IASI Code Collection

Generated on Sat Jul 12 2025 08:47:14 for IASI Code Collection by Doxygen 1.9.8

Sat Jul 12 2025 08:47:14

1 IASI Code Collection

The Juelich Rapid Spectral Simulation Code (JURASSIC) is a fast infrared radiative transfer model for the analysis of atmospheric remote sensing measurements.

1.1 Introduction

The source code of JURASSIC is available from the `git repository`. Please see the `README.md` in the git repository for introductory information. More information can be found in the `user manual`.

This doxygen manual contains information about the algorithms and data structures used in the code. Please refer to the 'jurassic.h' documentation for a first overview.

1.2 References

For citing the model in scientific publications, please see `CITATION.cff` and refer to the following papers:

Baumeister, P. F. and Hoffmann, L.: Fast infrared radiative transfer calculations using graphics processing units: JURASSIC-GPU v2.0, Geosci. Model Dev., 15, 1855–1874, `https://doi.org/10.5194/gmd-15-1855-2022`*, 2022.*

Hoffmann, L., and M. J. Alexander, Retrieval of stratospheric temperatures from Atmospheric Infrared Sounder radiance measurements for gravity wave studies, J. Geophys. Res., 114, D07105, `https://doi.org/10.1029/2008JD011241`*, 2009.*

Hoffmann, L., Kaufmann, M., Spang, R., Müller, R., Remedios, J. J., Moore, D. P., Volk, C. M., von Clarmann, T., and Riese, M.: Envisat MIPAS measurements of CFC-11: retrieval, validation, and climatology, Atmos. Chem. Phys., 8, 3671-3688, `https://doi.org/10.5194/acp-8-3671-2008`*, 2008.*

Additional references are collected here: `https://slcs-jsc.github.io/jurassic/references`

1.3 License

JURASSIC is being develop at the Jülich Supercomputing Centre, Forschungszentrum Jülich, Germany.

1.4 Contributing

We are interested in supporting operational and research applications with JURASSIC.

You can submit bug reports or feature requests on the `issue tracker`.

Proposed code changes and fixes can be submitted as `pull requests`.

Please do not hesitate to contact us if you have any questions or need assistance.

1.5 Contact

Dr. Lars Hoffmann

Jülich Supercomputing Centre, Forschungszentrum Jülich

e-mail: `l.hoffmann@fz-juelich.de`

2 Data Structure Index

2.1 Data Structures

Here are the data structures with brief descriptions:

3 File Index

3.1 File List

Here is a list of all files with brief descriptions:

4 Data Structure Documentation

4.1 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 [ppv].
- double k [NW][NP]

  Extinction [km^-1].
- double clz

  Cloud layer height [km].
- double cldz

  Cloud layer depth [km].
- double clk [NCL]

  Cloud layer extinction [km^-1].
- double sfz

  Surface height [km].
- double sfp

  Surface pressure [hPa].
- double sft

  Surface temperature [K].
- double sfeps [NSF]

  Surface emissivity.

4.1.1 Detailed Description

Atmospheric data.

Definition at line 488 of file jurassic.h.

4.1.2 Field Documentation

np

```
int atm_t::np
```

Number of data points.

Definition at line 491 of file jurassic.h.

time

```
double atm_t::time[NP]
```

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

Definition at line 494 of file jurassic.h.

z

```
double atm_t::z[NP]
```

Altitude [km].

Definition at line 497 of file jurassic.h.

lon

```
double atm_t::lon[NP]
```

Longitude [deg].

Definition at line 500 of file jurassic.h.

lat

```
double atm_t::lat[NP]
```

Latitude [deg].

Definition at line 503 of file jurassic.h.

p

```
double atm_t::p[NP]
```

Pressure [hPa].

Definition at line 506 of file jurassic.h.

t

```
double atm_t::t[NP]
```

Temperature [K].

Definition at line 509 of file jurassic.h.

q

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

Volume mixing ratio [ppv].

Definition at line 512 of file jurassic.h.

k

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

Extinction [km^-1].

Definition at line 515 of file jurassic.h.

clz

`double atm_t::clz`

Cloud layer height [km].

Definition at line 518 of file jurassic.h.

cldz

`double atm_t::cldz`

Cloud layer depth [km].

Definition at line 521 of file jurassic.h.

clk

`double atm_t::clk[NCL]`

Cloud layer extinction [km^-1].

Definition at line 524 of file jurassic.h.

sfz

`double atm_t::sfz`

Surface height [km].

Definition at line 527 of file jurassic.h.

sfp

```
double atm_t::sfp
```

Surface pressure [hPa].

Definition at line 530 of file jurassic.h.

sft

```
double atm_t::sft
```

Surface temperature [K].

Definition at line 533 of file jurassic.h.

sfeps

```
double atm_t::sfeps[NSF]
```

Surface emissivity.

Definition at line 536 of file jurassic.h.

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

- jurassic.h

4.2 ctl2_t Struct Reference

Control parameters.

Data Fields

- double dt_met
  Time step of meteorological data [s].
- char met_geopot [LEN]
  Surface geopotential data file.
- int met_dx
  Stride for longitudes.
- int met_dy
  Stride for latitudes.
- int met_dp
  Stride for pressure levels.
- int met_sx
  Smoothing for longitudes.
- int met_sy
  Smoothing for latitudes.
- int met_sp
  Smoothing for pressure levels.

4.2.1 Detailed Description

Control parameters.

Definition at line 32 of file extract.c.

4.2.2 Field Documentation

dt_met

```
double ctl2_t::dt_met
```

Time step of meteorological data [s].

Definition at line 35 of file extract.c.

met_geopot

```
char ctl2_t::met_geopot[LEN]
```

Surface geopotential data file.

Definition at line 38 of file extract.c.

met_dx

```
int ctl2_t::met_dx
```

Stride for longitudes.

Definition at line 41 of file extract.c.

met_dy

```
int ctl2_t::met_dy
```

Stride for latitudes.

Definition at line 44 of file extract.c.

met_dp

```
int ctl2_t::met_dp
```

Stride for pressure levels.

Definition at line 47 of file extract.c.

met_sx

```
int ctl2_t::met_sx
```

Smoothing for longitudes.

Definition at line 50 of file extract.c.

met_sy

```
int ctl2_t::met_sy
```

Smoothing for latitudes.

Definition at line 53 of file extract.c.

met_sp

```
int ctl2_t::met_sp
```

Smoothing for pressure levels.

Definition at line 56 of file extract.c.

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

- extract.c

4.3 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.
- double nu [ND]

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

  Number of spectral windows.
- int window [ND]

  Window index of each channel.
- int ncl

  Number of cloud layer spectral grid points.
- double clnu [NCL]

  Cloud layer wavenumber [cm^-1].
- int nsf

  Number of surface layer spectral grid points.
- double sfnu [NSF]

  Surface layer wavenumber [cm^-1].
- int sftype

  Surface treatment (0=none, 1=emissions, 2=downward, 3=solar).
- double sfsza

  Solar zenith angle at the surface [deg] (-999=auto).
- char tblbase [LEN]

  Basename for table files and filter function files.
- int tblfmt

  Look-up table file format (1=ASCII, 2=binary).
- 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 ret_clz

  Retrieve cloud layer height (0=no, 1=yes).
- int ret_cldz

  Retrieve cloud layer depth (0=no, 1=yes).
- int ret_clk

  Retrieve cloud layer extinction (0=no, 1=yes).
- int ret_sfz

  Retrieve surface layer height (0=no, 1=yes).
- int ret_sfp

  Retrieve surface layer pressure (0=no, 1=yes).
- int ret_sft

  Retrieve surface layer temperature (0=no, 1=yes).
- int ret_sfeps

  Retrieve surface layer emissivity (0=no, 1=yes).
- int write_bbt

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

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

  Forward model (0=CGA, 1=EGA, 2=RFM).
- char rfmbin [LEN]

  Path to RFM binary.
- char rfmhit [LEN]

  HITRAN file for RFM.
- char rfmxsc [NG][LEN]

  Emitter cross-section files for RFM.

4.3.1 Detailed Description

Forward model control parameters.

Definition at line 541 of file jurassic.h.

4.3.2 Field Documentation

ng

`int ctl_t::ng`

Number of emitters.

Definition at line 544 of file jurassic.h.

emitter

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

Name of each emitter.

Definition at line 547 of file jurassic.h.

nd

`int ctl_t::nd`

Number of radiance channels.

Definition at line 550 of file jurassic.h.

nu

`double ctl_t::nu[ND]`

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

Definition at line 553 of file jurassic.h.

nw

`int ctl_t::nw`

Number of spectral windows.

Definition at line 556 of file jurassic.h.

window

`int ctl_t::window[ND]`

Window index of each channel.

Definition at line 559 of file jurassic.h.

ncl

`int ctl_t::ncl`

Number of cloud layer spectral grid points.

Definition at line 562 of file jurassic.h.

clnu

`double ctl_t::clnu[NCL]`

Cloud layer wavenumber [cm^-1].

Definition at line 565 of file jurassic.h.

nsf

`int ctl_t::nsf`

Number of surface layer spectral grid points.

Definition at line 568 of file jurassic.h.

sfnu

`double ctl_t::sfnu[NSF]`

Surface layer wavenumber [cm^-1].

Definition at line 571 of file jurassic.h.

sftype

`int ctl_t::sftype`

Surface treatment (0=none, 1=emissions, 2=downward, 3=solar).

Definition at line 574 of file jurassic.h.

sfsza

`double ctl_t::sfsza`

Solar zenith angle at the surface [deg] (-999=auto).

Definition at line 577 of file jurassic.h.

tblbase

`char ctl_t::tblbase[LEN]`

Basename for table files and filter function files.

Definition at line 580 of file jurassic.h.

tblfmt

`int ctl_t::tblfmt`

Look-up table file format (1=ASCII, 2=binary).

Definition at line 583 of file jurassic.h.

hydz

`double ctl_t::hydz`

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

Definition at line 586 of file jurassic.h.

ctm_co2

`int ctl_t::ctm_co2`

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

Definition at line 589 of file jurassic.h.

ctm_h2o

`int ctl_t::ctm_h2o`

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

Definition at line 592 of file jurassic.h.

ctm_n2

`int ctl_t::ctm_n2`

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

Definition at line 595 of file jurassic.h.

ctm_o2

```
int ctl_t::ctm_o2
```

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

Definition at line 598 of file jurassic.h.

refrac

```
int ctl_t::refrac
```

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

Definition at line 601 of file jurassic.h.

rayds

```
double ctl_t::rayds
```

Maximum step length for raytracing [km].

Definition at line 604 of file jurassic.h.

raydz

```
double ctl_t::raydz
```

Vertical step length for raytracing [km].

Definition at line 607 of file jurassic.h.

fov

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

Field-of-view data file.

Definition at line 610 of file jurassic.h.

retp_zmin

```
double ctl_t::retp_zmin
```

Minimum altitude for pressure retrieval [km].

Definition at line 613 of file jurassic.h.

retp_zmax

```
double ctl_t::retp_zmax
```

Maximum altitude for pressure retrieval [km].

Definition at line 616 of file jurassic.h.

rett_zmin

```
double ctl_t::rett_zmin
```

Minimum altitude for temperature retrieval [km].

Definition at line 619 of file jurassic.h.

rett_zmax

```
double ctl_t::rett_zmax
```

Maximum altitude for temperature retrieval [km].

Definition at line 622 of file jurassic.h.

retq_zmin

```
double ctl_t::retq_zmin[NG]
```

Minimum altitude for volume mixing ratio retrieval [km].

Definition at line 625 of file jurassic.h.

retq_zmax

```
double ctl_t::retq_zmax[NG]
```

Maximum altitude for volume mixing ratio retrieval [km].

Definition at line 628 of file jurassic.h.

retk_zmin

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

Minimum altitude for extinction retrieval [km].

Definition at line 631 of file jurassic.h.

retk_zmax

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

Maximum altitude for extinction retrieval [km].

Definition at line 634 of file jurassic.h.

ret_clz

```
int ctl_t::ret_clz
```

Retrieve cloud layer height (0=no, 1=yes).

Definition at line 637 of file jurassic.h.

ret_cldz

```
int ctl_t::ret_cldz
```

Retrieve cloud layer depth (0=no, 1=yes).

Definition at line 640 of file jurassic.h.

ret_clk

```
int ctl_t::ret_clk
```

Retrieve cloud layer extinction (0=no, 1=yes).

Definition at line 643 of file jurassic.h.

ret_sfz

```
int ctl_t::ret_sfz
```

Retrieve surface layer height (0=no, 1=yes).

Definition at line 646 of file jurassic.h.

ret_sfp

```
int ctl_t::ret_sfp
```

Retrieve surface layer pressure (0=no, 1=yes).

Definition at line 649 of file jurassic.h.

ret_sft

`int ctl_t::ret_sft`

Retrieve surface layer temperature (0=no, 1=yes).

Definition at line 652 of file jurassic.h.

ret_sfeps

`int ctl_t::ret_sfeps`

Retrieve surface layer emissivity (0=no, 1=yes).

Definition at line 655 of file jurassic.h.

write_bbt

`int ctl_t::write_bbt`

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

Definition at line 658 of file jurassic.h.

write_matrix

`int ctl_t::write_matrix`

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

Definition at line 661 of file jurassic.h.

formod

`int ctl_t::formod`

Forward model (0=CGA, 1=EGA, 2=RFM).

Definition at line 664 of file jurassic.h.

rfmbin

`char ctl_t::rfmbin[LEN]`

Path to RFM binary.

Definition at line 667 of file jurassic.h.

rfmhit

```
char ctl_t::rfmhit[LEN]
```

HITRAN file for RFM.

Definition at line 670 of file jurassic.h.

rfmxsc

```
char ctl_t::rfmxsc[NG][LEN]
```

Emitter cross-section files for RFM.

