00135     geo2cart(obsz, pert->lon[track0][pert->nxtrack / 2],
00136             pert->lat[track0][pert->nxtrack / 2], xs);
00137     for (i = 0; i < 3; i++) {
00138         xf[i] -= xs[i];
00139         xsf[i] -= xs[i];
00140     }
00141     alpha = 180. / M_PI * acos(DOTP(xf, xsf) / NORM(xf) / NORM(xsf));
00142     if (xtrack0 < pert->nxtrack / 2)
00143         alpha = -alpha;
00144
00145     /* Get ascending/descending flag... */
00146     asc = (pert->lat[track0] > 0 ? track0 : track0 + 1)[pert->nxtrack / 2]
00147         > pert->lat[track0 > 0 ? track0 - 1 : track0][pert->nxtrack / 2]);
00148
00149     /* Write results... */
00150     jsec2time(pert->time[track0][xtrack0], &year, &mon, &day,
00151             &hour, &min, &sec, &remain);
00152     fprintf(out,
00153             "%.2f %d-%02d-%02dT%02d:%02dZ %g %g %d %d %d %d %g %g\n",
00154             pert->time[track0][xtrack0], year, mon, day, hour, min, sec,
00155             pert->lon[track0][xtrack0], pert->lat[track0][xtrack0],
00156             track0, xtrack0, orb, asc, alpha, sqrt(dmin));
00157 }

```

Here is the call graph for this function:



## 5.43.1.2 int main ( int argc, char \* argv[ ] )

Definition at line 21 of file [overpass.c](#).

```

00023         {
00024
00025     static pert_t *pert;
00026
00027     FILE *out;
00028
00029     char pertname[LEN];
00030

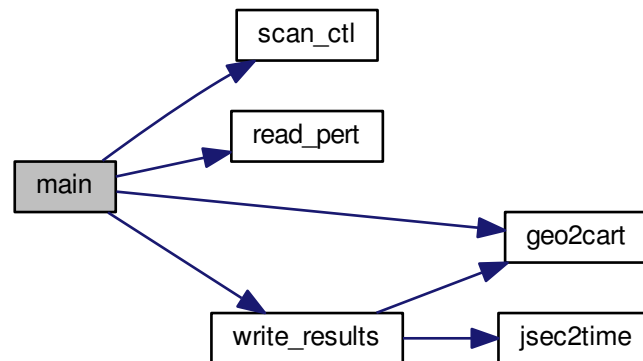
```

```

00031 double dmin = 1e100, lon0, lat0, orblat, rmax, obsz, x0[3], x1[3];
00032
00033 int orb = 0, track, track0 = 0, xtrack, xtrack0 = 0;
00034
00035 /* Check arguments... */
00036 if (argc < 6)
00037     ERRMSG("Give parameters: <ctl> <pert.nc> <lon0> <lat0> <overpass.tab>");
00038
00039 /* Get arguments... */
00040 lon0 = atof(argv[3]);
00041 lat0 = atof(argv[4]);
00042
00043 /* Get control parameters... */
00044 scan_ctl(argc, argv, "PERTNAME", -1, "4mu", pertname);
00045 orblat = scan_ctl(argc, argv, "ORBLAT", -1, "0", NULL);
00046 rmax = scan_ctl(argc, argv, "RMAX", -1, "100", NULL);
00047 obsz = scan_ctl(argc, argv, "OBSZ", -1, "", NULL);
00048
00049 /* Allocate... */
00050 ALLOC(pert, pert_t, 1);
00051
00052 /* Read perturbation data... */
00053 read_pert(argv[2], pertname, pert);
00054
00055 /* Get Cartesian coordinates... */
00056 geo2cart(0, lon0, lat0, x0);
00057
00058 /* Create file... */
00059 printf("Write overpass data file: %s\n", argv[5]);
00060 if (!(out = fopen(argv[5], "w")))
00061     ERRMSG("Cannot create file!");
00062
00063 /* Write header... */
00064 fprintf(out,
00065     "# $1 = time (seconds since 2000-01-01T00:00Z)\n"
00066     "# $2 = time (string)\n"
00067     "# $3 = longitude [deg]\n"
00068     "# $4 = latitude [deg]\n"
00069     "# $5 = along-track index\n"
00070     "# $6 = across-track index\n"
00071     "# $7 = orbit number\n"
00072     "# $8 = ascending (1=yes, 0=no)\n"
00073     "# $9 = scan angle [deg]\n" "# $10 = distance [km]\n\n");
00074
00075 /* Find nearest footprint... */
00076 for (track = 0; track < pert->ntrack; track++) {
00077
00078     /* Check for new orbit... */
00079     if (track > 0)
00080         if (pert->lat[track - 1][pert->nxtrack / 2] <= orblat
00081             && pert->lat[track][pert->nxtrack / 2] >= orblat) {
00082
00083         /* Write results... */
00084         if (sqrt(dmin) <= rmax)
00085             write_results(out, pert, track0, xtrack0, orb, dmin, obsz);
00086
00087         /* Set counters... */
00088         dmin = 1e100;
00089         orb++;
00090     }
00091
00092     /* Check distance of footprints... */
00093     for (xtrack = 0; xtrack < pert->nxtrack; xtrack++) {
00094         geo2cart(0, pert->lon[track][xtrack], pert->lat[track][xtrack], x1);
00095         if (DIST2(x0, x1) < dmin) {
00096             dmin = DIST2(x0, x1);
00097             track0 = track;
00098             xtrack0 = xtrack;
00099         }
00100     }
00101 }
00102
00103 /* Write results for last orbit... */
00104 if (sqrt(dmin) <= rmax)
00105     write_results(out, pert, track0, xtrack0, orb, dmin, obsz);
00106
00107 /* Close file... */
00108 fclose(out);
00109
00110 /* Free... */
00111 free(pert);
00112
00113 return EXIT_SUCCESS;
00114 }

```

Here is the call graph for this function:



## 5.44 overpass.c

```

00001 #include "libairs.h"
00002
00003 /* -----
00004  Functions...
00005 ----- */
00006
00007 /* Write results to file. */
00008 void write_results(
00009     FILE * out,
00010     pert_t * pert,
00011     int track0,
00012     int xtrack0,
00013     int orb,
00014     double dmin,
00015     double obsz);
00016
00017 /* -----
00018  Main...
00019 ----- */
00020
00021 int main(
00022     int argc,
00023     char *argv[]) {
00024
00025     static pert_t *pert;
00026
00027     FILE *out;
00028
00029     char pertname[LEN];
00030
00031     double dmin = 1e100, lon0, lat0, orblat, rmax, obsz, x0[3], x1[3];
00032
00033     int orb = 0, track, track0 = 0, xtrack, xtrack0 = 0;
00034
00035     /* Check arguments... */
00036     if (argc < 6)
00037         ERRMSG("Give parameters: <ctl> <pert.nc> <lon0> <lat0> <overpass.tab>");
00038
00039     /* Get arguments... */
00040     lon0 = atof(argv[3]);
00041     lat0 = atof(argv[4]);
00042
00043     /* Get control parameters... */
00044     scan_ctl(argc, argv, "PERTNAME", -1, "4mu", pertname);
00045     orblat = scan_ctl(argc, argv, "ORBLAT", -1, "0", NULL);
00046     rmax = scan_ctl(argc, argv, "RMAX", -1, "100", NULL);
00047     obsz = scan_ctl(argc, argv, "OBSZ", -1, "", NULL);
00048
00049     /* Allocate... */

```

```

00050  ALLOC(pert, pert_t, 1);
00051
00052  /* Read perturbation data... */
00053  read_pert(argv[2], pertname, pert);
00054
00055  /* Get Cartesian coordinates... */
00056  geo2cart(0, lon0, lat0, x0);
00057
00058  /* Create file... */
00059  printf("Write overpass data file: %s\n", argv[5]);
00060  if (!(out = fopen(argv[5], "w")))
00061      ERRMSG("Cannot create file!");
00062
00063  /* Write header... */
00064  fprintf(out,
00065          "# $1 = time (seconds since 2000-01-01T00:00Z)\n"
00066          "# $2 = time (string)\n"
00067          "# $3 = longitude [deg]\n"
00068          "# $4 = latitude [deg]\n"
00069          "# $5 = along-track index\n"
00070          "# $6 = across-track index\n"
00071          "# $7 = orbit number\n"
00072          "# $8 = ascending (1=yes, 0=no)\n"
00073          "# $9 = scan angle [deg]\n" "# $10 = distance [km]\n\n");
00074
00075  /* Find nearest footprint... */
00076  for (track = 0; track < pert->ntrack; track++) {
00077
00078      /* Check for new orbit... */
00079      if (track > 0)
00080          if (pert->lat[track - 1][pert->ntrack / 2] <= orblat
00081              && pert->lat[track][pert->ntrack / 2] >= orblat) {
00082
00083          /* Write results... */
00084          if (sqrt(dmin) <= rmax)
00085              write_results(out, pert, track0, xtrack0, orb, dmin, obsz);
00086
00087          /* Set counters... */
00088          dmin = 1e100;
00089          orb++;
00090      }
00091
00092      /* Check distance of footprints... */
00093      for (xtrack = 0; xtrack < pert->nxtrack; xtrack++) {
00094          geo2cart(0, pert->lon[track][xtrack], pert->lat[track][xtrack], x1);
00095          if (DIST2(x0, x1) < dmin) {
00096              dmin = DIST2(x0, x1);
00097              track0 = track;
00098              xtrack0 = xtrack;
00099          }
00100      }
00101  }
00102
00103  /* Write results for last orbit... */
00104  if (sqrt(dmin) <= rmax)
00105      write_results(out, pert, track0, xtrack0, orb, dmin, obsz);
00106
00107  /* Close file... */
00108  fclose(out);
00109
00110  /* Free... */
00111  free(pert);
00112
00113  return EXIT_SUCCESS;
00114 }
00115
00116 /*****
00117
00118 void write_results(
00119     FILE * out,
00120     pert_t * pert,
00121     int track0,
00122     int xtrack0,
00123     int orb,
00124     double dmin,
00125     double obsz) {
00126
00127     double alpha, xf[3], xs[3], xsf[3], remain;
00128
00129     int asc, i, year, mon, day, hour, min, sec;
00130
00131     /* Calculate scan angle... */
00132     geo2cart(0, pert->lon[track0][xtrack0], pert->lat[track0][xtrack0], xf);
00133     geo2cart(0, pert->lon[track0][pert->ntrack / 2],
00134             pert->lat[track0][pert->ntrack / 2], xsf);
00135     geo2cart(obsz, pert->lon[track0][pert->ntrack / 2],
00136             pert->lat[track0][pert->ntrack / 2], xs);

```

```

00137     for (i = 0; i < 3; i++) {
00138         xf[i] -= xs[i];
00139         xsf[i] -= xs[i];
00140     }
00141     alpha = 180. / M_PI * acos(DOTP(xf, xsf) / NORM(xf) / NORM(xsf));
00142     if (xtrack0 < pert->ntrack / 2)
00143         alpha = -alpha;
00144
00145     /* Get ascending/descending flag... */
00146     asc = (pert->lat[track0 > 0 ? track0 : track0 + 1][pert->ntrack / 2]
00147         > pert->lat[track0 > 0 ? track0 - 1 : track0][pert->ntrack / 2]);
00148
00149     /* Write results... */
00150     jsec2time(pert->time[track0][xtrack0], &year, &mon, &day,
00151         &hour, &min, &sec, &remain);
00152     fprintf(out,
00153         "%.2f %d-%02d-%02d%02d:%02d:%02dZ %g %g %d %d %d %d %g %g\n",
00154         pert->time[track0][xtrack0], year, mon, day, hour, min, sec,
00155         pert->lon[track0][xtrack0], pert->lat[track0][xtrack0],
00156         track0, xtrack0, orb, asc, alpha, sqrt(dmin));
00157 }

```

## 5.45 perturbation.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 20 of file [perturbation.c](#).

```

00022     {
00023
00024     static airs_rad_gran_t airs_rad_gran;
00025
00026     static pert_t *pert_4mu, *pert_15mu_low, *pert_15mu_high;
00027
00028     static wave_t wave;
00029
00030     static double var_dh = 100.;
00031
00032     static int list_4mu[N4]
00033     = { 2039, 2040, 2041, 2042, 2043, 2044, 2045, 2046, 2047, 2048,
00034         2049, 2050, 2051, 2052, 2053, 2054, 2055, 2056, 2057, 2058,
00035         2059, 2060, 2061, 2062, 2063, 2064, 2071, 2072, 2073, 2074,
00036         2075, 2076, 2077, 2078, 2079, 2080, 2081, 2082, 2083, 2084,
00037         2085, 2086
00038     };
00039
00040     static int list_15mu_low[N15_LOW]
00041     = { 4, 10, 16, 22, 29, 35, 41, 55, 83, 88, 94,
00042         100, 101, 106, 107, 112, 113, 118, 119, 124, 125
00043     };
00044
00045     static int list_15mu_high[N15_HIGH]
00046     = { 74, 75 };
00047
00048     static int ix, iy, dimid[2], i, n, ncid, track, track0, xtrack,
00049         time_varid, lon_varid, lat_varid, bt_4mu_varid, bt_4mu_pt_varid,
00050         bt_4mu_var_varid, bt_8mu_varid, bt_15mu_low_varid, bt_15mu_low_pt_varid,
00051         bt_15mu_low_var_varid, bt_15mu_high_varid, bt_15mu_high_pt_varid,
00052         bt_15mu_high_var_varid, iarg;
00053
00054     static size_t start[2], count[2];
00055
00056     /* Check arguments... */
00057     if (argc < 3)
00058         ERRMSG("Give parameters: <out.nc> <l1b_file1> [<l1b_file2> ...]");
00059
00060     /* Allocate... */
00061     ALLOC(pert_4mu, pert_t, 1);

```

```

00062  ALLOC(pert_15mu_low, pert_t, 1);
00063  ALLOC(pert_15mu_high, pert_t, 1);
00064
00065  /* -----
00066  Read HDF files...
00067  ----- */
00068
00069  /* Loop over HDF files... */
00070  for (iarg = 2; iarg < argc; iarg++) {
00071
00072      /* Read AIRS data... */
00073      printf("Read AIRS Level-1B data file: %s\n", argv[iarg]);
00074      airs_rad_rdr(argv[iarg], &airs_rad_gran);
00075
00076      /* Flag bad observations... */
00077      for (track = 0; track < AIRS_RAD_GEOTRACK; track++)
00078          for (xtrack = 0; xtrack < AIRS_RAD_GEOXTRACK; xtrack++)
00079              for (i = 0; i < AIRS_RAD_CHANNEL; i++)
00080                  if ((airs_rad_gran.state[track][xtrack] != 0)
00081                      || (airs_rad_gran.ExcludedChans[i] > 2)
00082                      || (airs_rad_gran.CalChanSummary[i] & 8)
00083                      || (airs_rad_gran.CalChanSummary[i] & (32 + 64))
00084                      || (airs_rad_gran.CalFlag[track][i] & 16)
00085                      || (airs_rad_gran.Longitude[track][xtrack] < -180)
00086                      || (airs_rad_gran.Longitude[track][xtrack] > 180)
00087                      || (airs_rad_gran.Latitude[track][xtrack] < -90)
00088                      || (airs_rad_gran.Latitude[track][xtrack] > 90))
00089                      airs_rad_gran.radiances[track][xtrack][i] = GSL_NAN;
00090                  else
00091                      airs_rad_gran.radiances[track][xtrack][i] *= 0.001f;
00092
00093      /* Save geolocation... */
00094      pert_4mu->ntrack += AIRS_RAD_GEOTRACK;
00095      if (pert_4mu->ntrack > PERT_NTRACK)
00096          ERRMSG("Too many granules!");
00097      pert_4mu->nxtrack = AIRS_RAD_GEOXTRACK;
00098      if (pert_4mu->nxtrack > PERT_NXTRACK)
00099          ERRMSG("Too many tracks!");
00100      for (track = 0; track < AIRS_RAD_GEOTRACK; track++)
00101          for (xtrack = 0; xtrack < AIRS_RAD_GEOXTRACK; xtrack++) {
00102              pert_4mu->time[track0 + track][xtrack]
00103                  = airs_rad_gran.Time[track][xtrack] - 220838400.;
00104              pert_4mu->lon[track0 + track][xtrack]
00105                  = airs_rad_gran.Longitude[track][xtrack];
00106              pert_4mu->lat[track0 + track][xtrack]
00107                  = airs_rad_gran.Latitude[track][xtrack];
00108          }
00109
00110      pert_15mu_low->ntrack += AIRS_RAD_GEOTRACK;
00111      if (pert_15mu_low->ntrack > PERT_NTRACK)
00112          ERRMSG("Too many granules!");
00113      pert_15mu_low->nxtrack = AIRS_RAD_GEOXTRACK;
00114      if (pert_15mu_low->nxtrack > PERT_NXTRACK)
00115          ERRMSG("Too many tracks!");
00116      for (track = 0; track < AIRS_RAD_GEOTRACK; track++)
00117          for (xtrack = 0; xtrack < AIRS_RAD_GEOXTRACK; xtrack++) {
00118              pert_15mu_low->time[track0 + track][xtrack]
00119                  = airs_rad_gran.Time[track][xtrack] - 220838400.;
00120              pert_15mu_low->lon[track0 + track][xtrack]
00121                  = airs_rad_gran.Longitude[track][xtrack];
00122              pert_15mu_low->lat[track0 + track][xtrack]
00123                  = airs_rad_gran.Latitude[track][xtrack];
00124          }
00125
00126      pert_15mu_high->ntrack += AIRS_RAD_GEOTRACK;
00127      if (pert_15mu_high->ntrack > PERT_NTRACK)
00128          ERRMSG("Too many granules!");
00129      pert_15mu_high->nxtrack = AIRS_RAD_GEOXTRACK;
00130      if (pert_15mu_high->nxtrack > PERT_NXTRACK)
00131          ERRMSG("Too many tracks!");
00132      for (track = 0; track < AIRS_RAD_GEOTRACK; track++)
00133          for (xtrack = 0; xtrack < AIRS_RAD_GEOXTRACK; xtrack++) {
00134              pert_15mu_high->time[track0 + track][xtrack]
00135                  = airs_rad_gran.Time[track][xtrack] - 220838400.;
00136              pert_15mu_high->lon[track0 + track][xtrack]
00137                  = airs_rad_gran.Longitude[track][xtrack];
00138              pert_15mu_high->lat[track0 + track][xtrack]
00139                  = airs_rad_gran.Latitude[track][xtrack];
00140          }
00141
00142      /* Get 8.1 micron brightness temperature... */
00143      for (track = 0; track < AIRS_RAD_GEOTRACK; track++)
00144          for (xtrack = 0; xtrack < AIRS_RAD_GEOXTRACK; xtrack++)
00145              pert_4mu->dc[track0 + track][xtrack]
00146                  = brightness(airs_rad_gran.radiances[track][xtrack][1290],
00147                              airs_rad_gran.nominal_freq[1290]);
00148

```

```

00149      /* Get 4.3 micron brightness temperature... */
00150      for (track = 0; track < AIRS_RAD_GEOTRACK; track++)
00151          for (xtrack = 0; xtrack < AIRS_RAD_GEOXTRACK; xtrack++) {
00152              n = 0;
00153              for (i = 0; i < N4; i++)
00154                  if (gsl_finite(airs_rad_gran.radiances[track][xtrack][list_4mu[i]])) {
00155                      pert_4mu->bt[track0 + track][xtrack]
00156                          +=
00157                          brightness(airs_rad_gran.radiances[track][xtrack][list_4mu[i]],
00158                                      airs_rad_gran.nominal_freq[list_4mu[i]]);
00159                      n++;
00160                  }
00161              if (n > 0.9 * N4)
00162                  pert_4mu->bt[track0 + track][xtrack] /= n;
00163              else
00164                  pert_4mu->bt[track0 + track][xtrack] = GSL_NAN;
00165          }
00166
00167      /* Get 15 micron brightness temperature (low altitudes)... */
00168      for (track = 0; track < AIRS_RAD_GEOTRACK; track++)
00169          for (xtrack = 0; xtrack < AIRS_RAD_GEOXTRACK; xtrack++) {
00170              n = 0;
00171              for (i = 0; i < N15_LOW; i++)
00172                  if (gsl_finite(airs_rad_gran.radiances
00173                                [track][xtrack][list_15mu_low[i]])) {
00174                      pert_15mu_low->bt[track0 + track][xtrack]
00175                          += brightness(airs_rad_gran.radiances
00176                                      [track][xtrack][list_15mu_low[i]],
00177                                      airs_rad_gran.nominal_freq[list_15mu_low[i]]);
00178                      n++;
00179                  }
00180              if (n > 0.9 * N15_LOW)
00181                  pert_15mu_low->bt[track0 + track][xtrack] /= n;
00182              else
00183                  pert_15mu_low->bt[track0 + track][xtrack] = GSL_NAN;
00184          }
00185
00186      /* Get 15 micron brightness temperature (high altitudes)... */
00187      for (track = 0; track < AIRS_RAD_GEOTRACK; track++)
00188          for (xtrack = 0; xtrack < AIRS_RAD_GEOXTRACK; xtrack++) {
00189              n = 0;
00190              for (i = 0; i < N15_HIGH; i++)
00191                  if (gsl_finite(airs_rad_gran.radiances
00192                                [track][xtrack][list_15mu_high[i]])) {
00193                      pert_15mu_high->bt[track0 + track][xtrack]
00194                          += brightness(airs_rad_gran.radiances
00195                                      [track][xtrack][list_15mu_high[i]],
00196                                      airs_rad_gran.nominal_freq[list_15mu_high[i]]);
00197                      n++;
00198                  }
00199              if (n > 0.9 * N15_HIGH)
00200                  pert_15mu_high->bt[track0 + track][xtrack] /= n;
00201              else
00202                  pert_15mu_high->bt[track0 + track][xtrack] = GSL_NAN;
00203          }
00204
00205      /* Increment track counter... */
00206      track0 += AIRS_RAD_GEOTRACK;
00207  }
00208
00209  /* -----
00210     Calculate perturbations and variances...
00211     ----- */
00212
00213  /* Convert to wave analysis struct... */
00214  pert2wave(pert_4mu, &wave,
00215            0, pert_4mu->ntrack - 1, 0, pert_4mu->nxtrack - 1);
00216
00217  /* Estimate background... */
00218  background_poly(&wave, 5, 0);
00219
00220  /* Compute variance... */
00221  variance(&wave, var_dh);
00222
00223  /* Copy data... */
00224  for (ix = 0; ix < wave.nx; ix++)
00225      for (iy = 0; iy < wave.ny; iy++) {
00226          pert_4mu->pt[iy][ix] = wave.pt[ix][iy];
00227          pert_4mu->var[iy][ix] = wave.var[ix][iy];
00228      }
00229
00230  /* Convert to wave analysis struct... */
00231  pert2wave(pert_15mu_low, &wave,
00232            0, pert_15mu_low->ntrack - 1, 0, pert_15mu_low->nxtrack - 1);
00233
00234  /* Estimate background... */
00235  background_poly(&wave, 5, 0);

```



```

00236
00237 /* Compute variance... */
00238 variance(&wave, var_dh);
00239
00240 /* Copy data... */
00241 for (ix = 0; ix < wave.nx; ix++)
00242     for (iy = 0; iy < wave.ny; iy++) {
00243         pert_15mu_low->pt[iy][ix] = wave.pt[ix][iy];
00244         pert_15mu_low->var[iy][ix] = wave.var[ix][iy];
00245     }
00246
00247 /* Convert to wave analysis struct... */
00248 pert2wave(pert_15mu_high, &wave,
00249           0, pert_15mu_high->ntrack - 1, 0, pert_15mu_high->ntrack - 1);
00250
00251 /* Estimate background... */
00252 background_poly(&wave, 5, 0);
00253
00254 /* Compute variance... */
00255 variance(&wave, var_dh);
00256
00257 /* Copy data... */
00258 for (ix = 0; ix < wave.nx; ix++)
00259     for (iy = 0; iy < wave.ny; iy++) {
00260         pert_15mu_high->pt[iy][ix] = wave.pt[ix][iy];
00261         pert_15mu_high->var[iy][ix] = wave.var[ix][iy];
00262     }
00263
00264 /* -----
00265    Write to netCDF file...
00266    ----- */
00267
00268 /* Create netCDF file... */
00269 NC(nc_create(argv[1], NC_CLOBBER, &ncid));
00270
00271 /* Set dimensions... */
00272 NC(nc_def_dim(ncid, "NTRACK", NC_UNLIMITED, &dimid[0]));
00273 NC(nc_def_dim(ncid, "NXTRACK", AIRS_RAD_GEOXTRACK, &dimid[1]));
00274
00275 /* Add variables... */
00276 NC(nc_def_var(ncid, "time", NC_DOUBLE, 2, dimid, &time_varid));
00277 add_att(ncid, time_varid, "s", "time (seconds since 2000-01-01T00:00Z)");
00278 NC(nc_def_var(ncid, "lon", NC_DOUBLE, 2, dimid, &lon_varid));
00279 add_att(ncid, lon_varid, "deg", "footprint longitude");
00280 NC(nc_def_var(ncid, "lat", NC_DOUBLE, 2, dimid, &lat_varid));
00281 add_att(ncid, lat_varid, "deg", "footprint latitude");
00282
00283 NC(nc_def_var(ncid, "bt_8mu", NC_FLOAT, 2, dimid, &bt_8mu_varid));
00284 add_att(ncid, bt_8mu_varid, "K", "brightness temperature at 8.1 micron");
00285
00286 NC(nc_def_var(ncid, "bt_4mu", NC_FLOAT, 2, dimid, &bt_4mu_varid));
00287 add_att(ncid, bt_4mu_varid, "K", "brightness temperature " " at 4.3 micron");
00288 NC(nc_def_var(ncid, "bt_4mu_pt", NC_FLOAT, 2, dimid, &bt_4mu_pt_varid));
00289 add_att(ncid, bt_4mu_pt_varid, "K", "brightness temperature perturbation"
00290        " at 4.3 micron");
00291 NC(nc_def_var(ncid, "bt_4mu_var", NC_FLOAT, 2, dimid, &bt_4mu_var_varid));
00292 add_att(ncid, bt_4mu_var_varid, "K^2", "brightness temperature variance"
00293        " at 4.3 micron");
00294
00295 NC(nc_def_var(ncid, "bt_15mu_low", NC_FLOAT, 2, dimid, &bt_15mu_low_varid));
00296 add_att(ncid, bt_15mu_low_varid, "K", "brightness temperature"
00297        " at 15 micron (low altitudes)");
00298 NC(nc_def_var(ncid, "bt_15mu_low_pt", NC_FLOAT, 2, dimid,
00299        &bt_15mu_low_pt_varid));
00300 add_att(ncid, bt_15mu_low_pt_varid, "K",
00301        "brightness temperature perturbation"
00302        " at 15 micron (low altitudes)");
00303 NC(nc_def_var
00304        (ncid, "bt_15mu_low_var", NC_FLOAT, 2, dimid, &bt_15mu_low_var_varid));
00305 add_att(ncid, bt_15mu_low_var_varid, "K^2",
00306        "brightness temperature variance" " at 15 micron (low altitudes)");
00307
00308 NC(nc_def_var(ncid, "bt_15mu_high", NC_FLOAT, 2, dimid,
00309        &bt_15mu_high_varid));
00310 add_att(ncid, bt_15mu_high_varid, "K", "brightness temperature"
00311        " at 15 micron (high altitudes)");
00312 NC(nc_def_var(ncid, "bt_15mu_high_pt", NC_FLOAT, 2, dimid,
00313        &bt_15mu_high_pt_varid));
00314 add_att(ncid, bt_15mu_high_pt_varid, "K",
00315        "brightness temperature perturbation"
00316        " at 15 micron (high altitudes)");
00317 NC(nc_def_var
00318        (ncid, "bt_15mu_high_var", NC_FLOAT, 2, dimid, &bt_15mu_high_var_varid));
00319 add_att(ncid, bt_15mu_high_var_varid, "K^2",
00320        "brightness temperature variance" " at 15 micron (high altitudes)");
00321
00322 /* Leave define mode... */

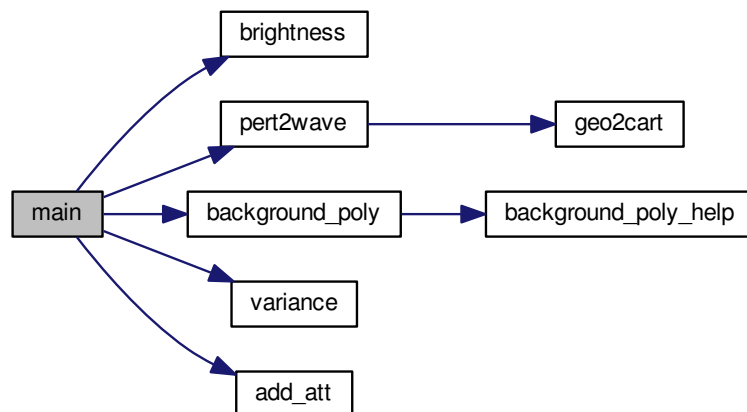
```

```

00323 NC(nc_enddef(ncid));
00324
00325 /* Loop over tracks... */
00326 for (track = 0; track < pert_4mu->ntrack; track++) {
00327
00328     /* Set array sizes... */
00329     start[0] = (size_t) track;
00330     start[1] = 0;
00331     count[0] = 1;
00332     count[1] = (size_t) pert_4mu->ntrack;
00333
00334     /* Write data... */
00335     NC(nc_put_vara_double(ncid, time_varid, start, count,
00336                          pert_4mu->time[track]));
00337     NC(nc_put_vara_double(ncid, lon_varid, start, count,
00338                          pert_4mu->lon[track]));
00339     NC(nc_put_vara_double(ncid, lat_varid, start, count,
00340                          pert_4mu->lat[track]));
00341
00342     NC(nc_put_vara_double(ncid, bt_8mu_varid, start, count,
00343                          pert_4mu->dc[track]));
00344
00345     NC(nc_put_vara_double(ncid, bt_4mu_varid, start, count,
00346                          pert_4mu->bt[track]));
00347     NC(nc_put_vara_double(ncid, bt_4mu_pt_varid, start, count,
00348                          pert_4mu->pt[track]));
00349     NC(nc_put_vara_double(ncid, bt_4mu_var_varid, start, count,
00350                          pert_4mu->var[track]));
00351
00352     NC(nc_put_vara_double(ncid, bt_15mu_low_varid, start, count,
00353                          pert_15mu_low->bt[track]));
00354     NC(nc_put_vara_double(ncid, bt_15mu_low_pt_varid, start, count,
00355                          pert_15mu_low->pt[track]));
00356     NC(nc_put_vara_double(ncid, bt_15mu_low_var_varid, start, count,
00357                          pert_15mu_low->var[track]));
00358
00359     NC(nc_put_vara_double(ncid, bt_15mu_high_varid, start, count,
00360                          pert_15mu_high->bt[track]));
00361     NC(nc_put_vara_double(ncid, bt_15mu_high_pt_varid, start, count,
00362                          pert_15mu_high->pt[track]));
00363     NC(nc_put_vara_double(ncid, bt_15mu_high_var_varid, start, count,
00364                          pert_15mu_high->var[track]));
00365 }
00366
00367 /* Close file... */
00368 NC(nc_close(ncid));
00369
00370 /* Free... */
00371 free(pert_4mu);
00372 free(pert_15mu_low);
00373 free(pert_15mu_high);
00374
00375 return EXIT_SUCCESS;
00376 }

```

Here is the call graph for this function:



## 5.46 perturbation.c

```

00001 #include "libairs.h"
00002
00003 /* -----
00004     Constants...
00005     ----- */
00006
00007 /* Number of 4 micron channels: */
00008 #define N4 42
00009
00010 /* Number of 15 micron channels (low altitudes): */
00011 #define N15_LOW 21
00012
00013 /* Number of 15 micron channels (high altitudes): */
00014 #define N15_HIGH 2
00015
00016 /* -----
00017     Main...
00018     ----- */
00019
00020 int main(
00021     int argc,
00022     char *argv[]) {
00023
00024     static airs_rad_gran_t airs_rad_gran;
00025
00026     static pert_t *pert_4mu, *pert_15mu_low, *pert_15mu_high;
00027
00028     static wave_t wave;
00029
00030     static double var_dh = 100.;
00031
00032     static int list_4mu[N4]
00033         = { 2039, 2040, 2041, 2042, 2043, 2044, 2045, 2046, 2047, 2048,
00034            2049, 2050, 2051, 2052, 2053, 2054, 2055, 2056, 2057, 2058,
00035            2059, 2060, 2061, 2062, 2063, 2064, 2071, 2072, 2073, 2074,
00036            2075, 2076, 2077, 2078, 2079, 2080, 2081, 2082, 2083, 2084,
00037            2085, 2086
00038     };
00039
00040     static int list_15mu_low[N15_LOW]
00041         = { 4, 10, 16, 22, 29, 35, 41, 55, 83, 88, 94,
00042            100, 101, 106, 107, 112, 113, 118, 119, 124, 125
00043     };
00044
00045     static int list_15mu_high[N15_HIGH]
00046         = { 74, 75 };
00047
00048     static int ix, iy, dimid[2], i, n, ncid, track, track0, xtrack,
00049         time_varid, lon_varid, lat_varid, bt_4mu_varid, bt_4mu_pt_varid,
00050         bt_4mu_var_varid, bt_8mu_varid, bt_15mu_low_varid, bt_15mu_low_pt_varid,
00051         bt_15mu_low_var_varid, bt_15mu_high_varid, bt_15mu_high_pt_varid,
00052         bt_15mu_high_var_varid, iarg;
00053
00054     static size_t start[2], count[2];
00055
00056     /* Check arguments... */
00057     if (argc < 3)
00058         ERRMSG("Give parameters: <out.nc> <l1b_file1> [<l1b_file2> ...]");
00059
00060     /* Allocate... */
00061     ALLOC(pert_4mu, pert_t, 1);
00062     ALLOC(pert_15mu_low, pert_t, 1);
00063     ALLOC(pert_15mu_high, pert_t, 1);
00064
00065     /* -----
00066         Read HDF files...
00067         ----- */
00068
00069     /* Loop over HDF files... */
00070     for (iarg = 2; iarg < argc; iarg++) {
00071
00072         /* Read AIRS data... */
00073         printf("Read AIRS Level-1B data file: %s\n", argv[iarg]);
00074         airs_rad_rdr(argv[iarg], &airs_rad_gran);
00075
00076         /* Flag bad observations... */
00077         for (track = 0; track < AIRS_RAD_GEOTRACK; track++)
00078             for (xtrack = 0; xtrack < AIRS_RAD_GEOXTRACK; xtrack++)
00079                 for (i = 0; i < AIRS_RAD_CHANNEL; i++)
00080                     if ((airs_rad_gran.state[track][xtrack] != 0)
00081                         || (airs_rad_gran.ExcludedChans[i] > 2)
00082                         || (airs_rad_gran.CalChanSummary[i] & 8)
00083                         || (airs_rad_gran.CalChanSummary[i] & (32 + 64))
00084                         || (airs_rad_gran.CalFlag[track][i] & 16)

```

```

00085         || (airs_rad_gran.Longitude[track][xtrack] < -180)
00086         || (airs_rad_gran.Longitude[track][xtrack] > 180)
00087         || (airs_rad_gran.Latitude[track][xtrack] < -90)
00088         || (airs_rad_gran.Latitude[track][xtrack] > 90))
00089         airs_rad_gran.radiances[track][xtrack][i] = GSL_NAN;
00090     else
00091         airs_rad_gran.radiances[track][xtrack][i] *= 0.001f;
00092
00093     /* Save geolocation... */
00094     pert_4mu->ntrack += AIRS_RAD_GEOTRACK;
00095     if (pert_4mu->ntrack > PERT_NTRACK)
00096         ERRMSG("Too many granules!");
00097     pert_4mu->nxtrack = AIRS_RAD_GEOXTRACK;
00098     if (pert_4mu->nxtrack > PERT_NXTRACK)
00099         ERRMSG("Too many tracks!");
00100     for (track = 0; track < AIRS_RAD_GEOTRACK; track++)
00101         for (xtrack = 0; xtrack < AIRS_RAD_GEOXTRACK; xtrack++) {
00102             pert_4mu->time[track0 + track][xtrack]
00103                 = airs_rad_gran.Time[track][xtrack] - 220838400.;
00104             pert_4mu->lon[track0 + track][xtrack]
00105                 = airs_rad_gran.Longitude[track][xtrack];
00106             pert_4mu->lat[track0 + track][xtrack]
00107                 = airs_rad_gran.Latitude[track][xtrack];
00108         }
00109
00110     pert_15mu_low->ntrack += AIRS_RAD_GEOTRACK;
00111     if (pert_15mu_low->ntrack > PERT_NTRACK)
00112         ERRMSG("Too many granules!");
00113     pert_15mu_low->nxtrack = AIRS_RAD_GEOXTRACK;
00114     if (pert_15mu_low->nxtrack > PERT_NXTRACK)
00115         ERRMSG("Too many tracks!");
00116     for (track = 0; track < AIRS_RAD_GEOTRACK; track++)
00117         for (xtrack = 0; xtrack < AIRS_RAD_GEOXTRACK; xtrack++) {
00118             pert_15mu_low->time[track0 + track][xtrack]
00119                 = airs_rad_gran.Time[track][xtrack] - 220838400.;
00120             pert_15mu_low->lon[track0 + track][xtrack]
00121                 = airs_rad_gran.Longitude[track][xtrack];
00122             pert_15mu_low->lat[track0 + track][xtrack]
00123                 = airs_rad_gran.Latitude[track][xtrack];
00124         }
00125
00126     pert_15mu_high->ntrack += AIRS_RAD_GEOTRACK;
00127     if (pert_15mu_high->ntrack > PERT_NTRACK)
00128         ERRMSG("Too many granules!");
00129     pert_15mu_high->nxtrack = AIRS_RAD_GEOXTRACK;
00130     if (pert_15mu_high->nxtrack > PERT_NXTRACK)
00131         ERRMSG("Too many tracks!");
00132     for (track = 0; track < AIRS_RAD_GEOTRACK; track++)
00133         for (xtrack = 0; xtrack < AIRS_RAD_GEOXTRACK; xtrack++) {
00134             pert_15mu_high->time[track0 + track][xtrack]
00135                 = airs_rad_gran.Time[track][xtrack] - 220838400.;
00136             pert_15mu_high->lon[track0 + track][xtrack]
00137                 = airs_rad_gran.Longitude[track][xtrack];
00138             pert_15mu_high->lat[track0 + track][xtrack]
00139                 = airs_rad_gran.Latitude[track][xtrack];
00140         }
00141
00142     /* Get 8.1 micron brightness temperature... */
00143     for (track = 0; track < AIRS_RAD_GEOTRACK; track++)
00144         for (xtrack = 0; xtrack < AIRS_RAD_GEOXTRACK; xtrack++)
00145             pert_4mu->dc[track0 + track][xtrack]
00146                 = brightness(airs_rad_gran.radiances[track][xtrack][1290],
00147                             airs_rad_gran.nominal_freq[1290]);
00148
00149     /* Get 4.3 micron brightness temperature... */
00150     for (track = 0; track < AIRS_RAD_GEOTRACK; track++)
00151         for (xtrack = 0; xtrack < AIRS_RAD_GEOXTRACK; xtrack++) {
00152             n = 0;
00153             for (i = 0; i < N4; i++)
00154                 if (gsl_finite(airs_rad_gran.radiances[track][xtrack][list_4mu[i]])) {
00155                     pert_4mu->bt[track0 + track][xtrack]
00156                         +=
00157                             brightness(airs_rad_gran.radiances[track][xtrack][list_4mu[i]],
00158                                         airs_rad_gran.nominal_freq[list_4mu[i]]);
00159                     n++;
00160                 }
00161             if (n > 0.9 * N4)
00162                 pert_4mu->bt[track0 + track][xtrack] /= n;
00163             else
00164                 pert_4mu->bt[track0 + track][xtrack] = GSL_NAN;
00165         }
00166
00167     /* Get 15 micron brightness temperature (low altitudes)... */
00168     for (track = 0; track < AIRS_RAD_GEOTRACK; track++)
00169         for (xtrack = 0; xtrack < AIRS_RAD_GEOXTRACK; xtrack++) {
00170             n = 0;
00171             for (i = 0; i < N15_LOW; i++)

```

```

00172         if (gsl_finite(airs_rad_gran.radiances
00173             [track][xtrack][list_15mu_low[i]])) {
00174             pert_15mu_low->bt[track0 + track][xtrack]
00175             += brightness(airs_rad_gran.radiances
00176                 [track][xtrack][list_15mu_low[i]],
00177                 airs_rad_gran.nominal_freq[list_15mu_low[i]]);
00178             n++;
00179         }
00180         if (n > 0.9 * N15_LOW)
00181             pert_15mu_low->bt[track0 + track][xtrack] /= n;
00182         else
00183             pert_15mu_low->bt[track0 + track][xtrack] = GSL_NAN;
00184     }
00185
00186     /* Get 15 micron brightness temperature (high altitudes)... */
00187     for (track = 0; track < AIRS_RAD_GEOTRACK; track++)
00188         for (xtrack = 0; xtrack < AIRS_RAD_GEOXTRACK; xtrack++) {
00189             n = 0;
00190             for (i = 0; i < N15_HIGH; i++)
00191                 if (gsl_finite(airs_rad_gran.radiances
00192                     [track][xtrack][list_15mu_high[i]])) {
00193                     pert_15mu_high->bt[track0 + track][xtrack]
00194                     += brightness(airs_rad_gran.radiances
00195                         [track][xtrack][list_15mu_high[i]],
00196                         airs_rad_gran.nominal_freq[list_15mu_high[i]]);
00197                     n++;
00198                 }
00199             if (n > 0.9 * N15_HIGH)
00200                 pert_15mu_high->bt[track0 + track][xtrack] /= n;
00201             else
00202                 pert_15mu_high->bt[track0 + track][xtrack] = GSL_NAN;
00203         }
00204
00205     /* Increment track counter... */
00206     track0 += AIRS_RAD_GEOTRACK;
00207 }
00208
00209 /* -----
00210 Calculate perturbations and variances...
00211 ----- */
00212
00213 /* Convert to wave analysis struct... */
00214 pert2wave(pert_4mu, &wave,
00215     0, pert_4mu->ntrack - 1, 0, pert_4mu->nxtrack - 1);
00216
00217 /* Estimate background... */
00218 background_poly(&wave, 5, 0);
00219
00220 /* Compute variance... */
00221 variance(&wave, var_dh);
00222
00223 /* Copy data... */
00224 for (ix = 0; ix < wave.nx; ix++)
00225     for (iy = 0; iy < wave.ny; iy++) {
00226         pert_4mu->pt[iy][ix] = wave.pt[ix][iy];
00227         pert_4mu->var[iy][ix] = wave.var[ix][iy];
00228     }
00229
00230 /* Convert to wave analysis struct... */
00231 pert2wave(pert_15mu_low, &wave,
00232     0, pert_15mu_low->ntrack - 1, 0, pert_15mu_low->nxtrack - 1);
00233
00234 /* Estimate background... */
00235 background_poly(&wave, 5, 0);
00236
00237 /* Compute variance... */
00238 variance(&wave, var_dh);
00239
00240 /* Copy data... */
00241 for (ix = 0; ix < wave.nx; ix++)
00242     for (iy = 0; iy < wave.ny; iy++) {
00243         pert_15mu_low->pt[iy][ix] = wave.pt[ix][iy];
00244         pert_15mu_low->var[iy][ix] = wave.var[ix][iy];
00245     }
00246
00247 /* Convert to wave analysis struct... */
00248 pert2wave(pert_15mu_high, &wave,
00249     0, pert_15mu_high->ntrack - 1, 0, pert_15mu_high->nxtrack - 1);
00250
00251 /* Estimate background... */
00252 background_poly(&wave, 5, 0);
00253
00254 /* Compute variance... */
00255 variance(&wave, var_dh);
00256
00257 /* Copy data... */
00258 for (ix = 0; ix < wave.nx; ix++)

```