Definition at line 673 of file jurassic.h.

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

- jurassic.h

4.4 iasi_l1_t Struct Reference

IASI Level-1 data.

```
#include <libiasi.h>
```

Data Fields

- size_t ntrack

  Number of along-track values.
- double time [L1_NTRACK][L1_NXTRACK]

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

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

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

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

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

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

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

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

4.4.1 Detailed Description

IASI Level-1 data.

Definition at line 86 of file libiasi.h.

4.4.2 Field Documentation

ntrack

```
size_t iasi_l1_t::ntrack
```

Number of along-track values.

Definition at line 89 of file libiasi.h.

time

```
double iasi_l1_t::time[L1_NTRACK][L1_NXTRACK]
```

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

Definition at line 92 of file libiasi.h.

lon

```
double iasi_l1_t::lon[L1_NTRACK][L1_NXTRACK]
```

Footprint longitude [deg].

Definition at line 95 of file libiasi.h.

lat

```
double iasi_l1_t::lat[L1_NTRACK][L1_NXTRACK]
```

Footprint latitude [deg].

Definition at line 98 of file libiasi.h.

sat_z

```
double iasi_l1_t::sat_z[L1_NTRACK]
```

Satellite altitude [km].

Definition at line 101 of file libiasi.h.

sat_lon

```
double iasi_l1_t::sat_lon[L1_NTRACK]
```

Satellite longitude [deg].

Definition at line 104 of file libiasi.h.

sat_lat

```
double iasi_l1_t::sat_lat[L1_NTRACK]
```

Satellite latitude [deg].

Definition at line 107 of file libiasi.h.

nu

```
double iasi_l1_t::nu[L1_NCHAN]
```

Channel frequencies [cm^-1].

Definition at line 110 of file libiasi.h.

rad

```
float iasi_l1_t::rad[L1_NTRACK][L1_NXTRACK][L1_NCHAN]
```

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

Definition at line 113 of file libiasi.h.

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

- libiasi.h

4.5 iasi_l2_t Struct Reference

IASI Level-2 data.

```
#include <libiasi.h>
```

Data Fields

- size_t ntrack

    Number of along-track values.
- 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].

4.5.1 Detailed Description

IASI Level-2 data.

Definition at line 118 of file libiasi.h.

4.5.2 Field Documentation

ntrack

```
size_t iasi_l2_t::ntrack
```

Number of along-track values.

Definition at line 121 of file libiasi.h.

time

```
double iasi_l2_t::time[L2_NTRACK][L2_NXTRACK]
```

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

Definition at line 124 of file libiasi.h.

z

```
double iasi_l2_t::z[L2_NTRACK][L2_NXTRACK][L2_NLAY]
```

Geopotential height [km].

Definition at line 127 of file libiasi.h.

lon

```
double iasi_l2_t::lon[L2_NTRACK][L2_NXTRACK]
```

Longitude [deg].

Definition at line 130 of file libiasi.h.

lat

```
double iasi_l2_t::lat[L2_NTRACK][L2_NXTRACK]
```

Latitude [deg].

Definition at line 133 of file libiasi.h.

p

```
double iasi_l2_t::p[L2_NLAY]
```

Pressure [hPa].

Definition at line 136 of file libiasi.h.

t

```
double iasi_l2_t::t[L2_NTRACK][L2_NXTRACK][L2_NLAY]
```

Temperature [K].

Definition at line 139 of file libiasi.h.

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

- libiasi.h

4.6 iasi_rad_t Struct Reference

IASI converted Level-1 radiation data.

```
#include <libiasi.h>
```

Data Fields

- int ntrack
  Number of along-track samples.
- double freq [IASI_L1_NCHAN]
  channel wavenumber [cm^-1]
- double Time [L1_NTRACK][L1_NXTRACK]
  Seconds since 2000-01-01 for each sounder pixel.
- double Longitude [L1_NTRACK][L1_NXTRACK]
  Longitude of the sounder pixel.
- double Latitude [L1_NTRACK][L1_NXTRACK]
  Latitude of the sounder pixel.
- float Rad [L1_NTRACK][L1_NXTRACK][IASI_L1_NCHAN]
  Radiance [W/(m^2 sr cm^-1)].
- double Sat_z [L1_NTRACK]
  Altitude of the satellite.
- double Sat_lon [L1_NTRACK]
  Estimated longitude of the satellite.
- double Sat_lat [L1_NTRACK]
  Estimated latitude of the satellite.

4.6.1 Detailed Description

IASI converted Level-1 radiation data.

Definition at line 208 of file libiasi.h.

4.6.2 Field Documentation

ntrack

```
int iasi_rad_t::ntrack
```

Number of along-track samples.

Definition at line 211 of file libiasi.h.

freq

```
double iasi_rad_t::freq[IASI_L1_NCHAN]
```

channel wavenumber [cm^-1]

Definition at line 214 of file libiasi.h.

Time

```
double iasi_rad_t::Time[L1_NTRACK][L1_NXTRACK]
```

Seconds since 2000-01-01 for each sounder pixel.

Definition at line 217 of file libiasi.h.

Longitude

```
double iasi_rad_t::Longitude[L1_NTRACK][L1_NXTRACK]
```

Longitude of the sounder pixel.

Definition at line 220 of file libiasi.h.

Latitude

```
double iasi_rad_t::Latitude[L1_NTRACK][L1_NXTRACK]
```

Latitude of the sounder pixel.

Definition at line 223 of file libiasi.h.

Rad

```
float iasi_rad_t::Rad[L1_NTRACK][L1_NXTRACK][IASI_L1_NCHAN]
```

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

Definition at line 226 of file libiasi.h.

Sat_z

```
double iasi_rad_t::Sat_z[L1_NTRACK]
```

Altitude of the satellite.

Definition at line 229 of file libiasi.h.

Sat_lon

```
double iasi_rad_t::Sat_lon[L1_NTRACK]
```

Estimated longitude of the satellite.

Definition at line 232 of file libiasi.h.

Sat_lat

```
double iasi_rad_t::Sat_lat[L1_NTRACK]
```

Estimated latitude of the satellite.

Definition at line 235 of file libiasi.h.

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

- libiasi.h

4.7 iasi_raw_t Struct Reference

IASI raw Level-1 data.

```
#include <libiasi.h>
```

Data Fields

- long ntrack

    Number of along-track samples.
- float IDefSpectDWn1b [L1_NTRACK]

    Constants for radiation spectrum (must be equal to the expected constant).
- int32_t IDefNsfirst1b [L1_NTRACK]

    Constants for radiation spectrum (must be equal to the expected constant).
- int32_t IDefNslast1b [L1_NTRACK]

    Constants for radiation spectrum (must be equal to the expected constant).
- double Time [L1_NTRACK][IASI_NXTRACK]

    Time (seconds since 2000-01-01T00:00Z).
- double Loc [L1_NTRACK][IASI_NXTRACK][IASI_PM][2]

    Location of the sounder pixel (long,lat).
- float Wavenumber [IASI_L1_NCHAN]

    Wavenumbers are computed with the expected values.
- short int Radiation [L1_NTRACK][IASI_NXTRACK][IASI_PM][IASI_L1_NCHAN]

    Radiance [W/(m^2 sr m^-1)].
- unsigned int Sat_z [L1_NTRACK]

    Satellite altitude [m].

4.7.1 Detailed Description

IASI raw Level-1 data.

Definition at line 176 of file libiasi.h.

4.7.2 Field Documentation

ntrack

```
long iasi_raw_t::ntrack
```

Number of along-track samples.

Definition at line 179 of file libiasi.h.

IDefSpectDWn1b

```
float iasi_raw_t::IDefSpectDWn1b[L1_NTRACK]
```

Constants for radiation spectrum (must be equal to the expected constant).

Definition at line 182 of file libiasi.h.

IDefNsfirst1b

```
int32_t iasi_raw_t::IDefNsfirst1b[L1_NTRACK]
```

Constants for radiation spectrum (must be equal to the expected constant).

Definition at line 185 of file libiasi.h.

IDefNslast1b

```
int32_t iasi_raw_t::IDefNslast1b[L1_NTRACK]
```

Constants for radiation spectrum (must be equal to the expected constant).

Definition at line 188 of file libiasi.h.

Time

```
double iasi_raw_t::Time[L1_NTRACK][IASI_NXTRACK]
```

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

Definition at line 191 of file libiasi.h.

Loc

```
double iasi_raw_t::Loc[L1_NTRACK][IASI_NXTRACK][IASI_PM][2]
```

Location of the sounder pixel (long,lat).

Definition at line 194 of file libiasi.h.

Wavenumber

```
float iasi_raw_t::Wavenumber[IASI_L1_NCHAN]
```

Wavenumbers are computed with the expected values.

Definition at line 197 of file libiasi.h.

Radiation

```
short int iasi_raw_t::Radiation[L1_NTRACK][IASI_NXTRACK][IASI_PM][IASI_L1_NCHAN]
```

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

Definition at line 200 of file libiasi.h.

Sat_z

```
unsigned int iasi_raw_t::Sat_z[L1_NTRACK]
```

Satellite altitude [m].

Definition at line 203 of file libiasi.h.

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

- libiasi.h

4.8 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 [NLOS][NG]
  Volume mixing ratio [ppv].
- double k [NLOS][ND]
  Extinction [km^-1].
- double sft
  Surface temperature [K].
- double sfeps [ND]
  Surface emissivity.
- double ds [NLOS]
  Segment length [km].
- double u [NLOS][NG]
  Column density [molecules/cm^2].
- double cgp [NLOS][NG]
  Curtis-Godson pressure [hPa].
- double cgt [NLOS][NG]
  Curtis-Godson temperature [K].
- double cgu [NLOS][NG]
  Curtis-Godson column density [molecules/cm^2].
- double eps [NLOS][ND]
  Segment emissivity.
- double src [NLOS][ND]
  Segment source function [W/(m^2 sr cm^-1)].

4.8.1 Detailed Description

Line-of-sight data.

Definition at line 678 of file jurassic.h.

4.8.2 Field Documentation

np

```
int los_t::np
```

Number of LOS points.

Definition at line 681 of file jurassic.h.

z

`double los_t::z[NLOS]`

Altitude [km].

Definition at line 684 of file jurassic.h.

lon

`double los_t::lon[NLOS]`

Longitude [deg].

Definition at line 687 of file jurassic.h.

lat

`double los_t::lat[NLOS]`

Latitude [deg].

Definition at line 690 of file jurassic.h.

p

`double los_t::p[NLOS]`

Pressure [hPa].

Definition at line 693 of file jurassic.h.

t

`double los_t::t[NLOS]`

Temperature [K].

Definition at line 696 of file jurassic.h.

q

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

Volume mixing ratio [ppv].

Definition at line 699 of file jurassic.h.

k

`double los_t::k[NLOS][ND]`

Extinction [km^-1].

Definition at line 702 of file jurassic.h.

sft

`double los_t::sft`

Surface temperature [K].

Definition at line 705 of file jurassic.h.

sfeps

`double los_t::sfeps[ND]`

Surface emissivity.

Definition at line 708 of file jurassic.h.

ds

`double los_t::ds[NLOS]`

Segment length [km].

Definition at line 711 of file jurassic.h.

u

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

Column density [molecules/cm^2].

Definition at line 714 of file jurassic.h.

cgp

`double los_t::cgp[NLOS][NG]`

Curtis-Godson pressure [hPa].

Definition at line 717 of file jurassic.h.

cgt

```
double los_t::cgt[NLOS][NG]
```

Curtis-Godson temperature [K].

Definition at line 720 of file jurassic.h.

cgu

```
double los_t::cgu[NLOS][NG]
```

Curtis-Godson column density [molecules/cm^2].

Definition at line 723 of file jurassic.h.

eps

```
double los_t::eps[NLOS][ND]
```

Segment emissivity.

Definition at line 726 of file jurassic.h.

src

```
double los_t::src[NLOS][ND]
```

Segment source function [W/(m^2 sr cm^-1)].

Definition at line 729 of file jurassic.h.

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

- jurassic.h

4.9 met_t Struct Reference

Meteorological data.

Data Fields

- double time
  Time [s].
- int nx
  Number of longitudes.
- int ny
  Number of latitudes.
- int np
  Number of pressure levels.
- double lon [EX]
  Longitude [deg].
- double lat [EY]
  Latitude [deg].
- double p [EP]
  Pressure [hPa].
- double ps [EX][EY]
  Surface pressure [hPa].
- double pt [EX][EY]
  Tropopause pressure [hPa].
- float z [EX][EY][EP]
  Geopotential height [km].
- float t [EX][EY][EP]
  Temperature [K].
- float u [EX][EY][EP]
  Zonal wind [m/s].
- float v [EX][EY][EP]
  Meridional wind [m/s].
- float w [EX][EY][EP]
  Vertical wind [hPa/s].
- float pv [EX][EY][EP]
  Potential vorticity [PVU].
- float h2o [EX][EY][EP]
  Water vapor volume mixing ratio [1].
- float o3 [EX][EY][EP]
  Ozone volume mixing ratio [1].
- float pl [EX][EY][EP]
  Pressure on model levels [hPa].

4.9.1 Detailed Description

Meteorological data.

Definition at line 61 of file extract.c.

4.9.2 Field Documentation

time

```
double met_t::time
```

Time [s].

Definition at line 64 of file extract.c.

nx

```
int met_t::nx
```

Number of longitudes.

Definition at line 67 of file extract.c.

ny

```
int met_t::ny
```

Number of latitudes.

Definition at line 70 of file extract.c.

np

```
int met_t::np
```

Number of pressure levels.

Definition at line 73 of file extract.c.

lon

```
double met_t::lon[EX]
```

Longitude [deg].

Definition at line 76 of file extract.c.

lat

```
double met_t::lat[EY]
```

Latitude [deg].

Definition at line 79 of file extract.c.

p

```
double met_t::p[EP]
```

Pressure [hPa].

Definition at line 82 of file extract.c.

ps

```
double met_t::ps[EX][EY]
```

Surface pressure [hPa].

Definition at line 85 of file extract.c.

pt

```
double met_t::pt[EX][EY]
```

Tropopause pressure [hPa].

Definition at line 88 of file extract.c.

z

```
float met_t::z[EX][EY][EP]
```

Geopotential height [km].

Definition at line 91 of file extract.c.

t

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

Temperature [K].

Definition at line 94 of file extract.c.

u

```
float met_t::u[EX][EY][EP]
```

Zonal wind [m/s].

Definition at line 97 of file extract.c.

v

```
float met_t::v[EX][EY][EP]
```

Meridional wind [m/s].

Definition at line 100 of file extract.c.

w

`float met_t::w[EX][EY][EP]`

Vertical wind [hPa/s].

Definition at line 103 of file extract.c.

pv

`float met_t::pv[EX][EY][EP]`

Potential vorticity [PVU].

Definition at line 106 of file extract.c.

h2o

`float met_t::h2o[EX][EY][EP]`

Water vapor volume mixing ratio [1].

Definition at line 109 of file extract.c.

o3

`float met_t::o3[EX][EY][EP]`

Ozone volume mixing ratio [1].

Definition at line 112 of file extract.c.

pl

`float met_t::pl[EX][EY][EP]`

Pressure on model levels [hPa].

Definition at line 115 of file extract.c.

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

- extract.c

4.10 ncd_t Struct Reference

Buffer for netCDF data.

Data Fields

- int ncid
  NetCDF file ID.
- int np
  Number of retrieval altitudes.
- int ntrack
  Number of tacks.
- 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^-1].
- float l1_rad [L1_NTRACK][L1_NXTRACK][L1_NCHAN]
  Radiance [W/(m^2 sr cm^-1)].
- 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].
- float ret_chisq [L1_NTRACK ∗L1_NXTRACK]
  chi^2 value of fit.

4.10.1 Detailed Description

Buffer for netCDF data.

Definition at line 44 of file retrieval.c.

4.10.2 Field Documentation

ncid

```
int ncd_t::ncid
```

NetCDF file ID.

Definition at line 47 of file retrieval.c.

np

```
int ncd_t::np
```

Number of retrieval altitudes.

Definition at line 50 of file retrieval.c.

ntrack

```
int ncd_t::ntrack
```

Number of tacks.

Definition at line 53 of file retrieval.c.

l1_time

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

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

Definition at line 56 of file retrieval.c.

l1_lon

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

Footprint longitude [deg].

Definition at line 59 of file retrieval.c.

l1_lat

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

Footprint latitude [deg].

Definition at line 62 of file retrieval.c.

l1_sat_z

```
double ncd_t::l1_sat_z[L1_NTRACK]
```

Satellite altitude [km].

Definition at line 65 of file retrieval.c.

l1_sat_lon

```
double ncd_t::l1_sat_lon[L1_NTRACK]
```

Satellite longitude [deg].

Definition at line 68 of file retrieval.c.

l1_sat_lat

```
double ncd_t::l1_sat_lat[L1_NTRACK]
```

Satellite latitude [deg].

Definition at line 71 of file retrieval.c.

l1_nu

```
double ncd_t::l1_nu[L1_NCHAN]
```

Channel frequencies [cm^-1].

Definition at line 74 of file retrieval.c.

l1_rad

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

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

Definition at line 77 of file retrieval.c.

l2_z

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

Altitude [km].

Definition at line 80 of file retrieval.c.

l2_p

```
double ncd_t::l2_p[L2_NLAY]
```

Pressure [hPa].

Definition at line 83 of file retrieval.c.

l2_t

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

Temperature [K].

Definition at line 86 of file retrieval.c.

ret_z

```
float ncd_t::ret_z[NP]
```

Altitude [km].

Definition at line 89 of file retrieval.c.

ret_p

```
float ncd_t::ret_p[L1_NTRACK *L1_NXTRACK]
```

Pressure [hPa].

Definition at line 92 of file retrieval.c.

ret_t

```
float ncd_t::ret_t[L1_NTRACK *L1_NXTRACK *NP]
```

Temperature [K].

Definition at line 95 of file retrieval.c.

ret_chisq

```
float ncd_t::ret_chisq[L1_NTRACK *L1_NXTRACK]
```

chi^2 value of fit.

Definition at line 98 of file retrieval.c.

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

- retrieval.c

4.11 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)].

4.11.1 Detailed Description

Observation geometry and radiance data.

Definition at line 734 of file jurassic.h.

4.11.2 Field Documentation

nr

```
int obs_t::nr
```

Number of ray paths.

Definition at line 737 of file jurassic.h.

time

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

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

Definition at line 740 of file jurassic.h.

obsz

`double obs_t::obsz[NR]`

Observer altitude [km].

Definition at line 743 of file jurassic.h.

obslon

`double obs_t::obslon[NR]`

Observer longitude [deg].

Definition at line 746 of file jurassic.h.

obslat

`double obs_t::obslat[NR]`

Observer latitude [deg].

Definition at line 749 of file jurassic.h.

vpz

`double obs_t::vpz[NR]`

View point altitude [km].

Definition at line 752 of file jurassic.h.

vplon

`double obs_t::vplon[NR]`

View point longitude [deg].

Definition at line 755 of file jurassic.h.

vplat

`double obs_t::vplat[NR]`

View point latitude [deg].

Definition at line 758 of file jurassic.h.

tpz

```
double obs_t::tpz[NR]
```

Tangent point altitude [km].

Definition at line 761 of file jurassic.h.

tplon

```
double obs_t::tplon[NR]
```

Tangent point longitude [deg].

Definition at line 764 of file jurassic.h.

tplat

```
double obs_t::tplat[NR]
```

Tangent point latitude [deg].

Definition at line 767 of file jurassic.h.

tau

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

Transmittance of ray path.

Definition at line 770 of file jurassic.h.

rad

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

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

Definition at line 773 of file jurassic.h.

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

- jurassic.h

4.12 pert_t Struct Reference

Perturbation data.

```
#include <libiasi.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].

4.12.1 Detailed Description

Perturbation data.

Definition at line 144 of file libiasi.h.

4.12.2 Field Documentation

ntrack

```
int pert_t::ntrack
```

Number of along-track values.

Definition at line 147 of file libiasi.h.

nxtrack

```
int pert_t::nxtrack
```

Number of across-track values.

Definition at line 150 of file libiasi.h.

time

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

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

Definition at line 153 of file libiasi.h.

lon

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

Longitude [deg].

Definition at line 156 of file libiasi.h.

lat

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

Latitude [deg].

Definition at line 159 of file libiasi.h.

dc

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

Brightness temperature (8 micron) [K].

Definition at line 162 of file libiasi.h.

bt

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

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

Definition at line 165 of file libiasi.h.

pt

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

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

Definition at line 168 of file libiasi.h.

var

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

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

Definition at line 171 of file libiasi.h.

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

- libiasi.h

4.13 ret_t Struct Reference

Retrieval control parameters.

Data Fields

- 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 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].

4.13.1 Detailed Description

Retrieval control parameters.

Definition at line 103 of file retrieval.c.

4.13.2 Field Documentation

kernel_recomp

```
int ret_t::kernel_recomp
```

Recomputation of kernel matrix (number of iterations).

Definition at line 106 of file retrieval.c.

conv_itmax

```
int ret_t::conv_itmax
```

Maximum number of iterations.

Definition at line 109 of file retrieval.c.

conv_dmin

```
double ret_t::conv_dmin
```

Minimum normalized step size in state space.

Definition at line 112 of file retrieval.c.

err_formod

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

Forward model error [%].

Definition at line 115 of file retrieval.c.

err_noise

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

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

Definition at line 118 of file retrieval.c.

err_press

`double ret_t::err_press`

Pressure error [%].

Definition at line 121 of file retrieval.c.

err_press_cz

`double ret_t::err_press_cz`

Vertical correlation length for pressure error [km].

Definition at line 124 of file retrieval.c.

err_press_ch

`double ret_t::err_press_ch`

Horizontal correlation length for pressure error [km].

Definition at line 127 of file retrieval.c.

err_temp

`double ret_t::err_temp`

Temperature error [K].

Definition at line 130 of file retrieval.c.

err_temp_cz

`double ret_t::err_temp_cz`

Vertical correlation length for temperature error [km].

Definition at line 133 of file retrieval.c.

err_temp_ch

`double ret_t::err_temp_ch`

Horizontal correlation length for temperature error [km].

Definition at line 136 of file retrieval.c.

err_q

```
double ret_t::err_q[NG]
```

Volume mixing ratio error [%].

Definition at line 139 of file retrieval.c.

err_q_cz

```
double ret_t::err_q_cz[NG]
```

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

Definition at line 142 of file retrieval.c.

err_q_ch

```
double ret_t::err_q_ch[NG]
```

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

Definition at line 145 of file retrieval.c.

err_k

```
double ret_t::err_k[NW]
```

Extinction error [1/km].

Definition at line 148 of file retrieval.c.

err_k_cz

```
double ret_t::err_k_cz[NW]
```

Vertical correlation length for extinction error [km].

Definition at line 151 of file retrieval.c.

err_k_ch

```
double ret_t::err_k_ch[NW]
```

Horizontal correlation length for extinction error [km].

Definition at line 154 of file retrieval.c.

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

- retrieval.c

4.14 tbl_t Struct Reference

Emissivity look-up tables.

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

Data Fields

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

4.14.1 Detailed Description

Emissivity look-up tables.

Definition at line 778 of file jurassic.h.

4.14.2 Field Documentation

np

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

Number of pressure levels.

Definition at line 781 of file jurassic.h.

nt

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

Number of temperatures.

Definition at line 784 of file jurassic.h.

nu

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

Number of column densities.

Definition at line 787 of file jurassic.h.

p

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

Pressure [hPa].

Definition at line 790 of file jurassic.h.

t

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

Temperature [K].

Definition at line 793 of file jurassic.h.

u

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

Column density [molecules/cm^2].

Definition at line 796 of file jurassic.h.

eps

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

Emissivity.

Definition at line 799 of file jurassic.h.

st

```
double tbl_t::st[TBLNS]
```

Source function temperature [K].

Definition at line 802 of file jurassic.h.

sr

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

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

Definition at line 805 of file jurassic.h.

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

- jurassic.h

4.15 wave_t Struct Reference

Wave analysis data.

```
#include <libiasi.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].

4.15.1 Detailed Description

Wave analysis data.

Definition at line 240 of file libiasi.h.

4.15.2 Field Documentation

nx

```
int wave_t::nx
```

Number of across-track values.

Definition at line 243 of file libiasi.h.

ny

```
int wave_t::ny
```

Number of along-track values.

Definition at line 246 of file libiasi.h.

time

```
double wave_t::time
```

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

Definition at line 249 of file libiasi.h.

z

```
double wave_t::z
```

Altitude [km].

Definition at line 252 of file libiasi.h.

lon

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

Longitude [deg].

Definition at line 255 of file libiasi.h.

lat

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

Latitude [deg].

Definition at line 258 of file libiasi.h.

x

`double wave_t::x[WX]`

Across-track distance [km].

Definition at line 261 of file libiasi.h.

y

`double wave_t::y[WY]`

Along-track distance [km].

Definition at line 264 of file libiasi.h.

temp

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

Temperature [K].

Definition at line 267 of file libiasi.h.

bg

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

Background [K].

Definition at line 270 of file libiasi.h.

pt

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

Perturbation [K].

Definition at line 273 of file libiasi.h.

var

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

Variance [K].

Definition at line 276 of file libiasi.h.

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

- libiasi.h

5 File Documentation

5.1 bands.c File Reference

Functions

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

5.1.1 Function Documentation

main()

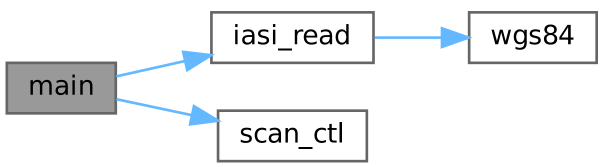
```
int main (
            int argc,
            char * argv[ ] )
```

Definition at line 14 of file bands.c.

```
                  {

  static iasi_rad_t *iasi_rad;

  static FILE *out;

  static double numin[NB], numax[NB], rad[NB];

  static int iarg, ib, ichan, n, nb, track, xtrack;

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

  /* Allocate... */
  ALLOC(iasi_rad, iasi_rad_t, 1);

  /* Get control parameters... */
  nb = (int) scan_ctl(argc, argv, "NB", -1, "1", NULL);
  if (nb > NB)
    ERRMSG("Too many bands!");
  for (ib = 0; ib < nb; ib++) {
    numin[ib] = scan_ctl(argc, argv, "NUMIN", ib, "", NULL);
    numax[ib] = scan_ctl(argc, argv, "NUMAX", ib, "", NULL);
  }

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

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

    /* Read IASI data... */
    printf("Read IASI Level-1C data file: %s\n", argv[iarg]);
    iasi_read(argv[iarg], iasi_rad);

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

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

```

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

      /* Loop over footprints... */
      for (xtrack = 0; xtrack < L1_NXTRACK; xtrack++) {

        /* Write output... */
        fprintf(out, "%.2f %.4f %.4f %.3f %.4f %.4f",
                iasi_rad->Time[track][xtrack],
                iasi_rad->Longitude[track][xtrack],
                iasi_rad->Latitude[track][xtrack],
                iasi_rad->Sat_z[track],
                iasi_rad->Sat_lon[track], iasi_rad->Sat_lat[track]);

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

          /* Get mean radiance... */
          n = 0;
          rad[ib] = 0;
          for (ichan = 0; ichan <= IASI_L1_NCHAN; ichan++)
            if (iasi_rad->freq[ichan] >= numin[ib]
                && iasi_rad->freq[ichan] <= numax[ib]
                && gsl_finite(iasi_rad->Rad[track][xtrack][ichan])) {
              rad[ib] += iasi_rad->Rad[track][xtrack][ichan];
              n++;
            }
          if (n > 0)
            rad[ib] /= n;
          else
            rad[ib] = GSL_NAN;

          /* Convert to brightness temperature... */
          rad[ib] = BRIGHT(rad[ib], 0.5 * (numin[ib] + numax[ib]));

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

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

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

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

  return EXIT_SUCCESS;
}
```

Here is the call graph for this function:



5.2 bands.c

Go to the documentation of this file.