```

00259     for (iy = 0; iy < wave.ny; iy++) {
00260         pert_15mu_high->pt[iy][ix] = wave.pt[ix][iy];
00261         pert_15mu_high->var[iy][ix] = wave.var[ix][iy];
00262     }
00263
00264     /* -----
00265        Write to netCDF file...
00266        ----- */
00267
00268     /* Create netCDF file... */
00269     NC(nc_create(argv[1], NC_CLOBBER, &ncid));
00270
00271     /* Set dimensions... */
00272     NC(nc_def_dim(ncid, "NTRACK", NC_UNLIMITED, &dimid[0]));
00273     NC(nc_def_dim(ncid, "NXTRACK", AIRS_RAD_GEOXTRACK, &dimid[1]));
00274
00275     /* Add variables... */
00276     NC(nc_def_var(ncid, "time", NC_DOUBLE, 2, dimid, &time_varid));
00277     add_att(ncid, time_varid, "s", "time (seconds since 2000-01-01T00:00Z)");
00278     NC(nc_def_var(ncid, "lon", NC_DOUBLE, 2, dimid, &lon_varid));
00279     add_att(ncid, lon_varid, "deg", "footprint longitude");
00280     NC(nc_def_var(ncid, "lat", NC_DOUBLE, 2, dimid, &lat_varid));
00281     add_att(ncid, lat_varid, "deg", "footprint latitude");
00282
00283     NC(nc_def_var(ncid, "bt_8mu", NC_FLOAT, 2, dimid, &bt_8mu_varid));
00284     add_att(ncid, bt_8mu_varid, "K", "brightness temperature at 8.1 micron");
00285
00286     NC(nc_def_var(ncid, "bt_4mu", NC_FLOAT, 2, dimid, &bt_4mu_varid));
00287     add_att(ncid, bt_4mu_varid, "K", "brightness temperature " " at 4.3 micron");
00288     NC(nc_def_var(ncid, "bt_4mu_pt", NC_FLOAT, 2, dimid, &bt_4mu_pt_varid));
00289     add_att(ncid, bt_4mu_pt_varid, "K", "brightness temperature perturbation"
00290            " at 4.3 micron");
00291     NC(nc_def_var(ncid, "bt_4mu_var", NC_FLOAT, 2, dimid, &bt_4mu_var_varid));
00292     add_att(ncid, bt_4mu_var_varid, "K^2", "brightness temperature variance"
00293            " at 4.3 micron");
00294
00295     NC(nc_def_var(ncid, "bt_15mu_low", NC_FLOAT, 2, dimid, &bt_15mu_low_varid));
00296     add_att(ncid, bt_15mu_low_varid, "K", "brightness temperature"
00297            " at 15 micron (low altitudes)");
00298     NC(nc_def_var(ncid, "bt_15mu_low_pt", NC_FLOAT, 2, dimid,
00299            &bt_15mu_low_pt_varid));
00300     add_att(ncid, bt_15mu_low_pt_varid, "K",
00301            "brightness temperature perturbation"
00302            " at 15 micron (low altitudes)");
00303     NC(nc_def_var
00304        (ncid, "bt_15mu_low_var", NC_FLOAT, 2, dimid, &bt_15mu_low_var_varid));
00305     add_att(ncid, bt_15mu_low_var_varid, "K^2",
00306            "brightness temperature variance" " at 15 micron (low altitudes)");
00307
00308     NC(nc_def_var(ncid, "bt_15mu_high", NC_FLOAT, 2, dimid,
00309            &bt_15mu_high_varid));
00310     add_att(ncid, bt_15mu_high_varid, "K", "brightness temperature"
00311            " at 15 micron (high altitudes)");
00312     NC(nc_def_var(ncid, "bt_15mu_high_pt", NC_FLOAT, 2, dimid,
00313            &bt_15mu_high_pt_varid));
00314     add_att(ncid, bt_15mu_high_pt_varid, "K",
00315            "brightness temperature perturbation"
00316            " at 15 micron (high altitudes)");
00317     NC(nc_def_var
00318        (ncid, "bt_15mu_high_var", NC_FLOAT, 2, dimid, &bt_15mu_high_var_varid));
00319     add_att(ncid, bt_15mu_high_var_varid, "K^2",
00320            "brightness temperature variance" " at 15 micron (high altitudes)");
00321
00322     /* Leave define mode... */
00323     NC(nc_enddef(ncid));
00324
00325     /* Loop over tracks... */
00326     for (track = 0; track < pert_4mu->ntrack; track++) {
00327
00328         /* Set array sizes... */
00329         start[0] = (size_t) track;
00330         start[1] = 0;
00331         count[0] = 1;
00332         count[1] = (size_t) pert_4mu->nxtrack;
00333
00334         /* Write data... */
00335         NC(nc_put_vara_double(ncid, time_varid, start, count,
00336            pert_4mu->time[track]));
00337         NC(nc_put_vara_double(ncid, lon_varid, start, count,
00338            pert_4mu->lon[track]));
00339         NC(nc_put_vara_double(ncid, lat_varid, start, count,
00340            pert_4mu->lat[track]));
00341
00342         NC(nc_put_vara_double(ncid, bt_8mu_varid, start, count,
00343            pert_4mu->dc[track]));
00344
00345         NC(nc_put_vara_double(ncid, bt_4mu_varid, start, count,

```

```

00346         pert_4mu->bt[track]));
00347     NC(nc_put_vara_double(ncid, bt_4mu_pt_varid, start, count,
00348         pert_4mu->pt[track]));
00349     NC(nc_put_vara_double(ncid, bt_4mu_var_varid, start, count,
00350         pert_4mu->var[track]));
00351
00352     NC(nc_put_vara_double(ncid, bt_15mu_low_varid, start, count,
00353         pert_15mu_low->bt[track]));
00354     NC(nc_put_vara_double(ncid, bt_15mu_low_pt_varid, start, count,
00355         pert_15mu_low->pt[track]));
00356     NC(nc_put_vara_double(ncid, bt_15mu_low_var_varid, start, count,
00357         pert_15mu_low->var[track]));
00358
00359     NC(nc_put_vara_double(ncid, bt_15mu_high_varid, start, count,
00360         pert_15mu_high->bt[track]));
00361     NC(nc_put_vara_double(ncid, bt_15mu_high_pt_varid, start, count,
00362         pert_15mu_high->pt[track]));
00363     NC(nc_put_vara_double(ncid, bt_15mu_high_var_varid, start, count,
00364         pert_15mu_high->var[track]));
00365 }
00366
00367 /* Close file... */
00368 NC(nc_close(ncid));
00369
00370 /* Free... */
00371 free(pert_4mu);
00372 free(pert_15mu_low);
00373 free(pert_15mu_high);
00374
00375 return EXIT_SUCCESS;
00376 }

```

## 5.47 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[ ])

### 5.47.1 Function Documentation

#### 5.47.1.1 double buoyancy ( double z0, double p0, double t0, double z1, double p1, double t1 )

Definition at line 204 of file [rayt.c](#).

```

00210     {
00211
00212     double theta0, theta1;
00213
00214     /* Get potential temperature... */
00215     theta0 = temp2theta(p0, t0);
00216     theta1 = temp2theta(p1, t1);
00217
00218     /* Get buoyancy frequency... */
00219     return sqrt(G0 / (0.5 * (theta0 + theta1)) * (theta1 - theta0) /
00220         ((z1 - z0) * 1e3));
00221 }

```

Here is the call graph for this function:



## 5.47.1.2 double scale\_height ( double t )

Definition at line 225 of file rayt.c.

```
00226         {
00227
00228     return 29.26 * t / 1e3;
00229 }
```

## 5.47.1.3 double temp2theta ( double p, double t )

Definition at line 233 of file rayt.c.

```
00235         {
00236
00237     return t * pow(P0 / p, 0.286);
00238 }
```

## 5.47.1.4 int main ( int argc, char \* argv[] )

Definition at line 36 of file rayt.c.

```
00038         {
00039
00040     FILE *in;
00041
00042     static double f0, k, omin, z[NZ], u[NZ], urel[NZ], v[NZ], bf[NZ], bf2[NZ],
00043         H[NZ], frel[NZ], osign[NZ], f1[NZ], f2[NZ], delta[NZ], a2[NZ], m[NZ],
00044         dxdz[NZ], cgz[NZ], dz, path[NZ], tim[NZ], costh, p[NZ], t[NZ], z0, w,
00045         wsum, dzw = 5 * 1e3, fgb, m0, alpha, lat;
00046
00047     static int iz, iz2, izcrit, izrefl, nz;
00048
00049     /* Check arguments... */
00050     if (argc != 8)
00051         ERRMSG("Give parameters: <atm.tab> <z_launch> <mode> "
00052             "<t_gb | lz_launch> <lx> <lat> <direct>");
00053
00054     /* Get launch level... */
00055     z0 = atof(argv[2]);
00056     lat = atof(argv[6]);
00057     alpha = atof(argv[7]);
00058
00059     /* Read atmosphere above launch level... */
00060     if (!(in = fopen(argv[1], "r")))
00061         ERRMSG("Cannot open atmospheric data file!");
00062     while (fscanf
00063         (in, "%lg %lg %lg %lg %lg", &z[nz], &p[nz], &t[nz], &u[nz], &v[nz])
00064         == 5)
00065     {
00066         if (z[nz] >= z0) {
00067             u[nz] =
00068                 cos(alpha * M_PI / 180.) * u[nz] + sin(alpha * M_PI / 180.) * v[nz];
00069             if (++nz > NZ)
00070                 ERRMSG("Too many altitude levels!");
00071         }
00072     }
00073     fclose(in);
00074
00075     /* Compute scale height and buoyancy frequency... */
00076     for (iz = 0; iz < nz; iz++) {
00077         if (iz < nz - 1)
00078             bf[iz] = buoyancy(z[iz], p[iz], t[iz], z[iz + 1], p[iz + 1], t[iz + 1]);
00079         else
00080             bf[iz] = bf[iz - 1];
00081         H[iz] = scale_height(t[iz]) * 1e3;
00082         z[iz] *= 1e3;
00083     }
00084
00085     /* Smooth N profile... */
00086     for (iz = 0; iz < nz; iz++) {
00087         bf2[iz] = wsum = 0;
00088         for (iz2 = 0; iz2 < nz; iz2++) {
00089             if (!gsl_finite(bf[iz2]) ||
00090                 !gsl_finite(bf[GSL_MAX(iz2 - 1, 0)])) ||
```



```

00089         !gsl_finite(bf[GSL_MIN(iz2 + 1, nz - 1)]))
00090         continue;
00091         w =
00092         (fabs(z[iz] - z[iz2]) < dzw) ? 1.0 - fabs(z[iz] - z[iz2]) / dzw : 0.0;
00093         bf2[iz] += w * bf[iz2];
00094         wsum += w;
00095     }
00096     bf2[iz] /= wsum;
00097 }
00098 for (iz = 0; iz < nz; iz++)
00099     bf[iz] = bf2[iz];
00100
00101 /* Get horizontal wavenumber... */
00102 k = 2 * M_PI / (atof(argv[5]) * 1e3);
00103
00104 /* Get minimum gravity wave frequency (Coriolis parameter)... */
00105 omin = 2 * 2 * M_PI / 86400. * sin(lat / 180. * M_PI);
00106
00107 /* Get initial frequencies... */
00108 if (argv[3][0] == 't') {
00109
00110     /* Get ground-based frequency... */
00111     fgb = 2 * M_PI / (atof(argv[4]) * 60.);
00112
00113     /* Get intrinsic frequency at launch level... */
00114     f0 = fgb - k * u[0];
00115 } else if (argv[3][0] == 'l') {
00116
00117     /* Get vertical wavenumber... */
00118     m0 = 2 * M_PI / (atof(argv[4]) * 1e3);
00119
00120     /* Get intrinsic frequency at launch level... */
00121     f0 =
00122         sqrt((bf[0] * bf[0] * k * k +
00123             omin * omin * (m0 * m0 + 0.25 / (H[0] * H[0])))
00124             / (m0 * m0 + k * k + 0.25 / (H[0] * H[0])));
00126
00127     /* Get ground-based frequency... */
00128     fgb = f0 + k * u[0];
00129 } else
00130     ERRMSG("Set <mode> to 't_gb' or 'lz_launch'!");
00132
00133 /* Loop over layers... */
00134 for (iz = 0; iz < nz; iz++) {
00135     urel[iz] = u[iz] - u[0];
00136     frel[iz] = f0 - k * urel[iz];
00137     osign[iz] = frel[iz] / fabs(frel[iz]);
00138     f1[iz] = (bf[iz] * bf[iz] - frel[iz] * frel[iz]) / frel[iz];
00139     f2[iz] = (frel[iz] * frel[iz] - omin * omin) / frel[iz];
00140     delta[iz] = k * k * (1 + f1[iz] / f2[iz]);
00141     a2[iz] = 1. / 4. / (H[iz] * H[iz]);
00142     m[iz] = (-osign[iz]) * k * sqrt((f1[iz] / f2[iz]) - (a2[iz] / (k * k)));
00143     dxdz[iz] = (u[iz] * delta[iz] + k * f1[iz]) / (-1 * m[iz] * f2[iz]);
00144     dz = z[1] - z[0];
00145     cgz[iz] = f2[iz] * (-1. * m[iz]) / (k * k + m[iz] * m[iz] + a2[iz]);
00146 }
00147
00148 /* Integrate via trapezoidal rule... */
00149 for (iz = 1; iz < nz; iz++) {
00150     path[iz] = path[iz - 1] + dz * .5 * (dxdz[iz - 1] + dxdz[iz]);
00151     tim[iz] = tim[iz - 1] + dz * 2. / (cgz[iz - 1] + cgz[iz]);
00152 }
00153
00154 /* Find critical level... */
00155 for (izcrit = 0; izcrit < nz; izcrit++)
00156     if (f0 / fabs(f0) * frel[izcrit] / fabs(omin) <= 1)
00157         break;
00158
00159 /* Find trapping/reflection level... */
00160 for (izrefl = 0; izrefl < nz; izrefl++) {
00161     costh = fabs(f0 - k * urel[izrefl])
00162         / sqrt(bf[izrefl] * bf[izrefl]
00163             * (1 -
00164                 (1 -
00165                     (omin / bf[izrefl]) * (omin / bf[izrefl])) / (k * k /
00166                         a2[izrefl] +
00167                         1)));
00168     if (costh >= 1.0)
00169         break;
00170 }
00171
00172 /* Filter data... */
00173 for (iz = 0; iz < nz; iz++)
00174     if (iz >= izcrit || iz >= izrefl)
00175         path[iz] = tim[iz] = m[iz] = frel[iz] = cgz[iz] = sqrt(-1.0);

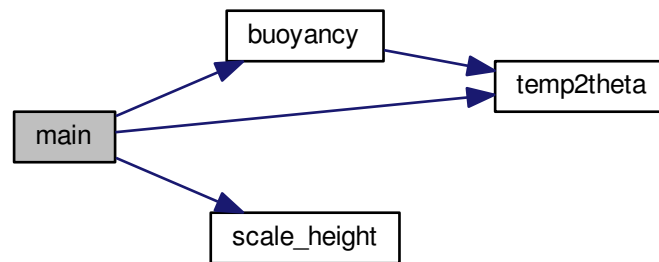
```

```

00176
00177  /* Write output... */
00178  printf("# $1 = latitude [deg]\n"
00179         "# $2 = altitude [km]\n"
00180         "# $3 = pressure [hPa]\n"
00181         "# $4 = temperature [K]\n"
00182         "# $5 = potential temperature [K]\n"
00183         "# $6 = wind speed [m/s]\n"
00184         "# $7 = buoyancy frequency [1/s]\n"
00185         "# $8 = scale height [km]\n"
00186         "# $9 = horizontal distance [km]\n"
00187         "# $10 = propagation time [min]\n"
00188         "# $11 = vertical wavelength [km]\n"
00189         "# $12 = wave period [min]\n"
00190         "# $13 = vertical group velocity [m/s]\n\n");
00191  for (iz = 0; iz < nz; iz++)
00192      printf("%g %g %g %g %g %g %g %g %g %g %g %g %g\n",
00193            lat, z[iz] / 1e3, p[iz], t[iz], temp2theta(p[iz], t[iz]), u[iz],
00194            bf[iz], H[iz] / 1e3, path[iz] / 1e3, tim[iz] / 60,
00195            fabs(2 * M_PI / m[iz] / 1e3), 2. * M_PI / frel[iz] / 60., cgz[iz]);
00196  printf("\n# z_crit= %g km\n# z_refl= %g km\n",
00197        z[izcrit - 1] / 1e3, z[izrefl - 1] / 1e3);
00198
00199  return EXIT_SUCCESS;
00200 }

```

Here is the call graph for this function:



## 5.48 rayt.c

```

00001 #include "libairs.h"
00002
00003 /* -----
00004     Dimensions...
00005     ----- */
00006
00007 /* Maximum number of levels. */
00008 #define NZ 1000
00009
00010 /* -----
00011     Functions...
00012     ----- */
00013
00014 /* Compute buoyancy frequency. */
00015 double buoyancy(
00016     double z0,
00017     double p0,
00018     double t0,
00019     double z1,
00020     double p1,
00021     double t1);
00022
00023 /* Compute scale height. */
00024 double scale_height(
00025     double t);
00026

```

```

00027 /* Convert temperature to potential temperature. */
00028 double temp2theta(
00029     double p,
00030     double t);
00031
00032 /* -----
00033     Main...
00034     ----- */
00035
00036 int main(
00037     int argc,
00038     char *argv[]) {
00039
00040     FILE *in;
00041
00042     static double f0, k, omin, z[NZ], u[NZ], urel[NZ], v[NZ], bf[NZ], bf2[NZ],
00043         H[NZ], frel[NZ], osign[NZ], f1[NZ], f2[NZ], delta[NZ], a2[NZ], m[NZ],
00044         dxdz[NZ], cgz[NZ], dz, path[NZ], tim[NZ], costh, p[NZ], t[NZ], z0, w,
00045         wsum, dzw = 5 * 1e3, fgb, m0, alpha, lat;
00046
00047     static int iz, iz2, izcrit, izrefl, nz;
00048
00049     /* Check arguments... */
00050     if (argc != 8)
00051         ERRMSG("Give parameters: <atm.tab> <z_launch> <mode> "
00052             "<t_gb | lz_launch> <lx> <lat> <direct>");
00053
00054     /* Get launch level... */
00055     z0 = atof(argv[2]);
00056     lat = atof(argv[6]);
00057     alpha = atof(argv[7]);
00058
00059     /* Read atmosphere above launch level... */
00060     if (!(in = fopen(argv[1], "r")))
00061         ERRMSG("Cannot open atmospheric data file!");
00062     while (fscanf
00063         (in, "%lg %lg %lg %lg %lg", &z[nz], &p[nz], &t[nz], &u[nz], &v[nz])
00064         == 5)
00065     {
00066         if (z[nz] >= z0) {
00067             u[nz] =
00068                 cos(alpha * M_PI / 180.) * u[nz] + sin(alpha * M_PI / 180.) * v[nz];
00069             if (++nz > NZ)
00070                 ERRMSG("Too many altitude levels!");
00071         }
00072     }
00073     fclose(in);
00074
00075     /* Compute scale height and buoyancy frequency... */
00076     for (iz = 0; iz < nz; iz++) {
00077         if (iz < nz - 1)
00078             bf[iz] = buoyancy(z[iz], p[iz], t[iz], z[iz + 1], p[iz + 1], t[iz + 1]);
00079         else
00080             bf[iz] = bf[iz - 1];
00081         H[iz] = scale_height(t[iz]) * 1e3;
00082         z[iz] *= 1e3;
00083     }
00084
00085     /* Smooth N profile... */
00086     for (iz = 0; iz < nz; iz++) {
00087         bf2[iz] = wsum = 0;
00088         for (iz2 = 0; iz2 < nz; iz2++) {
00089             if (!gsl_finite(bf[iz2]) ||
00090                 !gsl_finite(bf[GSL_MAX(iz2 - 1, 0)]) ||
00091                 !gsl_finite(bf[GSL_MIN(iz2 + 1, nz - 1)]))
00092                 continue;
00093             w =
00094                 (fabs(z[iz] - z[iz2]) < dzw) ? 1.0 - fabs(z[iz] - z[iz2]) / dzw : 0.0;
00095             bf2[iz] += w * bf[iz2];
00096             wsum += w;
00097         }
00098         bf2[iz] /= wsum;
00099     }
00100
00101     for (iz = 0; iz < nz; iz++)
00102         bf[iz] = bf2[iz];
00103
00104     /* Get horizontal wavenumber... */
00105     k = 2 * M_PI / (atof(argv[5]) * 1e3);
00106
00107     /* Get minimum gravity wave frequency (Coriolis parameter)... */
00108     omin = 2 * 2 * M_PI / 86400. * sin(lat / 180. * M_PI);
00109
00110     /* Get initial frequencies... */
00111     if (argv[3][0] == 't') {
00112         /* Get ground-based frequency... */
00113         fgb = 2 * M_PI / (atof(argv[4]) * 60.);
00114
00115         /* Get intrinsic frequency at launch level... */

```

```

00114     f0 = fgb - k * u[0];
00115
00116 } else if (argv[3][0] == 'l') {
00117
00118     /* Get vertical wavenumber... */
00119     m0 = 2 * M_PI / (atof(argv[4]) * 1e3);
00120
00121     /* Get intrinsic frequency at launch level... */
00122     f0 =
00123         sqrt((bf[0] * bf[0] * k * k +
00124             omin * omin * (m0 * m0 + 0.25 / (H[0] * H[0])))
00125             / (m0 * m0 + k * k + 0.25 / (H[0] * H[0])));
00126
00127     /* Get ground-based frequency... */
00128     fgb = f0 + k * u[0];
00129
00130 } else
00131     ERRMSG("Set <mode> to 't_gb' or 'lz_launch'!");
00132
00133 /* Loop over layers... */
00134 for (iz = 0; iz < nz; iz++) {
00135     urel[iz] = u[iz] - u[0];
00136     frel[iz] = f0 - k * urel[iz];
00137     osign[iz] = frel[iz] / fabs(frel[iz]);
00138     fl[iz] = (bf[iz] * bf[iz] - frel[iz] * frel[iz]) / frel[iz];
00139     f2[iz] = (frel[iz] * frel[iz] - omin * omin) / frel[iz];
00140     delta[iz] = k * k * (1 + fl[iz] / f2[iz]);
00141     a2[iz] = 1. / 4. / (H[iz] * H[iz]);
00142     m[iz] = (-osign[iz]) * k * sqrt((fl[iz] / f2[iz]) - (a2[iz] / (k * k)));
00143     dxdz[iz] = (u[iz] * delta[iz] + k * fl[iz]) / (-1 * m[iz] * f2[iz]);
00144     dz = z[1] - z[0];
00145     cgz[iz] = f2[iz] * (-1. * m[iz]) / (k * k + m[iz] * m[iz] + a2[iz]);
00146 }
00147
00148 /* Integrate via trapezoidal rule... */
00149 for (iz = 1; iz < nz; iz++) {
00150     path[iz] = path[iz - 1] + dz * .5 * (dxdz[iz - 1] + dxdz[iz]);
00151     tim[iz] = tim[iz - 1] + dz * 2. / (cgz[iz - 1] + cgz[iz]);
00152 }
00153
00154 /* Find critical level... */
00155 for (izcrit = 0; izcrit < nz; izcrit++)
00156     if (f0 / fabs(f0) * frel[izcrit] / fabs(omin) <= 1)
00157         break;
00158
00159 /* Find trapping/reflection level... */
00160 for (izrefl = 0; izrefl < nz; izrefl++) {
00161     costh = fabs(f0 - k * urel[izrefl])
00162         / sqrt(bf[izrefl] * bf[izrefl])
00163         * (1 -
00164             (1 -
00165                 (omin / bf[izrefl]) * (omin / bf[izrefl])) / (k * k /
00166                     a2[izrefl] +
00167                     1)));
00168     if (costh >= 1.0)
00169         break;
00170 }
00171
00172 /* Filter data... */
00173 for (iz = 0; iz < nz; iz++)
00174     if (iz >= izcrit || iz >= izrefl)
00175         path[iz] = tim[iz] = m[iz] = frel[iz] = cgz[iz] = sqrt(-1.0);
00176
00177 /* Write output... */
00178 printf("# $1 = latitude [deg]\n"
00179     "# $2 = altitude [km]\n"
00180     "# $3 = pressure [hPa]\n"
00181     "# $4 = temperature [K]\n"
00182     "# $5 = potential temperature [K]\n"
00183     "# $6 = wind speed [m/s]\n"
00184     "# $7 = buoyancy frequency [1/s]\n"
00185     "# $8 = scale height [km]\n"
00186     "# $9 = horizontal distance [km]\n"
00187     "# $10 = propagation time [min]\n"
00188     "# $11 = vertical wavelength [km]\n"
00189     "# $12 = wave period [min]\n"
00190     "# $13 = vertical group velocity [m/s]\n\n");
00191 for (iz = 0; iz < nz; iz++)
00192     printf("%g %g %g %g %g %g %g %g %g %g %g %g %g\n",
00193         lat, z[iz] / 1e3, p[iz], t[iz], temp2theta(p[iz], t[iz]), u[iz],
00194         bf[iz], H[iz] / 1e3, path[iz] / 1e3, tim[iz] / 60,
00195         fabs(2 * M_PI / m[iz] / 1e3), 2. * M_PI / frel[iz] / 60., cgz[iz]);
00196 printf("\n# z_crit= %g km\n# z_refl= %g km\n",
00197     z[izcrit - 1] / 1e3, z[izrefl - 1] / 1e3);
00198
00199 return EXIT_SUCCESS;
00200 }

```

```

00201
00202 /*****
00203
00204 double buoyancy(
00205     double z0,
00206     double p0,
00207     double t0,
00208     double z1,
00209     double p1,
00210     double t1) {
00211
00212     double theta0, theta1;
00213
00214     /* Get potential temperature... */
00215     theta0 = temp2theta(p0, t0);
00216     theta1 = temp2theta(p1, t1);
00217
00218     /* Get buoyancy frequency... */
00219     return sqrt(G0 / (0.5 * (theta0 + theta1)) * (theta1 - theta0) /
00220                ((z1 - z0) * 1e3));
00221 }
00222
00223 /*****
00224
00225 double scale_height(
00226     double t) {
00227
00228     return 29.26 * t / 1e3;
00229 }
00230
00231 /*****
00232
00233 double temp2theta(
00234     double p,
00235     double t) {
00236
00237     return t * pow(P0 / p, 0.286);
00238 }

```

## 5.49 ret2tab.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 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\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]),
00064             CHECK(airs_ret_gran.H2OMMRStd[track][xtrack][lay]),
00065             CHECK(airs_ret_gran.O3VMRStd[track][xtrack][lay]),
00066             CHECK(airs_ret_gran.COVMRLevStd[track][xtrack][lay]),
00067             CHECK(airs_ret_gran.CH4VMRLevStd[track][xtrack][lay]));
00068     }
00069
00070 /* Close file... */
00071 fclose(out);
00072
00073 return EXIT_SUCCESS;
00074 }

```

## 5.50 ret2tab.c

```

00001 #include "libairs.h"
00002
00003 /* -----
00004     Macros...
00005     ----- */
00006
00007 /* Replace dummy values by nan. */
00008 #define CHECK(x) ((x) != -9999 ? (x) : GSL_NAN)
00009
00010 /* -----
00011     Main...
00012     ----- */
00013
00014 int main(
00015     int argc,
00016     char *argv[]) {
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\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]),
00064                 CHECK(airs_ret_gran.H2OMMRStd[track][xtrack][lay]),
00065                 CHECK(airs_ret_gran.O3VMRStd[track][xtrack][lay]),
00066                 CHECK(airs_ret_gran.COVMRLevStd[track][xtrack][lay]),
00067                 CHECK(airs_ret_gran.CH4VMRLevStd[track][xtrack][lay]));
00068     }
00069
00070     /* Close file... */
00071     fclose(out);
00072
00073     return EXIT_SUCCESS;
00074 }

```

## 5.51 retrieval.c File Reference

### Data Structures

- struct [ncd\\_t](#)  
*Buffer for netCDF data.*
- 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)  
*Create variable in netCDF file.*
- void [buffer\\_nc](#) ([atm\\_t](#) \*atm, double chisq, [ncd\\_t](#) \*ncd, int track, int xtrack, int np0, int np1)  
*Buffer netCDF data.*
- double [cost\\_function](#) (gsl\_vector \*dx, gsl\_vector \*dy, gsl\_matrix \*s\_a\_inv, gsl\_vector \*sig\_eps\_inv)  
*Compute cost function.*
- void [fill\\_gaps](#) (double x[L2\_NTRACK][L2\_NXTRACK][L2\_NLAY], double cx, double cy)  
*Fill data gaps in L2 data.*
- void [init\\_l2](#) ([ncd\\_t](#) \*ncd, int track, int xtrack, [ctl\\_t](#) \*ctl, [atm\\_t](#) \*atm)  
*Initialize with AIRS Level-2 data.*
- void [matrix\\_invert](#) (gsl\_matrix \*a)  
*Invert symmetric matrix.*
- void [matrix\\_product](#) (gsl\_matrix \*a, gsl\_vector \*b, int transpose, gsl\_matrix \*c)  
*Compute matrix product  $A^T B$  or  $ABA^T$  for diagonal matrix  $B$ .*
- 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)  
*Carry out optimal estimation retrieval.*
- void [read\\_nc](#) (char \*filename, [ncd\\_t](#) \*ncd)  
*Read netCDF file.*

- void `read_ret_ctl` (int argc, char \*argv[], `ctl_t` \*ctl, `ret_t` \*ret)  
*Read retrieval control parameters.*
- void `set_cov_apr` (`ret_t` \*ret, `ctl_t` \*ctl, `atm_t` \*atm, int \*iqa, int \*ipa, gsl\_matrix \*s\_a)  
*Set a priori covariance.*
- 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)  
*Set measurement errors.*
- double `sza` (double sec, double lon, double lat)  
*Calculate solar zenith angle.*
- void `write_nc` (char \*filename, `ncd_t` \*ncd)  
*Write to netCDF file...*
- int `main` (int argc, char \*argv[])

### 5.51.1 Function Documentation

**5.51.1.1** void `add_var` ( int *ncid*, const char \* *varname*, const char \* *unit*, const char \* *longname*, int *type*, int *dimid*[], int \* *varid*, int *ndims* )

Create variable in netCDF file.

Add variable to netCDF file.

Definition at line 483 of file `retrieval.c`.

```
00491         {
00492
00493     /* Check if variable exists... */
00494     if (nc_inq_varid(ncid, varname, varid) != NC_NOERR) {
00495
00496     /* Define variable... */
00497     NC(nc_def_var(ncid, varname, type, ndims, dimid, varid));
00498
00499     /* Set long name... */
00500     NC(nc_put_att_text
00501         (ncid, *varid, "long_name", strlen(longname), longname));
00502
00503     /* Set units... */
00504     NC(nc_put_att_text(ncid, *varid, "units", strlen(unit), unit));
00505     }
00506 }
```

**5.51.1.2** void `buffer_nc` ( `atm_t` \* *atm*, double *chisq*, `ncd_t` \* *ncd*, int *track*, int *xtrack*, int *np0*, int *np1* )

Buffer netCDF data.

Definition at line 510 of file `retrieval.c`.

```
00517         {
00518
00519     int ip;
00520
00521     /* Set number of data points... */
00522     ncd->np = np1 - np0 + 1;
00523
00524     /* Save retrieval data... */
00525     for (ip = np0; ip <= np1; ip++) {
00526         ncd->ret_z[ip - np0] = (float) atm->z[ip];
00527         ncd->ret_p[track * L1_NXTRACK + xtrack] = (float) atm->p[np0];
00528         ncd->ret_t[(track * L1_NXTRACK + xtrack) * ncd->np + ip - np0] =
00529             (gsl_finite(chisq) ? (float) atm->t[ip] : GSL_NAN);
00530     }
00531 }
```



### 5.51.1.3 double cost\_function ( gsl\_vector \* dx, gsl\_vector \* dy, gsl\_matrix \* s\_a\_inv, gsl\_vector \* sig\_eps\_inv )

Compute cost function.

Definition at line 535 of file [retrieval.c](#).

```

00539             {
00540
00541         gsl_vector *x_aux, *y_aux;
00542
00543         double chisq_a, chisq_m = 0;
00544
00545         size_t i, m, n;
00546
00547         /* Get sizes... */
00548         m = dy->size;
00549         n = dx->size;
00550
00551         /* Allocate... */
00552         x_aux = gsl_vector_alloc(n);
00553         y_aux = gsl_vector_alloc(m);
00554
00555         /* Determine normalized cost function...
00556          (chi^2 = 1/m * [dy^T * S_eps^{-1} * dy + dx^T * S_a^{-1} * dx]) */
00557         for (i = 0; i < m; i++)
00558             chisq_m +=
00559                 gsl_pow_2(gsl_vector_get(dy, i) * gsl_vector_get(sig_eps_inv, i));
00560         gsl_blas_dgemv(CblasNoTrans, 1.0, s_a_inv, dx, 0.0, x_aux);
00561         gsl_blas_ddot(dx, x_aux, &chisq_a);
00562
00563         /* Free... */
00564         gsl_vector_free(x_aux);
00565         gsl_vector_free(y_aux);
00566
00567         /* Return cost function value... */
00568         return (chisq_m + chisq_a) / (double) m;
00569     }

```

### 5.51.1.4 void fill\_gaps ( double x[L2\_NTRACK][L2\_NXTRACK][L2\_NLAY], double cx, double cy )

Fill data gaps in L2 data.

Definition at line 573 of file [retrieval.c](#).

```

00576         {
00577
00578         double help[L2_NTRACK][L2_NXTRACK], w, wsum;
00579
00580         int lay, track, track2, xtrack, xtrack2;
00581
00582         /* Loop over layers... */
00583         for (lay = 0; lay < L2_NLAY; lay++) {
00584
00585             /* Loop over grid points... */
00586             for (track = 0; track < L2_NTRACK; track++)
00587                 for (xtrack = 0; xtrack < L2_NXTRACK; xtrack++) {
00588
00589                     /* Init... */
00590                     help[track][xtrack] = 0;
00591                     wsum = 0;
00592
00593                     /* Average data points... */
00594                     for (track2 = 0; track2 < L2_NTRACK; track2++)
00595                         for (xtrack2 = 0; xtrack2 < L2_NXTRACK; xtrack2++)
00596                             if (gsl_finite(x[track2][xtrack2][lay])
00597                                 && x[track2][xtrack2][lay] > 0) {
00598                                 w = exp(-gsl_pow_2((xtrack - xtrack2) / cx)
00599                                         - gsl_pow_2((track - track2) / cy));
00600                                 help[track][xtrack] += w * x[track2][xtrack2][lay];
00601                                 wsum += w;
00602                             }
00603
00604                     /* Normalize... */
00605                     if (wsum > 0)
00606                         help[track][xtrack] /= wsum;
00607                     else

```

```

00608         help[track][xtrack] = GSL_NAN;
00609     }
00610
00611     /* Copy grid points... */
00612     for (track = 0; track < L2_NTRACK; track++)
00613         for (xtrack = 0; xtrack < L2_NXTRACK; xtrack++)
00614             x[track][xtrack][lay] = help[track][xtrack];
00615     }
00616 }

```

#### 5.51.1.5 void init\_l2( ncd\_t \*ncd, int track, int xtrack, ctl\_t \*ctl, atm\_t \*atm )

Initialize with AIRS Level-2 data.

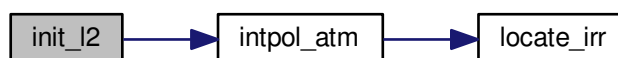
Definition at line 620 of file [retrieval.c](#).

```

00625     {
00626
00627     static atm_t atm_airs;
00628
00629     double k[NW], p, q[NG], t, w, zmax = 0, zmin = 1000;
00630
00631     int ip, lay;
00632
00633     /* Reset track- and xtrack-index to match Level-2 data... */
00634     track /= 3;
00635     xtrack /= 3;
00636
00637     /* Store AIRS data in atmospheric data struct... */
00638     atm_airs.np = 0;
00639     for (lay = 0; lay < L2_NLAY; lay++)
00640         if (gsl_finite(ncd->l2_z[track][xtrack][lay])) {
00641             atm_airs.z[atm_airs.np] = ncd->l2_z[track][xtrack][lay];
00642             atm_airs.p[atm_airs.np] = ncd->l2_p[lay];
00643             atm_airs.t[atm_airs.np] = ncd->l2_t[track][xtrack][lay];
00644             if ((++atm_airs.np) > NP)
00645                 ERRMSG("Too many layers!");
00646         }
00647
00648     /* Check number of levels... */
00649     if (atm_airs.np <= 0)
00650         return;
00651
00652     /* Get height range of AIRS data... */
00653     for (ip = 0; ip < atm_airs.np; ip++) {
00654         zmax = GSL_MAX(zmax, atm_airs.z[ip]);
00655         zmin = GSL_MIN(zmin, atm_airs.z[ip]);
00656     }
00657
00658     /* Merge AIRS data... */
00659     for (ip = 0; ip < atm->np; ip++) {
00660
00661         /* Interpolate AIRS data... */
00662         intpol_atm(ctl, &atm_airs, atm->z[ip], &p, &t, q, k);
00663
00664         /* Weighting factor... */
00665         w = 1;
00666         if (atm->z[ip] > zmax)
00667             w = GSL_MAX(1 - (atm->z[ip] - zmax) / 50, 0);
00668         if (atm->z[ip] < zmin)
00669             w = GSL_MAX(1 - (zmin - atm->z[ip]) / 50, 0);
00670
00671         /* Merge... */
00672         atm->t[ip] = w * t + (1 - w) * atm->t[ip];
00673         atm->p[ip] = w * p + (1 - w) * atm->p[ip];
00674     }
00675 }

```

Here is the call graph for this function:



### 5.51.1.6 void matrix\_invert ( gsl\_matrix \* a )

Invert symmetric matrix.

Definition at line 679 of file [retrieval.c](#).

```

00680         {
00681
00682     size_t diag = 1, i, j, n;
00683
00684     /* Get size... */
00685     n = a->size1;
00686
00687     /* Check if matrix is diagonal... */
00688     for (i = 0; i < n && diag; i++)
00689         for (j = i + 1; j < n; j++)
00690             if (gsl_matrix_get(a, i, j) != 0) {
00691                 diag = 0;
00692                 break;
00693             }
00694
00695     /* Quick inversion of diagonal matrix... */
00696     if (diag)
00697         for (i = 0; i < n; i++)
00698             gsl_matrix_set(a, i, i, 1 / gsl_matrix_get(a, i, i));
00699
00700     /* Matrix inversion by means of Cholesky decomposition... */
00701     else {
00702         gsl_linalg_cholesky_decomp(a);
00703         gsl_linalg_cholesky_invert(a);
00704     }
00705 }
```

### 5.51.1.7 void matrix\_product ( gsl\_matrix \* a, gsl\_vector \* b, int transpose, gsl\_matrix \* c )

Compute matrix product  $A^T B A$  or  $A B A^T$  for diagonal matrix B.

Definition at line 709 of file [retrieval.c](#).

```

00713         {
00714
00715     gsl_matrix *aux;
00716
00717     size_t i, j, m, n;
00718
00719     /* Set sizes... */
00720     m = a->size1;
00721     n = a->size2;
00722
00723     /* Allocate... */
00724     aux = gsl_matrix_alloc(m, n);
00725
00726     /* Compute A^T B A... */
00727     if (transpose == 1) {
00728
00729         /* Compute B^1/2 A... */
00730         for (i = 0; i < m; i++)
00731             for (j = 0; j < n; j++)
00732                 gsl_matrix_set(aux, i, j,
00733                     gsl_vector_get(b, i) * gsl_matrix_get(a, i, j));
00734
00735         /* Compute A^T B A = (B^1/2 A)^T (B^1/2 A)... */
00736         gsl_blas_dgemm(CblasTrans, CblasNoTrans, 1.0, aux, aux, 0.0, c);
00737     }
00738
00739     /* Compute A B A^T... */
00740     else if (transpose == 2) {
00741
00742         /* Compute A B^1/2... */
00743         for (i = 0; i < m; i++)
00744             for (j = 0; j < n; j++)
00745                 gsl_matrix_set(aux, i, j,
00746                     gsl_matrix_get(a, i, j) * gsl_vector_get(b, j));
00747
00748         /* Compute A B A^T = (A B^1/2) (A B^1/2)^T... */
00749         gsl_blas_dgemm(CblasNoTrans, CblasTrans, 1.0, aux, aux, 0.0, c);
00750     }
00751
00752     /* Free... */
00753     gsl_matrix_free(aux);
00754 }
```

**5.51.18** `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 )`

Carry out optimal estimation retrieval.