```
#include "libiasi.h"

```

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

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

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

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

  static iasi_rad_t *iasi_rad;

  static FILE *out;

  static double numin[NB], numax[NB], rad[NB];

  static int iarg, ib, ichan, n, nb, track, xtrack;

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

  /* Allocate... */
  ALLOC(iasi_rad, iasi_rad_t, 1);

  /* Get control parameters... */
  nb = (int) scan_ctl(argc, argv, "NB", -1, "1", NULL);
  if (nb > NB)
    ERRMSG("Too many bands!");
  for (ib = 0; ib < nb; ib++) {
    numin[ib] = scan_ctl(argc, argv, "NUMIN", ib, "", NULL);
    numax[ib] = scan_ctl(argc, argv, "NUMAX", ib, "", NULL);
  }

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

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

    /* Read IASI data... */
    printf("Read IASI Level-1C data file: %s\n", argv[iarg]);
    iasi_read(argv[iarg], iasi_rad);

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

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

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

      /* Loop over footprints... */
      for (xtrack = 0; xtrack < L1_NXTRACK; xtrack++) {

        /* Write output... */
        fprintf(out, "%.2f %.4f %.4f %.3f %.4f %.4f",
                iasi_rad->Time[track][xtrack],
                iasi_rad->Longitude[track][xtrack],
                iasi_rad->Latitude[track][xtrack],
                iasi_rad->Sat_z[track],
                iasi_rad->Sat_lon[track], iasi_rad->Sat_lat[track]);

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

          /* Get mean radiance... */
```

```
        n = 0;
        rad[ib] = 0;
        for (ichan = 0; ichan <= IASI_L1_NCHAN; ichan++)
          if (iasi_rad->freq[ichan] >= numin[ib]
              && iasi_rad->freq[ichan] <= numax[ib]
              && gsl_finite(iasi_rad->Rad[track][xtrack][ichan])) {
            rad[ib] += iasi_rad->Rad[track][xtrack][ichan];
            n++;
          }
        if (n > 0)
          rad[ib] /= n;
        else
          rad[ib] = GSL_NAN;

        /* Convert to brightness temperature... */
        rad[ib] = BRIGHT(rad[ib], 0.5 * (numin[ib] + numax[ib]));

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

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

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

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

return EXIT_SUCCESS;
}
```

5.3 extract.c File Reference

Data Structures

- struct ctl2_t

  Control parameters.
- struct met_t

  Meteorological data.

Functions

- void get_met (ctl2_t ∗ctl2, char ∗metbase, double t, met_t ∗met0, met_t ∗met1)

  Get meteorological data for given timestep.
- void get_met_help (double t, int direct, char ∗metbase, double dt_met, char ∗filename)

  Get meteorological data for timestep.
- void intpol_met_2d (double array[EX][EY], int ix, int iy, double wx, double wy, double ∗var)

  Linear interpolation of 2-D meteorological data.
- void intpol_met_3d (float array[EX][EY][EP], int ip, int ix, int iy, double wp, double wx, double wy, double ∗var)

  Linear interpolation of 3-D meteorological data.
- void intpol_met_space (met_t ∗met, double p, double lon, double lat, double ∗ps, double ∗pt, double ∗z, double ∗t, double ∗u, double ∗v, double ∗w, double ∗pv, double ∗h2o, double ∗o3)

  Spatial interpolation of meteorological data.
- void intpol_met_time (met_t ∗met0, met_t ∗met1, double ts, double p, double lon, double lat, double ∗ps, double ∗pt, double ∗z, double ∗t, double ∗u, double ∗v, double ∗w, double ∗pv, double ∗h2o, double ∗o3)

  Temporal interpolation of meteorological data.
- void read_ctl2 (int argc, char ∗argv[ ], ctl2_t ∗ctl2)

  Read control parameters.
- void read_met (ctl2_t ∗ctl2, char ∗filename, met_t ∗met)

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

  Extrapolate meteorological data at lower boundary.
- void read_met_geopot (ctl2_t ∗ctl2, met_t ∗met)

  Calculate geopotential heights.
- void read_met_help (int ncid, char ∗varname, char ∗varname2, met_t ∗met, float dest[EX][EY][EP], float scl)

  Read and convert variable from meteorological data file.
- void read_met_periodic (met_t ∗met)

  Create meteorological data with periodic boundary conditions.
- void read_met_sample (ctl2_t ∗ctl2, met_t ∗met)

  Downsampling of meteorological data.
- int main (int argc, char ∗argv[ ])

Variables

- int iasi_chan [L1_NCHAN]

  List of IASI channels (don't change).

5.3.1 Function Documentation

get_met()

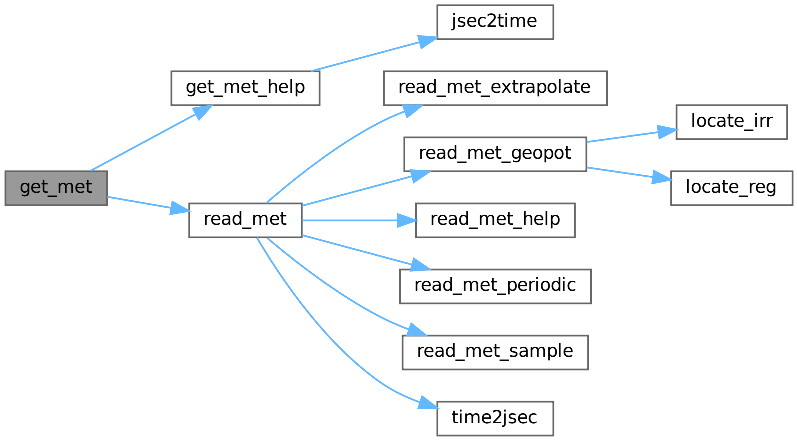
```
void get_met (
            ctl2_t * ctl2,
            char * metbase,
            double t,
            met_t * met0,
            met_t * met1 )
```

Get meteorological data for given timestep.

Definition at line 330 of file extract.c.

```
                      {

  char filename[LEN];

  static int init;

  /* Init... */
  if (!init) {
    init = 1;

    get_met_help(t, -1, metbase, ctl2->dt_met, filename);
    read_met(ctl2, filename, met0);

    get_met_help(t + 1.0, 1, metbase, ctl2->dt_met, filename);
    read_met(ctl2, filename, met1);
  }

  /* Read new data for forward trajectories... */
  if (t > met1->time) {
    memcpy(met0, met1, sizeof(met_t));
    get_met_help(t, 1, metbase, ctl2->dt_met, filename);
    read_met(ctl2, filename, met1);
  }
}
```

Here is the call graph for this function:



get_met_help()

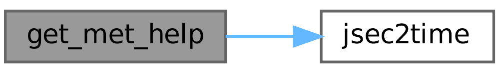
```
void get_met_help (
            double t,
            int direct,
            char * metbase,
            double dt_met,
            char * filename )
```

Get meteorological data for timestep.

Definition at line 362 of file extract.c.

```
                 {

  double t6, r;

  int year, mon, day, hour, min, sec;

  /* Round time to fixed intervals... */
  if (direct == -1)
    t6 = floor(t / dt_met) * dt_met;
  else
    t6 = ceil(t / dt_met) * dt_met;

  /* Decode time... */
  jsec2time(t6, &year, &mon, &day, &hour, &min, &sec, &r);

  /* Set filename... */
  sprintf(filename, "%s_%d_%02d_%02d_%02d.nc", metbase, year, mon, day, hour);
}
```

Here is the call graph for this function:



intpol_met_2d()

```
void intpol_met_2d (
            double array[EX][EY],
            int ix,
            int iy,
            double wx,
            double wy,
            double * var )
```

Linear interpolation of 2-D meteorological data.

Definition at line 388 of file extract.c.

```
                  {

  double aux00, aux01, aux10, aux11;

  /* Set variables... */
  aux00 = array[ix][iy];
  aux01 = array[ix][iy + 1];
  aux10 = array[ix + 1][iy];
  aux11 = array[ix + 1][iy + 1];

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

intpol_met_3d()

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

Linear interpolation of 3-D meteorological data.

Definition at line 412 of file extract.c.

```
                  {

  double aux00, aux01, aux10, aux11;

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

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

intpol_met_space()

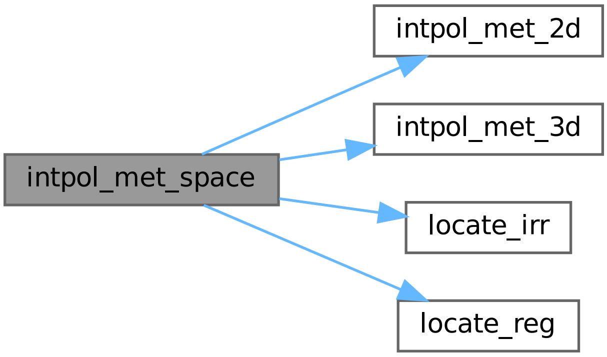
```
void intpol_met_space (
            met_t * met,
            double p,
            double lon,
            double lat,
            double * ps,
            double * pt,
            double * z,
            double * t,
            double * u,
            double * v,
            double * w,
            double * pv,
            double * h2o,
            double * o3 )
```

Spatial interpolation of meteorological data.

Definition at line 442 of file extract.c.

```
                  {

  double wp, wx, wy;

  int ip, ix, iy;

  /* Check longitude... */
  if (met->lon[met->nx - 1] > 180 && lon < 0)
    lon += 360;

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

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

  /* Interpolate... */
  if (ps != NULL)
    intpol_met_2d(met->ps, ix, iy, wx, wy, ps);
  if (pt != NULL)
    intpol_met_2d(met->pt, ix, iy, wx, wy, pt);
  if (z != NULL)
    intpol_met_3d(met->z, ip, ix, iy, wp, wx, wy, z);
  if (t != NULL)
    intpol_met_3d(met->t, ip, ix, iy, wp, wx, wy, t);
  if (u != NULL)
    intpol_met_3d(met->u, ip, ix, iy, wp, wx, wy, u);
  if (v != NULL)
    intpol_met_3d(met->v, ip, ix, iy, wp, wx, wy, v);
  if (w != NULL)
    intpol_met_3d(met->w, ip, ix, iy, wp, wx, wy, w);
  if (pv != NULL)
    intpol_met_3d(met->pv, ip, ix, iy, wp, wx, wy, pv);
  if (h2o != NULL)
    intpol_met_3d(met->h2o, ip, ix, iy, wp, wx, wy, h2o);
  if (o3 != NULL)
    intpol_met_3d(met->o3, ip, ix, iy, wp, wx, wy, o3);
}
```

Here is the call graph for this function:



intpol_met_time()

```
void intpol_met_time (
            met_t * met0,
            met_t * met1,
            double ts,
            double p,
            double lon,
            double lat,
            double * ps,
            double * pt,
            double * z,
            double * t,
            double * u,
            double * v,
            double * w,
            double * pv,
            double * h2o,
            double * o3 )
```

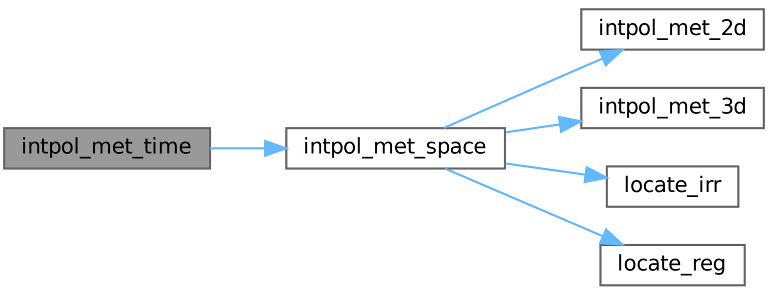
Temporal interpolation of meteorological data.

Definition at line 501 of file extract.c.

```
                 {

  double h2o0, h2o1, o30, o31, ps0, ps1, pt0, pt1, pv0, pv1, t0, t1, u0, u1,
    v0, v1, w0, w1, wt, z0, z1;

  /* Spatial interpolation... */
  intpol_met_space(met0, p, lon, lat,
                   ps == NULL ? NULL : &ps0,
                   pt == NULL ? NULL : &pt0,
                   z == NULL ? NULL : &z0,
                   t == NULL ? NULL : &t0,
                   u == NULL ? NULL : &u0,
                   v == NULL ? NULL : &v0,
                   w == NULL ? NULL : &w0,
                   pv == NULL ? NULL : &pv0,
                   h2o == NULL ? NULL : &h2o0, o3 == NULL ? NULL : &o30);
  intpol_met_space(met1, p, lon, lat,
```

```
                    ps == NULL ? NULL : &ps1,
                    pt == NULL ? NULL : &pt1,
                    z == NULL ? NULL : &z1,
                    t == NULL ? NULL : &t1,
                    u == NULL ? NULL : &u1,
                    v == NULL ? NULL : &v1,
                    w == NULL ? NULL : &w1,
                    pv == NULL ? NULL : &pv1,
                    h2o == NULL ? NULL : &h2o1, o3 == NULL ? NULL : &o31);

  /* Get weighting factor... */
  wt = (met1->time - ts) / (met1->time - met0->time);

  /* Interpolate... */
  if (ps != NULL)
    *ps = wt * (ps0 - ps1) + ps1;
  if (pt != NULL)
    *pt = wt * (pt0 - pt1) + pt1;
  if (z != NULL)
    *z = wt * (z0 - z1) + z1;
  if (t != NULL)
    *t = wt * (t0 - t1) + t1;
  if (u != NULL)
    *u = wt * (u0 - u1) + u1;
  if (v != NULL)
    *v = wt * (v0 - v1) + v1;
  if (w != NULL)
    *w = wt * (w0 - w1) + w1;
  if (pv != NULL)
    *pv = wt * (pv0 - pv1) + pv1;
  if (h2o != NULL)
    *h2o = wt * (h2o0 - h2o1) + h2o1;
  if (o3 != NULL)
    *o3 = wt * (o30 - o31) + o31;
}
```

Here is the call graph for this function:



read_ctl2()

```
void read_ctl2 (
            int argc,
            char * argv[],
            ctl2_t * ctl2 )
```

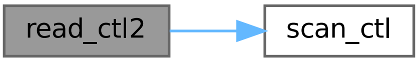
Read control parameters.