Definition at line 758 of file [retrieval.c](#).

```

00765         {
00766
00767     static int ipa[N], iqa[N];
00768
00769     gsl_matrix *a, *cov, *k_i, *s_a_inv;
00770     gsl_vector *b, *dx, *dy, *sig_eps_inv, *sig_formod, *sig_noise,
00771         *x_a, *x_i, *x_step, *y_aux, *y_i, *y_m;
00772
00773     double chisq_old, disq = 0, lmpar = 0.001;
00774
00775     int ig, ip, it = 0, it2, iw;
00776
00777     size_t i, m, n;
00778
00779     /* -----
00780        Initialize...
00781        ----- */
00782
00783     /* Get sizes... */
00784     m = obs2y(ctl, obs_meas, NULL, NULL, NULL);
00785     n = atm2x(ctl, atm_apr, NULL, iqa, ipa);
00786     if (m <= 0 || n <= 0) {
00787         *chisq = GSL_NAN;
00788         return;
00789     }
00790
00791     /* Allocate... */
00792     a = gsl_matrix_alloc(n, n);
00793     cov = gsl_matrix_alloc(n, n);
00794     k_i = gsl_matrix_alloc(m, n);
00795     s_a_inv = gsl_matrix_alloc(n, n);
00796
00797     b = gsl_vector_alloc(n);
00798     dx = gsl_vector_alloc(n);
00799     dy = gsl_vector_alloc(m);
00800     sig_eps_inv = gsl_vector_alloc(m);
00801     sig_formod = gsl_vector_alloc(m);
00802     sig_noise = gsl_vector_alloc(m);
00803     x_a = gsl_vector_alloc(n);
00804     x_i = gsl_vector_alloc(n);
00805     x_step = gsl_vector_alloc(n);
00806     y_aux = gsl_vector_alloc(m);
00807     y_i = gsl_vector_alloc(m);
00808     y_m = gsl_vector_alloc(m);
00809
00810     /* Set initial state... */
00811     copy_atm(ctl, atm_i, atm_apr, 0);
00812     copy_obs(ctl, obs_i, obs_meas, 0);
00813     formod(ctl, atm_i, obs_i);
00814
00815     /* Set state vectors and observation vectors... */
00816     atm2x(ctl, atm_apr, x_a, NULL, NULL);
00817     atm2x(ctl, atm_i, x_i, NULL, NULL);
00818     obs2y(ctl, obs_meas, y_m, NULL, NULL);
00819     obs2y(ctl, obs_i, y_i, NULL, NULL);
00820
00821     /* Set inverse a priori covariance S_a^-1... */
00822     set_cov_apr(ret, ctl, atm_apr, iqa, ipa, s_a_inv);
00823     matrix_invert(s_a_inv);
00824
00825     /* Get measurement errors... */
00826     set_cov_meas(ret, ctl, obs_meas, sig_noise, sig_formod, sig_eps_inv);
00827
00828     /* Determine dx = x_i - x_a and dy = y - F(x_i) ... */
00829     gsl_vector_memcpy(dx, x_i);
00830     gsl_vector_sub(dx, x_a);
00831     gsl_vector_memcpy(dy, y_m);
00832     gsl_vector_sub(dy, y_i);
00833
00834     /* Compute cost function... */
00835     *chisq = cost_function(dx, dy, s_a_inv, sig_eps_inv);
00836
00837     /* Compute initial kernel... */
00838     kernel(ctl, atm_i, obs_i, k_i);
00839
00840     /* -----

```

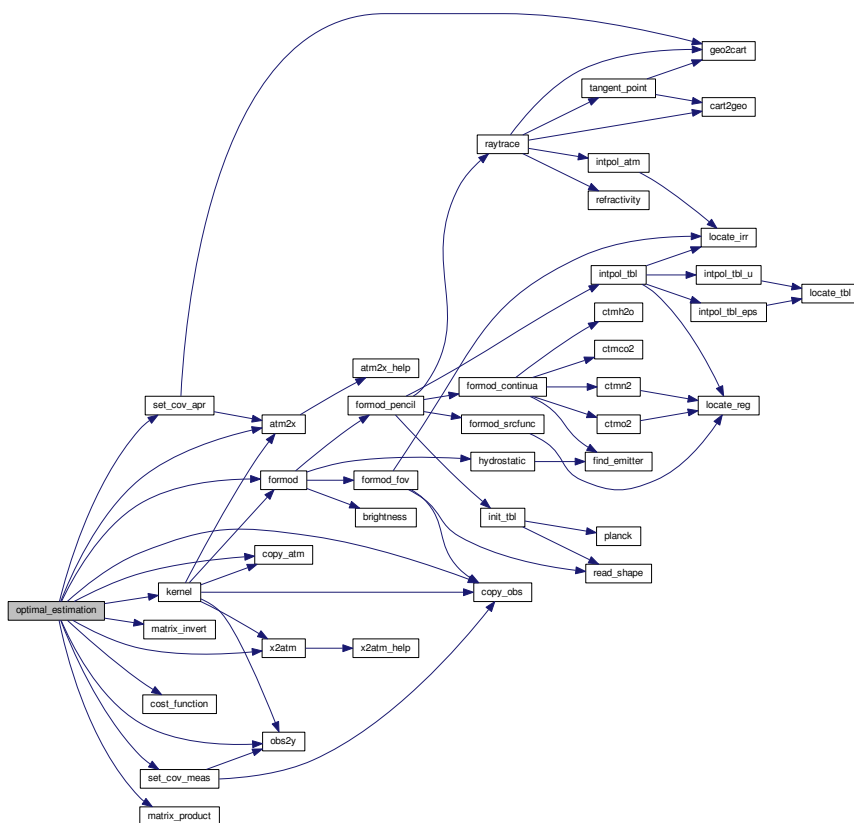
```

00841     Levenberg-Marquardt minimization...
00842     ----- */
00843
00844     /* Outer loop... */
00845     for (it = 1; it <= ret->conv_itmax; it++) {
00846
00847         /* Store current cost function value... */
00848         chisq_old = *chisq;
00849
00850         /* Compute kernel matrix K_i... */
00851         if (it > 1 && it % ret->kernel_recomp == 0)
00852             kernel(ctl, atm_i, obs_i, k_i);
00853
00854         /* Compute K_i^T * S_eps^{-1} * K_i ... */
00855         if (it == 1 || it % ret->kernel_recomp == 0)
00856             matrix_product(k_i, sig_eps_inv, 1, cov);
00857
00858         /* Determine b = K_i^T * S_eps^{-1} * dy - S_a^{-1} * dx ... */
00859         for (i = 0; i < m; i++)
00860             gsl_vector_set(y_aux, i, gsl_vector_get(dy, i)
00861                 * gsl_pow_2(gsl_vector_get(sig_eps_inv, i)));
00862         gsl_blas_dgemv(CblasTrans, 1.0, k_i, y_aux, 0.0, b);
00863         gsl_blas_dgemv(CblasNoTrans, -1.0, s_a_inv, dx, 1.0, b);
00864
00865         /* Inner loop... */
00866         for (it2 = 0; it2 < 20; it2++) {
00867
00868             /* Compute A = (1 + lmpar) * S_a^{-1} + K_i^T * S_eps^{-1} * K_i ... */
00869             gsl_matrix_memcpy(a, s_a_inv);
00870             gsl_matrix_scale(a, 1 + lmpar);
00871             gsl_matrix_add(a, cov);
00872
00873             /* Solve A * x_step = b by means of Cholesky decomposition... */
00874             gsl_linalg_cholesky_decomp(a);
00875             gsl_linalg_cholesky_solve(a, b, x_step);
00876
00877             /* Update atmospheric state... */
00878             gsl_vector_add(x_i, x_step);
00879             copy_atm(ctl, atm_i, atm_apr, 0);
00880             copy_obs(ctl, obs_i, obs_meas, 0);
00881             x2atm(ctl, x_i, atm_i);
00882
00883             /* Check atmospheric state... */
00884             for (ip = 0; ip < atm_i->np; ip++) {
00885                 atm_i->p[ip] = GSL_MIN(GSL_MAX(atm_i->p[ip], 5e-7), 5e4);
00886                 atm_i->t[ip] = GSL_MIN(GSL_MAX(atm_i->t[ip], 100), 400);
00887                 for (ig = 0; ig < ctl->ng; ig++)
00888                     atm_i->q[ig][ip] = GSL_MIN(GSL_MAX(atm_i->q[ig][ip], 0), 1);
00889                 for (iw = 0; iw < ctl->nw; iw++)
00890                     atm_i->k[iw][ip] = GSL_MAX(atm_i->k[iw][ip], 0);
00891             }
00892
00893             /* Forward calculation... */
00894             formod(ctl, atm_i, obs_i);
00895             obs2y(ctl, obs_i, y_i, NULL, NULL);
00896
00897             /* Determine dx = x_i - x_a and dy = y - F(x_i) ... */
00898             gsl_vector_memcpy(dx, x_i);
00899             gsl_vector_sub(dx, x_a);
00900             gsl_vector_memcpy(dy, y_m);
00901             gsl_vector_sub(dy, y_i);
00902
00903             /* Compute cost function... */
00904             *chisq = cost_function(dx, dy, s_a_inv, sig_eps_inv);
00905
00906             /* Modify Levenberg-Marquardt parameter... */
00907             if (*chisq > chisq_old) {
00908                 lmpar *= 10;
00909                 gsl_vector_sub(x_i, x_step);
00910             } else {
00911                 lmpar /= 10;
00912                 break;
00913             }
00914         }
00915
00916         /* Get normalized step size in state space... */
00917         gsl_blas_ddot(x_step, b, &disq);
00918         disq /= (double) n;
00919
00920         /* Convergence test... */
00921         if ((it == 1 || it % ret->kernel_recomp == 0) && disq < ret->
conv_dmin)
00922             break;
00923     }
00924
00925     /* -----
00926     Finalize...

```

```
00927 ----- */
00928
00929     gsl_matrix_free(a);
00930     gsl_matrix_free(cov);
00931     gsl_matrix_free(k_i);
00932     gsl_matrix_free(s_a_inv);
00933
00934     gsl_vector_free(b);
00935     gsl_vector_free(dx);
00936     gsl_vector_free(dy);
00937     gsl_vector_free(sig_eps_inv);
00938     gsl_vector_free(sig_formod);
00939     gsl_vector_free(sig_noise);
00940     gsl_vector_free(x_a);
00941     gsl_vector_free(x_i);
00942     gsl_vector_free(x_step);
00943     gsl_vector_free(y_aux);
00944     gsl_vector_free(y_i);
00945     gsl_vector_free(y_m);
00946 }
```

Here is the call graph for this function:



**5.51.1.9** void read\_nc ( char \* *filename*, ncd\_t \* *ncd* )

Read netCDF file.

Definition at line 950 of file retrieval.c.

```
00952
00953
00954     int varid;
00955
```

```

00956  /* Open netCDF file... */
00957  printf("Read netCDF file: %s\n", filename);
00958  NC(nc_open(filename, NC_WRITE, &ncd->ncid));
00959
00960  /* Read Level-1 data... */
00961  NC(nc_inq_varid(ncd->ncid, "l1_time", &varid));
00962  NC(nc_get_var_double(ncd->ncid, varid, ncd->l1_time[0]));
00963  NC(nc_inq_varid(ncd->ncid, "l1_lon", &varid));
00964  NC(nc_get_var_double(ncd->ncid, varid, ncd->l1_lon[0]));
00965  NC(nc_inq_varid(ncd->ncid, "l1_lat", &varid));
00966  NC(nc_get_var_double(ncd->ncid, varid, ncd->l1_lat[0]));
00967  NC(nc_inq_varid(ncd->ncid, "l1_sat_z", &varid));
00968  NC(nc_get_var_double(ncd->ncid, varid, ncd->l1_sat_z));
00969  NC(nc_inq_varid(ncd->ncid, "l1_sat_lon", &varid));
00970  NC(nc_get_var_double(ncd->ncid, varid, ncd->l1_sat_lon));
00971  NC(nc_inq_varid(ncd->ncid, "l1_sat_lat", &varid));
00972  NC(nc_get_var_double(ncd->ncid, varid, ncd->l1_sat_lat));
00973  NC(nc_inq_varid(ncd->ncid, "l1_nu", &varid));
00974  NC(nc_get_var_double(ncd->ncid, varid, ncd->l1_nu));
00975  NC(nc_inq_varid(ncd->ncid, "l1_rad", &varid));
00976  NC(nc_get_var_float(ncd->ncid, varid, ncd->l1_rad[0][0]));
00977
00978  /* Read Level-2 data... */
00979  NC(nc_inq_varid(ncd->ncid, "l2_z", &varid));
00980  NC(nc_get_var_double(ncd->ncid, varid, ncd->l2_z[0][0]));
00981  NC(nc_inq_varid(ncd->ncid, "l2_press", &varid));
00982  NC(nc_get_var_double(ncd->ncid, varid, ncd->l2_p));
00983  NC(nc_inq_varid(ncd->ncid, "l2_temp", &varid));
00984  NC(nc_get_var_double(ncd->ncid, varid, ncd->l2_t[0][0]));
00985 }

```

**5.51.1.10 void read\_ret\_ctl ( int argc, char \* argv[], ctl\_t \* ctl, ret\_t \* ret )**

Read retrieval control parameters.

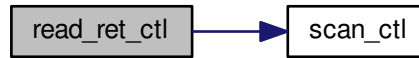
Definition at line 989 of file [retrieval.c](#).

```

00993      {
00994
00995      int id, ig, iw;
00996
00997      /* Iteration control... */
00998      ret->kernel_recomp =
00999          (int) scan_ctl(argc, argv, "KERNEL_RECOMP", -1, "3", NULL);
01000      ret->conv_itmax = (int) scan_ctl(argc, argv, "CONV_ITMAX", -1, "30", NULL);
01001      ret->conv_dmin = scan_ctl(argc, argv, "CONV_DMIN", -1, "0.1", NULL);
01002
01003      for (id = 0; id < ctl->nd; id++)
01004          ret->err_formod[id] = scan_ctl(argc, argv, "ERR_FORMOD", id, "0", NULL);
01005
01006      for (id = 0; id < ctl->nd; id++)
01007          ret->err_noise[id] = scan_ctl(argc, argv, "ERR_NOISE", id, "0", NULL);
01008
01009      ret->err_press = scan_ctl(argc, argv, "ERR_PRESS", -1, "0", NULL);
01010      ret->err_press_cz = scan_ctl(argc, argv, "ERR_PRESS_CZ", -1, "-999", NULL);
01011      ret->err_press_ch = scan_ctl(argc, argv, "ERR_PRESS_CH", -1, "-999", NULL);
01012
01013      ret->err_temp = scan_ctl(argc, argv, "ERR_TEMP", -1, "0", NULL);
01014      ret->err_temp_cz = scan_ctl(argc, argv, "ERR_TEMP_CZ", -1, "-999", NULL);
01015      ret->err_temp_ch = scan_ctl(argc, argv, "ERR_TEMP_CH", -1, "-999", NULL);
01016
01017      for (ig = 0; ig < ctl->ng; ig++) {
01018          ret->err_q[ig] = scan_ctl(argc, argv, "ERR_Q", ig, "0", NULL);
01019          ret->err_q_cz[ig] = scan_ctl(argc, argv, "ERR_Q_CZ", ig, "-999", NULL);
01020          ret->err_q_ch[ig] = scan_ctl(argc, argv, "ERR_Q_CH", ig, "-999", NULL);
01021      }
01022
01023      for (iw = 0; iw < ctl->nw; iw++) {
01024          ret->err_k[iw] = scan_ctl(argc, argv, "ERR_K", iw, "0", NULL);
01025          ret->err_k_cz[iw] = scan_ctl(argc, argv, "ERR_K_CZ", iw, "-999", NULL);
01026          ret->err_k_ch[iw] = scan_ctl(argc, argv, "ERR_K_CH", iw, "-999", NULL);
01027      }
01028 }

```

Here is the call graph for this function:



#### 5.51.1.11 void set\_cov\_apr ( ret\_t \* ret, ctl\_t \* ctl, atm\_t \* atm, int \* iqa, int \* ipa, gsl\_matrix \* s\_a )

Set a priori covariance.

Definition at line 1032 of file [retrieval.c](#).

```

01038         {
01039
01040     gsl_vector *x_a;
01041
01042     double ch, cz, rho, x0[3], x1[3];
01043
01044     int ig, iw;
01045
01046     size_t i, j, n;
01047
01048     /* Get sizes... */
01049     n = s_a->size1;
01050
01051     /* Allocate... */
01052     x_a = gsl_vector_alloc(n);
01053
01054     /* Get sigma vector... */
01055     atm2x(ctl, atm, x_a, NULL, NULL);
01056     for (i = 0; i < n; i++) {
01057         if (iqa[i] == IDXP)
01058             gsl_vector_set(x_a, i, ret->err_press / 100 * gsl_vector_get(x_a, i));
01059         if (iqa[i] == IDXT)
01060             gsl_vector_set(x_a, i, ret->err_temp);
01061         for (ig = 0; ig < ctl->ng; ig++)
01062             if (iqa[i] == IDXQ(ig))
01063                 gsl_vector_set(x_a, i, ret->err_q[ig] / 100 * gsl_vector_get(x_a, i));
01064         for (iw = 0; iw < ctl->nw; iw++)
01065             if (iqa[i] == IDXK(iw))
01066                 gsl_vector_set(x_a, i, ret->err_k[iw]);
01067     }
01068
01069     /* Check standard deviations... */
01070     for (i = 0; i < n; i++)
01071         if (gsl_pow_2(gsl_vector_get(x_a, i)) <= 0)
01072             ERRMSG("Check a priori data (zero standard deviation)!");
01073
01074     /* Initialize diagonal covariance... */
01075     gsl_matrix_set_zero(s_a);
01076     for (i = 0; i < n; i++)
01077         gsl_matrix_set(s_a, i, i, gsl_pow_2(gsl_vector_get(x_a, i)));
01078
01079     /* Loop over matrix elements... */
01080     for (i = 0; i < n; i++)
01081         for (j = 0; j < n; j++)
01082             if (i != j && iqa[i] == iqa[j]) {
01083
01084                 /* Initialize... */
01085                 cz = ch = 0;
01086
01087                 /* Set correlation lengths for pressure... */
01088                 if (iqa[i] == IDXP) {
01089                     cz = ret->err_press_cz;
01090                     ch = ret->err_press_ch;
01091                 }
01092
01093                 /* Set correlation lengths for temperature... */
01094                 if (iqa[i] == IDXT) {

```

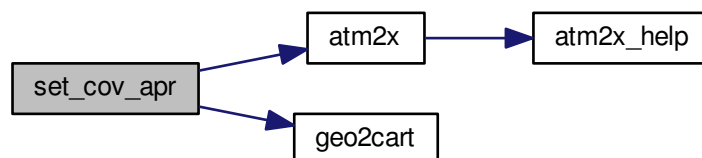


```

01095         cz = ret->err_temp_cz;
01096         ch = ret->err_temp_ch;
01097     }
01098
01099     /* Set correlation lengths for volume mixing ratios... */
01100     for (ig = 0; ig < ctl->ng; ig++)
01101         if (ipa[i] == IDXQ(ig)) {
01102             cz = ret->err_q_cz[ig];
01103             ch = ret->err_q_ch[ig];
01104         }
01105
01106     /* Set correlation lengths for extinction... */
01107     for (iw = 0; iw < ctl->nw; iw++)
01108         if (ipa[i] == IDXK(iw)) {
01109             cz = ret->err_k_cz[iw];
01110             ch = ret->err_k_ch[iw];
01111         }
01112
01113     /* Compute correlations... */
01114     if (cz > 0 && ch > 0) {
01115
01116         /* Get Cartesian coordinates... */
01117         geo2cart(0, atm->lon[ipa[i]], atm->lat[ipa[i]], x0);
01118         geo2cart(0, atm->lon[ipa[j]], atm->lat[ipa[j]], x1);
01119
01120         /* Compute correlations... */
01121         rho =
01122             exp(-DIST(x0, x1) / ch -
01123                 fabs(atm->z[ipa[i]] - atm->z[ipa[j]]) / cz);
01124
01125         /* Set covariance... */
01126         gsl_matrix_set(s_a, i, j, gsl_vector_get(x_a, i)
01127                     * gsl_vector_get(x_a, j) * rho);
01128     }
01129 }
01130
01131 /* Free... */
01132 gsl_vector_free(x_a);
01133 }

```

Here is the call graph for this function:



**5.51.1.12** `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 )`

Set measurement errors.

Definition at line 1137 of file `retrieval.c`.

```

01143     {
01144
01145         static obs_t obs_err;
01146
01147         int id, ir;
01148
01149         size_t i, m;
01150

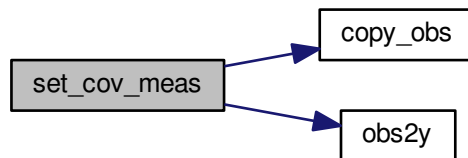
```

```

01151  /* Get size... */
01152  m = sig_eps_inv->size;
01153
01154  /* Noise error (always considered in retrieval fit)... */
01155  copy_obs(ctl, &obs_err, obs, 1);
01156  for (ir = 0; ir < obs_err.nr; ir++)
01157      for (id = 0; id < ctl->nd; id++)
01158          obs_err.rad[id][ir]
01159              = (gsl_finite(obs->rad[id][ir]) ? ret->err_noise[id] : GSL_NAN);
01160  obs2y(ctl, &obs_err, sig_noise, NULL, NULL);
01161
01162  /* Forward model error (always considered in retrieval fit)... */
01163  copy_obs(ctl, &obs_err, obs, 1);
01164  for (ir = 0; ir < obs_err.nr; ir++)
01165      for (id = 0; id < ctl->nd; id++)
01166          obs_err.rad[id][ir]
01167              = fabs(ret->err_formod[id] / 100 * obs->rad[id][ir]);
01168  obs2y(ctl, &obs_err, sig_formod, NULL, NULL);
01169
01170  /* Total error... */
01171  for (i = 0; i < m; i++)
01172      gsl_vector_set(sig_eps_inv, i,
01173                    1 / sqrt(gsl_pow_2(gsl_vector_get(sig_noise, i))
01174                             + gsl_pow_2(gsl_vector_get(sig_formod, i))));
01175
01176  /* Check standard deviations... */
01177  for (i = 0; i < m; i++)
01178      if (gsl_vector_get(sig_eps_inv, i) <= 0)
01179          ERRMSG("Check measurement errors (zero standard deviation)!");
01180 }

```

Here is the call graph for this function:



#### 5.51.1.13 double sza ( double sec, double lon, double lat )

Calculate solar zenith angle.

Definition at line 1184 of file [retrieval.c](#).

```

01187  {
01188
01189  double D, dec, e, g, GMST, h, L, LST, q, ra;
01190
01191  /* Number of days and fraction with respect to 2000-01-01T12:00Z... */
01192  D = sec / 86400 - 0.5;
01193
01194  /* Geocentric apparent ecliptic longitude [rad]... */
01195  g = (357.529 + 0.98560028 * D) * M_PI / 180;
01196  q = 280.459 + 0.98564736 * D;
01197  L = (q + 1.915 * sin(g) + 0.020 * sin(2 * g)) * M_PI / 180;
01198
01199  /* Mean obliquity of the ecliptic [rad]... */
01200  e = (23.439 - 0.00000036 * D) * M_PI / 180;
01201
01202  /* Declination [rad]... */
01203  dec = asin(sin(e) * sin(L));
01204
01205  /* Right ascension [rad]... */

```

```

01206   ra = atan2(cos(e) * sin(L), cos(L));
01207
01208   /* Greenwich Mean Sidereal Time [h]... */
01209   GMST = 18.697374558 + 24.06570982441908 * D;
01210
01211   /* Local Sidereal Time [h]... */
01212   LST = GMST + lon / 15;
01213
01214   /* Hour angle [rad]... */
01215   h = LST / 12 * M_PI - ra;
01216
01217   /* Convert latitude... */
01218   lat *= M_PI / 180;
01219
01220   /* Return solar zenith angle [deg]... */
01221   return acos(sin(lat) * sin(dec) +
01222              cos(lat) * cos(dec) * cos(h)) * 180 / M_PI;
01223 }

```

#### 5.51.1.14 void write\_nc ( char \* filename, ncd\_t \* ncd )

Write to netCDF file...

Definition at line 1227 of file [retrieval.c](#).

```

01229   {
01230
01231   int dimid[10], p_id, t_id, z_id;
01232
01233   /* Create netCDF file... */
01234   printf("Write netCDF file: %s\n", filename);
01235
01236   /* Read existing dimensions... */
01237   NC(nc_inq_dimid(ncd->ncid, "L1_NTRACK", &dimid[0]));
01238   NC(nc_inq_dimid(ncd->ncid, "L1_NXTRACK", &dimid[1]));
01239
01240   /* Set define mode... */
01241   NC(nc_redef(ncd->ncid));
01242
01243   /* Set new dimensions... */
01244   if (nc_inq_dimid(ncd->ncid, "RET_NP", &dimid[2]) != NC_NOERR)
01245       NC(nc_def_dim(ncd->ncid, "RET_NP", (size_t) ncd->np, &dimid[2]));
01246
01247   /* Set new variables... */
01248   add_var(ncd->ncid, "ret_z", "km", "altitude", NC_FLOAT, &dimid[2], &z_id,
01249          1);
01250   add_var(ncd->ncid, "ret_press", "hPa", "pressure", NC_FLOAT, dimid, &p_id,
01251          2);
01252   add_var(ncd->ncid, "ret_temp", "K", "temperature", NC_FLOAT, dimid, &t_id,
01253          3);
01254
01255   /* Leave define mode... */
01256   NC(nc_enddef(ncd->ncid));
01257
01258   /* Write data... */
01259   NC(nc_put_var_float(ncd->ncid, z_id, ncd->ret_z));
01260   NC(nc_put_var_float(ncd->ncid, p_id, ncd->ret_p));
01261   NC(nc_put_var_float(ncd->ncid, t_id, ncd->ret_t));
01262
01263   /* Close netCDF file... */
01264   NC(nc_close(ncd->ncid));
01265 }

```

Here is the call graph for this function:



## 5.51.1.15 int main ( int argc, char \* argv[] )

Definition at line 263 of file [retrieval.c](#).

```

00265         {
00266
00267     static ctl_t  ctl;
00268     static atm_t  atm_apr, atm_clim, atm_i;
00269     static obs_t  obs_i, obs_meas;
00270     static ncd_t  ncd;
00271     static ret_t  ret;
00272
00273     FILE *in;
00274
00275     char filename[LEN];
00276
00277     double chisq, chisq_min, chisq_max, chisq_mean, sx, sy, sza_thresh, z[NP];
00278
00279     int channel[ND], i, id, ip, iz, m, nz, ntask = -1, rank, size,
00280         np0, npl, track, track0, track1, xtrack, xtrack0, xtrack1;
00281
00282     /* -----
00283        Init...
00284        ----- */
00285
00286     /* MPI... */
00287     MPI_Init(&argc, &argv);
00288     MPI_Comm_rank(MPI_COMM_WORLD, &rank);
00289     MPI_Comm_size(MPI_COMM_WORLD, &size);
00290
00291     /* Measure CPU time... */
00292     TIMER("total", 1);
00293
00294     /* Check arguments... */
00295     if (argc < 3)
00296         ERRMSG("Give parameters: <ctl> <filelist>");
00297
00298     /* Read control parameters... */
00299     read_ctl(argc, argv, &ctl);
00300     read_ret_ctl(argc, argv, &ctl, &ret);
00301
00302     /* Read retrieval grid... */
00303     nz = (int) scan_ctl(argc, argv, "NZ", -1, "", NULL);
00304     if (nz > NP)
00305         ERRMSG("Too many altitudes!");
00306     for (iz = 0; iz < nz; iz++)
00307         z[iz] = scan_ctl(argc, argv, "Z", iz, "", NULL);
00308
00309     /* Read track range... */
00310     track0 = (int) scan_ctl(argc, argv, "TRACK_MIN", -1, "0", NULL);
00311     track1 = (int) scan_ctl(argc, argv, "TRACK_MAX", -1, "134", NULL);
00312
00313     /* Read xtrack range... */
00314     xtrack0 = (int) scan_ctl(argc, argv, "XTRACK_MIN", -1, "0", NULL);
00315     xtrack1 = (int) scan_ctl(argc, argv, "XTRACK_MAX", -1, "89", NULL);
00316
00317     /* Read height range... */
00318     np0 = (int) scan_ctl(argc, argv, "NP_MIN", -1, "0", NULL);
00319     npl = (int) scan_ctl(argc, argv, "NP_MAX", -1, "100", NULL);
00320     npl = GSL_MIN(npl, nz - 1);
00321
00322     /* Background smoothing... */
00323     sx = scan_ctl(argc, argv, "SX", -1, "8", NULL);
00324     sy = scan_ctl(argc, argv, "SY", -1, "2", NULL);
00325
00326     /* SZA threshold... */
00327     sza_thresh = scan_ctl(argc, argv, "SZA", -1, "96", NULL);
00328
00329     /* -----
00330        Distribute granules...
00331        ----- */
00332
00333     /* Open filelist... */
00334     printf("Read filelist: %s\n", argv[2]);
00335     if (!(in = fopen(argv[2], "r")))
00336         ERRMSG("Cannot open filelist!");
00337
00338     /* Loop over netCDF files... */
00339     while (fscanf(in, "%s", filename) != EOF) {
00340
00341         /* Distribute files with MPI... */
00342         if ((++ntask) % size != rank)
00343             continue;
00344

```

```

00345      /* Write info... */
00346      printf("Retrieve file %s on rank %d of %d (with %d threads)...\n",
00347             filename, rank + 1, size, omp_get_max_threads());
00348
00349      /* -----
00350      Initialize retrieval...
00351      ----- */
00352
00353      /* Read netCDF file... */
00354      read_nc(filename, &ncd);
00355
00356      /* Identify radiance channels... */
00357      for (id = 0; id < ctl.nd; id++) {
00358          channel[id] = -999;
00359          for (i = 0; i < L1_NCHAN; i++)
00360              if (fabs(ctl.nu[id] - ncd.ll_nu[i]) < 0.1)
00361                  channel[id] = i;
00362          if (channel[id] < 0)
00363              ERRMSG("Cannot identify radiance channel!");
00364      }
00365
00366      /* Fill data gaps... */
00367      fill_gaps(ncd.ll_t, sx, sy);
00368      fill_gaps(ncd.ll_z, sx, sy);
00369
00370      /* Set climatological data for center of granule... */
00371      atm_clim.np = nz;
00372      for (iz = 0; iz < nz; iz++)
00373          atm_clim.z[iz] = z[iz];
00374      climatology(&ctl, &atm_clim);
00375
00376      /* -----
00377      Retrieval...
00378      ----- */
00379
00380      /* Get chi^2 statistics... */
00381      chisq_min = 1e100;
00382      chisq_max = -1e100;
00383      chisq_mean = 0;
00384      m = 0;
00385
00386      /* Loop over swaths... */
00387      for (track = track0; track <= track1; track++) {
00388
00389          /* Measure CPU time... */
00390          TIMER("retrieval", 1);
00391
00392          /* Loop over scan... */
00393          for (xtrack = xtrack0; xtrack <= xtrack1; xtrack++) {
00394
00395              /* Store observation data... */
00396              obs_meas.nr = 1;
00397              obs_meas.time[0] = ncd.ll_time[track][xtrack];
00398              obs_meas.obsz[0] = ncd.ll_sat_z[track];
00399              obs_meas.obslon[0] = ncd.ll_sat_lon[track];
00400              obs_meas.obsplat[0] = ncd.ll_sat_lat[track];
00401              obs_meas.vplon[0] = ncd.ll_lon[track][xtrack];
00402              obs_meas.vplat[0] = ncd.ll_lat[track][xtrack];
00403              for (id = 0; id < ctl.nd; id++)
00404                  obs_meas.rad[id][0] = ncd.ll_rad[track][xtrack][channel[id]];
00405
00406              /* Flag out 4 micron channels for daytime measurements... */
00407              if (sza(obs_meas.time[0], obs_meas.obslon[0], obs_meas.
00408                  obsplat[0])
00409                  < sza_thresh)
00410                  for (id = 0; id < ctl.nd; id++)
00411                      if (ctl.nu[id] >= 2000)
00412                          obs_meas.rad[id][0] = GSL_NAN;
00413
00414              /* Prepare atmospheric data... */
00415              copy_atm(&ctl, &atm_apr, &atm_clim, 0);
00416              for (ip = 0; ip < atm_apr.np; ip++) {
00417                  atm_apr.time[ip] = obs_meas.time[0];
00418                  atm_apr.lon[ip] = obs_meas.vplon[0];
00419                  atm_apr.lat[ip] = obs_meas.vplat[0];
00420              }
00421
00422              /* Merge Level-2 data... */
00423              init_l2(&ncd, track, xtrack, &ctl, &atm_apr);
00424
00425              /* Retrieval... */
00426              optimal_estimation(&ret, &ctl, &obs_meas, &obs_i,
00427                               &atm_apr, &atm_i, &chisq);
00428
00429              /* Get chi^2 statistics... */
00430              if (gsl_finite(chisq)) {
00431                  chisq_min = GSL_MIN(chisq_min, chisq);
00432              }
00433          }
00434      }

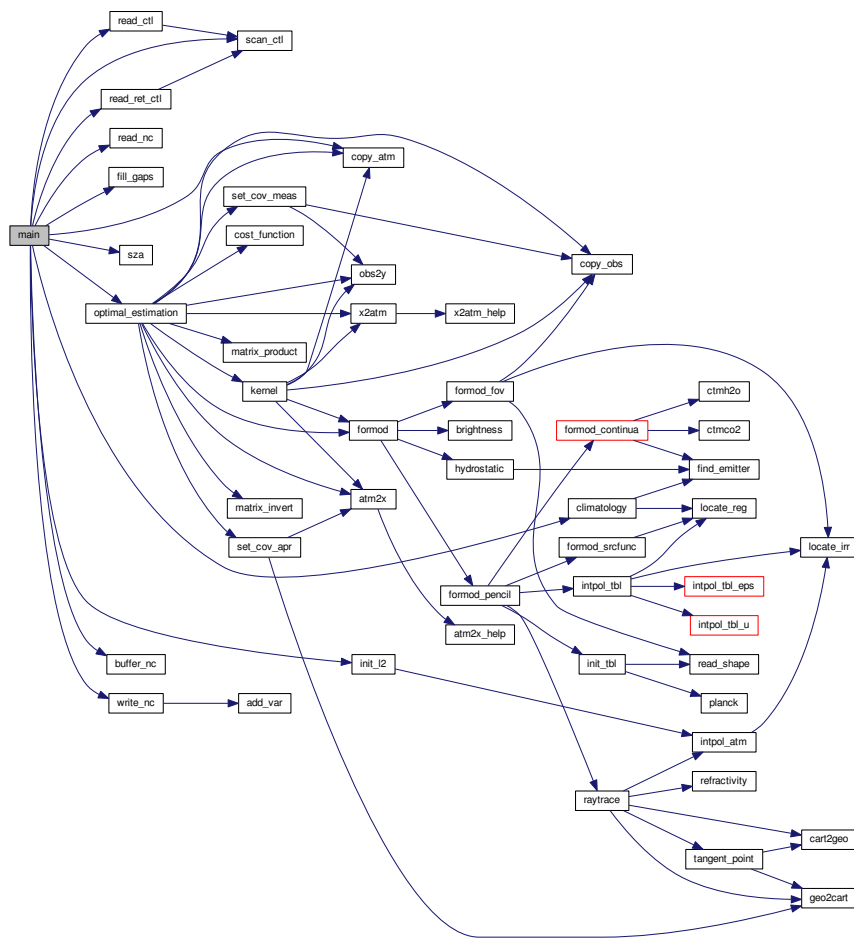
```

```

00431         chisq_max = GSL_MAX(chisq_max, chisq);
00432         chisq_mean += chisq;
00433         m++;
00434     }
00435
00436     /* Buffer results... */
00437     buffer_nc(&atm_i, chisq, &ncd, track, xtrack, np0, np1);
00438 }
00439
00440 /* Measure CPU time... */
00441 TIMER("retrieval", 3);
00442 }
00443
00444 /* -----
00445    Finalize...
00446    ----- */
00447
00448 /* Write netCDF file... */
00449 write_nc(filename, &ncd);
00450
00451 /* Write info... */
00452 printf("chi^2: min= %g / mean= %g / max= %g / m= %d\n",
00453        chisq_min, chisq_mean / m, chisq_max, m);
00454 printf("Retrieval finished on rank %d of %d!\n", rank, size);
00455 }
00456
00457 /* Close file list... */
00458 fclose(in);
00459
00460 /* Measure CPU time... */
00461 TIMER("total", 3);
00462
00463 /* Report memory usage... */
00464 printf("MEMORY_ATM = %g MByte\n", 4. * sizeof(atm_t) / 1024. / 1024.);
00465 printf("MEMORY_CTL = %g MByte\n", 1. * sizeof(ctl_t) / 1024. / 1024.);
00466 printf("MEMORY_NCD = %g MByte\n", 1. * sizeof(ncd_t) / 1024. / 1024.);
00467 printf("MEMORY_OBS = %g MByte\n", 3. * sizeof(atm_t) / 1024. / 1024.);
00468 printf("MEMORY_RET = %g MByte\n", 1. * sizeof(ret_t) / 1024. / 1024.);
00469 printf("MEMORY_TBL = %g MByte\n", 1. * sizeof(tbl_t) / 1024. / 1024.);
00470
00471 /* Report problem size... */
00472 printf("SIZE_TASKS = %d\n", size);
00473 printf("SIZE_THREADS = %d\n", omp_get_max_threads());
00474
00475 /* MPI... */
00476 MPI_Finalize();
00477
00478 return EXIT_SUCCESS;
00479 }

```

Here is the call graph for this function:



## 5.52 retrieval.c

```

00001 #include <mpi.h>
00002 #include <omp.h>
00003 #include <netcdf.h>
00004 #include "jurassic.h"
00005
00006 /* -----
00007     Macros...
00008     ----- */
00009
00011 #define NC(cmd) {
00012     if ((cmd) != NC_NOERR)
00013         ERRMSG(nc_strerror(cmd));
00014 }
00015
00016 /* -----
00017     Dimensions...
00018     ----- */
00019
00021 #define L1_NCHAN 34
00022
00024 #define L1_NTRACK 135
00025
00027 #define L1_NXTRACK 90
00028
00030 #define L2_NLAY 27
00031
00033 #define L2_NTRACK 45
00034

```

```

00036 #define L2_NXTRACK 30
00037
00038 /* -----
00039     Structs...
00040     ----- */
00041
00042 typedef struct {
00043     int ncid;
00044     int np;
00045     double l1_time[L1_NTRACK][L1_NXTRACK];
00046     double l1_lon[L1_NTRACK][L1_NXTRACK];
00047     double l1_lat[L1_NTRACK][L1_NXTRACK];
00048     double l1_sat_z[L1_NTRACK];
00049     double l1_sat_lon[L1_NTRACK];
00050     double l1_sat_lat[L1_NTRACK];
00051     double l1_nu[L1_NCHAN];
00052     float l1_rad[L1_NTRACK][L1_NXTRACK][L1_NCHAN];
00053     double l2_z[L2_NTRACK][L2_NXTRACK][L2_NLAY];
00054     double l2_p[L2_NLAY];
00055     double l2_t[L2_NTRACK][L2_NXTRACK][L2_NLAY];
00056     float ret_z[NP];
00057     float ret_p[L1_NTRACK * L1_NXTRACK];
00058     float ret_t[L1_NTRACK * L1_NXTRACK * NP];
00059 } ncd_t;
00060
00061 typedef struct {
00062     int kernel_recomp;
00063     int conv_itmax;
00064     double conv_dmin;
00065     double err_formod[ND];
00066     double err_noise[ND];
00067     double err_press;
00068     double err_press_cz;
00069     double err_press_ch;
00070     double err_temp;
00071     double err_temp_cz;
00072     double err_temp_ch;
00073     double err_q[NG];
00074     double err_q_cz[NG];
00075     double err_q_ch[NG];
00076     double err_k[NW];
00077     double err_k_cz[NW];
00078     double err_k_ch[NW];
00079 } ret_t;
00080
00081 /* -----
00082     Functions...
00083     ----- */
00084
00085 void add_var(
00086     int ncid,
00087     const char *varname,

```



```
00159     const char *unit,
00160     const char *longname,
00161     int type,
00162     int dimid[],
00163     int *varid,
00164     int ndims);
00165
00166 void buffer_nc(
00167     atm_t * atm,
00168     double chisq,
00169     ncd_t * ncd,
00170     int track,
00171     int xtrack,
00172     int np0,
00173     int np1);
00174
00175 double cost_function(
00176     gsl_vector * dx,
00177     gsl_vector * dy,
00178     gsl_matrix * s_a_inv,
00179     gsl_vector * sig_eps_inv);
00180
00181 void fill_gaps(
00182     double x[L2_NTRACK][L2_NXTRACK][L2_NLAY],
00183     double cx,
00184     double cy);
00185
00186 void init_l2(
00187     ncd_t * ncd,
00188     int track,
00189     int xtrack,
00190     ctl_t * ctl,
00191     atm_t * atm);
00192
00193 void matrix_invert(
00194     gsl_matrix * a);
00195
00196 void matrix_product(
00197     gsl_matrix * a,
00198     gsl_vector * b,
00199     int transpose,
00200     gsl_matrix * c);
00201
00202 void optimal_estimation(
00203     ret_t * ret,
00204     ctl_t * ctl,
00205     obs_t * obs_meas,
00206     obs_t * obs_i,
00207     atm_t * atm_apr,
00208     atm_t * atm_i,
00209     double *chisq);
00210
00211 void read_nc(
00212     char *filename,
00213     ncd_t * ncd);
00214
00215 void read_ret_ctl(
00216     int argc,
00217     char *argv[],
00218     ctl_t * ctl,
00219     ret_t * ret);
00220
00221 void set_cov_apr(
00222     ret_t * ret,
00223     ctl_t * ctl,
00224     atm_t * atm,
00225     int *iqa,
00226     int *ipa,
00227     gsl_matrix * s_a);
00228
00229 void set_cov_meas(
00230     ret_t * ret,
00231     ctl_t * ctl,
00232     obs_t * obs,
00233     gsl_vector * sig_noise,
00234     gsl_vector * sig_formod,
00235     gsl_vector * sig_eps_inv);
00236
00237 double sza(
00238     double sec,
00239     double lon,
00240     double lat);
00241
00242 void write_nc(
00243     char *filename,
00244     ncd_t * ncd);
00245
```

```

00259 /* -----
00260     Main...
00261     ----- */
00262
00263 int main(
00264     int argc,
00265     char *argv[]) {
00266
00267     static ctl_t ctl;
00268     static atm_t atm_apr, atm_clim, atm_i;
00269     static obs_t obs_i, obs_meas;
00270     static ncd_t ncd;
00271     static ret_t ret;
00272
00273     FILE *in;
00274
00275     char filename[LEN];
00276
00277     double chisq, chisq_min, chisq_max, chisq_mean, sx, sy, sza_thresh, z[NP];
00278
00279     int channel[ND], i, id, ip, iz, m, nz, ntask = -1, rank, size,
00280         np0, npl, track, track0, track1, xtrack, xtrack0, xtrack1;
00281
00282     /* -----
00283         Init...
00284         ----- */
00285
00286     /* MPI... */
00287     MPI_Init(&argc, &argv);
00288     MPI_Comm_rank(MPI_COMM_WORLD, &rank);
00289     MPI_Comm_size(MPI_COMM_WORLD, &size);
00290
00291     /* Measure CPU time... */
00292     TIMER("total", 1);
00293
00294     /* Check arguments... */
00295     if (argc < 3)
00296         ERRMSG("Give parameters: <ctl> <filelist>");
00297
00298     /* Read control parameters... */
00299     read_ctl(argc, argv, &ctl);
00300     read_ret_ctl(argc, argv, &ctl, &ret);
00301
00302     /* Read retrieval grid... */
00303     nz = (int) scan_ctl(argc, argv, "NZ", -1, "", NULL);
00304     if (nz > NP)
00305         ERRMSG("Too many altitudes!");
00306     for (iz = 0; iz < nz; iz++)
00307         z[iz] = scan_ctl(argc, argv, "Z", iz, "", NULL);
00308
00309     /* Read track range... */
00310     track0 = (int) scan_ctl(argc, argv, "TRACK_MIN", -1, "0", NULL);
00311     track1 = (int) scan_ctl(argc, argv, "TRACK_MAX", -1, "134", NULL);
00312
00313     /* Read xtrack range... */
00314     xtrack0 = (int) scan_ctl(argc, argv, "XTRACK_MIN", -1, "0", NULL);
00315     xtrack1 = (int) scan_ctl(argc, argv, "XTRACK_MAX", -1, "89", NULL);
00316
00317     /* Read height range... */
00318     np0 = (int) scan_ctl(argc, argv, "NP_MIN", -1, "0", NULL);
00319     npl = (int) scan_ctl(argc, argv, "NP_MAX", -1, "100", NULL);
00320     npl = GSL_MIN(npl, nz - 1);
00321
00322     /* Background smoothing... */
00323     sx = scan_ctl(argc, argv, "SX", -1, "8", NULL);
00324     sy = scan_ctl(argc, argv, "SY", -1, "2", NULL);
00325
00326     /* SZA threshold... */
00327     sza_thresh = scan_ctl(argc, argv, "SZA", -1, "96", NULL);
00328
00329     /* -----
00330         Distribute granules...
00331         ----- */
00332
00333     /* Open filelist... */
00334     printf("Read filelist: %s\n", argv[2]);
00335     if (!(in = fopen(argv[2], "r")))
00336         ERRMSG("Cannot open filelist!");
00337
00338     /* Loop over netCDF files... */
00339     while (fscanf(in, "%s", filename) != EOF) {
00340
00341         /* Distribute files with MPI... */
00342         if ((++ntask) % size != rank)
00343             continue;
00344
00345         /* Write info... */