Definition at line 572 of file extract.c.

```
                     {

  /* Meteorological data... */
  ctl2->dt_met = scan_ctl(argc, argv, "DT_MET", -1, "21600", NULL);
  scan_ctl(argc, argv, "MET_GEOPOT", -1, "", ctl2->met_geopot);
```

```
  ctl2->met_dx = (int) scan_ctl(argc, argv, "MET_DX", -1, "1", NULL);
  ctl2->met_dy = (int) scan_ctl(argc, argv, "MET_DY", -1, "1", NULL);
  ctl2->met_dp = (int) scan_ctl(argc, argv, "MET_DP", -1, "1", NULL);
  ctl2->met_sx = (int) scan_ctl(argc, argv, "MET_SX", -1, "20", NULL);
  ctl2->met_sy = (int) scan_ctl(argc, argv, "MET_SY", -1, "10", NULL);
  ctl2->met_sp = (int) scan_ctl(argc, argv, "MET_SP", -1, "1", NULL);
}
```

Here is the call graph for this function:



read_met()

```
void read_met (
            ctl2_t * ctl2,
            char * filename,
            met_t * met )
```

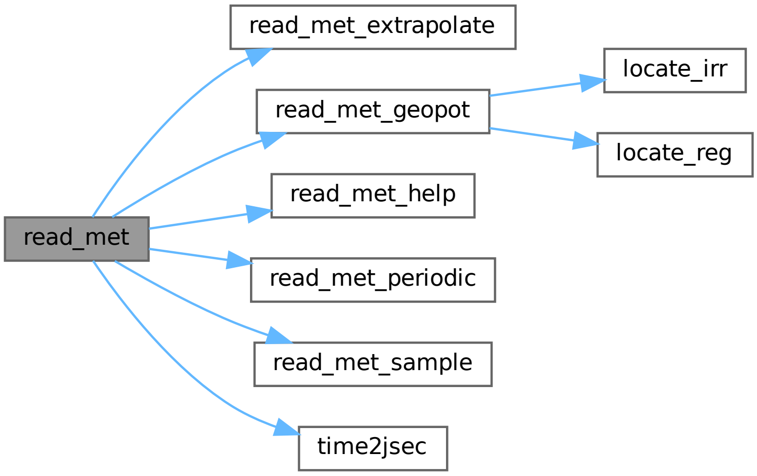
Read meteorological data file.

Definition at line 590 of file extract.c.

```
                {

  char levname[LEN], tstr[10];

  static float help[EX * EY];

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

  size_t np, nx, ny;

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

  /* Get time from filename... */
  sprintf(tstr, "%.4s", &filename[strlen(filename) - 16]);
  year = atoi(tstr);
  sprintf(tstr, "%.2s", &filename[strlen(filename) - 11]);
  mon = atoi(tstr);
  sprintf(tstr, "%.2s", &filename[strlen(filename) - 8]);
  day = atoi(tstr);
  sprintf(tstr, "%.2s", &filename[strlen(filename) - 5]);
  hour = atoi(tstr);
  time2jsec(year, mon, day, hour, 0, 0, 0, &met->time);

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

  /* Get dimensions... */
  NC(nc_inq_dimid(ncid, "lon", &dimid));
  NC(nc_inq_dimlen(ncid, dimid, &nx));
  if (nx < 2 || nx > EX)
    ERRMSG("Number of longitudes out of range!");

  NC(nc_inq_dimid(ncid, "lat", &dimid));
  NC(nc_inq_dimlen(ncid, dimid, &ny));
  if (ny < 2 || ny > EY)
    ERRMSG("Number of latitudes out of range!");

  sprintf(levname, "lev");
  NC(nc_inq_dimid(ncid, levname, &dimid));
  NC(nc_inq_dimlen(ncid, dimid, &np));
```

```
  if (np == 1) {
    sprintf(levname, "lev_2");
    NC(nc_inq_dimid(ncid, levname, &dimid));
    NC(nc_inq_dimlen(ncid, dimid, &np));
  }
  if (np < 2 || np > EP)
    ERRMSG("Number of levels out of range!");

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

  /* Get horizontal grid... */
  NC(nc_inq_varid(ncid, "lon", &varid));
  NC(nc_get_var_double(ncid, varid, met->lon));
  NC(nc_inq_varid(ncid, "lat", &varid));
  NC(nc_get_var_double(ncid, varid, met->lat));

  /* Read meteorological data... */
  read_met_help(ncid, "t", "T", met, met->t, 1.0);
  read_met_help(ncid, "u", "U", met, met->u, 1.0);
  read_met_help(ncid, "v", "V", met, met->v, 1.0);
  read_met_help(ncid, "w", "W", met, met->w, 0.01f);
  read_met_help(ncid, "q", "Q", met, met->h2o, 1.608f);
  read_met_help(ncid, "o3", "O3", met, met->o3, 0.602f);

  /* Read pressure levels from file... */
  NC(nc_inq_varid(ncid, levname, &varid));
  NC(nc_get_var_double(ncid, varid, met->p));
  for (ip = 0; ip < met->np; ip++)
    met->p[ip] /= 100.;

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

  /* Check ordering of pressure levels... */
  for (ip = 1; ip < met->np; ip++)
    if (met->p[ip - 1] < met->p[ip])
      ERRMSG("Pressure levels must be descending!");

  /* Read surface pressure... */
  if (nc_inq_varid(ncid, "ps", &varid) == NC_NOERR
      || nc_inq_varid(ncid, "PS", &varid) == NC_NOERR) {
    NC(nc_get_var_float(ncid, varid, help));
    for (iy = 0; iy < met->ny; iy++)
      for (ix = 0; ix < met->nx; ix++)
        met->ps[ix][iy] = help[iy * met->nx + ix] / 100.;
  } else if (nc_inq_varid(ncid, "lnsp", &varid) == NC_NOERR
             || nc_inq_varid(ncid, "LNSP", &varid) == NC_NOERR) {
    NC(nc_get_var_float(ncid, varid, help));
    for (iy = 0; iy < met->ny; iy++)
      for (ix = 0; ix < met->nx; ix++)
        met->ps[ix][iy] = exp(help[iy * met->nx + ix]) / 100.;
  } else
    for (ix = 0; ix < met->nx; ix++)
      for (iy = 0; iy < met->ny; iy++)
        met->ps[ix][iy] = met->p[0];

  /* Create periodic boundary conditions... */
  read_met_periodic(met);

  /* Calculate geopotential heights... */
  read_met_geopot(ctl2, met);

  /* Downsampling... */
  read_met_sample(ctl2, met);

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

Here is the call graph for this function:



read_met_extrapolate()

```
void read_met_extrapolate (
            met_t * met )
```

Extrapolate meteorological data at lower boundary.

Definition at line 708 of file extract.c.

```
                      {

  int ip, ip0, ix, iy;

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

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

      /* Extrapolate... */
      for (ip = ip0; ip >= 0; ip--) {
        met->t[ix][iy][ip] = met->t[ix][iy][ip + 1];
        met->u[ix][iy][ip] = met->u[ix][iy][ip + 1];
        met->v[ix][iy][ip] = met->v[ix][iy][ip + 1];
        met->w[ix][iy][ip] = met->w[ix][iy][ip + 1];
        met->h2o[ix][iy][ip] = met->h2o[ix][iy][ip + 1];
        met->o3[ix][iy][ip] = met->o3[ix][iy][ip + 1];
      }
    }
}
```

read_met_geopot()

```
void read_met_geopot (
            ctl2_t * ctl2,
            met_t * met )
```

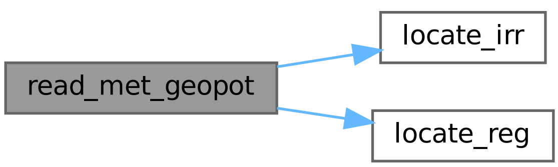
Calculate geopotential heights.

Definition at line 739 of file extract.c.

```
                     {

  static double topo_lat[EY], topo_lon[EX], topo_z[EX][EY];

  static int init, topo_nx = -1, topo_ny;

  FILE *in;

  char line[LEN];

  double data[30], lat, lon, rlat, rlon, rlon_old = -999, rz, ts, z0, z1;

  float help[EX][EY];

  int ip, ip0, ix, ix2, ix3, iy, iy2, n, tx, ty;

  /* Initialize geopotential heights... */
  for (ix = 0; ix < met->nx; ix++)
    for (iy = 0; iy < met->ny; iy++)
      for (ip = 0; ip < met->np; ip++)
        met->z[ix][iy][ip] = GSL_NAN;

  /* Check filename... */
  if (ctl2->met_geopot[0] == '-')
    return;

  /* Read surface geopotential... */
  if (!init) {

    /* Write info... */
    printf("Read surface geopotential: %s\n", ctl2->met_geopot);

    /* Open file... */
    if (!(in = fopen(ctl2->met_geopot, "r")))
      ERRMSG("Cannot open file!");

    /* Read data... */
    while (fgets(line, LEN, in))
      if (sscanf(line, "%lg %lg %lg", &rlon, &rlat, &rz) == 3) {
        if (rlon != rlon_old) {
          if ((++topo_nx) >= EX)
            ERRMSG("Too many longitudes!");
          topo_ny = 0;
        }
        rlon_old = rlon;
        topo_lon[topo_nx] = rlon;
        topo_lat[topo_ny] = rlat;
        topo_z[topo_nx][topo_ny] = rz;
        if ((++topo_ny) >= EY)
          ERRMSG("Too many latitudes!");
      }
    if ((++topo_nx) >= EX)
      ERRMSG("Too many longitudes!");

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

    /* Check grid spacing... */
    if (fabs(met->lon[0] - met->lon[1]) != fabs(topo_lon[0] - topo_lon[1])
        || fabs(met->lat[0] - met->lat[1]) != fabs(topo_lat[0] - topo_lat[1]))
      printf("Warning: Grid spacing does not match!\n");

    /* Set init flag... */
    init = 1;
  }

  /* Apply hydrostatic equation to calculate geopotential heights... */
  for (ix = 0; ix < met->nx; ix++)
    for (iy = 0; iy < met->ny; iy++) {

      /* Get surface height... */
      lon = met->lon[ix];
      if (lon < topo_lon[0])
```

```
        lon += 360;
      else if (lon > topo_lon[topo_nx - 1])
        lon -= 360;
      lat = met->lat[iy];
      tx = locate_reg(topo_lon, topo_nx, lon);
      ty = locate_reg(topo_lat, topo_ny, lat);
      z0 = LIN(topo_lon[tx], topo_z[tx][ty],
               topo_lon[tx + 1], topo_z[tx + 1][ty], lon);
      z1 = LIN(topo_lon[tx], topo_z[tx][ty + 1],
               topo_lon[tx + 1], topo_z[tx + 1][ty + 1], lon);
      z0 = LIN(topo_lat[ty], z0, topo_lat[ty + 1], z1, lat);

      /* Find surface pressure level... */
      ip0 = locate_irr(met->p, met->np, met->ps[ix][iy]);

      /* Get surface temperature... */
      ts = LIN(met->p[ip0], met->t[ix][iy][ip0],
               met->p[ip0 + 1], met->t[ix][iy][ip0 + 1], met->ps[ix][iy]);

      /* Upper part of profile... */
      met->z[ix][iy][ip0 + 1]
        = (float) (z0 + 8.31441 / 28.9647 / G0
                   * 0.5 * (ts + met->t[ix][iy][ip0 + 1])
                   * log(met->ps[ix][iy] / met->p[ip0 + 1]));
      for (ip = ip0 + 2; ip < met->np; ip++)
        met->z[ix][iy][ip]
          = (float) (met->z[ix][iy][ip - 1] + 8.31441 / 28.9647 / G0
                     * 0.5 * (met->t[ix][iy][ip - 1] + met->t[ix][iy][ip])
                     * log(met->p[ip - 1] / met->p[ip]));
    }

  /* Smooth fields... */
  for (ip = 0; ip < met->np; ip++) {

    /* Median filter... */
    for (ix = 0; ix < met->nx; ix++)
      for (iy = 0; iy < met->nx; iy++) {
        n = 0;
        for (ix2 = ix - 2; ix2 <= ix + 2; ix2++) {
          ix3 = ix2;
          if (ix3 < 0)
            ix3 += met->nx;
          if (ix3 >= met->nx)
            ix3 -= met->nx;
          for (iy2 = GSL_MAX(iy - 2, 0); iy2 <= GSL_MIN(iy + 2, met->ny - 1);
               iy2++)
            if (gsl_finite(met->z[ix3][iy2][ip])) {
              data[n] = met->z[ix3][iy2][ip];
              n++;
            }
        }
        if (n > 0) {
          gsl_sort(data, 1, (size_t) n);
          help[ix][iy] = (float)
            gsl_stats_median_from_sorted_data(data, 1, (size_t) n);
        } else
          help[ix][iy] = GSL_NAN;
      }

    /* Copy data... */
    for (ix = 0; ix < met->nx; ix++)
      for (iy = 0; iy < met->nx; iy++)
        met->z[ix][iy][ip] = help[ix][iy];
  }
}
```

Here is the call graph for this function:



read_met_help()

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

Read and convert variable from meteorological data file.

Definition at line 882 of file extract.c.