```

```

00346     printf("Retrieve file %s on rank %d of %d (with %d threads)...\n",
00347           filename, rank + 1, size, omp_get_max_threads());
00348
00349     /* -----
00350     Initialize retrieval...
00351     ----- */
00352
00353     /* Read netCDF file... */
00354     read_nc(filename, &ncd);
00355
00356     /* Identify radiance channels... */
00357     for (id = 0; id < ctl.nd; id++) {
00358         channel[id] = -999;
00359         for (i = 0; i < Ll_NCHAN; i++)
00360             if (fabs(ctl.nu[id] - ncd.ll_nu[i]) < 0.1)
00361                 channel[id] = i;
00362         if (channel[id] < 0)
00363             ERRMSG("Cannot identify radiance channel!");
00364     }
00365
00366     /* Fill data gaps... */
00367     fill_gaps(ncd.ll_t, sx, sy);
00368     fill_gaps(ncd.ll_z, sx, sy);
00369
00370     /* Set climatological data for center of granule... */
00371     atm_clim.np = nz;
00372     for (iz = 0; iz < nz; iz++)
00373         atm_clim.z[iz] = z[iz];
00374     climatology(&ctl, &atm_clim);
00375
00376     /* -----
00377     Retrieval...
00378     ----- */
00379
00380     /* Get chi^2 statistics... */
00381     chisq_min = 1e100;
00382     chisq_max = -1e100;
00383     chisq_mean = 0;
00384     m = 0;
00385
00386     /* Loop over swaths... */
00387     for (track = track0; track <= track1; track++) {
00388
00389         /* Measure CPU time... */
00390         TIMER("retrieval", 1);
00391
00392         /* Loop over scan... */
00393         for (xtrack = xtrack0; xtrack <= xtrack1; xtrack++) {
00394
00395             /* Store observation data... */
00396             obs_meas.nr = 1;
00397             obs_meas.time[0] = ncd.ll_time[track][xtrack];
00398             obs_meas.obsz[0] = ncd.ll_sat_z[track];
00399             obs_meas.obslon[0] = ncd.ll_sat_lon[track];
00400             obs_meas.obsLAT[0] = ncd.ll_sat_lat[track];
00401             obs_meas.vplon[0] = ncd.ll_lon[track][xtrack];
00402             obs_meas.vplat[0] = ncd.ll_lat[track][xtrack];
00403             for (id = 0; id < ctl.nd; id++)
00404                 obs_meas.rad[id][0] = ncd.ll_rad[track][xtrack][channel[id]];
00405
00406             /* Flag out 4 micron channels for daytime measurements... */
00407             if (sza(obs_meas.time[0], obs_meas.obslon[0], obs_meas.
obsLAT[0])
00408                 < sza_thresh)
00409                 for (id = 0; id < ctl.nd; id++)
00410                     if (ctl.nu[id] >= 2000)
00411                         obs_meas.rad[id][0] = GSL_NAN;
00412
00413             /* Prepare atmospheric data... */
00414             copy_atm(&ctl, &atm_apr, &atm_clim, 0);
00415             for (ip = 0; ip < atm_apr.np; ip++) {
00416                 atm_apr.time[ip] = obs_meas.time[0];
00417                 atm_apr.lon[ip] = obs_meas.vplon[0];
00418                 atm_apr.lat[ip] = obs_meas.vplat[0];
00419             }
00420
00421             /* Merge Level-2 data... */
00422             init_l2(&ncd, track, xtrack, &ctl, &atm_apr);
00423
00424             /* Retrieval... */
00425             optimal_estimation(&ret, &ctl, &obs_meas, &obs_i,
00426                               &atm_apr, &atm_i, &chisq);
00427
00428             /* Get chi^2 statistics... */
00429             if (gsl_finite(chisq)) {
00430                 chisq_min = GSL_MIN(chisq_min, chisq);
00431                 chisq_max = GSL_MAX(chisq_max, chisq);

```

```

00432         chisq_mean += chisq;
00433         m++;
00434     }
00435
00436     /* Buffer results... */
00437     buffer_nc(&atm_i, chisq, &ncd, track, xtrack, np0, npl);
00438 }
00439
00440 /* Measure CPU time... */
00441 TIMER("retrieval", 3);
00442 }
00443
00444 /* -----
00445     Finalize...
00446     ----- */
00447
00448 /* Write netCDF file... */
00449 write_nc(filename, &ncd);
00450
00451 /* Write info... */
00452 printf("chi^2: min= %g / mean= %g / max= %g / m= %d\n",
00453        chisq_min, chisq_mean / m, chisq_max, m);
00454 printf("Retrieval finished on rank %d of %d!\n", rank, size);
00455 }
00456
00457 /* Close file list... */
00458 fclose(in);
00459
00460 /* Measure CPU time... */
00461 TIMER("total", 3);
00462
00463 /* Report memory usage... */
00464 printf("MEMORY_ATM = %g MByte\n", 4. * sizeof(atm_t) / 1024. / 1024.);
00465 printf("MEMORY_CTL = %g MByte\n", 1. * sizeof(ctl_t) / 1024. / 1024.);
00466 printf("MEMORY_NCD = %g MByte\n", 1. * sizeof(ncd_t) / 1024. / 1024.);
00467 printf("MEMORY_OBS = %g MByte\n", 3. * sizeof(atm_t) / 1024. / 1024.);
00468 printf("MEMORY_RET = %g MByte\n", 1. * sizeof(ret_t) / 1024. / 1024.);
00469 printf("MEMORY_TBL = %g MByte\n", 1. * sizeof(tbl_t) / 1024. / 1024.);
00470
00471 /* Report problem size... */
00472 printf("SIZE_TASKS = %d\n", size);
00473 printf("SIZE_THREADS = %d\n", omp_get_max_threads());
00474
00475 /* MPI... */
00476 MPI_Finalize();
00477
00478 return EXIT_SUCCESS;
00479 }
00480
00481 /*****
00482
00483 void add_var(
00484     int ncid,
00485     const char *varname,
00486     const char *unit,
00487     const char *longname,
00488     int type,
00489     int dimid[],
00490     int *varid,
00491     int ndims) {
00492
00493     /* Check if variable exists... */
00494     if (nc_inq_varid(ncid, varname, varid) != NC_NOERR) {
00495
00496         /* Define variable... */
00497         NC(nc_def_var(ncid, varname, type, ndims, dimid, varid));
00498
00499         /* Set long name... */
00500         NC(nc_put_att_text
00501            (ncid, *varid, "long_name", strlen(longname), longname));
00502
00503         /* Set units... */
00504         NC(nc_put_att_text(ncid, *varid, "units", strlen(unit), unit));
00505     }
00506 }
00507
00508 /*****
00509
00510 void buffer_nc(
00511     atm_t * atm,
00512     double chisq,
00513     ncd_t * ncd,
00514     int track,
00515     int xtrack,
00516     int np0,
00517     int npl) {
00518

```

```

00519     int ip;
00520
00521     /* Set number of data points... */
00522     ncd->np = np1 - np0 + 1;
00523
00524     /* Save retrieval data... */
00525     for (ip = np0; ip <= np1; ip++) {
00526         ncd->ret_z[ip - np0] = (float) atm->z[ip];
00527         ncd->ret_p[track * L1_NXTRACK + xtrack] = (float) atm->p[np0];
00528         ncd->ret_t[(track * L1_NXTRACK + xtrack) * ncd->np + ip - np0] =
00529             (gsl_finite(chisq) ? (float) atm->t[ip] : GSL_NAN);
00530     }
00531 }
00532
00533 /*****
00534
00535 double cost_function(
00536     gsl_vector * dx,
00537     gsl_vector * dy,
00538     gsl_matrix * s_a_inv,
00539     gsl_vector * sig_eps_inv) {
00540
00541     gsl_vector *x_aux, *y_aux;
00542
00543     double chisq_a, chisq_m = 0;
00544
00545     size_t i, m, n;
00546
00547     /* Get sizes... */
00548     m = dy->size;
00549     n = dx->size;
00550
00551     /* Allocate... */
00552     x_aux = gsl_vector_alloc(n);
00553     y_aux = gsl_vector_alloc(m);
00554
00555     /* Determine normalized cost function...
00556        (chi^2 = 1/m * [dy^T * S_eps^{-1} * dy + dx^T * S_a^{-1} * dx]) */
00557     for (i = 0; i < m; i++)
00558         chisq_m +=
00559             gsl_pow_2(gsl_vector_get(dy, i) * gsl_vector_get(sig_eps_inv, i));
00560     gsl_blas_dgemv(CblasNoTrans, 1.0, s_a_inv, dx, 0.0, x_aux);
00561     gsl_blas_ddot(dx, x_aux, &chisq_a);
00562
00563     /* Free... */
00564     gsl_vector_free(x_aux);
00565     gsl_vector_free(y_aux);
00566
00567     /* Return cost function value... */
00568     return (chisq_m + chisq_a) / (double) m;
00569 }
00570
00571 /*****
00572
00573 void fill_gaps(
00574     double x[L2_NTRACK][L2_NXTRACK][L2_NLAY],
00575     double cx,
00576     double cy) {
00577
00578     double help[L2_NTRACK][L2_NXTRACK], w, wsum;
00579
00580     int lay, track, track2, xtrack, xtrack2;
00581
00582     /* Loop over layers... */
00583     for (lay = 0; lay < L2_NLAY; lay++) {
00584
00585         /* Loop over grid points... */
00586         for (xtrack = 0; xtrack < L2_NXTRACK; xtrack++)
00587             for (xtrack2 = 0; xtrack2 < L2_NXTRACK; xtrack2++) {
00588
00589                 /* Init... */
00590                 help[track][xtrack] = 0;
00591                 wsum = 0;
00592
00593                 /* Average data points... */
00594                 for (track2 = 0; track2 < L2_NTRACK; track2++)
00595                     for (xtrack2 = 0; xtrack2 < L2_NXTRACK; xtrack2++)
00596                         if (gsl_finite(x[track2][xtrack2][lay])
00597                             && x[track2][xtrack2][lay] > 0) {
00598                             w = exp(-gsl_pow_2((xtrack - xtrack2) / cx)
00599                                     - gsl_pow_2((track - track2) / cy));
00600                             help[track][xtrack] += w * x[track2][xtrack2][lay];
00601                             wsum += w;
00602                         }
00603
00604                 /* Normalize... */
00605                 if (wsum > 0)

```

```

00606         help[track][xtrack] /= wsum;
00607     else
00608         help[track][xtrack] = GSL_NAN;
00609     }
00610
00611     /* Copy grid points... */
00612     for (track = 0; track < L2_NTRACK; track++)
00613         for (xtrack = 0; xtrack < L2_NXTRACK; xtrack++)
00614             x[track][xtrack][lay] = help[track][xtrack];
00615     }
00616 }
00617
00618 /*****
00619
00620 void init_l2(
00621     ncd_t * ncd,
00622     int track,
00623     int xtrack,
00624     ctl_t * ctl,
00625     atm_t * atm) {
00626
00627     static atm_t atm_airs;
00628
00629     double k[NW], p, q[NG], t, w, zmax = 0, zmin = 1000;
00630
00631     int ip, lay;
00632
00633     /* Reset track- and xtrack-index to match Level-2 data... */
00634     track /= 3;
00635     xtrack /= 3;
00636
00637     /* Store AIRS data in atmospheric data struct... */
00638     atm_airs.np = 0;
00639     for (lay = 0; lay < L2_NLAY; lay++)
00640         if (gsl_finite(ncd->l2_z[track][xtrack][lay])) {
00641             atm_airs.z[atm_airs.np] = ncd->l2_z[track][xtrack][lay];
00642             atm_airs.p[atm_airs.np] = ncd->l2_p[lay];
00643             atm_airs.t[atm_airs.np] = ncd->l2_t[track][xtrack][lay];
00644             if (++atm_airs.np > NP)
00645                 ERRMSG("Too many layers!");
00646         }
00647
00648     /* Check number of levels... */
00649     if (atm_airs.np <= 0)
00650         return;
00651
00652     /* Get height range of AIRS data... */
00653     for (ip = 0; ip < atm_airs.np; ip++) {
00654         zmax = GSL_MAX(zmax, atm_airs.z[ip]);
00655         zmin = GSL_MIN(zmin, atm_airs.z[ip]);
00656     }
00657
00658     /* Merge AIRS data... */
00659     for (ip = 0; ip < atm->np; ip++) {
00660
00661         /* Interpolate AIRS data... */
00662         intpol_atm(ctl, &atm_airs, atm->z[ip], &p, &t, q, k);
00663
00664         /* Weighting factor... */
00665         w = 1;
00666         if (atm->z[ip] > zmax)
00667             w = GSL_MAX(1 - (atm->z[ip] - zmax) / 50, 0);
00668         if (atm->z[ip] < zmin)
00669             w = GSL_MAX(1 - (zmin - atm->z[ip]) / 50, 0);
00670
00671         /* Merge... */
00672         atm->t[ip] = w * t + (1 - w) * atm->t[ip];
00673         atm->p[ip] = w * p + (1 - w) * atm->p[ip];
00674     }
00675 }
00676
00677 /*****
00678
00679 void matrix_invert(
00680     gsl_matrix * a) {
00681
00682     size_t diag = 1, i, j, n;
00683
00684     /* Get size... */
00685     n = a->size1;
00686
00687     /* Check if matrix is diagonal... */
00688     for (i = 0; i < n && diag; i++)
00689         for (j = i + 1; j < n; j++)
00690             if (gsl_matrix_get(a, i, j) != 0) {
00691                 diag = 0;
00692                 break;

```

```

00693     }
00694
00695     /* Quick inversion of diagonal matrix... */
00696     if (diag)
00697         for (i = 0; i < n; i++)
00698             gsl_matrix_set(a, i, i, 1 / gsl_matrix_get(a, i, i));
00699
00700     /* Matrix inversion by means of Cholesky decomposition... */
00701     else {
00702         gsl_linalg_cholesky_decomp(a);
00703         gsl_linalg_cholesky_invert(a);
00704     }
00705 }
00706
00707 /*****
00708
00709 void matrix_product(
00710     gsl_matrix * a,
00711     gsl_vector * b,
00712     int transpose,
00713     gsl_matrix * c) {
00714
00715     gsl_matrix *aux;
00716
00717     size_t i, j, m, n;
00718
00719     /* Set sizes... */
00720     m = a->size1;
00721     n = a->size2;
00722
00723     /* Allocate... */
00724     aux = gsl_matrix_alloc(m, n);
00725
00726     /* Compute A^T B A... */
00727     if (transpose == 1) {
00728
00729         /* Compute B^1/2 A... */
00730         for (i = 0; i < m; i++)
00731             for (j = 0; j < n; j++)
00732                 gsl_matrix_set(aux, i, j,
00733                     gsl_vector_get(b, i) * gsl_matrix_get(a, i, j));
00734
00735         /* Compute A^T B A = (B^1/2 A)^T (B^1/2 A)... */
00736         gsl_blas_dgemm(CblasTrans, CblasNoTrans, 1.0, aux, aux, 0.0, c);
00737     }
00738
00739     /* Compute A B A^T... */
00740     else if (transpose == 2) {
00741
00742         /* Compute A B^1/2... */
00743         for (i = 0; i < m; i++)
00744             for (j = 0; j < n; j++)
00745                 gsl_matrix_set(aux, i, j,
00746                     gsl_matrix_get(a, i, j) * gsl_vector_get(b, j));
00747
00748         /* Compute A B A^T = (A B^1/2) (A B^1/2)^T... */
00749         gsl_blas_dgemm(CblasNoTrans, CblasTrans, 1.0, aux, aux, 0.0, c);
00750     }
00751
00752     /* Free... */
00753     gsl_matrix_free(aux);
00754 }
00755
00756 /*****
00757
00758 void optimal_estimation(
00759     ret_t * ret,
00760     ctl_t * ctl,
00761     obs_t * obs_meas,
00762     obs_t * obs_i,
00763     atm_t * atm_apr,
00764     atm_t * atm_i,
00765     double *chisq) {
00766
00767     static int ipa[N], iqa[N];
00768
00769     gsl_matrix *a, *cov, *k_i, *s_a_inv;
00770     gsl_vector *b, *dx, *dy, *sig_eps_inv, *sig_formod, *sig_noise,
00771         *x_a, *x_i, *x_step, *y_aux, *y_i, *y_m;
00772
00773     double chisq_old, disq = 0, lmpar = 0.001;
00774
00775     int ig, ip, it = 0, it2, iw;
00776
00777     size_t i, m, n;
00778
00779     /* -----

```

```

00780     Initialize...
00781     ----- */
00782
00783     /* Get sizes... */
00784     m = obs2y(ctl, obs_meas, NULL, NULL, NULL);
00785     n = atm2x(ctl, atm_apr, NULL, iqa, ipa);
00786     if (m <= 0 || n <= 0) {
00787         *chisq = GSL_NAN;
00788         return;
00789     }
00790
00791     /* Allocate... */
00792     a = gsl_matrix_alloc(n, n);
00793     cov = gsl_matrix_alloc(n, n);
00794     k_i = gsl_matrix_alloc(m, n);
00795     s_a_inv = gsl_matrix_alloc(n, n);
00796
00797     b = gsl_vector_alloc(n);
00798     dx = gsl_vector_alloc(n);
00799     dy = gsl_vector_alloc(m);
00800     sig_eps_inv = gsl_vector_alloc(m);
00801     sig_formod = gsl_vector_alloc(m);
00802     sig_noise = gsl_vector_alloc(m);
00803     x_a = gsl_vector_alloc(n);
00804     x_i = gsl_vector_alloc(n);
00805     x_step = gsl_vector_alloc(n);
00806     y_aux = gsl_vector_alloc(m);
00807     y_i = gsl_vector_alloc(m);
00808     y_m = gsl_vector_alloc(m);
00809
00810     /* Set initial state... */
00811     copy_atm(ctl, atm_i, atm_apr, 0);
00812     copy_obs(ctl, obs_i, obs_meas, 0);
00813     formod(ctl, atm_i, obs_i);
00814
00815     /* Set state vectors and observation vectors... */
00816     atm2x(ctl, atm_apr, x_a, NULL, NULL);
00817     atm2x(ctl, atm_i, x_i, NULL, NULL);
00818     obs2y(ctl, obs_meas, y_m, NULL, NULL);
00819     obs2y(ctl, obs_i, y_i, NULL, NULL);
00820
00821     /* Set inverse a priori covariance S_a^-1... */
00822     set_cov_apr(ret, ctl, atm_apr, iqa, ipa, s_a_inv);
00823     matrix_invert(s_a_inv);
00824
00825     /* Get measurement errors... */
00826     set_cov_meas(ret, ctl, obs_meas, sig_noise, sig_formod, sig_eps_inv);
00827
00828     /* Determine dx = x_i - x_a and dy = y - F(x_i) ... */
00829     gsl_vector_memcpy(dx, x_i);
00830     gsl_vector_sub(dx, x_a);
00831     gsl_vector_memcpy(dy, y_m);
00832     gsl_vector_sub(dy, y_i);
00833
00834     /* Compute cost function... */
00835     *chisq = cost_function(dx, dy, s_a_inv, sig_eps_inv);
00836
00837     /* Compute initial kernel... */
00838     kernel(ctl, atm_i, obs_i, k_i);
00839
00840     /* -----
00841        Levenberg-Marquardt minimization...
00842        ----- */
00843
00844     /* Outer loop... */
00845     for (it = 1; it <= ret->conv_itmax; it++) {
00846
00847         /* Store current cost function value... */
00848         chisq_old = *chisq;
00849
00850         /* Compute kernel matrix K_i... */
00851         if (it > 1 && it % ret->kernel_recomp == 0)
00852             kernel(ctl, atm_i, obs_i, k_i);
00853
00854         /* Compute K_i^T * S_eps^{-1} * K_i ... */
00855         if (it == 1 || it % ret->kernel_recomp == 0)
00856             matrix_product(k_i, sig_eps_inv, 1, cov);
00857
00858         /* Determine b = K_i^T * S_eps^{-1} * dy - S_a^{-1} * dx ... */
00859         for (i = 0; i < m; i++)
00860             gsl_vector_set(y_aux, i, gsl_vector_get(dy, i)
00861                 * gsl_pow_2(gsl_vector_get(sig_eps_inv, i)));
00862         gsl_blas_dgemv(CblasTrans, 1.0, k_i, y_aux, 0.0, b);
00863         gsl_blas_dgemv(CblasNoTrans, -1.0, s_a_inv, dx, 1.0, b);
00864
00865         /* Inner loop... */
00866         for (it2 = 0; it2 < 20; it2++) {

```



```

00867
00868 /* Compute A = (1 + lmpar) * S_a^{-1} + K_i^T * S_eps^{-1} * K_i ... */
00869 gsl_matrix_memcpy(a, s_a_inv);
00870 gsl_matrix_scale(a, 1 + lmpar);
00871 gsl_matrix_add(a, cov);
00872
00873 /* Solve A * x_step = b by means of Cholesky decomposition... */
00874 gsl_linalg_cholesky_decomp(a);
00875 gsl_linalg_cholesky_solve(a, b, x_step);
00876
00877 /* Update atmospheric state... */
00878 gsl_vector_add(x_i, x_step);
00879 copy_atm(ctl, atm_i, atm_apr, 0);
00880 copy_obs(ctl, obs_i, obs_meas, 0);
00881 x2atm(ctl, x_i, atm_i);
00882
00883 /* Check atmospheric state... */
00884 for (ip = 0; ip < atm_i->np; ip++) {
00885     atm_i->p[ip] = GSL_MIN(GSL_MAX(atm_i->p[ip], 5e-7), 5e4);
00886     atm_i->t[ip] = GSL_MIN(GSL_MAX(atm_i->t[ip], 100), 400);
00887     for (ig = 0; ig < ctl->ng; ig++)
00888         atm_i->q[ig][ip] = GSL_MIN(GSL_MAX(atm_i->q[ig][ip], 0), 1);
00889     for (iw = 0; iw < ctl->nw; iw++)
00890         atm_i->k[iw][ip] = GSL_MAX(atm_i->k[iw][ip], 0);
00891 }
00892
00893 /* Forward calculation... */
00894 formod(ctl, atm_i, obs_i);
00895 obs2y(ctl, obs_i, y_i, NULL, NULL);
00896
00897 /* Determine dx = x_i - x_a and dy = y - F(x_i) ... */
00898 gsl_vector_memcpy(dx, x_i);
00899 gsl_vector_sub(dx, x_a);
00900 gsl_vector_memcpy(dy, y_m);
00901 gsl_vector_sub(dy, y_i);
00902
00903 /* Compute cost function... */
00904 *chisq = cost_function(dx, dy, s_a_inv, sig_eps_inv);
00905
00906 /* Modify Levenberg-Marquardt parameter... */
00907 if (*chisq > chisq_old) {
00908     lmpar *= 10;
00909     gsl_vector_sub(x_i, x_step);
00910 } else {
00911     lmpar /= 10;
00912     break;
00913 }
00914 }
00915
00916 /* Get normalized step size in state space... */
00917 gsl_blas_ddot(x_step, b, &disq);
00918 disq /= (double) n;
00919
00920 /* Convergence test... */
00921 if ((it == 1 || it % ret->kernel_recomp == 0) && disq < ret->
conv_dmin)
00922     break;
00923 }
00924
00925 /* -----
00926 Finalize...
00927 ----- */
00928
00929 gsl_matrix_free(a);
00930 gsl_matrix_free(cov);
00931 gsl_matrix_free(k_i);
00932 gsl_matrix_free(s_a_inv);
00933
00934 gsl_vector_free(b);
00935 gsl_vector_free(dx);
00936 gsl_vector_free(dy);
00937 gsl_vector_free(sig_eps_inv);
00938 gsl_vector_free(sig_formod);
00939 gsl_vector_free(sig_noise);
00940 gsl_vector_free(x_a);
00941 gsl_vector_free(x_i);
00942 gsl_vector_free(x_step);
00943 gsl_vector_free(y_aux);
00944 gsl_vector_free(y_i);
00945 gsl_vector_free(y_m);
00946 }
00947
00948 /*****
00949 void read_nc(
00950     char *filename,
00951     ncd_t * ncd) {

```

```

00953
00954     int varid;
00955
00956     /* Open netCDF file... */
00957     printf("Read netCDF file: %s\n", filename);
00958     NC(nc_open(filename, NC_WRITE, &ncd->ncid));
00959
00960     /* Read Level-1 data... */
00961     NC(nc_inq_varid(ncd->ncid, "l1_time", &varid));
00962     NC(nc_get_var_double(ncd->ncid, varid, ncd->l1_time[0]));
00963     NC(nc_inq_varid(ncd->ncid, "l1_lon", &varid));
00964     NC(nc_get_var_double(ncd->ncid, varid, ncd->l1_lon[0]));
00965     NC(nc_inq_varid(ncd->ncid, "l1_lat", &varid));
00966     NC(nc_get_var_double(ncd->ncid, varid, ncd->l1_lat[0]));
00967     NC(nc_inq_varid(ncd->ncid, "l1_sat_z", &varid));
00968     NC(nc_get_var_double(ncd->ncid, varid, ncd->l1_sat_z));
00969     NC(nc_inq_varid(ncd->ncid, "l1_sat_lon", &varid));
00970     NC(nc_get_var_double(ncd->ncid, varid, ncd->l1_sat_lon));
00971     NC(nc_inq_varid(ncd->ncid, "l1_sat_lat", &varid));
00972     NC(nc_get_var_double(ncd->ncid, varid, ncd->l1_sat_lat));
00973     NC(nc_inq_varid(ncd->ncid, "l1_nu", &varid));
00974     NC(nc_get_var_double(ncd->ncid, varid, ncd->l1_nu));
00975     NC(nc_inq_varid(ncd->ncid, "l1_rad", &varid));
00976     NC(nc_get_var_float(ncd->ncid, varid, ncd->l1_rad[0][0]));
00977
00978     /* Read Level-2 data... */
00979     NC(nc_inq_varid(ncd->ncid, "l2_z", &varid));
00980     NC(nc_get_var_double(ncd->ncid, varid, ncd->l2_z[0][0]));
00981     NC(nc_inq_varid(ncd->ncid, "l2_press", &varid));
00982     NC(nc_get_var_double(ncd->ncid, varid, ncd->l2_p));
00983     NC(nc_inq_varid(ncd->ncid, "l2_temp", &varid));
00984     NC(nc_get_var_double(ncd->ncid, varid, ncd->l2_t[0][0]));
00985 }
00986
00987 /*****
00988
00989 void read_ret_ctl(
00990     int argc,
00991     char *argv[],
00992     ctl_t * ctl,
00993     ret_t * ret) {
00994
00995     int id, ig, iw;
00996
00997     /* Iteration control... */
00998     ret->kernel_recomp =
00999         (int) scan_ctl(argc, argv, "KERNEL_RECOMP", -1, "3", NULL);
01000     ret->conv_itmax = (int) scan_ctl(argc, argv, "CONV_ITMAX", -1, "30", NULL);
01001     ret->conv_dmin = scan_ctl(argc, argv, "CONV_DMIN", -1, "0.1", NULL);
01002
01003     for (id = 0; id < ctl->nd; id++)
01004         ret->err_formod[id] = scan_ctl(argc, argv, "ERR_FORMOD", id, "0", NULL);
01005
01006     for (id = 0; id < ctl->nd; id++)
01007         ret->err_noise[id] = scan_ctl(argc, argv, "ERR_NOISE", id, "0", NULL);
01008
01009     ret->err_press = scan_ctl(argc, argv, "ERR_PRESS", -1, "0", NULL);
01010     ret->err_press_cz = scan_ctl(argc, argv, "ERR_PRESS_CZ", -1, "-999", NULL);
01011     ret->err_press_ch = scan_ctl(argc, argv, "ERR_PRESS_CH", -1, "-999", NULL);
01012
01013     ret->err_temp = scan_ctl(argc, argv, "ERR_TEMP", -1, "0", NULL);
01014     ret->err_temp_cz = scan_ctl(argc, argv, "ERR_TEMP_CZ", -1, "-999", NULL);
01015     ret->err_temp_ch = scan_ctl(argc, argv, "ERR_TEMP_CH", -1, "-999", NULL);
01016
01017     for (ig = 0; ig < ctl->ng; ig++) {
01018         ret->err_q[ig] = scan_ctl(argc, argv, "ERR_Q", ig, "0", NULL);
01019         ret->err_q_cz[ig] = scan_ctl(argc, argv, "ERR_Q_CZ", ig, "-999", NULL);
01020         ret->err_q_ch[ig] = scan_ctl(argc, argv, "ERR_Q_CH", ig, "-999", NULL);
01021     }
01022
01023     for (iw = 0; iw < ctl->nw; iw++) {
01024         ret->err_k[iw] = scan_ctl(argc, argv, "ERR_K", iw, "0", NULL);
01025         ret->err_k_cz[iw] = scan_ctl(argc, argv, "ERR_K_CZ", iw, "-999", NULL);
01026         ret->err_k_ch[iw] = scan_ctl(argc, argv, "ERR_K_CH", iw, "-999", NULL);
01027     }
01028 }
01029
01030 /*****
01031
01032 void set_cov_apr(
01033     ret_t * ret,
01034     ctl_t * ctl,
01035     atm_t * atm,
01036     int *iqa,
01037     int *ipa,
01038     gsl_matrix * s_a) {
01039

```

```

01040     gsl_vector *x_a;
01041
01042     double ch, cz, rho, x0[3], x1[3];
01043
01044     int ig, iw;
01045
01046     size_t i, j, n;
01047
01048     /* Get sizes... */
01049     n = s_a->szel;
01050
01051     /* Allocate... */
01052     x_a = gsl_vector_alloc(n);
01053
01054     /* Get sigma vector... */
01055     atm2x(ctl, atm, x_a, NULL, NULL);
01056     for (i = 0; i < n; i++) {
01057         if (iqa[i] == IDXP)
01058             gsl_vector_set(x_a, i, ret->err_press / 100 * gsl_vector_get(x_a, i));
01059         if (iqa[i] == IDXT)
01060             gsl_vector_set(x_a, i, ret->err_temp);
01061         for (ig = 0; ig < ctl->ng; ig++)
01062             if (iqa[i] == IDXQ(ig))
01063                 gsl_vector_set(x_a, i, ret->err_q[ig] / 100 * gsl_vector_get(x_a, i));
01064         for (iw = 0; iw < ctl->nw; iw++)
01065             if (iqa[i] == IDXK(iw))
01066                 gsl_vector_set(x_a, i, ret->err_k[iw]);
01067     }
01068
01069     /* Check standard deviations... */
01070     for (i = 0; i < n; i++)
01071         if (gsl_pow_2(gsl_vector_get(x_a, i)) <= 0)
01072             ERRMSG("Check a priori data (zero standard deviation)!");
01073
01074     /* Initialize diagonal covariance... */
01075     gsl_matrix_set_zero(s_a);
01076     for (i = 0; i < n; i++)
01077         gsl_matrix_set(s_a, i, i, gsl_pow_2(gsl_vector_get(x_a, i)));
01078
01079     /* Loop over matrix elements... */
01080     for (i = 0; i < n; i++)
01081         for (j = 0; j < n; j++)
01082             if (i != j && iqa[i] == iqa[j]) {
01083
01084                 /* Initialize... */
01085                 cz = ch = 0;
01086
01087                 /* Set correlation lengths for pressure... */
01088                 if (iqa[i] == IDXP) {
01089                     cz = ret->err_press_cz;
01090                     ch = ret->err_press_ch;
01091                 }
01092
01093                 /* Set correlation lengths for temperature... */
01094                 if (iqa[i] == IDXT) {
01095                     cz = ret->err_temp_cz;
01096                     ch = ret->err_temp_ch;
01097                 }
01098
01099                 /* Set correlation lengths for volume mixing ratios... */
01100                 for (ig = 0; ig < ctl->ng; ig++)
01101                     if (iqa[i] == IDXQ(ig)) {
01102                         cz = ret->err_q_cz[ig];
01103                         ch = ret->err_q_ch[ig];
01104                     }
01105
01106                 /* Set correlation lengths for extinction... */
01107                 for (iw = 0; iw < ctl->nw; iw++)
01108                     if (iqa[i] == IDXK(iw)) {
01109                         cz = ret->err_k_cz[iw];
01110                         ch = ret->err_k_ch[iw];
01111                     }
01112
01113                 /* Compute correlations... */
01114                 if (cz > 0 && ch > 0) {
01115
01116                     /* Get Cartesian coordinates... */
01117                     geo2cart(0, atm->lon[ipa[i]], atm->lat[ipa[i]], x0);
01118                     geo2cart(0, atm->lon[ipa[j]], atm->lat[ipa[j]], x1);
01119
01120                     /* Compute correlations... */
01121                     rho =
01122                         exp(-DIST(x0, x1) / ch -
01123                             fabs(atm->z[ipa[i]] - atm->z[ipa[j]]) / cz);
01124
01125                     /* Set covariance... */
01126                     gsl_matrix_set(s_a, i, j, gsl_vector_get(x_a, i)

```

```

01127             * gsl_vector_get(x_a, j) * rho);
01128         }
01129     }
01130
01131     /* Free... */
01132     gsl_vector_free(x_a);
01133 }
01134
01135 /*****
01136
01137 void set_cov_meas(
01138     ret_t * ret,
01139     ctl_t * ctl,
01140     obs_t * obs,
01141     gsl_vector * sig_noise,
01142     gsl_vector * sig_formod,
01143     gsl_vector * sig_eps_inv) {
01144
01145     static obs_t obs_err;
01146
01147     int id, ir;
01148
01149     size_t i, m;
01150
01151     /* Get size... */
01152     m = sig_eps_inv->size;
01153
01154     /* Noise error (always considered in retrieval fit)... */
01155     copy_obs(ctl, &obs_err, obs, 1);
01156     for (ir = 0; ir < obs_err.nr; ir++)
01157         for (id = 0; id < ctl->nd; id++)
01158             obs_err.rad[id][ir]
01159                 = (gsl_finite(obs->rad[id][ir]) ? ret->err_noise[id] : GSL_NAN);
01160     obs2y(ctl, &obs_err, sig_noise, NULL, NULL);
01161
01162     /* Forward model error (always considered in retrieval fit)... */
01163     copy_obs(ctl, &obs_err, obs, 1);
01164     for (ir = 0; ir < obs_err.nr; ir++)
01165         for (id = 0; id < ctl->nd; id++)
01166             obs_err.rad[id][ir]
01167                 = fabs(ret->err_formod[id] / 100 * obs->rad[id][ir]);
01168     obs2y(ctl, &obs_err, sig_formod, NULL, NULL);
01169
01170     /* Total error... */
01171     for (i = 0; i < m; i++)
01172         gsl_vector_set(sig_eps_inv, i,
01173             1 / sqrt(gsl_pow_2(gsl_vector_get(sig_noise, i))
01174                 + gsl_pow_2(gsl_vector_get(sig_formod, i))));
01175
01176     /* Check standard deviations... */
01177     for (i = 0; i < m; i++)
01178         if (gsl_vector_get(sig_eps_inv, i) <= 0)
01179             ERRMSG("Check measurement errors (zero standard deviation)!");
01180 }
01181
01182 /*****
01183
01184 double sza(
01185     double sec,
01186     double lon,
01187     double lat) {
01188
01189     double D, dec, e, g, GMST, h, L, LST, q, ra;
01190
01191     /* Number of days and fraction with respect to 2000-01-01T12:00Z... */
01192     D = sec / 86400 - 0.5;
01193
01194     /* Geocentric apparent ecliptic longitude [rad]... */
01195     g = (357.529 + 0.98560028 * D) * M_PI / 180;
01196     q = 280.459 + 0.98564736 * D;
01197     L = (q + 1.915 * sin(g) + 0.020 * sin(2 * g)) * M_PI / 180;
01198
01199     /* Mean obliquity of the ecliptic [rad]... */
01200     e = (23.439 - 0.00000036 * D) * M_PI / 180;
01201
01202     /* Declination [rad]... */
01203     dec = asin(sin(e) * sin(L));
01204
01205     /* Right ascension [rad]... */
01206     ra = atan2(cos(e) * sin(L), cos(L));
01207
01208     /* Greenwich Mean Sidereal Time [h]... */
01209     GMST = 18.697374558 + 24.06570982441908 * D;
01210
01211     /* Local Sidereal Time [h]... */
01212     LST = GMST + lon / 15;
01213

```

```

01214  /* Hour angle [rad]... */
01215  h = LST / 12 * M_PI - ra;
01216
01217  /* Convert latitude... */
01218  lat *= M_PI / 180;
01219
01220  /* Return solar zenith angle [deg]... */
01221  return acos(sin(lat) * sin(dec) +
01222             cos(lat) * cos(dec) * cos(h)) * 180 / M_PI;
01223 }
01224
01225 /*****
01226
01227 void write_nc(
01228     char *filename,
01229     ncd_t *ncd) {
01230
01231     int dimid[10], p_id, t_id, z_id;
01232
01233     /* Create netCDF file... */
01234     printf("Write netCDF file: %s\n", filename);
01235
01236     /* Read existing dimensions... */
01237     NC(nc_inq_dimid(ncd->ncid, "L1_NTRACK", &dimid[0]));
01238     NC(nc_inq_dimid(ncd->ncid, "L1_NXTRACK", &dimid[1]));
01239
01240     /* Set define mode... */
01241     NC(nc_redef(ncd->ncid));
01242
01243     /* Set new dimensions... */
01244     if (nc_inq_dimid(ncd->ncid, "RET_NP", &dimid[2]) != NC_NOERR)
01245         NC(nc_def_dim(ncd->ncid, "RET_NP", (size_t) ncd->np, &dimid[2]));
01246
01247     /* Set new variables... */
01248     add_var(ncd->ncid, "ret_z", "km", "altitude", NC_FLOAT, &dimid[2], &z_id,
01249            1);
01250     add_var(ncd->ncid, "ret_press", "hPa", "pressure", NC_FLOAT, dimid, &p_id,
01251            2);
01252     add_var(ncd->ncid, "ret_temp", "K", "temperature", NC_FLOAT, dimid, &t_id,
01253            3);
01254
01255     /* Leave define mode... */
01256     NC(nc_enddef(ncd->ncid));
01257
01258     /* Write data... */
01259     NC(nc_put_var_float(ncd->ncid, z_id, ncd->ret_z));
01260     NC(nc_put_var_float(ncd->ncid, p_id, ncd->ret_p));
01261     NC(nc_put_var_float(ncd->ncid, t_id, ncd->ret_t));
01262
01263     /* Close netCDF file... */
01264     NC(nc_close(ncd->ncid));
01265 }

```

## 5.53 sampling.c File Reference

### Functions

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

#### 5.53.1 Function Documentation

##### 5.53.1.1 int main ( int argc, char \* argv[] )

Definition at line 3 of file [sampling.c](#).

```

00005     {
00006
00007     static pert_t *pert;
00008
00009     double d, dmin, dmax, dmu, x0[3], x1[3], x2[3];
00010
00011     int i, itrack, ixtrack, n;
00012
00013     /* Check arguments... */

```

```

00014     if (argc < 3)
00015         ERRMSG("Give parameters: <ctl> <pert.nc>");
00016
00017     /* Allocate... */
00018     ALLOC(pert, pert_t, 1);
00019
00020     /* Read perturbation data... */
00021     read_pert(argv[2], "4mu", pert);
00022
00023     /* Init... */
00024     dmin = 1e100;
00025     dmax = -1e100;
00026     dmu = 0;
00027     n = 0;
00028
00029     /* Get swath width... */
00030     for (itrack = 0; itrack < pert->ntrack; itrack++) {
00031         geo2cart(0, pert->lon[itrack][0], pert->lat[itrack][0], x0);
00032         geo2cart(0, pert->lon[itrack][pert->ntrack - 1],
00033                 pert->lat[itrack][pert->ntrack - 1], x1);
00034         d = 2. * RE * asin(DIST(x0, x1) / (2. * RE));
00035         dmin = GSL_MIN(dmin, d);
00036         dmax = GSL_MAX(dmax, d);
00037         dmu += d;
00038         n++;
00039     }
00040
00041     /* Write output... */
00042     printf("\nmean_swath_width= %.1f km\n", dmu / n);
00043     printf("minimum_swath_width= %.1f km\n", dmin);
00044     printf("maximum_swath_width= %.1f km\n", dmax);
00045
00046     /* Init... */
00047     dmin = 1e100;
00048     dmax = -1e100;
00049     dmu = 0;
00050     n = 0;
00051
00052     /* Get across-track sampling distances... */
00053     for (itrack = 0; itrack < pert->ntrack; itrack++) {
00054         for (ixtrack = 0; ixtrack < pert->ntrack - 1; ixtrack++) {
00055             geo2cart(0, pert->lon[itrack][ixtrack], pert->lat[itrack][ixtrack], x0);
00056             geo2cart(0, pert->lon[itrack][ixtrack + 1],
00057                     pert->lat[itrack][ixtrack + 1], x1);
00058             d = 2. * RE * asin(DIST(x0, x1) / (2. * RE));
00059             dmin = GSL_MIN(dmin, d);
00060             dmax = GSL_MAX(dmax, d);
00061             dmu += d;
00062             n++;
00063         }
00064     }
00065
00066     /* Write output... */
00067     printf("\nmean_across_track_sampling_distance= %.1f km\n", dmu / n);
00068     printf("minimum_across_track_sampling_distance= %.1f km\n", dmin);
00069     printf("maximum_across_track_sampling_distance= %.1f km\n", dmax);
00070
00071     /* Init... */
00072     dmin = 1e100;
00073     dmax = -1e100;
00074     dmu = 0;
00075     n = 0;
00076
00077     /* Get along-track sampling distances... */
00078     for (itrack = 0; itrack < pert->ntrack - 1; itrack++) {
00079         for (ixtrack = 0; ixtrack < pert->ntrack; ixtrack++) {
00080             geo2cart(0, pert->lon[itrack][ixtrack], pert->lat[itrack][ixtrack], x0);
00081             geo2cart(0, pert->lon[itrack + 1][ixtrack],
00082                     pert->lat[itrack + 1][ixtrack], x1);
00083             d = 2. * RE * asin(DIST(x0, x1) / (2. * RE));
00084             dmin = GSL_MIN(dmin, d);
00085             dmax = GSL_MAX(dmax, d);
00086             dmu += d;
00087             n++;
00088         }
00089     }
00090
00091     /* Write output... */
00092     printf("\nmean_along_track_sampling_distance= %.1f km\n", dmu / n);
00093     printf("minimum_along_track_sampling_distance= %.1f km\n", dmin);
00094     printf("maximum_along_track_sampling_distance= %.1f km\n", dmax);
00095
00096     /* Init... */
00097     dmin = 1e100;
00098     dmax = -1e100;
00099     dmu = 0;
00100     n = 0;

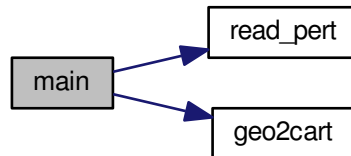
```

```

00101
00102 /* Get angle between along-track and across-track direction... */
00103 for (itrack = 0; itrack < pert->ntrack - 1; itrack++) {
00104     geo2cart(0, pert->lon[itrack][pert->nxtrack / 2],
00105             pert->lat[itrack][pert->nxtrack / 2], x0);
00106     geo2cart(0, pert->lon[itrack][pert->nxtrack / 2 + 1],
00107             pert->lat[itrack][pert->nxtrack / 2 + 1], x1);
00108     geo2cart(0, pert->lon[itrack + 1][pert->nxtrack / 2],
00109             pert->lat[itrack + 1][pert->nxtrack / 2], x2);
00110     for (i = 0; i < 3; i++) {
00111         x1[i] -= x0[i];
00112         x2[i] -= x0[i];
00113     }
00114     d = acos(DOTP(x1, x2) / (NORM(x1) * NORM(x2))) * 180. / M_PI;
00115     dmin = GSL_MIN(dmin, d);
00116     dmax = GSL_MAX(dmax, d);
00117     dmu += d;
00118     n++;
00119 }
00120
00121 /* Write output... */
00122 printf("\nmean_across_track_angle= %.1f deg\n", dmu / n);
00123 printf("minimum_across_track_angle= %.1f deg\n", dmin);
00124 printf("maximum_across_track_angle= %.1f deg\n", dmax);
00125
00126 /* Free... */
00127 free(pert);
00128
00129 return EXIT_SUCCESS;
00130 }