```
                {

  static float help[EX * EY * EP];

  int ip, ix, iy, varid;

  /* Check if variable exists... */
  if (nc_inq_varid(ncid, varname, &varid) != NC_NOERR)
    if (nc_inq_varid(ncid, varname2, &varid) != NC_NOERR)
      return;

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

  /* Copy and check data... */
  for (ix = 0; ix < met->nx; ix++)
    for (iy = 0; iy < met->ny; iy++)
      for (ip = 0; ip < met->np; ip++) {
        dest[ix][iy][ip] = help[(ip * met->ny + iy) * met->nx + ix];
        if (fabsf(dest[ix][iy][ip]) < 1e14f)
          dest[ix][iy][ip] *= scl;
        else
          dest[ix][iy][ip] = GSL_NAN;
      }
}
```

read_met_periodic()

```
void read_met_periodic (
            met_t * met )
```

Create meteorological data with periodic boundary conditions.

Definition at line 916 of file extract.c.

```
                  {

  int ip, iy;

  /* Check longitudes... */
  if (!(fabs(met->lon[met->nx - 1] - met->lon[0]
             + met->lon[1] - met->lon[0] - 360) < 0.01))
    return;

  /* Increase longitude counter... */
  if ((++met->nx) > EX)
    ERRMSG("Cannot create periodic boundary conditions!");

  /* Set longitude... */
  met->lon[met->nx - 1] = met->lon[met->nx - 2] + met->lon[1] - met->lon[0];

  /* Loop over latitudes and pressure levels... */
  for (iy = 0; iy < met->ny; iy++)
    for (ip = 0; ip < met->np; ip++) {
      met->ps[met->nx - 1][iy] = met->ps[0][iy];
      met->pt[met->nx - 1][iy] = met->pt[0][iy];
      met->z[met->nx - 1][iy][ip] = met->z[0][iy][ip];
      met->t[met->nx - 1][iy][ip] = met->t[0][iy][ip];
      met->u[met->nx - 1][iy][ip] = met->u[0][iy][ip];
      met->v[met->nx - 1][iy][ip] = met->v[0][iy][ip];
      met->w[met->nx - 1][iy][ip] = met->w[0][iy][ip];
      met->pv[met->nx - 1][iy][ip] = met->pv[0][iy][ip];
      met->h2o[met->nx - 1][iy][ip] = met->h2o[0][iy][ip];
      met->o3[met->nx - 1][iy][ip] = met->o3[0][iy][ip];
    }
}
```

read_met_sample()

```
void read_met_sample (
            ctl2_t * ctl2,
            met_t * met )
```

Downsampling of meteorological data.

Definition at line 951 of file extract.c.

```
                    {

  met_t *help;

  float w, wsum;

  int ip, ip2, ix, ix2, ix3, iy, iy2;

  /* Check parameters... */
  if (ctl2->met_dp <= 1 && ctl2->met_dx <= 1 && ctl2->met_dy <= 1
      && ctl2->met_sp <= 1 && ctl2->met_sx <= 1 && ctl2->met_sy <= 1)
    return;

  /* Allocate... */
  ALLOC(help, met_t, 1);

  /* Copy data... */
  help->nx = met->nx;
  help->ny = met->ny;
  help->np = met->np;
  memcpy(help->lon, met->lon, sizeof(met->lon));
  memcpy(help->lat, met->lat, sizeof(met->lat));
  memcpy(help->p, met->p, sizeof(met->p));

  /* Smoothing... */
  for (ix = 0; ix < met->nx; ix += ctl2->met_dx) {
    for (iy = 0; iy < met->ny; iy += ctl2->met_dy) {
      for (ip = 0; ip < met->np; ip += ctl2->met_dp) {
        help->ps[ix][iy] = 0;
        help->pt[ix][iy] = 0;
        help->z[ix][iy][ip] = 0;
        help->t[ix][iy][ip] = 0;
        help->u[ix][iy][ip] = 0;
        help->v[ix][iy][ip] = 0;
        help->w[ix][iy][ip] = 0;
        help->pv[ix][iy][ip] = 0;
        help->h2o[ix][iy][ip] = 0;
        help->o3[ix][iy][ip] = 0;
        wsum = 0;
        for (ix2 = ix - ctl2->met_sx + 1; ix2 <= ix + ctl2->met_sx - 1; ix2++) {
          ix3 = ix2;
          if (ix3 < 0)
            ix3 += met->nx;
          else if (ix3 >= met->nx)
            ix3 -= met->nx;

          for (iy2 = GSL_MAX(iy - ctl2->met_sy + 1, 0);
               iy2 <= GSL_MIN(iy + ctl2->met_sy - 1, met->ny - 1); iy2++)
            for (ip2 = GSL_MAX(ip - ctl2->met_sp + 1, 0);
                 ip2 <= GSL_MIN(ip + ctl2->met_sp - 1, met->np - 1); ip2++) {
              w = (1.0f - (float)abs(ix - ix2) /  (float)ctl2->met_sx)
                * (1.0f - (float)abs(iy - iy2) /  (float)ctl2->met_sy)
                * (1.0f - (float)abs(ip - ip2) /  (float)ctl2->met_sp);
              help->ps[ix][iy] += w * met->ps[ix3][iy2];
              help->pt[ix][iy] += w * met->pt[ix3][iy2];
              help->z[ix][iy][ip] += w * met->z[ix3][iy2][ip2];
              help->t[ix][iy][ip] += w * met->t[ix3][iy2][ip2];
              help->u[ix][iy][ip] += w * met->u[ix3][iy2][ip2];
              help->v[ix][iy][ip] += w * met->v[ix3][iy2][ip2];
              help->w[ix][iy][ip] += w * met->w[ix3][iy2][ip2];
              help->pv[ix][iy][ip] += w * met->pv[ix3][iy2][ip2];
              help->h2o[ix][iy][ip] += w * met->h2o[ix3][iy2][ip2];
              help->o3[ix][iy][ip] += w * met->o3[ix3][iy2][ip2];
              wsum += w;
            }
        }
        help->ps[ix][iy] /= wsum;
        help->pt[ix][iy] /= wsum;
        help->t[ix][iy][ip] /= wsum;
        help->z[ix][iy][ip] /= wsum;
        help->u[ix][iy][ip] /= wsum;
        help->v[ix][iy][ip] /= wsum;
        help->w[ix][iy][ip] /= wsum;
```

```
        help->pv[ix][iy][ip] /= wsum;
        help->h2o[ix][iy][ip] /= wsum;
        help->o3[ix][iy][ip] /= wsum;
      }
    }
  }

  /* Downsampling... */
  met->nx = 0;
  for (ix = 0; ix < help->nx; ix += ctl2->met_dx) {
    met->lon[met->nx] = help->lon[ix];
    met->ny = 0;
    for (iy = 0; iy < help->ny; iy += ctl2->met_dy) {
      met->lat[met->ny] = help->lat[iy];
      met->ps[met->nx][met->ny] = help->ps[ix][iy];
      met->pt[met->nx][met->ny] = help->pt[ix][iy];
      met->np = 0;
      for (ip = 0; ip < help->np; ip += ctl2->met_dp) {
        met->p[met->np] = help->p[ip];
        met->z[met->nx][met->ny][met->np] = help->z[ix][iy][ip];
        met->t[met->nx][met->ny][met->np] = help->t[ix][iy][ip];
        met->u[met->nx][met->ny][met->np] = help->u[ix][iy][ip];
        met->v[met->nx][met->ny][met->np] = help->v[ix][iy][ip];
        met->w[met->nx][met->ny][met->np] = help->w[ix][iy][ip];
        met->pv[met->nx][met->ny][met->np] = help->pv[ix][iy][ip];
        met->h2o[met->nx][met->ny][met->np] = help->h2o[ix][iy][ip];
        met->o3[met->nx][met->ny][met->np] = help->o3[ix][iy][ip];
        met->np++;
      }
      met->ny++;
    }
    met->nx++;
  }

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

main()

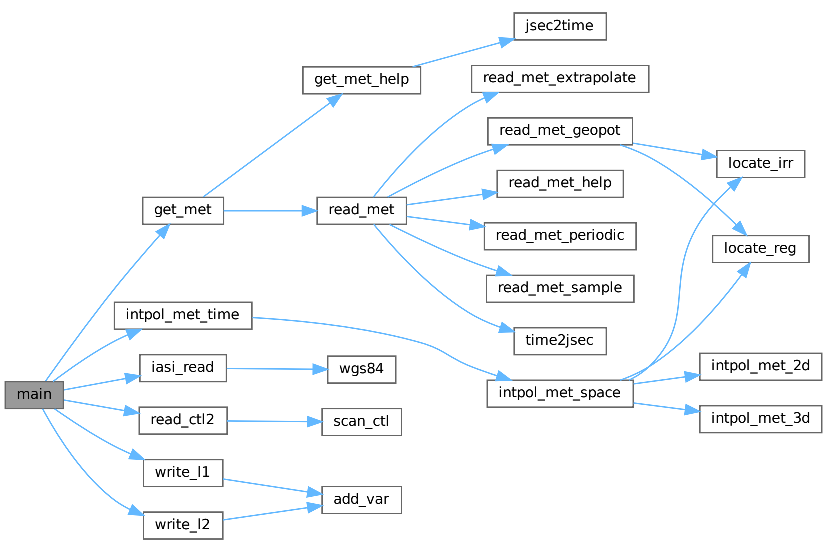
```
int main (
        int argc,
        char * argv[] )
```

Definition at line 238 of file extract.c.

```
                  {

  static iasi_rad_t *iasi_rad;

  static iasi_l1_t l1;
  static iasi_l2_t l2;

  static ctl2_t ctl2;

  met_t *met0, *met1;

  double ts;

  int ichan, lay, track = 0, xtrack;

  /* Check arguments... */
  if (argc < 4)
    ERRMSG("Give parameters: <ctl> <iasi_l1_file> <metbase> <out.nc>");

  /* Allocate... */
  ALLOC(iasi_rad, iasi_rad_t, 1);
  ALLOC(met0, met_t, 1);
  ALLOC(met1, met_t, 1);

  /* Read control parameters... */
  read_ctl2(argc, argv, &ctl2);

  /* Read IASI data... */
  iasi_read(argv[2], iasi_rad);

  /* Copy data to struct... */
  l1.ntrack = (size_t) iasi_rad->ntrack;
  for (track = 0; track < iasi_rad->ntrack; track++)
    for (xtrack = 0; xtrack < L1_NXTRACK; xtrack++) {
      l1.time[track][xtrack]
        = iasi_rad->Time[track][xtrack];
```

```
      l1.lon[track][xtrack]
        = iasi_rad->Longitude[track][xtrack];
      l1.lat[track][xtrack]
        = iasi_rad->Latitude[track][xtrack];
      l1.sat_z[track]
        = iasi_rad->Sat_z[track];
      l1.sat_lon[track]
        = iasi_rad->Sat_lon[track];
      l1.sat_lat[track]
        = iasi_rad->Sat_lat[track];
      for (ichan = 0; ichan < L1_NCHAN; ichan++) {
        l1.nu[ichan]
          = iasi_rad->freq[iasi_chan[ichan]];
        l1.rad[track][xtrack][ichan]
          = iasi_rad->Rad[track][xtrack][iasi_chan[ichan]];
      }
    }

  /* Write netCDF file... */
  write_l1(argv[4], &l1);

  /* Read meteo data... */
  l2.ntrack = l1.ntrack;
  ts = l1.time[l2.ntrack / 2][L1_NXTRACK / 2];
  get_met(&ctl2, argv[3], ts, met0, met1);

  /* Interpolate meteo data... */
  for (track = 0; track < (int) l2.ntrack; track++)
    for (xtrack = 0; xtrack < L1_NXTRACK; xtrack++)
      for (lay = 0; lay < L2_NLAY; lay++) {
        l2.time[track][xtrack] = l1.time[track][xtrack];
        l2.lon[track][xtrack] = l1.lon[track][xtrack];
        l2.lat[track][xtrack] = l1.lat[track][xtrack];
        l2.p[lay] = 1013.25 * exp(-2.5 * lay / 7.0);
        intpol_met_time(met0, met1, ts, l2.p[lay],
                        l2.lon[track][xtrack], l2.lat[track][xtrack],
                        NULL, NULL, &l2.z[track][xtrack][lay],
                        &l2.t[track][xtrack][lay],
                        NULL, NULL, NULL, NULL, NULL, NULL);
      }

  /* Write netCDF file... */
  write_l2(argv[4], &l2);

  /* Free... */
  free(iasi_rad);
  free(met0);
  free(met1);

  return EXIT_SUCCESS;
}
```

Here is the call graph for this function:



5.3.2 Variable Documentation

iasi_chan

```
int iasi_chan[L1_NCHAN]
```

Initial value:

```
= { 71, 88, 89, 90, 91, 92, 93, 94, 95, 96,
  97, 98, 99, 6712, 6720, 6735, 6742, 6749, 6750,
  6756, 6757, 6763, 6764, 6770, 6771, 6777, 6778,
  6784, 6791, 6797, 6838, 6855, 6866
}
```

List of IASI channels (don't change).