```

Here is the call graph for this function:



## 5.54 sampling.c

```

00001 #include "libairs.h"
00002
00003 int main(
00004     int argc,
00005     char *argv[]) {
00006
00007     static pert_t *pert;
00008
00009     double d, dmin, dmax, dmu, x0[3], x1[3], x2[3];
00010
00011     int i, itrack, ixtrack, n;
00012
00013     /* Check arguments... */
00014     if (argc < 3)
00015         ERRMSG("Give parameters: <ctl> <pert.nc>");
00016
00017     /* Allocate... */
00018     ALLOC(pert, pert_t, 1);
00019
00020     /* Read perturbation data... */
00021     read_pert(argv[2], "4mu", pert);
00022
00023     /* Init... */
00024     dmin = 1e100;
00025     dmax = -1e100;

```

```

00026     dmu = 0;
00027     n = 0;
00028
00029     /* Get swath width... */
00030     for (itrack = 0; itrack < pert->ntrack; itrack++) {
00031         geo2cart(0, pert->lon[itrack][0], pert->lat[itrack][0], x0);
00032         geo2cart(0, pert->lon[itrack][pert->ntrack - 1],
00033                 pert->lat[itrack][pert->ntrack - 1], x1);
00034         d = 2. * RE * asin(DIST(x0, x1) / (2. * RE));
00035         dmin = GSL_MIN(dmin, d);
00036         dmax = GSL_MAX(dmax, d);
00037         dmu += d;
00038         n++;
00039     }
00040
00041     /* Write output... */
00042     printf("\nmean_swath_width= %.1f km\n", dmu / n);
00043     printf("minimum_swath_width= %.1f km\n", dmin);
00044     printf("maximum_swath_width= %.1f km\n", dmax);
00045
00046     /* Init... */
00047     dmin = 1e100;
00048     dmax = -1e100;
00049     dmu = 0;
00050     n = 0;
00051
00052     /* Get across-track sampling distances... */
00053     for (itrack = 0; itrack < pert->ntrack; itrack++) {
00054         for (ixtrack = 0; ixtrack < pert->ntrack - 1; ixtrack++) {
00055             geo2cart(0, pert->lon[itrack][ixtrack], pert->lat[itrack][ixtrack], x0);
00056             geo2cart(0, pert->lon[itrack][ixtrack + 1],
00057                     pert->lat[itrack][ixtrack + 1], x1);
00058             d = 2. * RE * asin(DIST(x0, x1) / (2. * RE));
00059             dmin = GSL_MIN(dmin, d);
00060             dmax = GSL_MAX(dmax, d);
00061             dmu += d;
00062             n++;
00063         }
00064     }
00065
00066     /* Write output... */
00067     printf("\nmean_across_track_sampling_distance= %.1f km\n", dmu / n);
00068     printf("minimum_across_track_sampling_distance= %.1f km\n", dmin);
00069     printf("maximum_across_track_sampling_distance= %.1f km\n", dmax);
00070
00071     /* Init... */
00072     dmin = 1e100;
00073     dmax = -1e100;
00074     dmu = 0;
00075     n = 0;
00076
00077     /* Get along-track sampling distances... */
00078     for (itrack = 0; itrack < pert->ntrack - 1; itrack++) {
00079         for (ixtrack = 0; ixtrack < pert->ntrack; ixtrack++) {
00080             geo2cart(0, pert->lon[itrack][ixtrack], pert->lat[itrack][ixtrack], x0);
00081             geo2cart(0, pert->lon[itrack + 1][ixtrack],
00082                     pert->lat[itrack + 1][ixtrack], x1);
00083             d = 2. * RE * asin(DIST(x0, x1) / (2. * RE));
00084             dmin = GSL_MIN(dmin, d);
00085             dmax = GSL_MAX(dmax, d);
00086             dmu += d;
00087             n++;
00088         }
00089     }
00090
00091     /* Write output... */
00092     printf("\nmean_along_track_sampling_distance= %.1f km\n", dmu / n);
00093     printf("minimum_along_track_sampling_distance= %.1f km\n", dmin);
00094     printf("maximum_along_track_sampling_distance= %.1f km\n", dmax);
00095
00096     /* Init... */
00097     dmin = 1e100;
00098     dmax = -1e100;
00099     dmu = 0;
00100     n = 0;
00101
00102     /* Get angle between along-track and across-track direction... */
00103     for (itrack = 0; itrack < pert->ntrack - 1; itrack++) {
00104         geo2cart(0, pert->lon[itrack][pert->ntrack / 2],
00105                 pert->lat[itrack][pert->ntrack / 2], x0);
00106         geo2cart(0, pert->lon[itrack][pert->ntrack / 2 + 1],
00107                 pert->lat[itrack][pert->ntrack / 2 + 1], x1);
00108         geo2cart(0, pert->lon[itrack + 1][pert->ntrack / 2],
00109                 pert->lat[itrack + 1][pert->ntrack / 2], x2);
00110         for (i = 0; i < 3; i++) {
00111             x1[i] -= x0[i];
00112             x2[i] -= x0[i];

```



```

00113     }
00114     d = acos(DOTP(x1, x2) / (NORM(x1) * NORM(x2))) * 180. / M_PI;
00115     dmin = GSL_MIN(dmin, d);
00116     dmax = GSL_MAX(dmax, d);
00117     dmu += d;
00118     n++;
00119 }
00120
00121 /* Write output... */
00122 printf("\nmean_across_track_angle= %.1f deg\n", dmu / n);
00123 printf("minimum_across_track_angle= %.1f deg\n", dmin);
00124 printf("maximum_across_track_angle= %.1f deg\n", dmax);
00125
00126 /* Free... */
00127 free(pert);
00128
00129 return EXIT_SUCCESS;
00130 }

```

## 5.55 spec2tab.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 [spec2tab.c](#).

```

00005     {
00006
00007     static airs_rad_gran_t airs_rad_gran;
00008
00009     FILE *out;
00010
00011     double dmin = 1e100, x0[3], x1[3];
00012
00013     int ichan, track = -1, track2, xtrack = -1, xtrack2;
00014
00015     /* Check arguments... */
00016     if (argc != 6)
00017         ERRMSG("Give parameters: <airs_llb_file> "
00018              " [index <track> <xtrack> | geo <lon> <lat>] <spec.tab>");
00019
00020     /* Read AIRS data... */
00021     printf("Read AIRS Level-1B data file: %s\n", argv[1]);
00022     airs_rad_rdr(argv[1], &airs_rad_gran);
00023
00024     /* Get indices... */
00025     if (argv[2][0] == 'i') {
00026         track = atoi(argv[3]);
00027         xtrack = atoi(argv[4]);
00028     }
00029
00030     /* Find nearest footprint... */
00031     else {
00032         geo2cart(0, atof(argv[3]), atof(argv[4]), x0);
00033         for (track2 = 0; track2 < AIRS_RAD_GEOTRACK; track2++)
00034             for (xtrack2 = 0; xtrack2 < AIRS_RAD_GEOXTRACK; xtrack2++) {
00035                 geo2cart(0, airs_rad_gran.Longitude[track2][xtrack2],
00036                        airs_rad_gran.Latitude[track2][xtrack2], x1);
00037                 if (DIST2(x0, x1) < dmin) {
00038                     dmin = DIST2(x0, x1);
00039                     track = track2;
00040                     xtrack = xtrack2;
00041                 }
00042             }
00043         if (dmin > 2500)
00044             ERRMSG("Geolocation not covered by granule!");
00045     }
00046
00047     /* Check indices... */

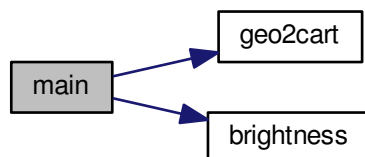
```

```

00048 if (track < 0 || track >= AIRS_RAD_GEOTRACK)
00049     ERRMSG("Along-track index out of range!");
00050 if (xtrack < 0 || xtrack >= AIRS_RAD_GEOXTRACK)
00051     ERRMSG("Across-track index out of range!");
00052
00053 /* Flag bad observations... */
00054 for (ichan = 0; ichan < AIRS_RAD_CHANNEL; ichan++)
00055     if ((airs_rad_gran.state[track][xtrack] != 0)
00056         || (airs_rad_gran.ExcludedChans[ichan] > 2)
00057         || (airs_rad_gran.CalChanSummary[ichan] & 8)
00058         || (airs_rad_gran.CalChanSummary[ichan] & (32 + 64))
00059         || (airs_rad_gran.CalFlag[track][ichan] & 16))
00060         airs_rad_gran.radiances[track][xtrack][ichan]
00061         = (float) sqrt(-1.0);
00062
00063 /* Create file... */
00064 printf("Write spectrum: %s\n", argv[5]);
00065 if (!(out = fopen(argv[5], "w")))
00066     ERRMSG("Cannot create file!");
00067
00068 /* Write header... */
00069 fprintf(out,
00070         "# $1 = time (seconds since 01-JAN-2000, 00:00 UTC)\n"
00071         "# $2 = satellite longitude [deg]\n"
00072         "# $3 = satellite latitude [deg]\n"
00073         "# $4 = footprint longitude [deg]\n"
00074         "# $5 = footprint latitude [deg]\n"
00075         "# $6 = wavenumber [cm^-1]\n"
00076         "# $7 = brightness temperature [K]\n"
00077         "# $8 = radiance [W/(m^2 sr cm^-1)]\n\n");
00078
00079 /* Write data... */
00080 for (ichan = 0; ichan < AIRS_RAD_CHANNEL; ichan++) {
00081     if (ichan > 0)
00082         if (fabs(airs_rad_gran.nominal_freq[ichan]
00083                 - airs_rad_gran.nominal_freq[ichan - 1]) > 1.2)
00084             fprintf(out, "\n");
00085     fprintf(out, "%.2f %g %g %g %g %g %g\n",
00086             airs_rad_gran.Time[track][xtrack] - 220838400,
00087             airs_rad_gran.sat_lon[track],
00088             airs_rad_gran.sat_lat[track],
00089             airs_rad_gran.Longitude[track][xtrack],
00090             airs_rad_gran.Latitude[track][xtrack],
00091             airs_rad_gran.nominal_freq[ichan],
00092             brightness(airs_rad_gran.radiances[track][xtrack][ichan] * 1e-3,
00093                       airs_rad_gran.nominal_freq[ichan]),
00094             airs_rad_gran.radiances[track][xtrack][ichan] * 1e-3);
00095 }
00096
00097 /* Close file... */
00098 fclose(out);
00099
00100 return EXIT_SUCCESS;
00101 }

```

Here is the call graph for this function:



## 5.56 spec2tab.c

```
00001 #include "libairs.h"
```

```

00002
00003 int main(
00004     int argc,
00005     char *argv[]) {
00006
00007     static airs_rad_gran_t airs_rad_gran;
00008
00009     FILE *out;
00010
00011     double dmin = 1e100, x0[3], x1[3];
00012
00013     int ichan, track = -1, track2, xtrack = -1, xtrack2;
00014
00015     /* Check arguments... */
00016     if (argc != 6)
00017         ERRMSG("Give parameters: <airs_llb_file> "
00018             "[index <track> <xtrack> | geo <lon> <lat>] <spec.tab>");
00019
00020     /* Read AIRS data... */
00021     printf("Read AIRS Level-1B data file: %s\n", argv[1]);
00022     airs_rad_rdr(argv[1], &airs_rad_gran);
00023
00024     /* Get indices... */
00025     if (argv[2][0] == 'i') {
00026         track = atoi(argv[3]);
00027         xtrack = atoi(argv[4]);
00028     }
00029
00030     /* Find nearest footprint... */
00031     else {
00032         geo2cart(0, atof(argv[3]), atof(argv[4]), x0);
00033         for (track2 = 0; track2 < AIRS_RAD_GEOTRACK; track2++)
00034             for (xtrack2 = 0; xtrack2 < AIRS_RAD_GEOXTRACK; xtrack2++) {
00035                 geo2cart(0, airs_rad_gran.Longitude[track2][xtrack2],
00036                     airs_rad_gran.Latitude[track2][xtrack2], x1);
00037                 if (DIST2(x0, x1) < dmin) {
00038                     dmin = DIST2(x0, x1);
00039                     track = track2;
00040                     xtrack = xtrack2;
00041                 }
00042             }
00043         if (dmin > 2500)
00044             ERRMSG("Geolocation not covered by granule!");
00045     }
00046
00047     /* Check indices... */
00048     if (track < 0 || track >= AIRS_RAD_GEOTRACK)
00049         ERRMSG("Along-track index out of range!");
00050     if (xtrack < 0 || xtrack >= AIRS_RAD_GEOXTRACK)
00051         ERRMSG("Across-track index out of range!");
00052
00053     /* Flag bad observations... */
00054     for (ichan = 0; ichan < AIRS_RAD_CHANNEL; ichan++)
00055         if ((airs_rad_gran.state[track][xtrack] != 0)
00056             || (airs_rad_gran.ExcludedChans[ichan] > 2)
00057             || (airs_rad_gran.CalChanSummary[ichan] & 8)
00058             || (airs_rad_gran.CalChanSummary[ichan] & (32 + 64))
00059             || (airs_rad_gran.CalFlag[track][ichan] & 16))
00060             airs_rad_gran.radiances[track][xtrack][ichan]
00061                 = (float) sqrt(-1.0);
00062
00063     /* Create file... */
00064     printf("Write spectrum: %s\n", argv[5]);
00065     if (!(out = fopen(argv[5], "w")))
00066         ERRMSG("Cannot create file!");
00067
00068     /* Write header... */
00069     fprintf(out,
00070         "# $1 = time (seconds since 01-JAN-2000, 00:00 UTC)\n"
00071         "# $2 = satellite longitude [deg]\n"
00072         "# $3 = satellite latitude [deg]\n"
00073         "# $4 = footprint longitude [deg]\n"
00074         "# $5 = footprint latitude [deg]\n"
00075         "# $6 = wavenumber [cm^-1]\n"
00076         "# $7 = brightness temperature [K]\n"
00077         "# $8 = radiance [W/(m^2 sr cm^-1)]\n\n");
00078
00079     /* Write data... */
00080     for (ichan = 0; ichan < AIRS_RAD_CHANNEL; ichan++) {
00081         if (ichan > 0)
00082             if (fabs(airs_rad_gran.nominal_freq[ichan]
00083                 - airs_rad_gran.nominal_freq[ichan - 1]) > 1.2)
00084                 fprintf(out, "\n");
00085         fprintf(out, "%.2f %g %g %g %g %g %g %g\n",
00086             airs_rad_gran.Time[track][xtrack] - 220838400,
00087             airs_rad_gran.sat_lon[track],
00088             airs_rad_gran.sat_lat[track],

```

```

00089         airs_rad_gran.Longitude[track][xtrack],
00090         airs_rad_gran.Latitude[track][xtrack],
00091         airs_rad_gran.nominal_freq[ichan],
00092         brightness(airs_rad_gran.radiances[track][xtrack][ichan] * 1e-3,
00093         airs_rad_gran.nominal_freq[ichan]),
00094         airs_rad_gran.radiances[track][xtrack][ichan] * 1e-3);
00095     }
00096
00097     /* Close file... */
00098     fclose(out);
00099
00100     return EXIT_SUCCESS;
00101 }

```

## 5.57 spec\_ana.c File Reference

### Functions

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

#### 5.57.1 Function Documentation

##### 5.57.1.1 int main ( int argc, char \* argv[] )

Definition at line 3 of file [spec\\_ana.c](#).

```

00005     {
00006
00007     static wave_t wave;
00008     static pert_t *pert;
00009
00010     char method[LEN], pertname[LEN];
00011
00012     double var_dh, Amax, phimax, lhmax, alphamax, betamax;
00013
00014     int bg_poly_x, bg_poly_y, bg_smooth_x, bg_smooth_y, inter_x,
00015         dtrack, dxtrack, track0, xtrack0;
00016
00017     /* Check arguments... */
00018     if (argc < 3)
00019         ERRMSG("Give parameters: <ctl> <pert.nc>");
00020
00021     /* Get control parameters... */
00022     scan_ctl(argc, argv, "PERTNAME", -1, "4mu", pertname);
00023     track0 = (int) scan_ctl(argc, argv, "TRACK0", -1, "", NULL);
00024     xtrack0 = (int) scan_ctl(argc, argv, "XTRACK0", -1, "", NULL);
00025     dtrack = (int) scan_ctl(argc, argv, "DTRACK", -1, "20", NULL);
00026     dxtrack = (int) scan_ctl(argc, argv, "DXTRACK", -1, "20", NULL);
00027     inter_x = (int) scan_ctl(argc, argv, "INTER_X", -1, "0", NULL);
00028     bg_poly_x = (int) scan_ctl(argc, argv, "BG_POLY_X", -1, "5", NULL);
00029     bg_poly_y = (int) scan_ctl(argc, argv, "BG_POLY_Y", -1, "0", NULL);
00030     bg_smooth_x = (int) scan_ctl(argc, argv, "BG_SMOOTH_X", -1, "0", NULL);
00031     bg_smooth_y = (int) scan_ctl(argc, argv, "BG_SMOOTH_Y", -1, "7", NULL);
00032     var_dh = scan_ctl(argc, argv, "VAR_DH", -1, "100", NULL);
00033     scan_ctl(argc, argv, "METHOD", -1, "P", method);
00034
00035     /* Allocate... */
00036     ALLOC(pert, pert_t, 1);
00037
00038     /* Read perturbation data... */
00039     read_pert(argv[2], pertname, pert);
00040
00041     /* Check indices... */
00042     if (track0 < 0 || track0 >= pert->ntrack)
00043         ERRMSG("Along-track index out of range!");
00044     if (xtrack0 < 0 || xtrack0 >= pert->nxtrack)
00045         ERRMSG("Across-track index out of range!");
00046
00047     /* Convert to wave analysis struct... */
00048     pert2wave(pert, &wave,
00049         track0 - dtrack, track0 + dtrack,
00050         xtrack0 - dxtrack, xtrack0 + dxtrack);
00051
00052     /* Interpolate to regular grid... */

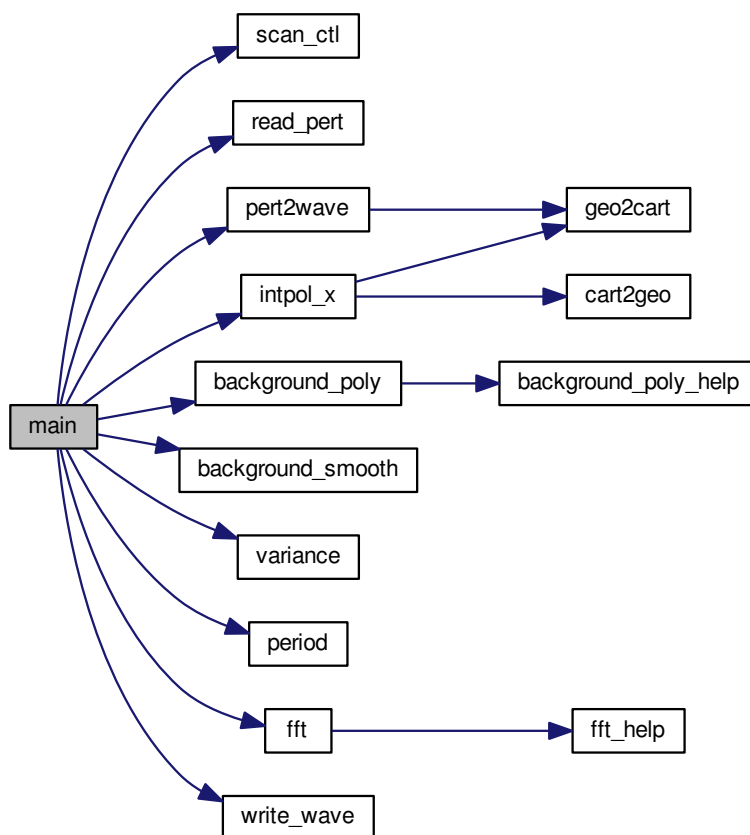
```

```

00053  intpol_x(&wave, inter_x);
00054
00055  /* Estimate background... */
00056  background_poly(&wave, bg_poly_x, bg_poly_y);
00057  background_smooth(&wave, bg_smooth_x, bg_smooth_y);
00058
00059  /* Compute variance... */
00060  variance(&wave, var_dh);
00061
00062  /* Get wave characteristics... */
00063  if (method[0] == 'p' || method[0] == 'P')
00064      period(&wave, &Amax, &phimax, &lhmax, &alphamax, &betamax, "period.tab");
00065  if (method[0] == 'f' || method[0] == 'F')
00066      fft(&wave, &Amax, &phimax, &lhmax, &alphamax, &betamax, "period.tab");
00067
00068  /* Save wave struct... */
00069  write_wave("wave.tab", &wave);
00070
00071  /* Write results... */
00072  PRINT("%g", Amax);
00073  PRINT("%g", phimax);
00074  PRINT("%g", lhmax);
00075  PRINT("%g", alphamax);
00076  PRINT("%g", betamax);
00077
00078  /* Free... */
00079  free(pert);
00080
00081  return EXIT_SUCCESS;
00082 }

```

Here is the call graph for this function:



## 5.58 spec\_ana.c

```

00001 #include "libairs.h"
00002
00003 int main(
00004     int argc,
00005     char *argv[]) {
00006
00007     static wave_t wave;
00008     static pert_t *pert;
00009
00010     char method[LEN], pertname[LEN];
00011
00012     double var_dh, Amax, phimax, lhmax, alphamax, betamax;
00013
00014     int bg_poly_x, bg_poly_y, bg_smooth_x, bg_smooth_y, inter_x,
00015         dtrack, dxtrack, track0, xtrack0;
00016
00017     /* Check arguments... */
00018     if (argc < 3)
00019         ERRMSG("Give parameters: <ctl> <pert.nc>");
00020
00021     /* Get control parameters... */
00022     scan_ctl(argc, argv, "PERTNAME", -1, "4mu", pertname);
00023     track0 = (int) scan_ctl(argc, argv, "TRACK0", -1, "", NULL);
00024     xtrack0 = (int) scan_ctl(argc, argv, "XTRACK0", -1, "", NULL);
00025     dtrack = (int) scan_ctl(argc, argv, "DTRACK", -1, "20", NULL);
00026     dxtrack = (int) scan_ctl(argc, argv, "DXTRACK", -1, "20", NULL);
00027     inter_x = (int) scan_ctl(argc, argv, "INTER_X", -1, "0", NULL);
00028     bg_poly_x = (int) scan_ctl(argc, argv, "BG_POLY_X", -1, "5", NULL);
00029     bg_poly_y = (int) scan_ctl(argc, argv, "BG_POLY_Y", -1, "0", NULL);
00030     bg_smooth_x = (int) scan_ctl(argc, argv, "BG_SMOOTH_X", -1, "0", NULL);
00031     bg_smooth_y = (int) scan_ctl(argc, argv, "BG_SMOOTH_Y", -1, "7", NULL);
00032     var_dh = scan_ctl(argc, argv, "VAR_DH", -1, "100", NULL);
00033     scan_ctl(argc, argv, "METHOD", -1, "P", method);
00034
00035     /* Allocate... */
00036     ALLOC(pert, pert_t, 1);
00037
00038     /* Read perturbation data... */
00039     read_pert(argv[2], pertname, pert);
00040
00041     /* Check indices... */
00042     if (track0 < 0 || track0 >= pert->ntrack)
00043         ERRMSG("Along-track index out of range!");
00044     if (xtrack0 < 0 || xtrack0 >= pert->nxtrack)
00045         ERRMSG("Across-track index out of range!");
00046
00047     /* Convert to wave analysis struct... */
00048     pert2wave(pert, &wave,
00049         track0 - dtrack, track0 + dtrack,
00050         xtrack0 - dxtrack, xtrack0 + dxtrack);
00051
00052     /* Interpolate to regular grid... */
00053     intpol_x(&wave, inter_x);
00054
00055     /* Estimate background... */
00056     background_poly(&wave, bg_poly_x, bg_poly_y);
00057     background_smooth(&wave, bg_smooth_x, bg_smooth_y);
00058
00059     /* Compute variance... */
00060     variance(&wave, var_dh);
00061
00062     /* Get wave characteristics... */
00063     if (method[0] == 'p' || method[0] == 'P')
00064         period(&wave, &Amax, &phimax, &lhmax, &alphamax, &betamax, "period.tab");
00065     if (method[0] == 'f' || method[0] == 'F')
00066         fft(&wave, &Amax, &phimax, &lhmax, &alphamax, &betamax, "period.tab");
00067
00068     /* Save wave struct... */
00069     write_wave("wave.tab", &wave);
00070
00071     /* Write results... */
00072     PRINT("%g", Amax);
00073     PRINT("%g", phimax);
00074     PRINT("%g", lhmax);
00075     PRINT("%g", alphamax);
00076     PRINT("%g", betamax);
00077
00078     /* Free... */
00079     free(pert);
00080
00081     return EXIT_SUCCESS;
00082 }

```

## 5.59 sza.c File Reference

### Functions

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

### 5.59.1 Function Documentation

#### 5.59.1.1 int main ( int argc, char \* argv[] )

Definition at line 3 of file [sza.c](#).

```

00005         {
00006
00007     double jsec, lon, lat;
00008
00009     /* Check arguments... */
00010     if (argc != 4)
00011         ERRMSG("Give parameters: <jsec> <lon> <lat>");
00012
00013     /* Read arguments... */
00014     jsec = atof(argv[1]);
00015     lon = atof(argv[2]);
00016     lat = atof(argv[3]);
00017
00018     /* Compute solar zenith angle... */
00019     printf("%g\n", sza(jsec, lon, lat));
00020
00021     return EXIT_SUCCESS;
00022 }
```

Here is the call graph for this function:



## 5.60 sza.c

```

00001 #include "libairs.h"
00002
00003 int main(
00004     int argc,
00005     char *argv[]) {
00006
00007     double jsec, lon, lat;
00008
00009     /* Check arguments... */
00010     if (argc != 4)
00011         ERRMSG("Give parameters: <jsec> <lon> <lat>");
00012
00013     /* Read arguments... */
00014     jsec = atof(argv[1]);
00015     lon = atof(argv[2]);
00016     lat = atof(argv[3]);
00017
00018     /* Compute solar zenith angle... */
00019     printf("%g\n", sza(jsec, lon, lat));
00020
00021     return EXIT_SUCCESS;
00022 }
```

## 5.61 var1d.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 3 of file [var1d.c](#).

```

00005         {
00006
00007     gsl_multifit_linear_workspace *work;
00008     gsl_matrix *cov, *X;
00009     gsl_vector *c, *xvec, *yvec, *yfit;
00010
00011     static double chisq, fwhm, lx, dlx, lxmin, lxmax, phi,
00012         var, var2, vmean, vmean2, width, w, wsum;
00013
00014     static int dim, i, i2, n;
00015
00016     /* Check arguments... */
00017     if (argc != 8)
00018         ERRMSG("Give parameters: <width> <n> <lxmin> <lxmax> <dlx> <fwhm> <dim>");
00019
00020     /* Get arguments... */
00021     width = atof(argv[1]);
00022     n = atoi(argv[2]);
00023     lxmin = atof(argv[3]);
00024     lxmax = atof(argv[4]);
00025     dlx = atoi(argv[5]);
00026     fwhm = atof(argv[6]);
00027     dim = atoi(argv[7]);
00028
00029     /* Initialize... */
00030     c = gsl_vector_alloc((size_t) dim);
00031     cov = gsl_matrix_alloc((size_t) dim, (size_t) dim);
00032     work = gsl_multifit_linear_alloc((size_t) n, (size_t) dim);
00033     X = gsl_matrix_alloc((size_t) n, (size_t) dim);
00034     xvec = gsl_vector_alloc((size_t) n);
00035     yvec = gsl_vector_alloc((size_t) n);
00036     yfit = gsl_vector_alloc((size_t) n);
00037
00038     /* Loop over wavelengths... */
00039     for (lx = lxmin; lx <= lxmax; lx += dlx) {
00040
00041         /* Initialize... */
00042         vmean = 0;
00043         vmean2 = 0;
00044
00045         /* Loop over phases... */
00046         for (phi = 0; phi < 2 * M_PI; phi += M_PI / 180) {
00047
00048             /* Initialize... */
00049             var = 0;
00050             var2 = 0;
00051             wsum = 0;
00052
00053             /* Set wave... */
00054             for (i = 0; i < n; i++) {
00055                 gsl_vector_set(xvec, (size_t) i, width / (n - 1.0) * i - width / 2.);
00056                 gsl_vector_set(yvec, (size_t) i,
00057                     sin(2 * M_PI / lx * gsl_vector_get(xvec, (size_t) i) +
00058                         phi));
00059                 if (fwhm > 0) {
00060                     w = gsl_ran_gaussian_pdf(gsl_vector_get(xvec, (size_t) i),
00061                         fwhm * lx / 2.3548);
00062                     gsl_vector_set(yvec, (size_t) i,
00063                         w * gsl_vector_get(yvec, (size_t) i));
00064                     wsum += w;
00065                 }
00066             }
00067             if (wsum > 0)
00068                 gsl_vector_scale(yvec, 1 / wsum);
00069

```



```

00070      /* Detrending... */
00071      for (i = 0; i < n; i++)
00072          for (i2 = 0; i2 < dim; i2++)
00073              gsl_matrix_set(X, (size_t) i, (size_t) i2,
00074                             pow(gsl_vector_get(xvec, (size_t) i), 1. * i2));
00075      gsl_multifit_linear(X, yvec, c, cov, &chisq, work);
00076      for (i = 0; i < n; i++)
00077          gsl_vector_set(yfit, (size_t) i, gsl_vector_get(yvec, (size_t) i)
00078                        - gsl_poly_eval(c->data, (int) dim,
00079                                       gsl_vector_get(xvec, (size_t) i)));
00080
00081      /* Compute variances... */
00082      for (i = 0; i < n; i++) {
00083          var += gsl_pow_2(gsl_vector_get(yfit, (size_t) i)) / (double) n;
00084          var2 += gsl_pow_2(gsl_vector_get(yvec, (size_t) i)) / (double) n;
00085      }
00086      vmean += var;
00087      vmean2 += var2;
00088  }
00089
00090      /* Write output... */
00091      printf("%g %g\n", lx, 100 * vmean / vmean2);
00092  }
00093
00094      return EXIT_SUCCESS;
00095  }

```

## 5.62 var1d.c

```

00001 #include "libairs.h"
00002
00003 int main(
00004     int argc,
00005     char *argv[]) {
00006
00007     gsl_multifit_linear_workspace *work;
00008     gsl_matrix *cov, *X;
00009     gsl_vector *c, *xvec, *yvec, *yfit;
00010
00011     static double chisq, fwhm, lx, dlx, lxmin, lxmax, phi,
00012                 var, var2, vmean, vmean2, width, w, wsum;
00013
00014     static int dim, i, i2, n;
00015
00016     /* Check arguments... */
00017     if (argc != 8)
00018         ERRMSG("Give parameters: <width> <n> <lxmin> <lxmax> <dlx> <fwhm> <dim>");
00019
00020     /* Get arguments... */
00021     width = atof(argv[1]);
00022     n = atoi(argv[2]);
00023     lxmin = atof(argv[3]);
00024     lxmax = atof(argv[4]);
00025     dlx = atoi(argv[5]);
00026     fwhm = atof(argv[6]);
00027     dim = atoi(argv[7]);
00028
00029     /* Initialize... */
00030     c = gsl_vector_alloc((size_t) dim);
00031     cov = gsl_matrix_alloc((size_t) dim, (size_t) dim);
00032     work = gsl_multifit_linear_alloc((size_t) n, (size_t) dim);
00033     X = gsl_matrix_alloc((size_t) n, (size_t) dim);
00034     xvec = gsl_vector_alloc((size_t) n);
00035     yvec = gsl_vector_alloc((size_t) n);
00036     yfit = gsl_vector_alloc((size_t) n);
00037
00038     /* Loop over wavelengths... */
00039     for (lx = lxmin; lx <= lxmax; lx += dlx) {
00040
00041         /* Initialize... */
00042         vmean = 0;
00043         vmean2 = 0;
00044
00045         /* Loop over phases... */
00046         for (phi = 0; phi < 2 * M_PI; phi += M_PI / 180) {
00047
00048             /* Initialize... */
00049             var = 0;
00050             var2 = 0;
00051             wsum = 0;
00052
00053             /* Set wave... */
00054             for (i = 0; i < n; i++) {

```

```

00055     gsl_vector_set(xvec, (size_t) i, width / (n - 1.0) * i - width / 2.);
00056     gsl_vector_set(yvec, (size_t) i,
00057         sin(2 * M_PI / lx * gsl_vector_get(xvec, (size_t) i) +
00058             phi));
00059     if (fwhm > 0) {
00060         w = gsl_ran_gaussian_pdf(gsl_vector_get(xvec, (size_t) i),
00061             fwhm * lx / 2.3548);
00062         gsl_vector_set(yvec, (size_t) i,
00063             w * gsl_vector_get(yvec, (size_t) i));
00064         wsum += w;
00065     }
00066 }
00067 if (wsum > 0)
00068     gsl_vector_scale(yvec, 1 / wsum);
00069
00070 /* Detrending... */
00071 for (i = 0; i < n; i++)
00072     for (i2 = 0; i2 < dim; i2++)
00073         gsl_matrix_set(X, (size_t) i, (size_t) i2,
00074             pow(gsl_vector_get(xvec, (size_t) i), 1. * i2));
00075 gsl_multifit_linear(X, yvec, c, cov, &chisq, work);
00076 for (i = 0; i < n; i++)
00077     gsl_vector_set(yfit, (size_t) i, gsl_vector_get(yvec, (size_t) i)
00078         - gsl_poly_eval(c->data, (int) dim,
00079             gsl_vector_get(xvec, (size_t) i)));
00080
00081 /* Compute variances... */
00082 for (i = 0; i < n; i++) {
00083     var += gsl_pow_2(gsl_vector_get(yfit, (size_t) i)) / (double) n;
00084     var2 += gsl_pow_2(gsl_vector_get(yvec, (size_t) i)) / (double) n;
00085 }
00086 vmean += var;
00087 vmean2 += var2;
00088 }
00089
00090 /* Write output... */
00091 printf("%g %g\n", lx, 100 * vmean / vmean2);
00092 }
00093
00094 return EXIT_SUCCESS;
00095 }

```

## 5.63 variance.c File Reference

### Functions

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

#### 5.63.1 Function Documentation

##### 5.63.1.1 int main ( int argc, char \* argv[] )

Definition at line 261 of file [variance.c](#).

```

00263     {
00264
00265     static pert_t *pert;
00266
00267     static wave_t *wave;
00268
00269     static FILE *in, *out;
00270
00271     static char pertname[LEN], set[LEN];
00272
00273     static double bt[NX][NY], bt_8mu[NX][NY], bt_8mu_min[NX][NY],
00274         bt_8mu_max[NX][NY], dt[NX][NY], mtime[NX][NY], glat[NY], glon[NX],
00275         fdc[NX][NY], fwg[NX][NY], fgw[NX][NY], fcw[NX][NY],
00276         mean[NX][NY], min[NX][NY], max[NX][NY], var[NX][NY],
00277         t_dc, t_gw, dt_trop, dc_hlat = 25, dc_tlim = 250, dt230,
00278         nesr, gauss_fwhm, var_dh, nu, lon0, lon1, lat0, lat1,
00279         thresh_dc, thresh_gw, lt, help[NX * NY];
00280
00281     static int asc, ix, iy, nx, ny, iarg, n[NX][NY],
00282         ndc[NX][NY], ngw[NX][NY], ncw[NX][NY], nwg[NX][NY],

```

```

00283     det_gw, det_cw, det_dc, det_wg, ilat, imon, nmin = 10,
00284     bg_poly_x, bg_poly_y, bg_smooth_x, bg_smooth_y,
00285     itrack, itrack2, ixtrack, ixtrack2, iradius = 30, output, ncid, varid,
00286     minid, maxid, lonid, latid, npid, dimid[10], help2[NX * NY];
00287
00288 /* Check arguments... */
00289 if (argc < 4)
00290     ERRMSG("Give parameters: <ctl> <var.tab> <pert1.nc> [<pert2.nc> ...]");
00291
00292 /* Get control parameters... */
00293 scan_ctl(argc, argv, "SET", -1, "full", set);
00294 scan_ctl(argc, argv, "PERTNAME", -1, "4mu", pertname);
00295 nx = (int) scan_ctl(argc, argv, "NX", -1, "360", NULL);
00296 lon0 = scan_ctl(argc, argv, "LON0", -1, "-180", NULL);
00297 lon1 = scan_ctl(argc, argv, "LON1", -1, "180", NULL);
00298 ny = (int) scan_ctl(argc, argv, "NY", -1, "180", NULL);
00299 lat0 = scan_ctl(argc, argv, "LAT0", -1, "-90", NULL);
00300 lat1 = scan_ctl(argc, argv, "LAT1", -1, "90", NULL);
00301 bg_poly_x = (int) scan_ctl(argc, argv, "BG_POLY_X", -1, "0", NULL);
00302 bg_poly_y = (int) scan_ctl(argc, argv, "BG_POLY_Y", -1, "0", NULL);
00303 bg_smooth_x = (int) scan_ctl(argc, argv, "BG_SMOOTH_X", -1, "0", NULL);
00304 bg_smooth_y = (int) scan_ctl(argc, argv, "BG_SMOOTH_Y", -1, "0", NULL);
00305 gauss_fwhm = scan_ctl(argc, argv, "GAUSS_FWHM", -1, "0", NULL);
00306 var_dh = scan_ctl(argc, argv, "VAR_DH", -1, "0", NULL);
00307 thresh_gw = scan_ctl(argc, argv, "THRESH_GW", -1, "-999", NULL);
00308 thresh_dc = scan_ctl(argc, argv, "THRESH_DC", -1, "-999", NULL);
00309 dt_trop = scan_ctl(argc, argv, "DT_TROP", -1, "0", NULL);
00310 dt230 = scan_ctl(argc, argv, "DT230", -1, "0.16", NULL);
00311 nu = scan_ctl(argc, argv, "NU", -1, "2345.0", NULL);
00312 output = (int) scan_ctl(argc, argv, "OUTPUT", -1, "1", NULL);
00313
00314 /* Allocate... */
00315 ALLOC(pert, pert_t, 1);
00316
00317 /* Check grid dimensions... */
00318 if (nx < 1 || nx > NX)
00319     ERRMSG("Set 1 <= NX <= MAX!");
00320 if (ny < 1 || ny > NY)
00321     ERRMSG("Set 1 <= NY <= MAX!");
00322
00323 /* Loop over perturbation files... */
00324 for (iarg = 3; iarg < argc; iarg++) {
00325
00326     /* Read perturbation data... */
00327     if (!(in = fopen(argv[iarg], "r")))
00328         continue;
00329     else {
00330         fclose(in);
00331         read_pert(argv[iarg], pertname, pert);
00332     }
00333
00334     /* Recalculate background and perturbations... */
00335     if (bg_poly_x > 0 || bg_poly_y > 0 ||
00336         bg_smooth_x > 0 || bg_smooth_y > 0 || gauss_fwhm > 0 || var_dh > 0) {
00337
00338         /* Allocate... */
00339         ALLOC(wave, wave_t, 1);
00340
00341         /* Convert to wave analysis struct... */
00342         pert2wave(pert, wave, 0, pert->ntrack - 1, 0, pert->ntrack - 1);
00343
00344         /* Estimate background... */
00345         background_poly(wave, bg_poly_x, bg_poly_y);
00346         background_smooth(wave, bg_smooth_x, bg_smooth_y);
00347
00348         /* Gaussian filter... */
00349         gauss(wave, gauss_fwhm);
00350
00351         /* Compute variance... */
00352         variance(wave, var_dh);
00353
00354         /* Copy data... */
00355         for (ix = 0; ix < wave->nx; ix++)
00356             for (iy = 0; iy < wave->ny; iy++) {
00357                 pert->pt[iy][ix] = wave->pt[ix][iy];
00358                 pert->var[iy][ix] = wave->var[ix][iy];
00359             }
00360
00361         /* Free... */
00362         free(wave);
00363     }
00364
00365     /* Detection... */
00366     for (itrack = 0; itrack < pert->ntrack; itrack++)
00367         for (ixtrack = 0; ixtrack < pert->ntrack; ixtrack++) {
00368
00369             /* Check data... */

```

```

00370     if (pert->time[itrack][ixtrack] < 0
00371         || pert->lon[itrack][ixtrack] < -180
00372         || pert->lon[itrack][ixtrack] > 180
00373         || pert->lat[itrack][ixtrack] < -90
00374         || pert->lat[itrack][ixtrack] > 90
00375         || pert->pt[itrack][ixtrack] < -100
00376         || pert->pt[itrack][ixtrack] > 100
00377         || !gsl_finite(pert->bt[itrack][ixtrack])
00378         || !gsl_finite(pert->pt[itrack][ixtrack])
00379         || !gsl_finite(pert->var[itrack][ixtrack])
00380         || !gsl_finite(pert->dc[itrack][ixtrack]))
00381     continue;
00382
00383     /* Get and check ascending/descending flag... */
00384     asc = (pert->lat[itrack] > 0 ? itrack : itrack + 1)[pert->ntrack / 2]
00385           > pert->lat[itrack >
00386               0 ? itrack - 1 : itrack][pert->ntrack / 2]);
00387     if ((!(set[0] == 'a' || set[0] == 'A') && !asc)
00388         || ((set[0] == 'd' || set[0] == 'D') && asc))
00389     continue;
00390
00391     /* Check am/pm flag... */
00392     lt = fmod(pert->time[itrack][ixtrack], 86400.) / 3600.;
00393     if ((!(set[0] == 'm' || set[0] == 'M') && lt > 12.)
00394         || ((set[0] == 'n' || set[0] == 'N') && lt < 12.))
00395     continue;
00396
00397     /* Get grid indices... */
00398     ix =
00399         (int) ((pert->lon[itrack][ixtrack] - lon0) / (lon1 -
00400                                                       lon0) * (double) nx);
00401     iy =
00402         (int) ((pert->lat[itrack][ixtrack] - lat0) / (lat1 -
00403                                                       lat0) * (double) ny);
00404     if (ix < 0 || ix >= nx || iy < 0 || iy >= ny)
00405     continue;
00406
00407     /* Get month index... */
00408     imon =
00409         (int) (fmod(pert->time[0][0] / 60. / 60. / 24. / 365.25, 1.) *
00410               NMON);
00411     if (imon < 0 || imon >= NMON)
00412     continue;
00413
00414     /* Get gravity wave detection threshold... */
00415     if (thresh_gw <= 0.0) {
00416         ilat = locate_irr(t_gw_lat, NLAT_GW, pert->lat[itrack][ixtrack]);
00417         if (asc)
00418             t_gw = LIN(t_gw_lat[ilat], t_gw_asc[imon][ilat],
00419                       t_gw_lat[ilat + 1], t_gw_asc[imon][ilat + 1],
00420                       pert->lat[itrack][ixtrack]);
00421         else
00422             t_gw = LIN(t_gw_lat[ilat], t_gw_dsc[imon][ilat],
00423                       t_gw_lat[ilat + 1], t_gw_dsc[imon][ilat + 1],
00424                       pert->lat[itrack][ixtrack]);
00425     } else
00426         t_gw = thresh_gw;
00427
00428     /* Get deep convection detection threshold... */
00429     if (thresh_dc <= 0.0) {
00430         ilat =
00431             locate_irr(t_trop_lat, NLAT_TROP, pert->lat[itrack][ixtrack]);
00432         t_dc =
00433             LIN(t_trop_lat[ilat], t_trop[imon][ilat], t_trop_lat[ilat + 1],
00434               t_trop[imon][ilat + 1], pert->lat[itrack][ixtrack]) + dt_trop;
00435     } else
00436         t_dc = thresh_dc + dt_trop;
00437
00438     /* Detection of gravity waves... */
00439     det_gw = (pert->var[itrack][ixtrack] >= t_gw);
00440
00441     /* Detection of convective waves... */
00442     det_cw = 0;
00443     if (det_gw)
00444         for (itrack2 = GSL_MAX(itrack - iradius, 0);
00445             itrack2 <= GSL_MIN(itrack + iradius, pert->ntrack - 1);
00446             itrack2++)
00447             for (ixtrack2 = GSL_MAX(ixtrack - iradius, 0);
00448                 ixtrack2 <= GSL_MIN(ixtrack + iradius, pert->ntrack - 1);
00449                 ixtrack2++) {
00450                 if (det_cw)
00451                     break;
00452                 det_cw = (pert->dc[itrack2][ixtrack2] <= t_dc);
00453             }
00454
00455     /* Detection of deep convection... */
00456     det_dc = (pert->dc[itrack][ixtrack] <= t_dc);

```

```

00457
00458     /* Detection of wave generation... */
00459     det_wg = 0;
00460     if (det_dc)
00461         for (itrack2 = GSL_MAX(itrack - iradius, 0);
00462              itrack2 <= GSL_MIN(itrack + iradius, pert->ntrack - 1);
00463              itrack2++)
00464             for (ixtrack2 = GSL_MAX(ixtrack - iradius, 0);
00465                  ixtrack2 <= GSL_MIN(ixtrack + iradius, pert->nxtrack - 1);
00466                  ixtrack2++) {
00467                 if (det_wg)
00468                     break;
00469                 det_wg = (pert->var[itrack2][ixtrack2] >= t_gw);
00470             }
00471
00472     /* Count events... */
00473     n[ix][iy]++;
00474     if (det_dc)
00475         ndc[ix][iy]++;
00476     if (det_wg)
00477         nwg[ix][iy]++;
00478     if (det_gw)
00479         ngw[ix][iy]++;
00480     if (det_cw)
00481         ncw[ix][iy]++;
00482
00483     /* Get statistics of perturbations... */
00484     mean[ix][iy] += pert->pt[itrack][ixtrack];
00485     var[ix][iy] += gsl_pow_2(pert->pt[itrack][ixtrack]);
00486     max[ix][iy] = GSL_MAX(max[ix][iy], pert->pt[itrack][ixtrack]);
00487     min[ix][iy] = GSL_MIN(min[ix][iy], pert->pt[itrack][ixtrack]);
00488
00489     /* Get statistics of brightness temperatures... */
00490     bt[ix][iy] += pert->bt[itrack][ixtrack];
00491     bt_8mu[ix][iy] += pert->dc[itrack][ixtrack];
00492     if (n[ix][iy] > 1) {
00493         bt_8mu_min[ix][iy]
00494             = GSL_MIN(bt_8mu_min[ix][iy], pert->dc[itrack][ixtrack]);
00495         bt_8mu_max[ix][iy]
00496             = GSL_MAX(bt_8mu_max[ix][iy], pert->dc[itrack][ixtrack]);
00497     } else {
00498         bt_8mu_min[ix][iy] = pert->dc[itrack][ixtrack];
00499         bt_8mu_max[ix][iy] = pert->dc[itrack][ixtrack];
00500     }
00501
00502     /* Get mean time... */
00503     mtime[ix][iy] += pert->time[itrack][ixtrack];
00504 }
00505 }
00506
00507 /* Analyze results... */
00508 for (ix = 0; ix < nx; ix++)
00509     for (iy = 0; iy < ny; iy++) {
00510
00511         /* Get geolocation... */
00512         mtime[ix][iy] /= (double) n[ix][iy];
00513         glon[ix]
00514             = lon0 + (ix + 0.5) / (double) nx * (
00515             lon1 - lon0);
00516         glat[iy]
00517             = lat0 + (iy + 0.5) / (double) ny * (
00518             lat1 - lat0);
00519
00520         /* Normalize brightness temperatures... */
00521         bt[ix][iy] /= (double) n[ix][iy];
00522         bt_8mu[ix][iy] /= (double) n[ix][iy];
00523
00524         /* Get fractions... */
00525         fdc[ix][iy] = (double) ndc[ix][iy] / (double) n[ix][iy] * 100.;
00526         fwg[ix][iy] = (double) nwg[ix][iy] / (double) ndc[ix][iy] * 100.;
00527         fgw[ix][iy] = (double) ngw[ix][iy] / (double) n[ix][iy] * 100.;
00528         fcw[ix][iy] = (double) ncw[ix][iy] / (double) ngw[ix][iy] * 100.;
00529
00530         /* Check number of observations... */
00531         if (n[ix][iy] < nmin) {
00532             fdc[ix][iy] = GSL_NAN;
00533             fwg[ix][iy] = GSL_NAN;
00534             fgw[ix][iy] = GSL_NAN;
00535             fcw[ix][iy] = GSL_NAN;
00536             bt_8mu[ix][iy] = GSL_NAN;
00537             bt_8mu_min[ix][iy] = GSL_NAN;
00538             bt_8mu_max[ix][iy] = GSL_NAN;
00539         }
00540
00541         /* Check detections of deep convection at high latitudes... */
00542         if (fabs(glat[iy]) > dc_hlat && bt_8mu[ix][iy] <= dc_tlim) {
00543             fdc[ix][iy] = GSL_NAN;

```

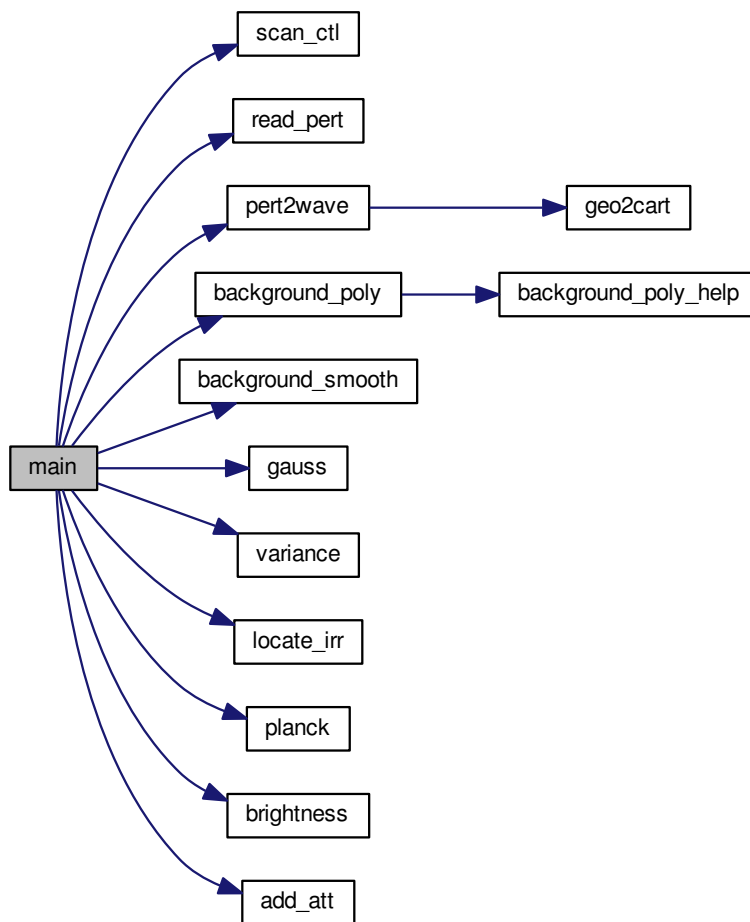
```

00544         fwg[ix][iy] = GSL_NAN;
00545         fcw[ix][iy] = GSL_NAN;
00546     }
00547
00548     /* Estimate noise... */
00549     if (dt230 > 0) {
00550         nesr = planck(230.0 + dt230, nu) - planck(230.0, nu);
00551         dt[ix][iy] =
00552             brightness(planck(bt[ix][iy], nu) + nesr, nu) - bt[ix][iy];
00553     }
00554
00555     /* Get mean perturbation and variance... */
00556     mean[ix][iy] /= (double) n[ix][iy];
00557     var[ix][iy] =
00558         var[ix][iy] / (double) n[ix][iy] - gsl_pow_2(mean[ix][iy]);
00559 }
00560
00561 /* Write ASCII file... */
00562 if (output == 1) {
00563
00564     /* Create file... */
00565     printf("Write variance statistics: %s\n", argv[2]);
00566     if (!out = fopen(argv[2], "w"))
00567         ERRMSG("Cannot create file!");
00568
00569     /* Write header... */
00570     fprintf(out,
00571         "# $1 = time [s]\n"
00572         "# $2 = longitude [deg]\n"
00573         "# $3 = latitude [deg]\n"
00574         "# $4 = number of footprints\n"
00575         "# $5 = fraction of convection events [%]\n"
00576         "# $6 = fraction of wave generating events [%]\n"
00577         "# $7 = fraction of gravity wave events [%]\n"
00578         "# $8 = fraction of convective wave events [%]\n"
00579         "# $9 = mean perturbation [K]\n"
00580         "# $10 = minimum perturbation [K]\n");
00581     fprintf(out,
00582         "# $11 = maximum perturbation [K]\n"
00583         "# $12 = variance [K^2]\n"
00584         "# $13 = mean surface temperature [K]\n"
00585         "# $14 = minimum surface temperature [K]\n"
00586         "# $15 = maximum surface temperature [K]\n"
00587         "# $16 = mean background temperature [K]\n"
00588         "# $17 = noise estimate [K]\n");
00589
00590     /* Write results... */
00591     for (iy = 0; iy < ny; iy++) {
00592         if (iy == 0 || nx > 1)
00593             fprintf(out, "\n");
00594         for (ix = 0; ix < nx; ix++)
00595             fprintf(out, "%2f %g %g %d %g %g %g %g %g %g %g %g %g %g %g\n",
00596                 mtime[ix][iy], glon[ix], glat[iy], n[ix][iy],
00597                 fdc[ix][iy], fwg[ix][iy], fgw[ix][iy], fcw[ix][iy],
00598                 mean[ix][iy], min[ix][iy], max[ix][iy], var[ix][iy],
00599                 bt_8mu[ix][iy], bt_8mu_min[ix][iy], bt_8mu_max[ix][iy],
00600                 bt[ix][iy], dt[ix][iy]);
00601     }
00602
00603     /* Close file... */
00604     fclose(out);
00605 }
00606
00607 /* Write netCDF file... */
00608 else if (output == 2) {
00609
00610     /* Create netCDF file... */
00611     printf("Write variance statistics: %s\n", argv[2]);
00612     NC(nc_create(argv[2], NC_CLOBBER, &ncid));
00613
00614     /* Set dimensions... */
00615     NC(nc_def_dim(ncid, "lat", (size_t) ny, &dimid[0]));
00616     NC(nc_def_dim(ncid, "lon", (size_t) nx, &dimid[1]));
00617
00618     /* Add variables... */
00619     NC(nc_def_var(ncid, "lat", NC_DOUBLE, 1, &dimid[0], &latid));
00620     add_att(ncid, latid, "deg", "latitude");
00621     NC(nc_def_var(ncid, "lon", NC_DOUBLE, 1, &dimid[1], &lonid));
00622     add_att(ncid, lonid, "deg", "longitude");
00623     NC(nc_def_var(ncid, "var", NC_FLOAT, 2, dimid, &varid));
00624     add_att(ncid, varid, "K^2", "brightness temperature variance");
00625     NC(nc_def_var(ncid, "min", NC_FLOAT, 2, dimid, &minid));
00626     add_att(ncid, minid, "K", "brightness temperature minimum");
00627     NC(nc_def_var(ncid, "max", NC_FLOAT, 2, dimid, &maxid));
00628     add_att(ncid, maxid, "K", "brightness temperature maximum");
00629     NC(nc_def_var(ncid, "np", NC_INT, 2, dimid, &npid));
00630     add_att(ncid, npid, "1", "number of footprints");

```

```
00631
00632     /* Leave define mode... */
00633     NC(nc_enddef(ncid));
00634
00635     /* Write data... */
00636     NC(nc_put_var_double(ncid, latid, glat));
00637     NC(nc_put_var_double(ncid, lonid, glon));
00638     for (ix = 0; ix < nx; ix++)
00639         for (iy = 0; iy < ny; iy++)
00640             help[iy * nx + ix] = var[ix][iy] - POW2(dt[ix][iy]);
00641     NC(nc_put_var_double(ncid, varid, help));
00642     for (ix = 0; ix < nx; ix++)
00643         for (iy = 0; iy < ny; iy++)
00644             help[iy * nx + ix] = min[ix][iy];
00645     NC(nc_put_var_double(ncid, minid, help));
00646     for (ix = 0; ix < nx; ix++)
00647         for (iy = 0; iy < ny; iy++)
00648             help[iy * nx + ix] = max[ix][iy];
00649     NC(nc_put_var_double(ncid, maxid, help));
00650     for (ix = 0; ix < nx; ix++)
00651         for (iy = 0; iy < ny; iy++)
00652             help2[iy * nx + ix] = n[ix][iy];
00653     NC(nc_put_var_int(ncid, npid, help2));
00654
00655     /* Close file... */
00656     NC(nc_close(ncid));
00657 }
00658
00659 else
00660     ERRMSG("Unknown output format!");
00661
00662 /* Free... */
00663 free(pert);
00664
00665 return EXIT_SUCCESS;
00666 }
```

Here is the call graph for this function:



## 5.64 variance.c

```

00001 #include "libairs.h"
00002
00003 /* -----
00004    Dimensions...
00005    ----- */
00006
00007 /* Number of latitudes for threshold tables. */
00008 #define NLAT_GW 19
00009 #define NLAT_SURF 6
00010 #define NLAT_TROP 73
00011
00012 /* Number of months for threshold tables. */
00013 #define NMON 12
00014
00015 /* Maximum number of longitudes. */
00016 #define NX 3600
00017
00018 /* Maximum number of latitudes. */
00019 #define NY 1800
00020
00021 /* -----
00022    Global variables...
00023    ----- */
00024

```



```

00025 /* Latitudes for gravity wave variance thresholds. */
00026 static double t_gw_lat[NLAT_GW]
00027 = { -90, -80, -70, -60, -50, -40, -30, -20, -10, 0,
00028    10, 20, 30, 40, 50, 60, 70, 80, 90
00029 };
00030
00031 /* Gravity wave variance thresholds (ascending orbits). */
00032 static double t_gw_asc[NMON][NLAT_GW]
00033 = { { 0.00387, 0.00422, 0.00633, 0.0124, 0.0216, 0.0324,
00034       0.0553, 0.0791, 0.0501, 0.0136, 0.0134, 0.0151,
00035       0.0522, 0.321, 0.697, 0.776, 0.696, 0.764, 0.771},
00036 { 0.00913, 0.00942, 0.00867, 0.00897, 0.0112, 0.0168,
00037   0.0314, 0.0484, 0.032, 0.0128, 0.0122, 0.0134,
00038   0.0382, 0.124, 0.345, 0.404, 0.545, 1.16, 1.18},
00039 { 0.0845, 0.0664, 0.0384, 0.0227, 0.0147, 0.0118,
00040   0.0141, 0.0184, 0.0162, 0.0123, 0.0124, 0.0124,
00041   0.0159, 0.0509, 0.085, 0.103, 0.188, 0.367, 0.529},
00042 { 0.265, 0.297, 0.216, 0.106, 0.0666, 0.0299,
00043   0.0169, 0.0129, 0.0116, 0.012, 0.0135, 0.0141,
00044   0.0134, 0.0137, 0.017, 0.0268, 0.0259, 0.0319, 0.0323},
00045 { 0.326, 0.44, 0.628, 0.567, 0.434, 0.235,
00046   0.0601, 0.0214, 0.0132, 0.0113, 0.0144, 0.0185,
00047   0.0179, 0.0142, 0.0116, 0.00945, 0.00865, 0.00918, 0.00878},
00048 { 0.537, 0.73, 1.39, 1.75, 1.35, 0.528,
00049   0.188, 0.0311, 0.0133, 0.0124, 0.0205, 0.0313,
00050   0.0297, 0.0216, 0.0166, 0.0131, 0.00983, 0.00606, 0.0049},
00051 { 0.382, 1.15, 1.57, 2.13, 1.66, 0.851,
00052   0.126, 0.0204, 0.0133, 0.0135, 0.0281, 0.0385,
00053   0.0375, 0.0312, 0.0223, 0.0143, 0.00949, 0.0061, 0.00493},
00054 { 0.226, 0.697, 1.68, 1.56, 1.14, 0.496,
00055   0.0616, 0.0143, 0.0126, 0.013, 0.0216, 0.0252,
00056   0.0241, 0.0206, 0.0152, 0.0106, 0.00976, 0.0105, 0.00998},
00057 { 0.236, 0.489, 0.648, 0.553, 0.524, 0.21,
00058   0.033, 0.0129, 0.0116, 0.0129, 0.0163, 0.0165,
00059   0.0153, 0.014, 0.0141, 0.0185, 0.0301, 0.0591, 0.0745},
00060 { 0.046, 0.082, 0.112, 0.0806, 0.0516, 0.0469,
00061   0.0225, 0.0139, 0.0127, 0.0121, 0.0125, 0.0138,
00062   0.0176, 0.0357, 0.0563, 0.062, 0.133, 0.327, 0.3},
00063 { 0.00669, 0.00867, 0.0117, 0.0117, 0.014, 0.015,
00064   0.0203, 0.0213, 0.0144, 0.0116, 0.0124, 0.0179,
00065   0.0574, 0.185, 0.346, 0.442, 0.54, 0.669, 0.664},
00066 { 0.00355, 0.00381, 0.00658, 0.0125, 0.0217, 0.0304,
00067   0.0424, 0.0515, 0.0315, 0.0139, 0.0137, 0.0161,
00068   0.0582, 0.306, 0.999, 1.2, 1.14, 0.621, 0.448}
00069 };
00070
00071 /* Gravity wave variance thresholds (descending orbits). */
00072 static double t_gw_dsc[NMON][NLAT_GW]
00073 = { { 0.00383, 0.00458, 0.00866, 0.019, 0.0348, 0.0598,
00074       0.144, 0.234, 0.135, 0.0373, 0.0325, 0.0377,
00075       0.0858, 0.497, 1.4, 1.32, 0.808, 0.771, 0.773},
00076 { 0.00999, 0.0123, 0.0141, 0.0148, 0.0177, 0.0286,
00077   0.0626, 0.102, 0.0717, 0.0302, 0.0261, 0.03,
00078   0.086, 0.268, 0.631, 0.716, 1.17, 1.24, 1.21},
00079 { 0.103, 0.096, 0.0715, 0.0535, 0.0343, 0.0245,
00080   0.025, 0.0315, 0.0303, 0.0233, 0.023, 0.0257,
00081   0.0353, 0.118, 0.197, 0.359, 0.541, 0.585, 0.586},
00082 { 0.272, 0.293, 0.276, 0.226, 0.146, 0.0689,
00083   0.0373, 0.0245, 0.0232, 0.0232, 0.0224, 0.0217,
00084   0.0242, 0.031, 0.0441, 0.0664, 0.0623, 0.053, 0.0361},
00085 { 0.331, 0.44, 0.641, 0.868, 0.824, 0.47,
00086   0.115, 0.0444, 0.0269, 0.0223, 0.0274, 0.0332,
00087   0.0273, 0.023, 0.0191, 0.0172, 0.0138, 0.0107, 0.00894},
00088 { 0.554, 0.716, 1.31, 2.29, 2.43, 1.05,
00089   0.41, 0.0651, 0.0269, 0.0257, 0.0447, 0.0622,
00090   0.0497, 0.0357, 0.0258, 0.0182, 0.0117, 0.00697, 0.00502},
00091 { 0.427, 0.905, 1.44, 2.78, 2.76, 1.52,
00092   0.278, 0.041, 0.0279, 0.0296, 0.0629, 0.0818,
00093   0.0758, 0.0534, 0.0356, 0.0227, 0.012, 0.00692, 0.00513},
00094 { 0.245, 0.74, 1.88, 2.32, 1.89, 0.883,
00095   0.122, 0.0292, 0.0264, 0.0289, 0.0516, 0.059,
00096   0.0495, 0.0373, 0.0268, 0.0185, 0.0163, 0.0131, 0.0103},
00097 { 0.272, 0.551, 0.812, 0.844, 0.852, 0.486,
00098   0.0842, 0.0269, 0.0225, 0.0239, 0.0322, 0.0324,
00099   0.0307, 0.0304, 0.035, 0.0484, 0.0692, 0.0956, 0.0948},
00100 { 0.0644, 0.125, 0.177, 0.135, 0.0922, 0.0899,
00101   0.0524, 0.0249, 0.0214, 0.0218, 0.0251, 0.0293,
00102   0.0403, 0.0903, 0.168, 0.246, 0.358, 0.378, 0.288},
00103 { 0.00676, 0.00923, 0.0148, 0.0195, 0.0261, 0.0286,
00104   0.0302, 0.0343, 0.0298, 0.024, 0.0252, 0.0403,
00105   0.131, 0.448, 0.681, 0.923, 0.839, 0.684, 0.629},
00106 { 0.00347, 0.00412, 0.00995, 0.0221, 0.0363, 0.0531,
00107   0.104, 0.168, 0.112, 0.0365, 0.0335, 0.0382,
00108   0.128, 0.563, 1.62, 1.87, 1.47, 0.652, 0.408}
00109 };
00110
00111 /* Latitudes for zonal mean tropopause temperatures. */

```

```

00112 static double t_trop_lat[NLAT_TROP]
00113 = { 90, 87.5, 85, 82.5, 80, 77.5, 75, 72.5, 70, 67.5, 65, 62.5, 60,
00114 57.5, 55, 52.5, 50, 47.5, 45, 42.5, 40, 37.5, 35, 32.5, 30, 27.5,
00115 25, 22.5, 20, 17.5, 15, 12.5, 10, 7.5, 5, 2.5, 0, -2.5, -5, -7.5,
00116 -10, -12.5, -15, -17.5, -20, -22.5, -25, -27.5, -30, -32.5, -35,
00117 -37.5, -40, -42.5, -45, -47.5, -50, -52.5, -55, -57.5, -60, -62.5,
00118 -65, -67.5, -70, -72.5, -75, -77.5, -80, -82.5, -85, -87.5, -90
00119 };
00120
00121 /* Zonal mean tropopause temperatures. */
00122 static double t_trop[NMON][NLAT_TROP]
00123 = { {211.152, 211.237, 211.434, 211.549, 211.614, 211.776, 211.974,
00124 212.234, 212.489, 212.808, 213.251, 213.692, 214.193, 214.591,
00125 214.985, 215.327, 215.658, 215.956, 216.236, 216.446, 216.738,
00126 216.836, 216.032, 213.607, 209.281, 205, 201.518, 198.969,
00127 197.123, 195.869, 195.001, 194.409, 193.985, 193.734, 193.617,
00128 193.573, 193.6, 193.642, 193.707, 193.856, 194.131, 194.558,
00129 195.121, 195.907, 196.91, 198.192, 199.744, 201.583, 203.672,
00130 206.012, 208.542, 211.135, 213.681, 216.085, 218.317, 220.329,
00131 222.071, 223.508, 224.612, 225.357, 225.761, 225.863, 225.657,
00132 225.287, 224.813, 224.571, 224.385, 224.3, 224.257, 224.173,
00133 223.786, 222.713, 222.11},
00134 {212.593, 212.621, 212.801, 212.888, 212.912, 213.054, 213.245,
00135 213.512, 213.726, 213.962, 214.259, 214.508, 214.823, 215.037,
00136 215.297, 215.545, 215.808, 216.063, 216.323, 216.539, 216.867,
00137 217.051, 216.532, 214.512, 210.371, 205.658, 201.758, 198.937,
00138 197.047, 195.817, 194.96, 194.386, 193.993, 193.771, 193.673,
00139 193.635, 193.658, 193.691, 193.744, 193.872, 194.126, 194.54,
00140 195.085, 195.847, 196.8, 198.013, 199.489, 201.261, 203.298,
00141 205.596, 208.082, 210.628, 213.156, 215.563, 217.822, 219.903,
00142 221.745, 223.311, 224.566, 225.451, 225.947, 226.079, 225.849,
00143 225.406, 224.889, 224.643, 224.431, 224.246, 224.079, 223.884,
00144 223.42, 222.402, 221.871},
00145 {215.529, 215.491, 215.539, 215.621, 215.691, 215.808, 215.847,
00146 215.881, 215.878, 215.907, 216.02, 216.113, 216.297, 216.342,
00147 216.38, 216.369, 216.342, 216.284, 216.185, 215.989, 215.855,
00148 215.626, 215.023, 213.432, 209.979, 205.886, 202.212, 199.414,
00149 197.488, 196.216, 195.327, 194.732, 194.347, 194.158, 194.095,
00150 194.079, 194.116, 194.154, 194.195, 194.302, 194.534, 194.922,
00151 195.461, 196.253, 197.288, 198.644, 200.309, 202.293, 204.553,
00152 207.033, 209.538, 211.911, 214.016, 215.862, 217.572, 219.179,
00153 220.655, 221.959, 223.052, 223.867, 224.344, 224.451, 224.179,
00154 223.706, 223.163, 222.876, 222.613, 222.385, 222.154, 221.842,
00155 221.304, 220.402, 220.06},
00156 {219.921, 219.916, 219.99, 219.989, 219.916, 219.867, 219.73,
00157 219.522, 219.16, 218.765, 218.448, 218.144, 217.99, 217.756,
00158 217.553, 217.311, 217.025, 216.684, 216.241, 215.649, 215.05,
00159 214.302, 213.219, 211.496, 208.729, 205.649, 202.594, 200.066,
00160 198.144, 196.733, 195.687, 194.991, 194.586, 194.429, 194.418,
00161 194.443, 194.492, 194.534, 194.59, 194.718, 194.997, 195.481,
00162 196.165, 197.159, 198.462, 200.142, 202.154, 204.533, 207.208,
00163 209.848, 212.088, 213.845, 215.222, 216.348, 217.384, 218.383,
00164 219.313, 220.131, 220.799, 221.271, 221.479, 221.405, 221.012,
00165 220.4, 219.702, 219.227, 218.827, 218.434, 217.977, 217.477,
00166 216.783, 215.974, 215.707},
00167 {225.363, 225.255, 225.064, 224.745, 224.351, 224, 223.551,
00168 222.966, 222.195, 221.435, 220.802, 220.245, 219.871, 219.424,
00169 218.99, 218.529, 218.013, 217.445, 216.76, 215.859, 214.723,
00170 213.049, 211.032, 208.767, 206.449, 204.302, 202.113, 200.187,
00171 198.501, 197.153, 196.117, 195.441, 195.121, 195.073, 195.146,
00172 195.212, 195.261, 195.288, 195.343, 195.485, 195.772, 196.284,
00173 197.018, 198.125, 199.624, 201.604, 204.073, 207.036, 210.193,
00174 212.853, 214.611, 215.635, 216.287, 216.801, 217.284, 217.716,
00175 218.057, 218.253, 218.282, 218.115, 217.729, 217.15, 216.376,
00176 215.449, 214.428, 213.574, 212.847, 212.281, 211.718, 211.211,
00177 210.616, 210.112, 210.056},
00178 {228.431, 228.261, 227.966, 227.457, 226.812, 226.208, 225.518,
00179 224.71, 223.701, 222.762, 222.045, 221.486, 221.142, 220.761,
00180 220.361, 219.896, 219.34, 218.646, 217.626, 215.983, 213.624,
00181 210.817, 208.017, 205.73, 203.8, 202.363, 200.96, 199.778,
00182 198.695, 197.845, 197.166, 196.743, 196.6, 196.66, 196.809,
00183 196.925, 196.985, 196.996, 197.033, 197.135, 197.335, 197.754,
00184 198.367, 199.335, 200.693, 202.564, 205.001, 208.084, 211.473,
00185 214.407, 216.208, 217.018, 217.314, 217.394, 217.371, 217.234,
00186 216.961, 216.517, 215.878, 215.027, 213.952, 212.697, 211.274,
00187 209.736, 208.172, 206.872, 205.84, 205.093, 204.32, 203.816,
00188 203.55, 203.49, 203.606},
00189 {229.01, 228.807, 228.45, 227.839, 227.084, 226.377, 225.589,
00190 224.712, 223.665, 222.724, 222.058, 221.658, 221.519, 221.376,
00191 221.136, 220.673, 219.926, 218.742, 216.744, 214.028, 210.994,
00192 208.374, 206.131, 204.563, 203.251, 202.328, 201.313, 200.411,
00193 199.531, 198.876, 198.356, 198.104, 198.088, 198.21, 198.385,
00194 198.502, 198.57, 198.601, 198.652, 198.731, 198.869, 199.207,
00195 199.737, 200.595, 201.802, 203.491, 205.771, 208.765, 212.241,
00196 215.403, 217.439, 218.251, 218.297, 217.988, 217.533, 216.941,
00197 216.161, 215.154, 213.887, 212.35, 210.525, 208.481, 206.287,
00198 204.068, 202.033, 200.405, 199.106, 198.225, 197.435, 197.02,

```

```

00199 197.133, 197.527, 197.808},
00200 {226.525, 226.354, 225.996, 225.433, 224.842, 224.358, 223.818,
00201 223.202, 222.426, 221.723, 221.266, 220.98, 220.893, 220.707,
00202 220.392, 219.928, 219.182, 218.015, 216.051, 213.399, 210.617,
00203 208.318, 206.311, 204.838, 203.515, 202.527, 201.397, 200.423,
00204 199.494, 198.848, 198.385, 198.212, 198.294, 198.49, 198.707,
00205 198.853, 198.933, 198.967, 199.01, 199.079, 199.207, 199.537,
00206 200.081, 200.968, 202.215, 203.946, 206.254, 209.291, 212.876,
00207 216.262, 218.487, 219.387, 219.436, 219.048, 218.405, 217.527,
00208 216.372, 214.919, 213.152, 211.096, 208.767, 206.247, 203.609,
00209 201.029, 198.763, 196.961, 195.578, 194.635, 193.923, 193.54,
00210 193.632, 193.944, 193.912},
00211 {223.293, 223.158, 222.945, 222.571, 222.126, 221.749, 221.362,
00212 220.946, 220.404, 219.946, 219.704, 219.599, 219.611, 219.429,
00213 219.124, 218.702, 218.063, 217.157, 215.827, 213.879, 211.352,
00214 208.833, 206.504, 204.728, 203.168, 201.992, 200.735, 199.74,
00215 198.833, 198.213, 197.801, 197.661, 197.765, 197.963, 198.182,
00216 198.336, 198.42, 198.456, 198.505, 198.609, 198.794, 199.19,
00217 199.796, 200.758, 202.089, 203.915, 206.262, 209.295, 212.807,
00218 216.083, 218.329, 219.47, 219.877, 219.846, 219.507, 218.85,
00219 217.84, 216.448, 214.652, 212.509, 210.083, 207.534, 204.982,
00220 202.596, 200.463, 198.769, 197.441, 196.546, 195.902, 195.472,
00221 195.193, 195.066, 195.006},
00222 {219.564, 219.492, 219.415, 219.191, 218.926, 218.801, 218.691,
00223 218.561, 218.298, 218.06, 217.982, 217.956, 218.038, 217.954,
00224 217.81, 217.532, 217.08, 216.439, 215.549, 214.31, 212.725,
00225 210.573, 208.019, 205.585, 203.459, 201.779, 200.162, 198.879,
00226 197.771, 196.987, 196.459, 196.19, 196.172, 196.274, 196.435,
00227 196.544, 196.601, 196.644, 196.727, 196.904, 197.184, 197.696,
00228 198.42, 199.497, 200.934, 202.825, 205.151, 208.005, 211.279,
00229 214.441, 216.87, 218.493, 219.498, 220.072, 220.353, 220.336,
00230 219.991, 219.271, 218.142, 216.636, 214.804, 212.776, 210.636,
00231 208.535, 206.516, 204.825, 203.383, 202.281, 201.365, 200.561,
00232 199.896, 199.415, 199.382},
00233 {215.926, 215.884, 215.897, 215.814, 215.689, 215.692, 215.707,
00234 215.767, 215.815, 215.92, 216.138, 216.327, 216.588, 216.668,
00235 216.664, 216.553, 216.373, 216.112, 215.711, 215.025, 214.106,
00236 212.596, 210.346, 207.503, 204.604, 202.251, 200.231, 198.607,
00237 197.228, 196.174, 195.382, 194.87, 194.61, 194.54, 194.579,
00238 194.615, 194.66, 194.709, 194.82, 195.074, 195.487, 196.103,
00239 196.904, 198.01, 199.43, 201.246, 203.431, 206.007, 208.905,
00240 211.81, 214.34, 216.36, 217.918, 219.141, 220.159, 220.965,
00241 221.514, 221.754, 221.637, 221.135, 220.226, 218.986, 217.475,
00242 215.879, 214.251, 212.918, 211.84, 211.026, 210.288, 209.553,
00243 208.791, 208.132, 208.053},
00244 {212.893, 212.911, 213.03, 213.109, 213.224, 213.453, 213.653,
00245 213.836, 213.98, 214.166, 214.481, 214.787, 215.179, 215.435,
00246 215.688, 215.908, 216.084, 216.217, 216.262, 216.123, 215.819,
00247 214.977, 213.173, 210.214, 206.619, 203.437, 200.836, 198.843,
00248 197.271, 196.078, 195.164, 194.509, 194.057, 193.82, 193.742,
00249 193.723, 193.762, 193.813, 193.903, 194.121, 194.49, 195.016,
00250 195.698, 196.627, 197.82, 199.359, 201.204, 203.355, 205.78,
00251 208.414, 211.057, 213.521, 215.662, 217.504, 219.133, 220.544,
00252 221.723, 222.631, 223.274, 223.649, 223.737, 223.547, 223.053,
00253 222.357, 221.52, 220.948, 220.527, 220.247, 220.013, 219.726,
00254 219.273, 218.506, 218.144}
00255 };
00256
00257 /* -----
00258 Main...
00259 ----- */
00260
00261 int main(
00262     int argc,
00263     char *argv[]) {
00264
00265     static pert_t *pert;
00266
00267     static wave_t *wave;
00268
00269     static FILE *in, *out;
00270
00271     static char pertname[LEN], set[LEN];
00272
00273     static double bt[NX][NY], bt_8mu[NX][NY], bt_8mu_min[NX][NY],
00274         bt_8mu_max[NX][NY], dt[NX][NY], mtime[NX][NY], glat[NY], glon[NX],
00275         fdc[NX][NY], fwg[NX][NY], fgw[NX][NY], fcw[NX][NY],
00276         mean[NX][NY], min[NX][NY], max[NX][NY], var[NX][NY],
00277         t_dc, t_gw, dt_trop, dc_hlat = 25, dc_tlim = 250, dt230,
00278         nesr, gauss_fwhm, var_dh, nu, lon0, lon1, lat0, lat1,
00279         thresh_dc, thresh_gw, lt, help[NX * NY];
00280
00281     static int asc, ix, iy, nx, ny, iarg, n[NX][NY],
00282         ndc[NX][NY], ngw[NX][NY], ncw[NX][NY], nwg[NX][NY],
00283         det_gw, det_cw, det_dc, det_wg, ilat, imon, nmin = 10,
00284         bg_poly_x, bg_poly_y, bg_smooth_x, bg_smooth_y,
00285         itrack, itrack2, ixtrack, ixtrack2, iradius = 30, output, ncid, varid,

```

```

00286     minid, maxid, lonid, latid, npid, dimid[10], help2[NX * NY];
00287
00288     /* Check arguments... */
00289     if (argc < 4)
00290         ERRMSG("Give parameters: <ctl> <var.tab> <pert1.nc> [<pert2.nc> ...]");
00291
00292     /* Get control parameters... */
00293     scan_ctl(argc, argv, "SET", -1, "full", set);
00294     scan_ctl(argc, argv, "PERTNAME", -1, "4mu", pertname);
00295     nx = (int) scan_ctl(argc, argv, "NX", -1, "360", NULL);
00296     lon0 = scan_ctl(argc, argv, "LON0", -1, "-180", NULL);
00297     lon1 = scan_ctl(argc, argv, "LON1", -1, "180", NULL);
00298     ny = (int) scan_ctl(argc, argv, "NY", -1, "180", NULL);
00299     lat0 = scan_ctl(argc, argv, "LAT0", -1, "-90", NULL);
00300     lat1 = scan_ctl(argc, argv, "LAT1", -1, "90", NULL);
00301     bg_poly_x = (int) scan_ctl(argc, argv, "BG_POLY_X", -1, "0", NULL);
00302     bg_poly_y = (int) scan_ctl(argc, argv, "BG_POLY_Y", -1, "0", NULL);
00303     bg_smooth_x = (int) scan_ctl(argc, argv, "BG_SMOOTH_X", -1, "0", NULL);
00304     bg_smooth_y = (int) scan_ctl(argc, argv, "BG_SMOOTH_Y", -1, "0", NULL);
00305     gauss_fwhm = scan_ctl(argc, argv, "GAUSS_FWHM", -1, "0", NULL);
00306     var_dh = scan_ctl(argc, argv, "VAR_DH", -1, "0", NULL);
00307     thresh_gw = scan_ctl(argc, argv, "THRESH_GW", -1, "-999", NULL);
00308     thresh_dc = scan_ctl(argc, argv, "THRESH_DC", -1, "-999", NULL);
00309     dt_trop = scan_ctl(argc, argv, "DT_TROP", -1, "0", NULL);
00310     dt230 = scan_ctl(argc, argv, "DT230", -1, "0.16", NULL);
00311     nu = scan_ctl(argc, argv, "NU", -1, "2345.0", NULL);
00312     output = (int) scan_ctl(argc, argv, "OUTPUT", -1, "1", NULL);
00313
00314     /* Allocate... */
00315     ALLOC(pert, pert_t, 1);
00316
00317     /* Check grid dimensions... */
00318     if (nx < 1 || nx > NX)
00319         ERRMSG("Set 1 <= NX <= MAX!");
00320     if (ny < 1 || ny > NY)
00321         ERRMSG("Set 1 <= NY <= MAX!");
00322
00323     /* Loop over perturbation files... */
00324     for (iarg = 3; iarg < argc; iarg++) {
00325
00326         /* Read perturbation data... */
00327         if (! (in = fopen(argv[iarg], "r")))
00328             continue;
00329         else {
00330             fclose(in);
00331             read_pert(argv[iarg], pertname, pert);
00332         }
00333
00334         /* Recalculate background and perturbations... */
00335         if (bg_poly_x > 0 || bg_poly_y > 0 ||
00336             bg_smooth_x > 0 || bg_smooth_y > 0 || gauss_fwhm > 0 || var_dh > 0) {
00337
00338             /* Allocate... */
00339             ALLOC(wave, wave_t, 1);
00340
00341             /* Convert to wave analysis struct... */
00342             pert2wave(pert, wave, 0, pert->ntrack - 1, 0, pert->ntrack - 1);
00343
00344             /* Estimate background... */
00345             background_poly(wave, bg_poly_x, bg_poly_y);
00346             background_smooth(wave, bg_smooth_x, bg_smooth_y);
00347
00348             /* Gaussian filter... */
00349             gauss(wave, gauss_fwhm);
00350
00351             /* Compute variance... */
00352             variance(wave, var_dh);
00353
00354             /* Copy data... */
00355             for (ix = 0; ix < wave->nx; ix++)
00356                 for (iy = 0; iy < wave->ny; iy++) {
00357                     pert->pt[iy][ix] = wave->pt[ix][iy];
00358                     pert->var[iy][ix] = wave->var[ix][iy];
00359                 }
00360
00361             /* Free... */
00362             free(wave);
00363         }
00364
00365         /* Detection... */
00366         for (itrack = 0; itrack < pert->ntrack; itrack++)
00367             for (ixtrack = 0; ixtrack < pert->ntrack; ixtrack++) {
00368
00369                 /* Check data... */
00370                 if (pert->time[itrack][ixtrack] < 0
00371                     || pert->lon[itrack][ixtrack] < -180
00372                     || pert->lon[itrack][ixtrack] > 180