Definition at line 21 of file extract.c.

```
00021                       { 71, 88, 89, 90, 91, 92, 93, 94, 95, 96,
00022   97, 98, 99, 6712, 6720, 6735, 6742, 6749, 6750,
00023   6756, 6757, 6763, 6764, 6770, 6771, 6777, 6778,
00024   6784, 6791, 6797, 6838, 6855, 6866
00025 };
```

5.4 extract.c

Go to the documentation of this file.

```
00001 #include "libiasi.h"
00002
00003 /* ------------------------------------------------------------
00004    Dimensions...
00005    ------------------------------------------------------------ */
00006
00008 #define EP 72
00009
00011 #define EX 362
00012
00014 #define EY 182
00015
00016 /* ------------------------------------------------------------
00017    Global variables...
00018    ------------------------------------------------------------ */
00019
00021 int iasi_chan[L1_NCHAN] = { 71, 88, 89, 90, 91, 92, 93, 94, 95, 96,
00022   97, 98, 99, 6712, 6720, 6735, 6742, 6749, 6750,
00023   6756, 6757, 6763, 6764, 6770, 6771, 6777, 6778,
00024   6784, 6791, 6797, 6838, 6855, 6866
00025 };
00026
00027 /* ------------------------------------------------------------
00028    Structs...
00029    ------------------------------------------------------------ */
00030
00032 typedef struct {
00033
00035   double dt_met;
00036
00038   char met_geopot[LEN];
00039
00041   int met_dx;
00042
00044   int met_dy;
00045
00047   int met_dp;
00048
00050   int met_sx;
00051
00053   int met_sy;
00054
00056   int met_sp;
00057
00058 } ctl2_t;
00059
00061 typedef struct {
00062
00064   double time;
00065
00067   int nx;
```

```
  int ny;

  int np;

  double lon[EX];

  double lat[EY];

  double p[EP];

  double ps[EX][EY];

  double pt[EX][EY];

  float z[EX][EY][EP];

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

  float u[EX][EY][EP];

  float v[EX][EY][EP];

  float w[EX][EY][EP];

  float pv[EX][EY][EP];

  float h2o[EX][EY][EP];

  float o3[EX][EY][EP];

  float pl[EX][EY][EP];

} met_t;

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

void get_met(
  ctl2_t * ctl2,
  char *metbase,
  double t,
  met_t * met0,
  met_t * met1);

void get_met_help(
  double t,
  int direct,
  char *metbase,
  double dt_met,
  char *filename);

void intpol_met_2d(
  double array[EX][EY],
  int ix,
  int iy,
  double wx,
  double wy,
  double *var);

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

void intpol_met_space(
  met_t * met,
  double p,
  double lon,
  double lat,
  double *ps,
  double *pt,
  double *z,
  double *t,
  double *u,
  double *v,
  double *w,
  double *pv,
  double *h2o,
  double *o3);

```

```
void intpol_met_time(
  met_t * met0,
  met_t * met1,
  double ts,
  double p,
  double lon,
  double lat,
  double *ps,
  double *pt,
  double *z,
  double *t,
  double *u,
  double *v,
  double *w,
  double *pv,
  double *h2o,
  double *o3);

void read_ctl2(
  int argc,
  char *argv[],
  ctl2_t * ctl2);

void read_met(
  ctl2_t * ctl2,
  char *filename,
  met_t * met);

void read_met_extrapolate(
  met_t * met);

void read_met_geopot(
  ctl2_t * ctl2,
  met_t * met);

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

void read_met_periodic(
  met_t * met);

void read_met_sample(
  ctl2_t * ctl2,
  met_t * met);

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

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

  static iasi_rad_t *iasi_rad;

  static iasi_l1_t l1;
  static iasi_l2_t l2;

  static ctl2_t ctl2;

  met_t *met0, *met1;

  double ts;

  int ichan, lay, track = 0, xtrack;

  /* Check arguments... */
  if (argc < 4)
    ERRMSG("Give parameters: <ctl> <iasi_l1_file> <metbase> <out.nc>");

  /* Allocate... */
  ALLOC(iasi_rad, iasi_rad_t, 1);
  ALLOC(met0, met_t, 1);
  ALLOC(met1, met_t, 1);

  /* Read control parameters... */
  read_ctl2(argc, argv, &ctl2);

  /* Read IASI data... */
  iasi_read(argv[2], iasi_rad);

  /* Copy data to struct... */
```

```
  l1.ntrack = (size_t) iasi_rad->ntrack;
  for (track = 0; track < iasi_rad->ntrack; track++)
    for (xtrack = 0; xtrack < L1_NXTRACK; xtrack++) {
      l1.time[track][xtrack]
        = iasi_rad->Time[track][xtrack];
      l1.lon[track][xtrack]
        = iasi_rad->Longitude[track][xtrack];
      l1.lat[track][xtrack]
        = iasi_rad->Latitude[track][xtrack];
      l1.sat_z[track]
        = iasi_rad->Sat_z[track];
      l1.sat_lon[track]
        = iasi_rad->Sat_lon[track];
      l1.sat_lat[track]
        = iasi_rad->Sat_lat[track];
      for (ichan = 0; ichan < L1_NCHAN; ichan++) {
        l1.nu[ichan]
          = iasi_rad->freq[iasi_chan[ichan]];
        l1.rad[track][xtrack][ichan]
          = iasi_rad->Rad[track][xtrack][iasi_chan[ichan]];
      }
    }

  /* Write netCDF file... */
  write_l1(argv[4], &l1);

  /* Read meteo data... */
  l2.ntrack = l1.ntrack;
  ts = l1.time[l2.ntrack / 2][L1_NXTRACK / 2];
  get_met(&ctl2, argv[3], ts, met0, met1);

  /* Interpolate meteo data... */
  for (track = 0; track < (int) l2.ntrack; track++)
    for (xtrack = 0; xtrack < L1_NXTRACK; xtrack++)
      for (lay = 0; lay < L2_NLAY; lay++) {
        l2.time[track][xtrack] = l1.time[track][xtrack];
        l2.lon[track][xtrack] = l1.lon[track][xtrack];
        l2.lat[track][xtrack] = l1.lat[track][xtrack];
        l2.p[lay] = 1013.25 * exp(-2.5 * lay / 7.0);
        intpol_met_time(met0, met1, ts, l2.p[lay],
                        l2.lon[track][xtrack], l2.lat[track][xtrack],
                        NULL, NULL, &l2.z[track][xtrack][lay],
                        &l2.t[track][xtrack][lay],
                        NULL, NULL, NULL, NULL, NULL, NULL);
      }

  /* Write netCDF file... */
  write_l2(argv[4], &l2);

  /* Free... */
  free(iasi_rad);
  free(met0);
  free(met1);

  return EXIT_SUCCESS;
}

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

void get_met(
  ctl2_t * ctl2,
  char *metbase,
  double t,
  met_t * met0,
  met_t * met1) {

  char filename[LEN];

  static int init;

  /* Init... */
  if (!init) {
    init = 1;

    get_met_help(t, -1, metbase, ctl2->dt_met, filename);
    read_met(ctl2, filename, met0);

    get_met_help(t + 1.0, 1, metbase, ctl2->dt_met, filename);
    read_met(ctl2, filename, met1);
  }

  /* Read new data for forward trajectories... */
  if (t > met1->time) {
    memcpy(met0, met1, sizeof(met_t));
    get_met_help(t, 1, metbase, ctl2->dt_met, filename);
    read_met(ctl2, filename, met1);
  }
```

```
}

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

void get_met_help(
  double t,
  int direct,
  char *metbase,
  double dt_met,
  char *filename) {

  double t6, r;

  int year, mon, day, hour, min, sec;

  /* Round time to fixed intervals... */
  if (direct == -1)
    t6 = floor(t / dt_met) * dt_met;
  else
    t6 = ceil(t / dt_met) * dt_met;

  /* Decode time... */
  jsec2time(t6, &year, &mon, &day, &hour, &min, &sec, &r);

  /* Set filename... */
  sprintf(filename, "%s_%d_%02d_%02d_%02d.nc", metbase, year, mon, day, hour);
}

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

void intpol_met_2d(
  double array[EX][EY],
  int ix,
  int iy,
  double wx,
  double wy,
  double *var) {

  double aux00, aux01, aux10, aux11;

  /* Set variables... */
  aux00 = array[ix][iy];
  aux01 = array[ix][iy + 1];
  aux10 = array[ix + 1][iy];
  aux11 = array[ix + 1][iy + 1];

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

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

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

  double aux00, aux01, aux10, aux11;

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

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

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

void intpol_met_space(
  met_t * met,
  double p,
```

```
  double lon,
  double lat,
  double *ps,
  double *pt,
  double *z,
  double *t,
  double *u,
  double *v,
  double *w,
  double *pv,
  double *h2o,
  double *o3) {

  double wp, wx, wy;

  int ip, ix, iy;

  /* Check longitude... */
  if (met->lon[met->nx - 1] > 180 && lon < 0)
    lon += 360;

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

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

  /* Interpolate... */
  if (ps != NULL)
    intpol_met_2d(met->ps, ix, iy, wx, wy, ps);
  if (pt != NULL)
    intpol_met_2d(met->pt, ix, iy, wx, wy, pt);
  if (z != NULL)
    intpol_met_3d(met->z, ip, ix, iy, wp, wx, wy, z);
  if (t != NULL)
    intpol_met_3d(met->t, ip, ix, iy, wp, wx, wy, t);
  if (u != NULL)
    intpol_met_3d(met->u, ip, ix, iy, wp, wx, wy, u);
  if (v != NULL)
    intpol_met_3d(met->v, ip, ix, iy, wp, wx, wy, v);
  if (w != NULL)
    intpol_met_3d(met->w, ip, ix, iy, wp, wx, wy, w);
  if (pv != NULL)
    intpol_met_3d(met->pv, ip, ix, iy, wp, wx, wy, pv);
  if (h2o != NULL)
    intpol_met_3d(met->h2o, ip, ix, iy, wp, wx, wy, h2o);
  if (o3 != NULL)
    intpol_met_3d(met->o3, ip, ix, iy, wp, wx, wy, o3);
}

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

void intpol_met_time(
  met_t * met0,
  met_t * met1,
  double ts,
  double p,
  double lon,
  double lat,
  double *ps,
  double *pt,
  double *z,
  double *t,
  double *u,
  double *v,
  double *w,
  double *pv,
  double *h2o,
  double *o3) {

  double h2o0, h2o1, o30, o31, ps0, ps1, pt0, pt1, pv0, pv1, t0, t1, u0, u1,
    v0, v1, w0, w1, wt, z0, z1;

  /* Spatial interpolation... */
  intpol_met_space(met0, p, lon, lat,
                   ps == NULL ? NULL : &ps0,
                   pt == NULL ? NULL : &pt0,
                   z == NULL ? NULL : &z0,
                   t == NULL ? NULL : &t0,
                   u == NULL ? NULL : &u0,
                   v == NULL ? NULL : &v0,
                   w == NULL ? NULL : &w0,
                   pv == NULL ? NULL : &pv0,
```

```
                    h2o == NULL ? NULL : &h2o0, o3 == NULL ? NULL : &o30);
  intpol_met_space(met1, p, lon, lat,
                    ps == NULL ? NULL : &ps1,
                    pt == NULL ? NULL : &pt1,
                    z == NULL ? NULL : &z1,
                    t == NULL ? NULL : &t1,
                    u == NULL ? NULL : &u1,
                    v == NULL ? NULL : &v1,
                    w == NULL ? NULL : &w1,
                    pv == NULL ? NULL : &pv1,
                    h2o == NULL ? NULL : &h2o1, o3 == NULL ? NULL : &o31);

  /* Get weighting factor... */
  wt = (met1->time - ts) / (met1->time - met0->time);

  /* Interpolate... */
  if (ps != NULL)
    *ps = wt * (ps0 - ps1) + ps1;
  if (pt != NULL)
    *pt = wt * (pt0 - pt1) + pt1;
  if (z != NULL)
    *z = wt * (z0 - z1) + z1;
  if (t != NULL)
    *t = wt * (t0 - t1) + t1;
  if (u != NULL)
    *u = wt * (u0 - u1) + u1;
  if (v != NULL)
    *v = wt * (v0 - v1) + v1;
  if (w != NULL)
    *w = wt * (w0 - w1) + w1;
  if (pv != NULL)
    *pv = wt * (pv0 - pv1) + pv1;
  if (h2o != NULL)
    *h2o = wt * (h2o0 - h2o1) + h2o1;
  if (o3 != NULL)
    *o3 = wt * (o30 - o31) + o31;
}

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

void read_ctl2(
  int argc,
  char *argv[],
  ctl2_t * ctl2) {

  /* Meteorological data... */
  ctl2->dt_met = scan_ctl(argc, argv, "DT_MET", -1, "21600", NULL);
  scan_ctl(argc, argv, "MET_GEOPOT", -1, "", ctl2->met_geopot);
  ctl2->met_dx = (int) scan_ctl(argc, argv, "MET_DX", -1, "1", NULL);
  ctl2->met_dy = (int) scan_ctl(argc, argv, "MET_DY", -1, "1", NULL);
  ctl2->met_dp = (int) scan_ctl(argc, argv, "MET_DP", -1, "1", NULL);
  ctl2->met_sx = (int) scan_ctl(argc, argv, "MET_SX", -1, "20", NULL);
  ctl2->met_sy = (int) scan_ctl(argc, argv, "MET_SY", -1, "10", NULL);
  ctl2->met_sp = (int) scan_ctl(argc, argv, "MET_SP", -1, "1", NULL);
}

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

void read_met(
  ctl2_t * ctl2,
  char *filename,
  met_t * met) {

  char levname[LEN], tstr[10];

  static float help[EX * EY];

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

  size_t np, nx, ny;

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

  /* Get time from filename... */
  sprintf(tstr, "%.4s", &filename[strlen(filename) - 16]);
  year = atoi(tstr);
  sprintf(tstr, "%.2s", &filename[strlen(filename) - 11]);
  mon = atoi(tstr);
  sprintf(tstr, "%.2s", &filename[strlen(filename) - 8]);
  day = atoi(tstr);
  sprintf(tstr, "%.2s", &filename[strlen(filename) - 5]);
  hour = atoi(tstr);
  time2jsec(year, mon, day, hour, 0, 0, &met->time);

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