```

```

00373         || pert->lat[itrack][ixtrack] < -90
00374         || pert->lat[itrack][ixtrack] > 90
00375         || pert->pt[itrack][ixtrack] < -100
00376         || pert->pt[itrack][ixtrack] > 100
00377         || !gsl_finite(pert->bt[itrack][ixtrack])
00378         || !gsl_finite(pert->pt[itrack][ixtrack])
00379         || !gsl_finite(pert->var[itrack][ixtrack])
00380         || !gsl_finite(pert->dc[itrack][ixtrack])
00381     continue;
00382
00383     /* Get and check ascending/descending flag... */
00384     asc = (pert->lat[itrack] > 0 ? itrack : itrack + 1)[pert->nxtrack / 2]
00385           > pert->lat[itrack >
00386               0 ? itrack - 1 : itrack][pert->nxtrack / 2]);
00387     if ((set[0] == 'a' || set[0] == 'A') && !asc)
00388         || ((set[0] == 'd' || set[0] == 'D') && asc)
00389         continue;
00390
00391     /* Check am/pm flag... */
00392     lt = fmod(pert->time[itrack][ixtrack], 86400.) / 3600.;
00393     if ((set[0] == 'm' || set[0] == 'M') && lt > 12.)
00394         || ((set[0] == 'n' || set[0] == 'N') && lt < 12.)
00395         continue;
00396
00397     /* Get grid indices... */
00398     ix =
00399         (int) ((pert->lon[itrack][ixtrack] - lon0) / (lon1 -
00400                                                       lon0) * (double) nx);
00401     iy =
00402         (int) ((pert->lat[itrack][ixtrack] - lat0) / (lat1 -
00403                                                       lat0) * (double) ny);
00404     if (ix < 0 || ix >= nx || iy < 0 || iy >= ny)
00405         continue;
00406
00407     /* Get month index... */
00408     imon =
00409         (int) (fmod(pert->time[0][0] / 60. / 60. / 24. / 365.25, 1.) *
00410               NMON);
00411     if (imon < 0 || imon >= NMON)
00412         continue;
00413
00414     /* Get gravity wave detection threshold... */
00415     if (thresh_gw <= 0.0) {
00416         ilat = locate_irr(t_gw_lat, NLAT_GW, pert->lat[itrack][ixtrack]);
00417         if (asc)
00418             t_gw = LIN(t_gw_lat[ilat], t_gw_asc[imon][ilat],
00419                       t_gw_lat[ilat + 1], t_gw_asc[imon][ilat + 1],
00420                       pert->lat[itrack][ixtrack]);
00421         else
00422             t_gw = LIN(t_gw_lat[ilat], t_gw_dsc[imon][ilat],
00423                       t_gw_lat[ilat + 1], t_gw_dsc[imon][ilat + 1],
00424                       pert->lat[itrack][ixtrack]);
00425     } else
00426         t_gw = thresh_gw;
00427
00428     /* Get deep convection detection threshold... */
00429     if (thresh_dc <= 0.0) {
00430         ilat =
00431             locate_irr(t_trop_lat, NLAT_TROP, pert->lat[itrack][ixtrack]);
00432         t_dc =
00433             LIN(t_trop_lat[ilat], t_trop[imon][ilat], t_trop_lat[ilat + 1],
00434               t_trop[imon][ilat + 1], pert->lat[itrack][ixtrack]) + dt_trop;
00435     } else
00436         t_dc = thresh_dc + dt_trop;
00437
00438     /* Detection of gravity waves... */
00439     det_gw = (pert->var[itrack][ixtrack] >= t_gw);
00440
00441     /* Detection of convective waves... */
00442     det_cw = 0;
00443     if (det_gw)
00444         for (itrack2 = GSL_MAX(itrack - iradius, 0);
00445              itrack2 <= GSL_MIN(itrack + iradius, pert->nxtrack - 1);
00446              itrack2++)
00447             for (ixtrack2 = GSL_MAX(ixtrack - iradius, 0);
00448                  ixtrack2 <= GSL_MIN(ixtrack + iradius, pert->nxtrack - 1);
00449                  ixtrack2++) {
00450                 if (det_cw)
00451                     break;
00452                 det_cw = (pert->dc[itrack2][ixtrack2] <= t_dc);
00453             }
00454
00455     /* Detection of deep convection... */
00456     det_dc = (pert->dc[itrack][ixtrack] <= t_dc);
00457
00458     /* Detection of wave generation... */
00459     det_wg = 0;

```

```

00460         if (det_dc)
00461             for (itrack2 = GSL_MAX(itrack - iradius, 0);
00462                  itrack2 <= GSL_MIN(itrack + iradius, pert->ntrack - 1);
00463                  itrack2++)
00464                 for (ixtrack2 = GSL_MAX(ixtrack - iradius, 0);
00465                      ixtrack2 <= GSL_MIN(ixtrack + iradius, pert->nxtrack - 1);
00466                      ixtrack2++) {
00467                     if (det_wg)
00468                         break;
00469                     det_wg = (pert->var[itrack2][ixtrack2] >= t_gw);
00470                 }
00471
00472         /* Count events... */
00473         n[ix][iy]++;
00474         if (det_dc)
00475             ndc[ix][iy]++;
00476         if (det_wg)
00477             nwg[ix][iy]++;
00478         if (det_gw)
00479             ngw[ix][iy]++;
00480         if (det_cw)
00481             ncw[ix][iy]++;
00482
00483         /* Get statistics of perturbations... */
00484         mean[ix][iy] += pert->pt[itrack][ixtrack];
00485         var[ix][iy] += gsl_pow_2(pert->pt[itrack][ixtrack]);
00486         max[ix][iy] = GSL_MAX(max[ix][iy], pert->pt[itrack][ixtrack]);
00487         min[ix][iy] = GSL_MIN(min[ix][iy], pert->pt[itrack][ixtrack]);
00488
00489         /* Get statistics of brightness temperatures... */
00490         bt[ix][iy] += pert->bt[itrack][ixtrack];
00491         bt_8mu[ix][iy] += pert->dc[itrack][ixtrack];
00492         if (n[ix][iy] > 1) {
00493             bt_8mu_min[ix][iy]
00494                 = GSL_MIN(bt_8mu_min[ix][iy], pert->dc[itrack][ixtrack]);
00495             bt_8mu_max[ix][iy]
00496                 = GSL_MAX(bt_8mu_max[ix][iy], pert->dc[itrack][ixtrack]);
00497         } else {
00498             bt_8mu_min[ix][iy] = pert->dc[itrack][ixtrack];
00499             bt_8mu_max[ix][iy] = pert->dc[itrack][ixtrack];
00500         }
00501
00502         /* Get mean time... */
00503         mtime[ix][iy] += pert->time[itrack][ixtrack];
00504     }
00505 }
00506
00507 /* Analyze results... */
00508 for (ix = 0; ix < nx; ix++)
00509     for (iy = 0; iy < ny; iy++) {
00510
00511         /* Get geolocation... */
00512         mtime[ix][iy] /= (double) n[ix][iy];
00513         glon[ix]
00514             = lon0 + (ix + 0.5) / (double) nx * (
00515             lon1 - lon0);
00516         glat[iy]
00517             = lat0 + (iy + 0.5) / (double) ny * (
00518             lat1 - lat0);
00519
00520         /* Normalize brightness temperatures... */
00521         bt[ix][iy] /= (double) n[ix][iy];
00522         bt_8mu[ix][iy] /= (double) n[ix][iy];
00523
00524         /* Get fractions... */
00525         fdc[ix][iy] = (double) ndc[ix][iy] / (double) n[ix][iy] * 100.;
00526         fwg[ix][iy] = (double) nwg[ix][iy] / (double) ndc[ix][iy] * 100.;
00527         fgw[ix][iy] = (double) ngw[ix][iy] / (double) n[ix][iy] * 100.;
00528         fcw[ix][iy] = (double) ncw[ix][iy] / (double) ngw[ix][iy] * 100.;
00529
00530         /* Check number of observations... */
00531         if (n[ix][iy] < nmin) {
00532             fdc[ix][iy] = GSL_NAN;
00533             fwg[ix][iy] = GSL_NAN;
00534             fgw[ix][iy] = GSL_NAN;
00535             fcw[ix][iy] = GSL_NAN;
00536             bt_8mu[ix][iy] = GSL_NAN;
00537             bt_8mu_min[ix][iy] = GSL_NAN;
00538             bt_8mu_max[ix][iy] = GSL_NAN;
00539         }
00540
00541         /* Check detections of deep convection at high latitudes... */
00542         if (fabs(glat[iy]) > dc_hlat && bt_8mu[ix][iy] <= dc_tlim) {
00543             fdc[ix][iy] = GSL_NAN;
00544             fwg[ix][iy] = GSL_NAN;
00545             fcw[ix][iy] = GSL_NAN;
00546         }

```

```

00547
00548     /* Estimate noise... */
00549     if (dt230 > 0) {
00550         nesr = planck(230.0 + dt230, nu) - planck(230.0, nu);
00551         dt[ix][iy] =
00552             brightness(planck(bt[ix][iy], nu) + nesr, nu) - bt[ix][iy];
00553     }
00554
00555     /* Get mean perturbation and variance... */
00556     mean[ix][iy] /= (double) n[ix][iy];
00557     var[ix][iy] =
00558         var[ix][iy] / (double) n[ix][iy] - gsl_pow_2(mean[ix][iy]);
00559 }
00560
00561 /* Write ASCII file... */
00562 if (output == 1) {
00563
00564     /* Create file... */
00565     printf("Write variance statistics: %s\n", argv[2]);
00566     if (!(out = fopen(argv[2], "w")))
00567         ERRMSG("Cannot create file!");
00568
00569     /* Write header... */
00570     fprintf(out,
00571         "# $1 = time [s]\n"
00572         "# $2 = longitude [deg]\n"
00573         "# $3 = latitude [deg]\n"
00574         "# $4 = number of footprints\n"
00575         "# $5 = fraction of convection events [%%]\n"
00576         "# $6 = fraction of wave generating events [%%]\n"
00577         "# $7 = fraction of gravity wave events [%%]\n"
00578         "# $8 = fraction of convective wave events [%%]\n"
00579         "# $9 = mean perturbation [K]\n"
00580         "# $10 = minimum perturbation [K]\n");
00581     fprintf(out,
00582         "# $11 = maximum perturbation [K]\n"
00583         "# $12 = variance [K^2]\n"
00584         "# $13 = mean surface temperature [K]\n"
00585         "# $14 = minimum surface temperature [K]\n"
00586         "# $15 = maximum surface temperature [K]\n"
00587         "# $16 = mean background temperature [K]\n"
00588         "# $17 = noise estimate [K]\n");
00589
00590     /* Write results... */
00591     for (iy = 0; iy < ny; iy++) {
00592         if (iy == 0 || nx > 1)
00593             fprintf(out, "\n");
00594         for (ix = 0; ix < nx; ix++)
00595             fprintf(out, "%2f %g %g %d %g %g %g %g %g %g %g %g %g %g %g\n",
00596                 mtime[ix][iy], glon[ix], glat[iy], n[ix][iy],
00597                 fdc[ix][iy], fwg[ix][iy], fgw[ix][iy], fcw[ix][iy],
00598                 mean[ix][iy], min[ix][iy], max[ix][iy], var[ix][iy],
00599                 bt_8mu[ix][iy], bt_8mu_min[ix][iy], bt_8mu_max[ix][iy],
00600                 bt[ix][iy], dt[ix][iy]);
00601     }
00602
00603     /* Close file... */
00604     fclose(out);
00605 }
00606
00607 /* Write netCDF file... */
00608 else if (output == 2) {
00609
00610     /* Create netCDF file... */
00611     printf("Write variance statistics: %s\n", argv[2]);
00612     NC(nc_create(argv[2], NC_CLOBBER, &ncid));
00613
00614     /* Set dimensions... */
00615     NC(nc_def_dim(ncid, "lat", (size_t) ny, &dimid[0]));
00616     NC(nc_def_dim(ncid, "lon", (size_t) nx, &dimid[1]));
00617
00618     /* Add variables... */
00619     NC(nc_def_var(ncid, "lat", NC_DOUBLE, 1, &dimid[0], &latid));
00620     add_att(ncid, latid, "deg", "latitude");
00621     NC(nc_def_var(ncid, "lon", NC_DOUBLE, 1, &dimid[1], &lonid));
00622     add_att(ncid, lonid, "deg", "longitude");
00623     NC(nc_def_var(ncid, "var", NC_FLOAT, 2, dimid, &varid));
00624     add_att(ncid, varid, "K^2", "brightness temperature variance");
00625     NC(nc_def_var(ncid, "min", NC_FLOAT, 2, dimid, &minid));
00626     add_att(ncid, minid, "K", "brightness temperature minimum");
00627     NC(nc_def_var(ncid, "max", NC_FLOAT, 2, dimid, &maxid));
00628     add_att(ncid, maxid, "K", "brightness temperature maximum");
00629     NC(nc_def_var(ncid, "np", NC_INT, 2, dimid, &npid));
00630     add_att(ncid, npid, "1", "number of footprints");
00631
00632     /* Leave define mode... */
00633     NC(nc_enddef(ncid));

```

```

00634
00635     /* Write data... */
00636     NC(nc_put_var_double(ncid, latid, glat));
00637     NC(nc_put_var_double(ncid, lonid, glon));
00638     for (ix = 0; ix < nx; ix++)
00639         for (iy = 0; iy < ny; iy++)
00640             help[iy * nx + ix] = var[ix][iy] - POW2(dt[ix][iy]);
00641     NC(nc_put_var_double(ncid, varid, help));
00642     for (ix = 0; ix < nx; ix++)
00643         for (iy = 0; iy < ny; iy++)
00644             help[iy * nx + ix] = min[ix][iy];
00645     NC(nc_put_var_double(ncid, minid, help));
00646     for (ix = 0; ix < nx; ix++)
00647         for (iy = 0; iy < ny; iy++)
00648             help[iy * nx + ix] = max[ix][iy];
00649     NC(nc_put_var_double(ncid, maxid, help));
00650     for (ix = 0; ix < nx; ix++)
00651         for (iy = 0; iy < ny; iy++)
00652             help2[iy * nx + ix] = n[ix][iy];
00653     NC(nc_put_var_int(ncid, npid, help2));
00654
00655     /* Close file... */
00656     NC(nc_close(ncid));
00657 }
00658
00659 else
00660     ERRMSG("Unknown output format!");
00661
00662 /* Free... */
00663 free(pert);
00664
00665 return EXIT_SUCCESS;
00666 }

```

## 5.65 volcano.c File Reference

### Functions

- double [get\\_noise](#) (double bt, double dt250, double nu)
- int [main](#) (int argc, char \*argv[])

#### 5.65.1 Function Documentation

##### 5.65.1.1 double [get\\_noise](#) ( double *bt*, double *dt250*, double *nu* )

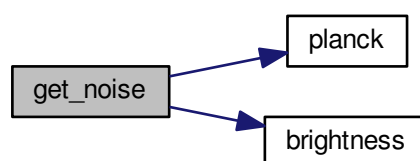
Definition at line [284](#) of file [volcano.c](#).

```

00287     {
00288
00289     double nesr;
00290
00291     nesr = planck(250.0 + dt250, nu) - planck(250.0, nu);
00292
00293     return brightness(planck(bt, nu) + nesr, nu) - bt;
00294 }

```

Here is the call graph for this function:





## 5.65.1.2 int main ( int argc, char \* argv[] )

Definition at line 17 of file [volcano.c](#).

```

00019         {
00020
00021     FILE *out;
00022
00023     static airs_rad_gran_t airs_rad_gran;
00024
00025     static double ci, ci_err, ci_nedt = 0.0783,
00026         ai_low, ai_low_err, ai_low_bt1, ai_low_bt1_nedt =
00027         0.3698, ai_low_bt2, ai_low_bt2_nedt =
00028         0.1177, ai_high, ai_high_err, ai_high_bt1, ai_high_bt1_nedt =
00029         0.0766, ai_high_bt2, ai_high_bt2_nedt =
00030         0.3706,
00031         ai_old, ai_old_err, ai_old_bt1, ai_old_bt1_nedt =
00032         0.3155, ai_old_bt2, ai_old_bt2_nedt =
00033         0.1177, si_high, si_high_err, si_high_bt1, si_high_bt1_nedt =
00034         0.1025, si_high_bt2, si_high_bt2_nedt =
00035         0.1373, si_low, si_low_err, si_low_bt1, si_low_bt1_nedt =
00036         0.0799, si_low_bt2, si_low_bt2_nedt =
00037         0.0909, si_old, si_old_err, si_old_bt1, si_old_bt1_nedt =
00038         0.1064, si_old_bt2, si_old_bt2_nedt =
00039         0.0909, si_oper, si_oper_err, si_oper_bt1, si_oper_bt1_nedt =
00040         0.0884, si_oper_bt2, si_oper_bt2_nedt = 0.1159;
00041
00042     static int ichan, track, xtrack, iarg, ai_low_nu1 = 641, ai_low_nu2 =
00043         901, ai_high_nu1 = 1295, ai_high_nu2 = 1162, ai_old_nu1 =
00044         559, ai_old_nu2 = 901, ci_nu = 1290, si_low_nu1 = 1601, si_low_nu2 =
00045         1526, si_high_nu1 = 1602, si_high_nu2 = 1551, si_old_nu1 =
00046         1591, si_old_nu2 = 1526, si_oper_nu1 = 1636, si_oper_nu2 = 1507;
00047
00048     /* Check arguments... */
00049     if (argc < 3)
00050         ERRMSG("Give parameters: <out.tab> <l1b_file1> [<l1b_file2> ...]");
00051
00052     /* Create file... */
00053     printf("Write volcanic emission data: %s\n", argv[1]);
00054     if (!(out = fopen(argv[1], "w")))
00055         ERRMSG("Cannot create file!");
00056
00057     /* Loop over HDF files... */
00058     for (iarg = 2; iarg < argc; iarg++) {
00059
00060         /* Read AIRS data... */
00061         printf("Read AIRS Level-1B data file: %s\n", argv[iarg]);
00062         airs_rad_rdr(argv[iarg], &airs_rad_gran);
00063
00064         /* Write header... */
00065         if (iarg == 2) {
00066             fprintf(out,
00067                 "# $1 = time [s]\n"
00068                 "# $2 = footprint longitude [deg]\n"
00069                 "# $3 = footprint latitude [deg]\n"
00070                 "# $4 = satellite altitude [km]\n"
00071                 "# $5 = satellite longitude [deg]\n"
00072                 "# $6 = satellite latitude [deg]\n");
00073             fprintf(out,
00074                 "# $7 = cloud index, BT(%.2f/cm) [K]\n"
00075                 "# $8 = cloud index error [K]\n"
00076                 "# $9 = ash index (low wavenumbers), "
00077                 "    BT(%.2f/cm) - BT(%.2f/cm) [K]\n"
00078                 "# $10 = ash index (low wavenumbers) error [K]\n"
00079                 "# $11 = ash index (high wavenumbers), "
00080                 "    BT(%.2f/cm) - BT(%.2f/cm) [K]\n"
00081                 "# $12 = ash index (high wavenumbers) error [K]\n"
00082                 "# $13 = ash index (Hoffmann et al., 2014), "
00083                 "    BT(%.2f/cm) - BT(%.2f/cm) [K]\n"
00084                 "# $14 = ash index (Hoffmann et al., 2014) error [K]\n",
00085                 airs_rad_gran.nominal_freq[ci_nu],
00086                 airs_rad_gran.nominal_freq[ai_low_nu1],
00087                 airs_rad_gran.nominal_freq[ai_low_nu2],
00088                 airs_rad_gran.nominal_freq[ai_high_nu1],
00089                 airs_rad_gran.nominal_freq[ai_high_nu2],
00090                 airs_rad_gran.nominal_freq[ai_old_nu1],
00091                 airs_rad_gran.nominal_freq[ai_old_nu2]);
00092             fprintf(out,
00093                 "# $15 = SO2 index (low concentrations), "
00094                 "    BT(%.2f/cm) - BT(%.2f/cm) [K]\n"
00095                 "# $16 = SO2 index (low concentrations) error [K]\n"
00096                 "# $17 = SO2 index (high concentrations), "
00097                 "    BT(%.2f/cm) - BT(%.2f/cm) [K]\n"
00098                 "# $18 = SO2 index (high concentrations) error [K]\n");

```

```

00099         "# $19 = SO2 index (operational),"
00100         " BT(%.2f/cm) - BT(%.2f/cm) [K]\n"
00101         "# $20 = SO2 index (operational) error [K]\n"
00102         "# $21 = SO2 index (Hoffmann et al., 2014),"
00103         " BT(%.2f/cm) - BT(%.2f/cm) [K]\n"
00104         "# $22 = SO2 index (Hoffmann et al., 2014) error [K]\n",
00105         airs_rad_gran.nominal_freq[si_low_nu1],
00106         airs_rad_gran.nominal_freq[si_low_nu2],
00107         airs_rad_gran.nominal_freq[si_high_nu1],
00108         airs_rad_gran.nominal_freq[si_high_nu2],
00109         airs_rad_gran.nominal_freq[si_oper_nu1],
00110         airs_rad_gran.nominal_freq[si_oper_nu2],
00111         airs_rad_gran.nominal_freq[si_old_nu1],
00112         airs_rad_gran.nominal_freq[si_old_nu2]);
00113     }
00114
00115     /* Flag bad observations... */
00116     for (track = 0; track < AIRS_RAD_GEOTRACK; track++)
00117         for (xtrack = 0; xtrack < AIRS_RAD_GEOXTRACK; xtrack++)
00118             for (ichan = 0; ichan < AIRS_RAD_CHANNEL; ichan++)
00119                 if ((airs_rad_gran.state[track][xtrack] != 0)
00120                     || (airs_rad_gran.ExcludedChans[ichan] > 2)
00121                     || (airs_rad_gran.CalChanSummary[ichan] & 8)
00122                     || (airs_rad_gran.CalChanSummary[ichan] & (32 + 64))
00123                     || (airs_rad_gran.CalFlag[track][ichan] & 16))
00124                     airs_rad_gran.radiances[track][xtrack][ichan] = GSL_NAN;
00125
00126     /* Loop over scans... */
00127     for (track = 0; track < AIRS_RAD_GEOTRACK; track++) {
00128
00129         /* Write output... */
00130         fprintf(out, "\n");
00131
00132         /* Loop over footprints... */
00133         for (xtrack = 0; xtrack < AIRS_RAD_GEOXTRACK; xtrack++) {
00134
00135             /* cloud index... */
00136             ci = brightness(airs_rad_gran.radiances[track][xtrack][ci_nu] * 0.001,
00137                            airs_rad_gran.nominal_freq[ci_nu]);
00138             ci_err = get_noise(ci, ci_nedt, airs_rad_gran.nominal_freq[ci_nu]);
00139
00140             /* ash index (low wavenumbers)... */
00141             ai_low_bt1 =
00142                 brightness(airs_rad_gran.radiances[track][xtrack][ai_low_nu1] *
00143                            0.001, airs_rad_gran.nominal_freq[ai_low_nu1]);
00144             ai_low_bt2 =
00145                 brightness(airs_rad_gran.radiances[track][xtrack][ai_low_nu2] *
00146                            0.001, airs_rad_gran.nominal_freq[ai_low_nu2]);
00147             ai_low = ai_low_bt1 - ai_low_bt2;
00148             ai_low_err = sqrt(gsl_pow_2(get_noise(ai_low_bt1, ai_low_bt1_nedt,
00149                                                  airs_rad_gran.nominal_freq
00150                                                  [ai_low_nu1]))
00151                               +
00152                               gsl_pow_2(get_noise
00153                                           (ai_low_bt2, ai_low_bt2_nedt,
00154                                            airs_rad_gran.nominal_freq
00155                                            [ai_low_nu2]))));
00156
00157             /* ash index (high wavenumbers)... */
00158             ai_high_bt1 =
00159                 brightness(airs_rad_gran.radiances[track][xtrack][ai_high_nu1] *
00160                            0.001, airs_rad_gran.nominal_freq[ai_high_nu1]);
00161             ai_high_bt2 =
00162                 brightness(airs_rad_gran.radiances[track][xtrack][ai_high_nu2] *
00163                            0.001, airs_rad_gran.nominal_freq[ai_high_nu2]);
00164             ai_high = ai_high_bt1 - ai_high_bt2;
00165             ai_high_err = sqrt(gsl_pow_2(get_noise(ai_high_bt1, ai_high_bt1_nedt,
00166                                                  airs_rad_gran.nominal_freq
00167                                                  [ai_high_nu1]))
00168                               +
00169                               gsl_pow_2(get_noise
00170                                           (ai_high_bt2, ai_high_bt2_nedt,
00171                                            airs_rad_gran.nominal_freq
00172                                            [ai_high_nu2]))));
00173
00174             /* ash index (old)... */
00175             ai_old_bt1 =
00176                 brightness(airs_rad_gran.radiances[track][xtrack][ai_old_nu1] *
00177                            0.001, airs_rad_gran.nominal_freq[ai_old_nu1]);
00178             ai_old_bt2 =
00179                 brightness(airs_rad_gran.radiances[track][xtrack][ai_old_nu2] *
00180                            0.001, airs_rad_gran.nominal_freq[ai_old_nu2]);
00181             ai_old = ai_old_bt1 - ai_old_bt2;
00182             ai_old_err = sqrt(gsl_pow_2(get_noise(ai_old_bt1, ai_old_bt1_nedt,
00183                                                  airs_rad_gran.nominal_freq
00184                                                  [ai_old_nu1]))
00185                               +

```

```

00186             gsl_pow_2(get_noise
00187                 (ai_old_bt2, ai_old_bt2_nedt,
00188                 airs_rad_gran.nominal_freq
00189                 [ai_old_nu2])));
00190
00191 /* SO2 index (low concentrations)... */
00192 si_low_bt1 =
00193     brightness(airs_rad_gran.radiances[track][xtrack][si_low_nu1] *
00194         0.001, airs_rad_gran.nominal_freq[si_low_nu1]);
00195 si_low_bt2 =
00196     brightness(airs_rad_gran.radiances[track][xtrack][si_low_nu2] *
00197         0.001, airs_rad_gran.nominal_freq[si_low_nu2]);
00198 si_low = si_low_bt1 - si_low_bt2;
00199 si_low_err = sqrt(gsl_pow_2(get_noise(si_low_bt1, si_low_bt1_nedt,
00200             airs_rad_gran.nominal_freq
00201             [si_low_nu1]))
00202             +
00203             gsl_pow_2(get_noise
00204                 (si_low_bt2, si_low_bt2_nedt,
00205                 airs_rad_gran.nominal_freq
00206                 [si_low_nu2])));
00207
00208 /* SO2 index (high concentrations)... */
00209 si_high_bt1 =
00210     brightness(airs_rad_gran.radiances[track][xtrack][si_high_nu1] *
00211         0.001, airs_rad_gran.nominal_freq[si_high_nu1]);
00212 si_high_bt2 =
00213     brightness(airs_rad_gran.radiances[track][xtrack][si_high_nu2] *
00214         0.001, airs_rad_gran.nominal_freq[si_high_nu2]);
00215 si_high = si_high_bt1 - si_high_bt2;
00216 si_high_err = sqrt(gsl_pow_2(get_noise(si_high_bt1, si_high_bt1_nedt,
00217             airs_rad_gran.nominal_freq
00218             [si_high_nu1]))
00219             +
00220             gsl_pow_2(get_noise
00221                 (si_high_bt2, si_high_bt2_nedt,
00222                 airs_rad_gran.nominal_freq
00223                 [si_high_nu2])));
00224
00225 /* SO2 index (operational)... */
00226 si_oper_bt1 =
00227     brightness(airs_rad_gran.radiances[track][xtrack][si_oper_nu1] *
00228         0.001, airs_rad_gran.nominal_freq[si_oper_nu1]);
00229 si_oper_bt2 =
00230     brightness(airs_rad_gran.radiances[track][xtrack][si_oper_nu2] *
00231         0.001, airs_rad_gran.nominal_freq[si_oper_nu2]);
00232 si_oper = si_oper_bt1 - si_oper_bt2;
00233 si_oper_err = sqrt(gsl_pow_2(get_noise(si_oper_bt1, si_oper_bt1_nedt,
00234             airs_rad_gran.nominal_freq
00235             [si_oper_nu1]))
00236             +
00237             gsl_pow_2(get_noise
00238                 (si_oper_bt2, si_oper_bt2_nedt,
00239                 airs_rad_gran.nominal_freq
00240                 [si_oper_nu2])));
00241
00242 /* SO2 index (old)... */
00243 si_old_bt1 =
00244     brightness(airs_rad_gran.radiances[track][xtrack][si_old_nu1] *
00245         0.001, airs_rad_gran.nominal_freq[si_old_nu1]);
00246 si_old_bt2 =
00247     brightness(airs_rad_gran.radiances[track][xtrack][si_old_nu2] *
00248         0.001, airs_rad_gran.nominal_freq[si_old_nu2]);
00249 si_old = si_old_bt1 - si_old_bt2;
00250 si_old_err = sqrt(gsl_pow_2(get_noise(si_old_bt1, si_old_bt1_nedt,
00251             airs_rad_gran.nominal_freq
00252             [si_old_nu1]))
00253             +
00254             gsl_pow_2(get_noise
00255                 (si_old_bt2, si_old_bt2_nedt,
00256                 airs_rad_gran.nominal_freq
00257                 [si_old_nu2])));
00258
00259 /* Write output... */
00260 fprintf(out,
00261     "%.2f %.4f %.4f %.3f %.4f %.4f %.2f %.2f %.2f %.2f %.2f "
00262     "%.2f %.2f %.2f %.2f %.2f %.2f %.2f %.2f %.2f\n",
00263     airs_rad_gran.Time[track][xtrack] - 220838400,
00264     airs_rad_gran.Longitude[track][xtrack],
00265     airs_rad_gran.Latitude[track][xtrack],
00266     airs_rad_gran.satheight[track],
00267     airs_rad_gran.sat_lon[track],
00268     airs_rad_gran.sat_lat[track],
00269     ci, ci_err, ai_low, ai_low_err, ai_high, ai_high_err, ai_old,
00270     ai_old_err, si_low, si_low_err, si_high, si_high_err, si_oper,
00271     si_oper_err, si_old, si_old_err);
00272 }

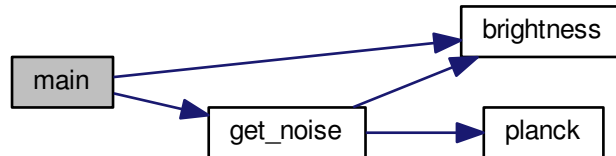
```

```

00273     }
00274 }
00275
00276 /* Close file... */
00277 fclose(out);
00278
00279 return EXIT_SUCCESS;
00280 }

```

Here is the call graph for this function:



## 5.66 volcano.c

```

00001 #include "libairs.h"
00002
00003 /* -----
00004    Functions...
00005    ----- */
00006
00007 /* Estimate noise. */
00008 double get_noise(
00009     double bt,
00010     double dt250,
00011     double nu);
00012
00013 /* -----
00014    Main...
00015    ----- */
00016
00017 int main(
00018     int argc,
00019     char *argv[]) {
00020
00021     FILE *out;
00022
00023     static airs_rad_gran_t airs_rad_gran;
00024
00025     static double ci, ci_err, ci_nedt = 0.0783,
00026         ai_low, ai_low_err, ai_low_bt1, ai_low_bt1_nedt =
00027         0.3698, ai_low_bt2, ai_low_bt2_nedt =
00028         0.1177, ai_high, ai_high_err, ai_high_bt1, ai_high_bt1_nedt =
00029         0.0766, ai_high_bt2, ai_high_bt2_nedt =
00030         0.3706,
00031         ai_old, ai_old_err, ai_old_bt1, ai_old_bt1_nedt =
00032         0.3155, ai_old_bt2, ai_old_bt2_nedt =
00033         0.1177, si_high, si_high_err, si_high_bt1, si_high_bt1_nedt =
00034         0.1025, si_high_bt2, si_high_bt2_nedt =
00035         0.1373, si_low, si_low_err, si_low_bt1, si_low_bt1_nedt =
00036         0.0799, si_low_bt2, si_low_bt2_nedt =
00037         0.0909, si_old, si_old_err, si_old_bt1, si_old_bt1_nedt =
00038         0.1064, si_old_bt2, si_old_bt2_nedt =
00039         0.0909, si_oper, si_oper_err, si_oper_bt1, si_oper_bt1_nedt =
00040         0.0884, si_oper_bt2, si_oper_bt2_nedt = 0.1159;
00041
00042     static int ichan, track, xtrack, iarg, ai_low_nu1 = 641, ai_low_nu2 =
00043         901, ai_high_nu1 = 1295, ai_high_nu2 = 1162, ai_old_nu1 =
00044         559, ai_old_nu2 = 901, ci_nu = 1290, si_low_nu1 = 1601, si_low_nu2 =
00045         1526, si_high_nu1 = 1602, si_high_nu2 = 1551, si_old_nu1 =
00046         1591, si_old_nu2 = 1526, si_oper_nu1 = 1636, si_oper_nu2 = 1507;
00047

```

```

00048  /* Check arguments... */
00049  if (argc < 3)
00050      ERRMSG("Give parameters: <out.tab> <l1b_file1> [<l1b_file2> ...]");
00051
00052  /* Create file... */
00053  printf("Write volcanic emission data: %s\n", argv[1]);
00054  if (!(out = fopen(argv[1], "w")))
00055      ERRMSG("Cannot create file!");
00056
00057  /* Loop over HDF files... */
00058  for (iarg = 2; iarg < argc; iarg++) {
00059
00060      /* Read AIRS data... */
00061      printf("Read AIRS Level-1B data file: %s\n", argv[iarg]);
00062      airs_rad_rdr(argv[iarg], &airs_rad_gran);
00063
00064      /* Write header... */
00065      if (iarg == 2) {
00066          fprintf(out,
00067              "# $1 = time [s]\n"
00068              "# $2 = footprint longitude [deg]\n"
00069              "# $3 = footprint latitude [deg]\n"
00070              "# $4 = satellite altitude [km]\n"
00071              "# $5 = satellite longitude [deg]\n"
00072              "# $6 = satellite latitude [deg]\n");
00073          fprintf(out,
00074              "# $7 = cloud index, BT(%.2f/cm) [K]\n"
00075              "# $8 = cloud index error [K]\n"
00076              "# $9 = ash index (low wavenumbers), "
00077              "BT(%.2f/cm) - BT(%.2f/cm) [K]\n"
00078              "# $10 = ash index (low wavenumbers) error [K]\n"
00079              "# $11 = ash index (high wavenumbers), "
00080              "BT(%.2f/cm) - BT(%.2f/cm) [K]\n"
00081              "# $12 = ash index (high wavenumbers) error [K]\n"
00082              "# $13 = ash index (Hoffmann et al., 2014), "
00083              "BT(%.2f/cm) - BT(%.2f/cm) [K]\n"
00084              "# $14 = ash index (Hoffmann et al., 2014) error [K]\n",
00085              airs_rad_gran.nominal_freq[ci_nu],
00086              airs_rad_gran.nominal_freq[ai_low_nu1],
00087              airs_rad_gran.nominal_freq[ai_low_nu2],
00088              airs_rad_gran.nominal_freq[ai_high_nu1],
00089              airs_rad_gran.nominal_freq[ai_high_nu2],
00090              airs_rad_gran.nominal_freq[ai_old_nu1],
00091              airs_rad_gran.nominal_freq[ai_old_nu2]);
00092          fprintf(out,
00093              "# $15 = SO2 index (low concentrations), "
00094              "BT(%.2f/cm) - BT(%.2f/cm) [K]\n"
00095              "# $16 = SO2 index (low concentrations) error [K]\n"
00096              "# $17 = SO2 index (high concentrations), "
00097              "BT(%.2f/cm) - BT(%.2f/cm) [K]\n"
00098              "# $18 = SO2 index (high concentrations) error [K]\n"
00099              "# $19 = SO2 index (operational), "
00100              "BT(%.2f/cm) - BT(%.2f/cm) [K]\n"
00101              "# $20 = SO2 index (operational) error [K]\n"
00102              "# $21 = SO2 index (Hoffmann et al., 2014), "
00103              "BT(%.2f/cm) - BT(%.2f/cm) [K]\n"
00104              "# $22 = SO2 index (Hoffmann et al., 2014) error [K]\n",
00105              airs_rad_gran.nominal_freq[si_low_nu1],
00106              airs_rad_gran.nominal_freq[si_low_nu2],
00107              airs_rad_gran.nominal_freq[si_high_nu1],
00108              airs_rad_gran.nominal_freq[si_high_nu2],
00109              airs_rad_gran.nominal_freq[si_oper_nu1],
00110              airs_rad_gran.nominal_freq[si_oper_nu2],
00111              airs_rad_gran.nominal_freq[si_old_nu1],
00112              airs_rad_gran.nominal_freq[si_old_nu2]);
00113      }
00114
00115      /* Flag bad observations... */
00116      for (track = 0; track < AIRS_RAD_GEOTRACK; track++)
00117          for (xtrack = 0; xtrack < AIRS_RAD_GEOXTRACK; xtrack++)
00118              for (ichan = 0; ichan < AIRS_RAD_CHANNEL; ichan++)
00119                  if ((airs_rad_gran.state[track][xtrack] != 0)
00120                      || (airs_rad_gran.ExcludedChans[ichan] > 2)
00121                      || (airs_rad_gran.CalChanSummary[ichan] & 8)
00122                      || (airs_rad_gran.CalChanSummary[ichan] & (32 + 64))
00123                      || (airs_rad_gran.CalFlag[track][ichan] & 16))
00124                      airs_rad_gran.radiances[track][xtrack][ichan] = GSL_NAN;
00125
00126      /* Loop over scans... */
00127      for (track = 0; track < AIRS_RAD_GEOTRACK; track++) {
00128
00129          /* Write output... */
00130          fprintf(out, "\n");
00131
00132          /* Loop over footprints... */
00133          for (xtrack = 0; xtrack < AIRS_RAD_GEOXTRACK; xtrack++) {
00134

```

```

00135      /* cloud index... */
00136      ci = brightness(airs_rad_gran.radiances[track][xtrack][ci_nu] * 0.001,
00137                     airs_rad_gran.nominal_freq[ci_nu]);
00138      ci_err = get_noise(ci, ci_nedt, airs_rad_gran.nominal_freq[ci_nu]);
00139
00140      /* ash index (low wavenumbers)... */
00141      ai_low_bt1 =
00142          brightness(airs_rad_gran.radiances[track][xtrack][ai_low_nu1] *
00143                   0.001, airs_rad_gran.nominal_freq[ai_low_nu1]);
00144      ai_low_bt2 =
00145          brightness(airs_rad_gran.radiances[track][xtrack][ai_low_nu2] *
00146                   0.001, airs_rad_gran.nominal_freq[ai_low_nu2]);
00147      ai_low = ai_low_bt1 - ai_low_bt2;
00148      ai_low_err = sqrt(gsl_pow_2(get_noise(ai_low_bt1, ai_low_bt1_nedt,
00149                                         airs_rad_gran.nominal_freq
00150                                         [ai_low_nu1]))
00151                      +
00152                      gsl_pow_2(get_noise
00153                                (ai_low_bt2, ai_low_bt2_nedt,
00154                                airs_rad_gran.nominal_freq
00155                                [ai_low_nu2])));
00156
00157      /* ash index (high wavenumbers)... */
00158      ai_high_bt1 =
00159          brightness(airs_rad_gran.radiances[track][xtrack][ai_high_nu1] *
00160                   0.001, airs_rad_gran.nominal_freq[ai_high_nu1]);
00161      ai_high_bt2 =
00162          brightness(airs_rad_gran.radiances[track][xtrack][ai_high_nu2] *
00163                   0.001, airs_rad_gran.nominal_freq[ai_high_nu2]);
00164      ai_high = ai_high_bt1 - ai_high_bt2;
00165      ai_high_err = sqrt(gsl_pow_2(get_noise(ai_high_bt1, ai_high_bt1_nedt,
00166                                         airs_rad_gran.nominal_freq
00167                                         [ai_high_nu1]))
00168                      +
00169                      gsl_pow_2(get_noise
00170                                (ai_high_bt2, ai_high_bt2_nedt,
00171                                airs_rad_gran.nominal_freq
00172                                [ai_high_nu2])));
00173
00174      /* ash index (old)... */
00175      ai_old_bt1 =
00176          brightness(airs_rad_gran.radiances[track][xtrack][ai_old_nu1] *
00177                   0.001, airs_rad_gran.nominal_freq[ai_old_nu1]);
00178      ai_old_bt2 =
00179          brightness(airs_rad_gran.radiances[track][xtrack][ai_old_nu2] *
00180                   0.001, airs_rad_gran.nominal_freq[ai_old_nu2]);
00181      ai_old = ai_old_bt1 - ai_old_bt2;
00182      ai_old_err = sqrt(gsl_pow_2(get_noise(ai_old_bt1, ai_old_bt1_nedt,
00183                                         airs_rad_gran.nominal_freq
00184                                         [ai_old_nu1]))
00185                      +
00186                      gsl_pow_2(get_noise
00187                                (ai_old_bt2, ai_old_bt2_nedt,
00188                                airs_rad_gran.nominal_freq
00189                                [ai_old_nu2])));
00190
00191      /* SO2 index (low concentrations)... */
00192      si_low_bt1 =
00193          brightness(airs_rad_gran.radiances[track][xtrack][si_low_nu1] *
00194                   0.001, airs_rad_gran.nominal_freq[si_low_nu1]);
00195      si_low_bt2 =
00196          brightness(airs_rad_gran.radiances[track][xtrack][si_low_nu2] *
00197                   0.001, airs_rad_gran.nominal_freq[si_low_nu2]);
00198      si_low = si_low_bt1 - si_low_bt2;
00199      si_low_err = sqrt(gsl_pow_2(get_noise(si_low_bt1, si_low_bt1_nedt,
00200                                         airs_rad_gran.nominal_freq
00201                                         [si_low_nu1]))
00202                      +
00203                      gsl_pow_2(get_noise
00204                                (si_low_bt2, si_low_bt2_nedt,
00205                                airs_rad_gran.nominal_freq
00206                                [si_low_nu2])));
00207
00208      /* SO2 index (high concentrations)... */
00209      si_high_bt1 =
00210          brightness(airs_rad_gran.radiances[track][xtrack][si_high_nu1] *
00211                   0.001, airs_rad_gran.nominal_freq[si_high_nu1]);
00212      si_high_bt2 =
00213          brightness(airs_rad_gran.radiances[track][xtrack][si_high_nu2] *
00214                   0.001, airs_rad_gran.nominal_freq[si_high_nu2]);
00215      si_high = si_high_bt1 - si_high_bt2;
00216      si_high_err = sqrt(gsl_pow_2(get_noise(si_high_bt1, si_high_bt1_nedt,
00217                                         airs_rad_gran.nominal_freq
00218                                         [si_high_nu1]))
00219                      +
00220                      gsl_pow_2(get_noise
00221                                (si_high_bt2, si_high_bt2_nedt,

```

```

00222             airs_rad_gran.nominal_freq
00223             [si_high_nu2])));
00224
00225     /* SO2 index (operational)... */
00226     si_oper_bt1 =
00227         brightness(airs_rad_gran.radiances[track][xtrack][si_oper_nu1] *
00228             0.001, airs_rad_gran.nominal_freq[si_oper_nu1]);
00229     si_oper_bt2 =
00230         brightness(airs_rad_gran.radiances[track][xtrack][si_oper_nu2] *
00231             0.001, airs_rad_gran.nominal_freq[si_oper_nu2]);
00232     si_oper = si_oper_bt1 - si_oper_bt2;
00233     si_oper_err = sqrt(gsl_pow_2(get_noise(si_oper_bt1, si_oper_bt1_nedt,
00234             airs_rad_gran.nominal_freq
00235             [si_oper_nu1]))
00236         +
00237         gsl_pow_2(get_noise
00238             (si_oper_bt2, si_oper_bt2_nedt,
00239             airs_rad_gran.nominal_freq
00240             [si_oper_nu2])));
00241
00242     /* SO2 index (old)... */
00243     si_old_bt1 =
00244         brightness(airs_rad_gran.radiances[track][xtrack][si_old_nu1] *
00245             0.001, airs_rad_gran.nominal_freq[si_old_nu1]);
00246     si_old_bt2 =
00247         brightness(airs_rad_gran.radiances[track][xtrack][si_old_nu2] *
00248             0.001, airs_rad_gran.nominal_freq[si_old_nu2]);
00249     si_old = si_old_bt1 - si_old_bt2;
00250     si_old_err = sqrt(gsl_pow_2(get_noise(si_old_bt1, si_old_bt1_nedt,
00251             airs_rad_gran.nominal_freq
00252             [si_old_nu1]))
00253         +
00254         gsl_pow_2(get_noise
00255             (si_old_bt2, si_old_bt2_nedt,
00256             airs_rad_gran.nominal_freq
00257             [si_old_nu2])));
00258
00259     /* Write output... */
00260     fprintf(out,
00261         "%2f %4f %4f %3f %4f %4f %2f %2f %2f %2f %2f %2f "
00262         "%2f %2f %2f %2f %2f %2f %2f %2f %2f %2f\n",
00263         airs_rad_gran.Time[track][xtrack] - 220838400,
00264         airs_rad_gran.Longitude[track][xtrack],
00265         airs_rad_gran.Latitude[track][xtrack],
00266         airs_rad_gran.satheight[track],
00267         airs_rad_gran.sat_lon[track],
00268         airs_rad_gran.sat_lat[track],
00269         ci, ci_err, ai_low, ai_low_err, ai_high, ai_high_err, ai_old,
00270         ai_old_err, si_low, si_low_err, si_high, si_high_err, si_oper,
00271         si_oper_err, si_old, si_old_err);
00272     }
00273 }
00274 }
00275
00276 /* Close file... */
00277 fclose(out);
00278
00279 return EXIT_SUCCESS;
00280 }
00281
00282 /*****
00283
00284 double get_noise(
00285     double bt,
00286     double dt250,
00287     double nu) {
00288
00289     double nesr;
00290
00291     nesr = planck(250.0 + dt250, nu) - planck(250.0, nu);
00292
00293     return brightness(planck(bt, nu) + nesr, nu) - bt;
00294 }

```

## 5.67 zm\_ret.c File Reference

### Functions

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

## 5.67.1 Function Documentation

## 5.67.1.1 int main ( int argc, char \* argv[] )

Definition at line 14 of file [zm\\_ret.c](#).

```

00016         {
00017
00018     static ret_t ret;
00019     static wave_t wave;
00020
00021     static double apr_tm[NPG][NLAT], apr_var[NPG][NLAT], apr_noise[NPG][NLAT],
00022         ret_tm[NPG][NLAT], ret_var[NPG][NLAT], ret_noise[NPG][NLAT],
00023         ret_time[NPG][NLAT], mu, sig_apr, sig_ret, tbg[NDS], tabg[NDS];
00024
00025     static int bg_poly_x, bg_poly_y, bg_smooth_x, bg_smooth_y,
00026         i, ids, ilat, ip, ix, iy, nlat, n[NPG][NLAT], ncid;
00027
00028     FILE *out;
00029
00030     /* Check arguments... */
00031     if (argc < 4)
00032         ERRMSG("Give parameters: <ctl> <zm.tab> <airs1.nc> [<airs2.nc> ...]");
00033
00034     /* Get control parameters... */
00035     bg_poly_x = (int) scan_ctl(argc, argv, "BG_POLY_X", -1, "5", NULL);
00036     bg_poly_y = (int) scan_ctl(argc, argv, "BG_POLY_Y", -1, "0", NULL);
00037     bg_smooth_x = (int) scan_ctl(argc, argv, "BG_SMOOTH_X", -1, "0", NULL);
00038     bg_smooth_y = (int) scan_ctl(argc, argv, "BG_SMOOTH_Y", -1, "0", NULL);
00039     nlat = (int) scan_ctl(argc, argv, "NLAT", -1, "36", NULL);
00040     if (nlat > NLAT)
00041         ERRMSG("Too many latitudes!");
00042
00043     /* Loop over files... */
00044     for (i = 3; i < argc; i++) {
00045
00046         /* Read AIRS data... */
00047         if (nc_open(argv[i], NC_WRITE, &ncid) != NC_NOERR)
00048             continue;
00049         else
00050             nc_close(ncid);
00051         read_retr(argv[i], &ret);
00052
00053         /* Loop over altitudes... */
00054         for (ip = 0; ip < ret.np; ip++) {
00055
00056             /* Compute background... */
00057             ret2wave(&ret, &wave, 1, ip);
00058             background_poly(&wave, bg_poly_x, bg_poly_y);
00059             background_smooth(&wave, bg_smooth_x, bg_smooth_y);
00060             for (ix = 0; ix < wave.nx; ix++)
00061                 for (iy = 0; iy < wave.ny; iy++)
00062                     tbg[iy * 90 + ix] = wave.bg[ix][iy];
00063             noise(&wave, &mu, &sig_ret);
00064             ret2wave(&ret, &wave, 2, ip);
00065             background_poly(&wave, bg_poly_x, bg_poly_y);
00066             background_smooth(&wave, bg_smooth_x, bg_smooth_y);
00067             for (ix = 0; ix < wave.nx; ix++)
00068                 for (iy = 0; iy < wave.ny; iy++)
00069                     tabg[iy * 90 + ix] = wave.bg[ix][iy];
00070             noise(&wave, &mu, &sig_apr);
00071
00072             /* Loop over data sets... */
00073             for (ids = 0; ids < ret.nds; ids++) {
00074
00075                 /* Check data... */
00076                 if (ret.lon[ids][ip] < -180 || ret.lon[ids][ip] > 180
00077                     || ret.lat[ids][ip] < -90 || ret.lat[ids][ip] > 90
00078                     || ret.t[ids][ip] < 110 || ret.t[ids][ip] > 390
00079                     || !gsl_finite(ret.t[ids][ip]))
00080                     continue;
00081
00082                 /* Get latitude index... */
00083                 ilat = (int) ((ret.lat[ids][ip] + 90.) / 180. * (double) nlat);
00084                 if (ilat < 0 || ilat >= nlat)
00085                     continue;
00086
00087                 /* Get zonal mean... */
00088                 if (gsl_finite(ret.t[ids][ip]) && gsl_finite(tbg[ids])) {
00089                     ret_time[ip][ilat] += ret.time[ids][ip];
00090                     ret_tm[ip][ilat] += ret.t[ids][ip];
00091                     ret_var[ip][ilat] += gsl_pow_2(ret.t[ids][ip] - tbg[ids]);

```

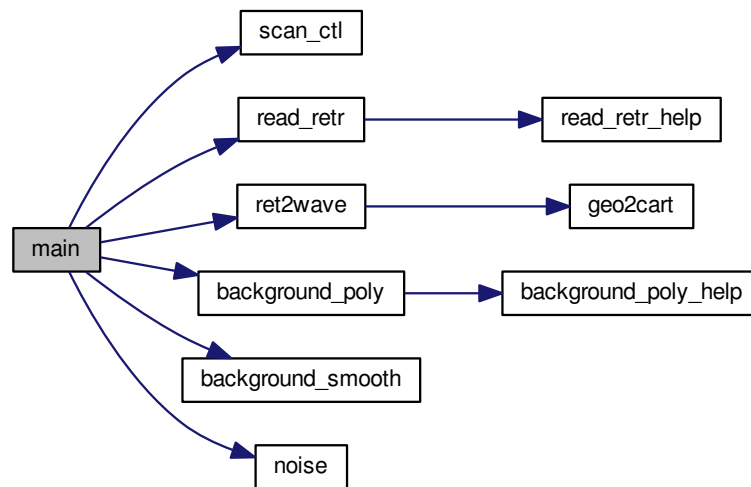


```

00092         ret_noise[ip][ilat] += gsl_pow_2(sig_ret);
00093         apr_tm[ip][ilat] += ret.t_apr[ids][ip];
00094         apr_var[ip][ilat] += gsl_pow_2(ret.t_apr[ids][ip] - tabg[ids]);
00095         apr_noise[ip][ilat] += gsl_pow_2(sig_apr);
00096         n[ip][ilat]++;
00097     }
00098 }
00099 }
00100 }
00101
00102 /* Create output file... */
00103 printf("Write AIRS zonal mean data: %s\n", argv[2]);
00104 if (! (out = fopen(argv[2], "w")))
00105     ERRMSG("Cannot create file!");
00106
00107 /* Write header... */
00108 fprintf(out,
00109     "# $1 = time (seconds since 01-JAN-2000, 00:00 UTC)\n"
00110     "# $2 = altitude [km]\n"
00111     "# $3 = latitude [deg]\n"
00112     "# $4 = mean temperature (retrieved) [K]\n"
00113     "# $5 = temperature variance (retrieved) [K^2]\n"
00114     "# $6 = noise estimate (retrieved) [K^2]\n"
00115     "# $7 = mean temperature (a priori) [K]\n"
00116     "# $8 = temperature variance (a priori) [K^2]\n"
00117     "# $9 = noise estimate (a priori) [K^2]\n"
00118     "# $10 = number of data points\n");
00119
00120 /* Loop over latitudes... */
00121 for (ilat = 0; ilat < nlat; ilat++) {
00122
00123     /* Write empty line... */
00124     fprintf(out, "\n");
00125
00126     /* Loop over altitudes... */
00127     for (ip = 0; ip < ret.np; ip++) {
00128
00129         /* Write data... */
00130         fprintf(out, "%.2f %g %g %g %g %g %g %g %d\n",
00131             ret_time[ip][ilat] / n[ip][ilat],
00132             ret.z[0][ip], (ilat + 0.5) / nlat * 180. - 90.,
00133             ret_tm[ip][ilat] / n[ip][ilat],
00134             sqrt(ret_var[ip][ilat] / n[ip][ilat]),
00135             sqrt(ret_noise[ip][ilat] / n[ip][ilat]),
00136             apr_tm[ip][ilat] / n[ip][ilat],
00137             sqrt(apr_var[ip][ilat] / n[ip][ilat]),
00138             sqrt(apr_noise[ip][ilat] / n[ip][ilat]), n[ip][ilat]);
00139     }
00140 }
00141
00142 /* Close file... */
00143 fclose(out);
00144
00145 return EXIT_SUCCESS;
00146 }

```

Here is the call graph for this function:



## 5.68 zm\_ret.c

```

00001 #include "libairs.h"
00002
00003 /* -----
00004     Dimensions...
00005     ----- */
00006
00007 /* Maximum number of latitudes. */
00008 #define NLAT 180
00009
00010 /* -----
00011     Main...
00012     ----- */
00013
00014 int main(
00015     int argc,
00016     char *argv[]) {
00017     static ret_t ret;
00018     static wave_t wave;
00019
00020     static double apr_tm[NPG][NLAT], apr_var[NPG][NLAT], apr_noise[NPG][NLAT],
00021         ret_tm[NPG][NLAT], ret_var[NPG][NLAT], ret_noise[NPG][NLAT],
00022         ret_time[NPG][NLAT], mu, sig_apr, sig_ret, tbg[NDS], tabg[NDS];
00023
00024     static int bg_poly_x, bg_poly_y, bg_smooth_x, bg_smooth_y,
00025         i, ids, ilat, ip, ix, iy, nlat, n[NPG][NLAT], ncid;
00026
00027     FILE *out;
00028
00029     /* Check arguments... */
00030     if (argc < 4)
00031         ERRMSG("Give parameters: <ctl> <zm.tab> <airs1.nc> [<airs2.nc> ...]");
00032
00033     /* Get control parameters... */
00034     bg_poly_x = (int) scan_ctl(argc, argv, "BG_POLY_X", -1, "5", NULL);
00035     bg_poly_y = (int) scan_ctl(argc, argv, "BG_POLY_Y", -1, "0", NULL);
00036     bg_smooth_x = (int) scan_ctl(argc, argv, "BG_SMOOTH_X", -1, "0", NULL);
00037     bg_smooth_y = (int) scan_ctl(argc, argv, "BG_SMOOTH_Y", -1, "0", NULL);
00038     nlat = (int) scan_ctl(argc, argv, "NLAT", -1, "36", NULL);
00039     if (nlat > NLAT)
00040         ERRMSG("Too many latitudes!");
00041
00042     /* Loop over files... */
00043     for (i = 3; i < argc; i++) {

```

```

00045
00046 /* Read AIRS data... */
00047 if (nc_open(argv[i], NC_WRITE, &ncid) != NC_NOERR)
00048     continue;
00049 else
00050     nc_close(ncid);
00051 read_retr(argv[i], &ret);
00052
00053 /* Loop over altitudes... */
00054 for (ip = 0; ip < ret.np; ip++) {
00055
00056     /* Compute background... */
00057     ret2wave(&ret, &wave, 1, ip);
00058     background_poly(&wave, bg_poly_x, bg_poly_y);
00059     background_smooth(&wave, bg_smooth_x, bg_smooth_y);
00060     for (ix = 0; ix < wave.nx; ix++)
00061         for (iy = 0; iy < wave.ny; iy++)
00062             tbg[iy * 90 + ix] = wave.bg[ix][iy];
00063     noise(&wave, &mu, &sig_ret);
00064     ret2wave(&ret, &wave, 2, ip);
00065     background_poly(&wave, bg_poly_x, bg_poly_y);
00066     background_smooth(&wave, bg_smooth_x, bg_smooth_y);
00067     for (ix = 0; ix < wave.nx; ix++)
00068         for (iy = 0; iy < wave.ny; iy++)
00069             tabg[iy * 90 + ix] = wave.bg[ix][iy];
00070     noise(&wave, &mu, &sig_apr);
00071
00072 /* Loop over data sets... */
00073 for (ids = 0; ids < ret.nds; ids++) {
00074
00075     /* Check data... */
00076     if (ret.lon[ids][ip] < -180 || ret.lon[ids][ip] > 180
00077         || ret.lat[ids][ip] < -90 || ret.lat[ids][ip] > 90
00078         || ret.t[ids][ip] < 110 || ret.t[ids][ip] > 390
00079         || !gsl_finite(ret.t[ids][ip]))
00080         continue;
00081
00082     /* Get latitude index... */
00083     ilat = (int) ((ret.lat[ids][ip] + 90.) / 180. * (double) nlat);
00084     if (ilat < 0 || ilat >= nlat)
00085         continue;
00086
00087     /* Get zonal mean... */
00088     if (gsl_finite(ret.t[ids][ip]) && gsl_finite(tbg[ids])) {
00089         ret_time[ip][ilat] += ret.time[ids][ip];
00090         ret_tm[ip][ilat] += ret.t[ids][ip];
00091         ret_var[ip][ilat] += gsl_pow_2(ret.t[ids][ip] - tbg[ids]);
00092         ret_noise[ip][ilat] += gsl_pow_2(sig_ret);
00093         apr_tm[ip][ilat] += ret.t_apr[ids][ip];
00094         apr_var[ip][ilat] += gsl_pow_2(ret.t_apr[ids][ip] - tabg[ids]);
00095         apr_noise[ip][ilat] += gsl_pow_2(sig_apr);
00096         n[ip][ilat]++;
00097     }
00098 }
00099 }
00100 }
00101
00102 /* Create output file... */
00103 printf("Write AIRS zonal mean data: %s\n", argv[2]);
00104 if (!(out = fopen(argv[2], "w")))
00105     ERRMSG("Cannot create file!");
00106
00107 /* Write header... */
00108 fprintf(out,
00109     "# $1 = time (seconds since 01-JAN-2000, 00:00 UTC)\n"
00110     "# $2 = altitude [km]\n"
00111     "# $3 = latitude [deg]\n"
00112     "# $4 = mean temperature (retrieved) [K]\n"
00113     "# $5 = temperature variance (retrieved) [K^2]\n"
00114     "# $6 = noise estimate (retrieved) [K^2]\n"
00115     "# $7 = mean temperature (a priori) [K]\n"
00116     "# $8 = temperature variance (a priori) [K^2]\n"
00117     "# $9 = noise estimate (a priori) [K^2]\n"
00118     "# $10 = number of data points\n");
00119
00120 /* Loop over latitudes... */
00121 for (ilat = 0; ilat < nlat; ilat++) {
00122
00123     /* Write empty line... */
00124     fprintf(out, "\n");
00125
00126     /* Loop over altitudes... */
00127     for (ip = 0; ip < ret.np; ip++) {
00128
00129         /* Write data... */
00130         fprintf(out, "%.2f %g %g %g %g %g %g %g %d\n",
00131             ret_time[ip][ilat] / n[ip][ilat],

```

```
00132         ret.z[0][ip], (ilat + 0.5) / nlat * 180. - 90.,
00133         ret_tm[ip][ilat] / n[ip][ilat],
00134         sqrt(ret_var[ip][ilat] / n[ip][ilat]),
00135         sqrt(ret_noise[ip][ilat] / n[ip][ilat]),
00136         apr_tm[ip][ilat] / n[ip][ilat],
00137         sqrt(apr_var[ip][ilat] / n[ip][ilat]),
00138         sqrt(apr_noise[ip][ilat] / n[ip][ilat]), n[ip][ilat]);
00139     }
00140 }
00141
00142 /* Close file... */
00143 fclose(out);
00144
00145 return EXIT_SUCCESS;
00146 }
```



## Index

add\_att  
    libairs.c, 295  
    libairs.h, 344  
add\_var  
    libairs.c, 295  
    libairs.h, 344  
    retrieval.c, 419  
airs\_chan  
    extract.c, 49  
airs\_l1\_t, 3  
    lat, 4  
    lon, 4  
    nu, 5  
    rad, 5  
    sat\_lat, 4  
    sat\_lon, 4  
    sat\_z, 4  
    time, 4  
airs\_l2\_t, 5  
    lat, 6  
    lon, 6  
    p, 6  
    t, 6  
    time, 6  
    z, 6  
atm2x  
    jurassic.c, 92  
    jurassic.h, 219  
atm2x\_help  
    jurassic.c, 92  
    jurassic.h, 219  
atm\_t, 6  
    k, 8  
    lat, 8  
    lon, 7  
    np, 7  
    p, 8  
    q, 8  
    t, 8  
    time, 7  
    z, 7  
  
background\_poly  
    libairs.c, 296  
    libairs.h, 344  
background\_poly\_help  
    libairs.c, 295  
    libairs.h, 345  
background\_smooth  
    libairs.c, 297  
    libairs.h, 346  
bg  
    wave\_t, 31  
brightness  
    jurassic.c, 93  
    jurassic.h, 220  
bt  
    pert\_t, 22  
buffer\_nc  
    retrieval.c, 419  
buoyancy  
    rayt.c, 410  
  
cart2geo  
    jurassic.c, 93  
    jurassic.h, 220  
chisq  
    ret\_t, 26  
climatology  
    jurassic.c, 93  
    jurassic.h, 220  
conv\_dmin  
    ret\_t, 26  
conv\_itmax  
    ret\_t, 26  
copy\_atm  
    jurassic.c, 127  
    jurassic.h, 254  
copy\_obs  
    jurassic.c, 127  
    jurassic.h, 254  
cost\_function  
    retrieval.c, 419  
create\_background  
    libairs.c, 298  
    libairs.h, 347  
create\_noise  
    libairs.c, 298  
    libairs.h, 347  
create\_wave  
    libairs.c, 299  
    libairs.h, 347  
ctl\_t, 8  
    ctm\_co2, 11  
    ctm\_h2o, 11  
    ctm\_n2, 11  
    ctm\_o2, 11  
    emitter, 10  
    fov, 11  
    hydz, 10  
    nd, 10  
    ng, 10  
    nu, 10  
    nw, 10  
    rayds, 11  
    raydz, 11  
    refrac, 11  
    retk\_zmax, 12  
    retk\_zmin, 12  
    retp\_zmax, 12

- retp\_zmin, [11](#)
  - retq\_zmax, [12](#)
  - retq\_zmin, [12](#)
  - rett\_zmax, [12](#)
  - rett\_zmin, [12](#)
  - tblbase, [10](#)
  - window, [10](#)
  - write\_bbt, [12](#)
  - write\_matrix, [12](#)
- ctm\_co2
  - ctl\_t, [11](#)
- ctm\_h2o
  - ctl\_t, [11](#)
- ctm\_n2
  - ctl\_t, [11](#)
- ctm\_o2
  - ctl\_t, [11](#)
- ctmco2
  - jurassic.c, [103](#)
  - jurassic.h, [230](#)
- ctmh2o
  - jurassic.c, [113](#)
  - jurassic.h, [240](#)
- ctmn2
  - jurassic.c, [125](#)
  - jurassic.h, [252](#)
- ctmo2
  - jurassic.c, [126](#)
  - jurassic.h, [253](#)
- day2doy
  - libairs.c, [299](#)
  - libairs.h, [348](#)
- day2doy.c, [32](#)
  - main, [32](#)
- dc
  - pert\_t, [22](#)
- diff\_apr.c, [33](#)
  - main, [34](#)
  - read\_nc, [33](#)
- diff\_ret.c, [38](#)
  - main, [38](#)
- distance.c, [40](#)
  - main, [40](#)
- doy2day
  - libairs.c, [299](#)
  - libairs.h, [348](#)
- doy2day.c, [41](#)
  - main, [42](#)
- ds
  - los\_t, [15](#)
- emitter
  - ctl\_t, [10](#)
- eps
  - tbl\_t, [29](#)
- err\_formod
  - ret\_t, [26](#)
- err\_k
  - ret\_t, [27](#)
- err\_k\_ch
  - ret\_t, [27](#)
- err\_k\_cz
  - ret\_t, [27](#)
- err\_noise
  - ret\_t, [26](#)
- err\_press
  - ret\_t, [26](#)
- err\_press\_ch
  - ret\_t, [26](#)
- err\_press\_cz
  - ret\_t, [26](#)
- err\_q
  - ret\_t, [27](#)
- err\_q\_ch
  - ret\_t, [27](#)
- err\_q\_cz
  - ret\_t, [27](#)
- err\_temp
  - ret\_t, [27](#)
- err\_temp\_ch
  - ret\_t, [27](#)
- err\_temp\_cz
  - ret\_t, [27](#)
- events.c, [42](#)
  - main, [43](#)
- extract.c, [47](#)
  - airs\_chan, [49](#)
  - gph2z, [47](#)
  - main, [47](#)
- fft
  - libairs.c, [300](#)
  - libairs.h, [349](#)
- fft\_help
  - libairs.c, [300](#)
  - libairs.h, [348](#)
- fill\_array
  - map\_pert.c, [374](#)
- fill\_gaps
  - retrieval.c, [420](#)
- find\_emitter
  - jurassic.c, [128](#)
  - jurassic.h, [255](#)
- formod
  - jurassic.c, [128](#)
  - jurassic.h, [255](#)
- formod\_continua
  - jurassic.c, [130](#)
  - jurassic.h, [257](#)
- formod\_fov
  - jurassic.c, [130](#)
  - jurassic.h, [257](#)
- formod\_pencil
  - jurassic.c, [132](#)
  - jurassic.h, [259](#)
- formod\_srcfunc
  - jurassic.c, [133](#)

- jurassic.h, 260
- fov
  - ctl\_t, 11
- gauss
  - libairs.c, 303
  - libairs.h, 351
- geo2cart
  - jurassic.c, 134
  - jurassic.h, 261
- get\_noise
  - volcano.c, 475
- get\_storm\_pos
  - hurricane.c, 52
- gph2z
  - extract.c, 47
- hamming
  - libairs.c, 303
  - libairs.h, 352
- hurricane.c, 51
  - get\_storm\_pos, 52
  - main, 53
  - read\_var, 52
- hydrostatic
  - jurassic.c, 134
  - jurassic.h, 261
- hydzt
  - ctl\_t, 10
- idx2name
  - jurassic.c, 135
  - jurassic.h, 262
- init\_l2
  - retrieval.c, 421
- init\_tbl
  - jurassic.c, 135
  - jurassic.h, 262
- intpol
  - issifm.c, 72
- intpol\_atm
  - jurassic.c, 137
  - jurassic.h, 264
- intpol\_tbl
  - jurassic.c, 138
  - jurassic.h, 265
- intpol\_tbl\_eps
  - jurassic.c, 140
  - jurassic.h, 267
- intpol\_tbl\_u
  - jurassic.c, 140
  - jurassic.h, 267
- intpol\_x
  - libairs.c, 304
  - libairs.h, 352
- island.c, 62
  - main, 62
- issifm.c, 71
  - intpol, 72
  - main, 75
  - smooth, 73
  - write\_nc, 74
- jsec2time
  - jurassic.c, 141
  - jurassic.h, 268
- jurassic.c, 90
  - atm2x, 92
  - atm2x\_help, 92
  - brightness, 93
  - cart2geo, 93
  - climatology, 93
  - copy\_atm, 127
  - copy\_obs, 127
  - ctmco2, 103
  - ctmh2o, 113
  - ctmn2, 125
  - ctmo2, 126
  - find\_emitter, 128
  - formod, 128
  - formod\_continua, 130
  - formod\_fov, 130
  - formod\_pencil, 132
  - formod\_srcfunc, 133
  - geo2cart, 134
  - hydrostatic, 134
  - idx2name, 135
  - init\_tbl, 135
  - intpol\_atm, 137
  - intpol\_tbl, 138
  - intpol\_tbl\_eps, 140
  - intpol\_tbl\_u, 140
  - jsec2time, 141
  - kernel, 141
  - locate\_irr, 143
  - locate\_reg, 143
  - locate\_tbl, 144
  - obs2y, 144
  - planck, 144
  - raytrace, 145
  - read\_atm, 147
  - read\_ctl, 148
  - read\_matrix, 149
  - read\_obs, 150
  - read\_shape, 151
  - refractivity, 151
  - scan\_ctl, 151
  - tangent\_point, 152
  - time2jsec, 153
  - timer, 153
  - write\_atm, 154
  - write\_matrix, 155
  - write\_obs, 157
  - x2atm, 158
  - x2atm\_help, 159
  - y2obs, 159
- jurassic.h, 216
  - atm2x, 219



- atm2x\_help, 219
  - brightness, 220
  - cart2geo, 220
  - climatology, 220
  - copy\_atm, 254
  - copy\_obs, 254
  - ctmco2, 230
  - ctmh2o, 240
  - ctmn2, 252
  - ctmo2, 253
  - find\_emitter, 255
  - formod, 255
  - formod\_continua, 257
  - formod\_fov, 257
  - formod\_pencil, 259
  - formod\_srcfunc, 260
  - geo2cart, 261
  - hydrostatic, 261
  - idx2name, 262
  - init\_tbl, 262
  - intpol\_atm, 264
  - intpol\_tbl, 265
  - intpol\_tbl\_eps, 267
  - intpol\_tbl\_u, 267
  - jsec2time, 268
  - kernel, 268
  - locate\_irr, 270
  - locate\_reg, 270
  - locate\_tbl, 271
  - obs2y, 271
  - planck, 271
  - raytrace, 272
  - read\_atm, 274
  - read\_ctl, 275
  - read\_matrix, 276
  - read\_obs, 277
  - read\_shape, 278
  - refractivity, 278
  - scan\_ctl, 278
  - tangent\_point, 279
  - time2jsec, 280
  - timer, 280
  - write\_atm, 281
  - write\_matrix, 282
  - write\_obs, 284
  - x2atm, 285
  - x2atm\_help, 286
  - y2obs, 286
- k
- atm\_t, 8
  - los\_t, 14
- kernel
- jurassic.c, 141
  - jurassic.h, 268
- kernel\_recomp
- ret\_t, 26
- l1\_lat
- ncd\_t, 16
- l1\_lon
- ncd\_t, 16
- l1\_nu
- ncd\_t, 17
- l1\_rad
- ncd\_t, 17
- l1\_sat\_lat
- ncd\_t, 17
- l1\_sat\_lon
- ncd\_t, 17
- l1\_sat\_z
- ncd\_t, 16
- l1\_time
- ncd\_t, 16
- l2\_p
- ncd\_t, 17
- l2\_t
- ncd\_t, 17
- l2\_z
- ncd\_t, 17
- lat
- airs\_l1\_t, 4
  - airs\_l2\_t, 6
  - atm\_t, 8
  - los\_t, 14
  - pert\_t, 22
  - ret\_t, 25
  - wave\_t, 31
- libairs.c, 293
- add\_att, 295
  - add\_var, 295
  - background\_poly, 296
  - background\_poly\_help, 295
  - background\_smooth, 297
  - create\_background, 298
  - create\_noise, 298
  - create\_wave, 299
  - day2doy, 299
  - doy2day, 299
  - fft, 300
  - fft\_help, 300
  - gauss, 303
  - hamming, 303
  - intpol\_x, 304
  - median, 305
  - merge\_y, 306
  - noise, 306
  - period, 307
  - pert2wave, 309
  - rad2wave, 315
  - read\_l1, 310
  - read\_l2, 310
  - read\_pert, 311
  - read\_retr, 312
  - read\_retr\_help, 314
  - read\_wave, 314
  - ret2wave, 316

- sza, [317](#)
  - variance, [318](#)
  - write\_l1, [318](#)
  - write\_l2, [319](#)
  - write\_wave, [320](#)
- libairs.h, [342](#)
  - add\_att, [344](#)
  - add\_var, [344](#)
  - background\_poly, [344](#)
  - background\_poly\_help, [345](#)
  - background\_smooth, [346](#)
  - create\_background, [347](#)
  - create\_noise, [347](#)
  - create\_wave, [347](#)
  - day2doy, [348](#)
  - doy2day, [348](#)
  - fft, [349](#)
  - fft\_help, [348](#)
  - gauss, [351](#)
  - hamming, [352](#)
  - intpol\_x, [352](#)
  - median, [354](#)
  - merge\_y, [354](#)
  - noise, [355](#)
  - period, [355](#)
  - pert2wave, [357](#)
  - rad2wave, [364](#)
  - read\_l1, [358](#)
  - read\_l2, [359](#)
  - read\_pert, [359](#)
  - read\_retr, [360](#)
  - read\_retr\_help, [363](#)
  - read\_wave, [363](#)
  - ret2wave, [365](#)
  - sza, [366](#)
  - variance, [366](#)
  - write\_l1, [367](#)
  - write\_l2, [368](#)
  - write\_wave, [369](#)
- locate\_irr
  - jurassic.c, [143](#)
  - jurassic.h, [270](#)
- locate\_reg
  - jurassic.c, [143](#)
  - jurassic.h, [270](#)
- locate\_tbl
  - jurassic.c, [144](#)
  - jurassic.h, [271](#)
- lon
  - airs\_l1\_t, [4](#)
  - airs\_l2\_t, [6](#)
  - atm\_t, [7](#)
  - los\_t, [14](#)
  - pert\_t, [21](#)
  - ret\_t, [24](#)
  - wave\_t, [31](#)
- los\_t, [13](#)
  - ds, [15](#)
- k, [14](#)
  - lat, [14](#)
  - lon, [14](#)
  - np, [14](#)
  - p, [14](#)
  - q, [14](#)
  - t, [14](#)
  - tsurf, [14](#)
  - u, [15](#)
  - z, [14](#)
- main
  - day2doy.c, [32](#)
  - diff\_apr.c, [34](#)
  - diff\_ret.c, [38](#)
  - distance.c, [40](#)
  - doy2day.c, [42](#)
  - events.c, [43](#)
  - extract.c, [47](#)
  - hurricane.c, [53](#)
  - island.c, [62](#)
  - issifm.c, [75](#)
  - map\_pert.c, [374](#)
  - map\_rad.c, [380](#)
  - map\_ret.c, [383](#)
  - noise\_pert.c, [386](#)
  - noise\_ret.c, [389](#)
  - optimize\_btd.c, [391](#)
  - orbit.c, [395](#)
  - overpass.c, [397](#)
  - perturbation.c, [401](#)
  - rayt.c, [411](#)
  - ret2tab.c, [416](#)
  - retrieval.c, [430](#)
  - sampling.c, [448](#)
  - spec2tab.c, [452](#)
  - spec\_ana.c, [455](#)
  - sza.c, [458](#)
  - var1d.c, [459](#)
  - variance.c, [461](#)
  - volcano.c, [475](#)
  - zm\_ret.c, [483](#)
- map\_pert.c, [374](#)
  - fill\_array, [374](#)
  - main, [374](#)
- map\_rad.c, [380](#)
  - main, [380](#)
- map\_ret.c, [383](#)
  - main, [383](#)
- matrix\_invert
  - retrieval.c, [422](#)
- matrix\_product
  - retrieval.c, [422](#)
- median
  - libairs.c, [305](#)
  - libairs.h, [354](#)
- merge\_y
  - libairs.c, [306](#)
  - libairs.h, [354](#)

- ncd\_t, 15
  - l1\_lat, 16
  - l1\_lon, 16
  - l1\_nu, 17
  - l1\_rad, 17
  - l1\_sat\_lat, 17
  - l1\_sat\_lon, 17
  - l1\_sat\_z, 16
  - l1\_time, 16
  - l2\_p, 17
  - l2\_t, 17
  - l2\_z, 17
  - ncid, 16
  - np, 16
  - ret\_p, 17
  - ret\_t, 18
  - ret\_z, 17
- ncid
  - ncd\_t, 16
- nd
  - ctl\_t, 10
- nds
  - ret\_t, 24
- ng
  - ctl\_t, 10
- noise
  - libairs.c, 306
  - libairs.h, 355
- noise\_pert.c, 386
  - main, 386
- noise\_ret.c, 389
  - main, 389
- np
  - atm\_t, 7
  - los\_t, 14
  - ncd\_t, 16
  - ret\_t, 24
  - tbl\_t, 28
- nr
  - obs\_t, 19
- nt
  - tbl\_t, 28
- ntrack
  - pert\_t, 21
- nu
  - airs\_l1\_t, 5
  - ctl\_t, 10
  - tbl\_t, 29
- nw
  - ctl\_t, 10
- nx
  - wave\_t, 30
- nxtrack
  - pert\_t, 21
- ny
  - wave\_t, 30
- obs2y
  - jurassic.c, 144
- jurassic.h, 271
- obs\_t, 18
  - nr, 19
  - obslat, 19
  - obslon, 19
  - obsz, 19
  - rad, 20
  - tau, 20
  - time, 19
  - tplat, 20
  - tplon, 20
  - tpz, 20
  - vplat, 19
  - vplon, 19
  - vpz, 19
- obslat
  - obs\_t, 19
- obslon
  - obs\_t, 19
- obsz
  - obs\_t, 19
- optimal\_estimation
  - retrieval.c, 422
- optimize\_btd.c, 391
  - main, 391
- orbit.c, 395
  - main, 395
- overpass.c, 396
  - main, 397
  - write\_results, 397
- p
  - airs\_l2\_t, 6
  - atm\_t, 8
  - los\_t, 14
  - ret\_t, 25
  - tbl\_t, 29
- period
  - libairs.c, 307
  - libairs.h, 355
- pert2wave
  - libairs.c, 309
  - libairs.h, 357
- pert\_t, 20
  - bt, 22
  - dc, 22
  - lat, 22
  - lon, 21
  - ntrack, 21
  - nxtrack, 21
  - pt, 22
  - time, 21
  - var, 22
- perturbation.c, 401
  - main, 401
- planck
  - jurassic.c, 144
  - jurassic.h, 271
- pt

- pert\_t, 22
- wave\_t, 31
- q
  - atm\_t, 8
  - los\_t, 14
- rad
  - airs\_l1\_t, 5
  - obs\_t, 20
- rad2wave
  - libairs.c, 315
  - libairs.h, 364
- rayds
  - ctl\_t, 11
- raydz
  - ctl\_t, 11
- rayt.c, 410
  - buoyancy, 410
  - main, 411
  - scale\_height, 410
  - temp2theta, 411
- raytrace
  - jurassic.c, 145
  - jurassic.h, 272
- read\_atm
  - jurassic.c, 147
  - jurassic.h, 274
- read\_ctl
  - jurassic.c, 148
  - jurassic.h, 275
- read\_l1
  - libairs.c, 310
  - libairs.h, 358
- read\_l2
  - libairs.c, 310
  - libairs.h, 359
- read\_matrix
  - jurassic.c, 149
  - jurassic.h, 276
- read\_nc
  - diff\_apr.c, 33
  - retrieval.c, 425
- read\_obs
  - jurassic.c, 150
  - jurassic.h, 277
- read\_pert
  - libairs.c, 311
  - libairs.h, 359
- read\_ret\_ctl
  - retrieval.c, 426
- read\_retr
  - libairs.c, 312
  - libairs.h, 360
- read\_retr\_help
  - libairs.c, 314
  - libairs.h, 363
- read\_shape
  - jurassic.c, 151
- jurassic.h, 278
- read\_var
  - hurricane.c, 52
- read\_wave
  - libairs.c, 314
  - libairs.h, 363
- refrac
  - ctl\_t, 11
- refractivity
  - jurassic.c, 151
  - jurassic.h, 278
- ret2tab.c, 416
  - main, 416
- ret2wave
  - libairs.c, 316
  - libairs.h, 365
- ret\_p
  - ncd\_t, 17
- ret\_t, 22
  - chisq, 26
  - conv\_dmin, 26
  - conv\_itmax, 26
  - err\_formod, 26
  - err\_k, 27
  - err\_k\_ch, 27
  - err\_k\_cz, 27
  - err\_noise, 26
  - err\_press, 26
  - err\_press\_ch, 26
  - err\_press\_cz, 26
  - err\_q, 27
  - err\_q\_ch, 27
  - err\_q\_cz, 27
  - err\_temp, 27
  - err\_temp\_ch, 27
  - err\_temp\_cz, 27
  - kernel\_recomp, 26
  - lat, 25
  - lon, 24
  - ncd\_t, 18
  - nds, 24
  - np, 24
  - p, 25
  - t, 25
  - t\_apr, 25
  - t\_cont, 25
  - t\_fm, 25
  - t\_noise, 25
  - t\_res, 25
  - t\_tot, 25
  - time, 24
  - z, 24
- ret\_z
  - ncd\_t, 17
- retk\_zmax
  - ctl\_t, 12
- retk\_zmin
  - ctl\_t, 12

- retp\_zmax
  - ctl\_t, 12
- retp\_zmin
  - ctl\_t, 11
- retq\_zmax
  - ctl\_t, 12
- retq\_zmin
  - ctl\_t, 12
- retrieval.c, 418
  - add\_var, 419
  - buffer\_nc, 419
  - cost\_function, 419
  - fill\_gaps, 420
  - init\_l2, 421
  - main, 430
  - matrix\_invert, 422
  - matrix\_product, 422
  - optimal\_estimation, 422
  - read\_nc, 425
  - read\_ret\_ctl, 426
  - set\_cov\_apr, 427
  - set\_cov\_meas, 428
  - sza, 429
  - write\_nc, 430
- rett\_zmax
  - ctl\_t, 12
- rett\_zmin
  - ctl\_t, 12
- sampling.c, 448
  - main, 448
- sat\_lat
  - airs\_l1\_t, 4
- sat\_lon
  - airs\_l1\_t, 4
- sat\_z
  - airs\_l1\_t, 4
- scale\_height
  - rayt.c, 410
- scan\_ctl
  - jurassic.c, 151
  - jurassic.h, 278
- set\_cov\_apr
  - retrieval.c, 427
- set\_cov\_meas
  - retrieval.c, 428
- smooth
  - issifm.c, 73
- spec2tab.c, 452
  - main, 452
- spec\_ana.c, 455
  - main, 455
- sr
  - tbl\_t, 29
- st
  - tbl\_t, 29
- sza
  - libairs.c, 317
  - libairs.h, 366
- retrieval.c, 429
- sza.c, 458
  - main, 458
- t
  - airs\_l2\_t, 6
  - atm\_t, 8
  - los\_t, 14
  - ret\_t, 25
  - tbl\_t, 29
- t\_apr
  - ret\_t, 25
- t\_cont
  - ret\_t, 25
- t\_fm
  - ret\_t, 25
- t\_noise
  - ret\_t, 25
- t\_res
  - ret\_t, 25
- t\_tot
  - ret\_t, 25
- tangent\_point
  - jurassic.c, 152
  - jurassic.h, 279
- tau
  - obs\_t, 20
- tbl\_t, 28
  - eps, 29
  - np, 28
  - nt, 28
  - nu, 29
  - p, 29
  - sr, 29
  - st, 29
  - t, 29
  - u, 29
- tblbase
  - ctl\_t, 10
- temp
  - wave\_t, 31
- temp2theta
  - rayt.c, 411
- time
  - airs\_l1\_t, 4
  - airs\_l2\_t, 6
  - atm\_t, 7
  - obs\_t, 19
  - pert\_t, 21
  - ret\_t, 24
  - wave\_t, 31
- time2jsec
  - jurassic.c, 153
  - jurassic.h, 280
- timer
  - jurassic.c, 153
  - jurassic.h, 280
- tplat
  - obs\_t, 20

tplon  
  obs\_t, 20  
tpz  
  obs\_t, 20  
tsurf  
  los\_t, 14  
u  
  los\_t, 15  
  tbl\_t, 29  
var  
  pert\_t, 22  
  wave\_t, 32  
var1d.c, 459  
  main, 459  
variance  
  libairs.c, 318  
  libairs.h, 366  
variance.c, 461  
  main, 461  
volcano.c, 475  
  get\_noise, 475  
  main, 475  
vplat  
  obs\_t, 19  
vplon  
  obs\_t, 19  
vpz  
  obs\_t, 19  
wave\_t, 30  
  bg, 31  
  lat, 31  
  lon, 31  
  nx, 30  
  ny, 30  
  pt, 31  
  temp, 31  
  time, 31  
  var, 32  
  x, 31  
  y, 31  
  z, 31  
window  
  ctl\_t, 10  
write\_atm  
  jurassic.c, 154  
  jurassic.h, 281  
write\_bbt  
  ctl\_t, 12  
write\_l1  
  libairs.c, 318  
  libairs.h, 367  
write\_l2  
  libairs.c, 319  
  libairs.h, 368  
write\_matrix  
  ctl\_t, 12  
  jurassic.c, 155  
  jurassic.h, 282  
write\_nc  
  issifm.c, 74  
  retrieval.c, 430  
write\_obs  
  jurassic.c, 157  
  jurassic.h, 284  
write\_results  
  overpass.c, 397  
write\_wave  
  libairs.c, 320  
  libairs.h, 369  
x  
  wave\_t, 31  
x2atm  
  jurassic.c, 158  
  jurassic.h, 285  
x2atm\_help  
  jurassic.c, 159  
  jurassic.h, 286  
y  
  wave\_t, 31  
y2obs  
  jurassic.c, 159  
  jurassic.h, 286  
z  
  airs\_l2\_t, 6  
  atm\_t, 7  
  los\_t, 14  
  ret\_t, 24  
  wave\_t, 31  
zm\_ret.c, 482  
  main, 483