```
  /* Get dimensions... */
  NC(nc_inq_dimid(ncid, "lon", &dimid));
  NC(nc_inq_dimlen(ncid, dimid, &nx));
  if (nx < 2 || nx > EX)
    ERRMSG("Number of longitudes out of range!");

  NC(nc_inq_dimid(ncid, "lat", &dimid));
  NC(nc_inq_dimlen(ncid, dimid, &ny));
  if (ny < 2 || ny > EY)
    ERRMSG("Number of latitudes out of range!");

  sprintf(levname, "lev");
  NC(nc_inq_dimid(ncid, levname, &dimid));
  NC(nc_inq_dimlen(ncid, dimid, &np));
  if (np == 1) {
    sprintf(levname, "lev_2");
    NC(nc_inq_dimid(ncid, levname, &dimid));
    NC(nc_inq_dimlen(ncid, dimid, &np));
  }
  if (np < 2 || np > EP)
    ERRMSG("Number of levels out of range!");

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

  /* Get horizontal grid... */
  NC(nc_inq_varid(ncid, "lon", &varid));
  NC(nc_get_var_double(ncid, varid, met->lon));
  NC(nc_inq_varid(ncid, "lat", &varid));
  NC(nc_get_var_double(ncid, varid, met->lat));

  /* Read meteorological data... */
  read_met_help(ncid, "t", "T", met, met->t, 1.0);
  read_met_help(ncid, "u", "U", met, met->u, 1.0);
  read_met_help(ncid, "v", "V", met, met->v, 1.0);
  read_met_help(ncid, "w", "W", met, met->w, 0.01f);
  read_met_help(ncid, "q", "Q", met, met->h2o, 1.608f);
  read_met_help(ncid, "o3", "O3", met, met->o3, 0.602f);

  /* Read pressure levels from file... */
  NC(nc_inq_varid(ncid, levname, &varid));
  NC(nc_get_var_double(ncid, varid, met->p));
  for (ip = 0; ip < met->np; ip++)
    met->p[ip] /= 100.;

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

  /* Check ordering of pressure levels... */
  for (ip = 1; ip < met->np; ip++)
    if (met->p[ip - 1] < met->p[ip])
      ERRMSG("Pressure levels must be descending!");

  /* Read surface pressure... */
  if (nc_inq_varid(ncid, "ps", &varid) == NC_NOERR
      || nc_inq_varid(ncid, "PS", &varid) == NC_NOERR) {
    NC(nc_get_var_float(ncid, varid, help));
    for (iy = 0; iy < met->ny; iy++)
      for (ix = 0; ix < met->nx; ix++)
        met->ps[ix][iy] = help[iy * met->nx + ix] / 100.;
  } else if (nc_inq_varid(ncid, "lnsp", &varid) == NC_NOERR
             || nc_inq_varid(ncid, "LNSP", &varid) == NC_NOERR) {
    NC(nc_get_var_float(ncid, varid, help));
    for (iy = 0; iy < met->ny; iy++)
      for (ix = 0; ix < met->nx; ix++)
        met->ps[ix][iy] = exp(help[iy * met->nx + ix]) / 100.;
  } else
    for (ix = 0; ix < met->nx; ix++)
      for (iy = 0; iy < met->ny; iy++)
        met->ps[ix][iy] = met->p[0];

  /* Create periodic boundary conditions... */
  read_met_periodic(met);

  /* Calculate geopotential heights... */
  read_met_geopot(ctl2, met);

  /* Downsampling... */
  read_met_sample(ctl2, met);

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

```
/*****************************************************************************/

void read_met_extrapolate(
  met_t * met) {

  int ip, ip0, ix, iy;

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

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

      /* Extrapolate... */
      for (ip = ip0; ip >= 0; ip--) {
        met->t[ix][iy][ip] = met->t[ix][iy][ip + 1];
        met->u[ix][iy][ip] = met->u[ix][iy][ip + 1];
        met->v[ix][iy][ip] = met->v[ix][iy][ip + 1];
        met->w[ix][iy][ip] = met->w[ix][iy][ip + 1];
        met->h2o[ix][iy][ip] = met->h2o[ix][iy][ip + 1];
        met->o3[ix][iy][ip] = met->o3[ix][iy][ip + 1];
      }
    }
}

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

void read_met_geopot(
  ctl2_t * ctl2,
  met_t * met) {

  static double topo_lat[EY], topo_lon[EX], topo_z[EX][EY];

  static int init, topo_nx = -1, topo_ny;

  FILE *in;

  char line[LEN];

  double data[30], lat, lon, rlat, rlon, rlon_old = -999, rz, ts, z0, z1;

  float help[EX][EY];

  int ip, ip0, ix, ix2, ix3, iy, iy2, n, tx, ty;

  /* Initialize geopotential heights... */
  for (ix = 0; ix < met->nx; ix++)
    for (iy = 0; iy < met->ny; iy++)
      for (ip = 0; ip < met->np; ip++)
        met->z[ix][iy][ip] = GSL_NAN;

  /* Check filename... */
  if (ctl2->met_geopot[0] == '-')
    return;

  /* Read surface geopotential... */
  if (!init) {

    /* Write info... */
    printf("Read surface geopotential: %s\n", ctl2->met_geopot);

    /* Open file... */
    if (!(in = fopen(ctl2->met_geopot, "r")))
      ERRMSG("Cannot open file!");

    /* Read data... */
    while (fgets(line, LEN, in))
      if (sscanf(line, "%lg %lg %lg", &rlon, &rlat, &rz) == 3) {
        if (rlon != rlon_old) {
          if ((++topo_nx) >= EX)
            ERRMSG("Too many longitudes!");
          topo_ny = 0;
        }
        rlon_old = rlon;
        topo_lon[topo_nx] = rlon;
        topo_lat[topo_ny] = rlat;
        topo_z[topo_nx][topo_ny] = rz;
        if ((++topo_ny) >= EY)
          ERRMSG("Too many latitudes!");
      }
    if ((++topo_nx) >= EX)
```

```
      ERRMSG("Too many longitudes!");

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

    /* Check grid spacing... */
    if (fabs(met->lon[0] - met->lon[1]) != fabs(topo_lon[0] - topo_lon[1])
        || fabs(met->lat[0] - met->lat[1]) != fabs(topo_lat[0] - topo_lat[1]))
      printf("Warning: Grid spacing does not match!\n");

    /* Set init flag... */
    init = 1;
  }

  /* Apply hydrostatic equation to calculate geopotential heights... */
  for (ix = 0; ix < met->nx; ix++)
    for (iy = 0; iy < met->ny; iy++) {

      /* Get surface height... */
      lon = met->lon[ix];
      if (lon < topo_lon[0])
        lon += 360;
      else if (lon > topo_lon[topo_nx - 1])
        lon -= 360;
      lat = met->lat[iy];
      tx = locate_reg(topo_lon, topo_nx, lon);
      ty = locate_reg(topo_lat, topo_ny, lat);
      z0 = LIN(topo_lon[tx], topo_z[tx][ty],
               topo_lon[tx + 1], topo_z[tx + 1][ty], lon);
      z1 = LIN(topo_lon[tx], topo_z[tx][ty + 1],
               topo_lon[tx + 1], topo_z[tx + 1][ty + 1], lon);
      z0 = LIN(topo_lat[ty], z0, topo_lat[ty + 1], z1, lat);

      /* Find surface pressure level... */
      ip0 = locate_irr(met->p, met->np, met->ps[ix][iy]);

      /* Get surface temperature... */
      ts = LIN(met->p[ip0], met->t[ix][iy][ip0],
               met->p[ip0 + 1], met->t[ix][iy][ip0 + 1], met->ps[ix][iy]);

      /* Upper part of profile... */
      met->z[ix][iy][ip0 + 1]
        = (float) (z0 + 8.31441 / 28.9647 / G0
                   * 0.5 * (ts + met->t[ix][iy][ip0 + 1])
                   * log(met->ps[ix][iy] / met->p[ip0 + 1]));
      for (ip = ip0 + 2; ip < met->np; ip++)
        met->z[ix][iy][ip]
          = (float) (met->z[ix][iy][ip - 1] + 8.31441 / 28.9647 / G0
                     * 0.5 * (met->t[ix][iy][ip - 1] + met->t[ix][iy][ip])
                     * log(met->p[ip - 1] / met->p[ip]));
    }

  /* Smooth fields... */
  for (ip = 0; ip < met->np; ip++) {

    /* Median filter... */
    for (ix = 0; ix < met->nx; ix++)
      for (iy = 0; iy < met->nx; iy++) {
        n = 0;
        for (ix2 = ix - 2; ix2 <= ix + 2; ix2++) {
          ix3 = ix2;
          if (ix3 < 0)
            ix3 += met->nx;
          if (ix3 >= met->nx)
            ix3 -= met->nx;
          for (iy2 = GSL_MAX(iy - 2, 0); iy2 <= GSL_MIN(iy + 2, met->ny - 1);
               iy2++)
            if (gsl_finite(met->z[ix3][iy2][ip])) {
              data[n] = met->z[ix3][iy2][ip];
              n++;
            }
        }
        if (n > 0) {
          gsl_sort(data, 1, (size_t) n);
          help[ix][iy] = (float)
            gsl_stats_median_from_sorted_data(data, 1, (size_t) n);
        } else
          help[ix][iy] = GSL_NAN;
      }

    /* Copy data... */
    for (ix = 0; ix < met->nx; ix++)
      for (iy = 0; iy < met->nx; iy++)
        met->z[ix][iy][ip] = help[ix][iy];
  }
}
```

```
/*****************************************************************************/

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

  static float help[EX * EY * EP];

  int ip, ix, iy, varid;

  /* Check if variable exists... */
  if (nc_inq_varid(ncid, varname, &varid) != NC_NOERR)
    if (nc_inq_varid(ncid, varname2, &varid) != NC_NOERR)
      return;

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

  /* Copy and check data... */
  for (ix = 0; ix < met->nx; ix++)
    for (iy = 0; iy < met->ny; iy++)
      for (ip = 0; ip < met->np; ip++) {
        dest[ix][iy][ip] = help[(ip * met->ny + iy) * met->nx + ix];
        if (fabsf(dest[ix][iy][ip]) < 1e14f)
          dest[ix][iy][ip] *= scl;
        else
          dest[ix][iy][ip] = GSL_NAN;
      }
}

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

void read_met_periodic(
  met_t * met) {

  int ip, iy;

  /* Check longitudes... */
  if (!(fabs(met->lon[met->nx - 1] - met->lon[0]
             + met->lon[1] - met->lon[0] - 360) < 0.01))
    return;

  /* Increase longitude counter... */
  if ((++met->nx) > EX)
    ERRMSG("Cannot create periodic boundary conditions!");

  /* Set longitude... */
  met->lon[met->nx - 1] = met->lon[met->nx - 2] + met->lon[1] - met->lon[0];

  /* Loop over latitudes and pressure levels... */
  for (iy = 0; iy < met->ny; iy++)
    for (ip = 0; ip < met->np; ip++) {
      met->ps[met->nx - 1][iy] = met->ps[0][iy];
      met->pt[met->nx - 1][iy] = met->pt[0][iy];
      met->z[met->nx - 1][iy][ip] = met->z[0][iy][ip];
      met->t[met->nx - 1][iy][ip] = met->t[0][iy][ip];
      met->u[met->nx - 1][iy][ip] = met->u[0][iy][ip];
      met->v[met->nx - 1][iy][ip] = met->v[0][iy][ip];
      met->w[met->nx - 1][iy][ip] = met->w[0][iy][ip];
      met->pv[met->nx - 1][iy][ip] = met->pv[0][iy][ip];
      met->h2o[met->nx - 1][iy][ip] = met->h2o[0][iy][ip];
      met->o3[met->nx - 1][iy][ip] = met->o3[0][iy][ip];
    }
}

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

void read_met_sample(
  ctl2_t * ctl2,
  met_t * met) {

  met_t *help;

  float w, wsum;

  int ip, ip2, ix, ix2, ix3, iy, iy2;

  /* Check parameters... */
  if (ctl2->met_dp <= 1 && ctl2->met_dx <= 1 && ctl2->met_dy <= 1
      && ctl2->met_sp <= 1 && ctl2->met_sx <= 1 && ctl2->met_sy <= 1)
    return;

  /* Allocate... */
```

```
  ALLOC(help, met_t, 1);

  /* Copy data... */
  help->nx = met->nx;
  help->ny = met->ny;
  help->np = met->np;
  memcpy(help->lon, met->lon, sizeof(met->lon));
  memcpy(help->lat, met->lat, sizeof(met->lat));
  memcpy(help->p, met->p, sizeof(met->p));

  /* Smoothing... */
  for (ix = 0; ix < met->nx; ix += ctl2->met_dx) {
    for (iy = 0; iy < met->ny; iy += ctl2->met_dy) {
      for (ip = 0; ip < met->np; ip += ctl2->met_dp) {
        help->ps[ix][iy] = 0;
        help->pt[ix][iy] = 0;
        help->z[ix][iy][ip] = 0;
        help->t[ix][iy][ip] = 0;
        help->u[ix][iy][ip] = 0;
        help->v[ix][iy][ip] = 0;
        help->w[ix][iy][ip] = 0;
        help->pv[ix][iy][ip] = 0;
        help->h2o[ix][iy][ip] = 0;
        help->o3[ix][iy][ip] = 0;
        wsum = 0;
        for (ix2 = ix - ctl2->met_sx + 1; ix2 <= ix + ctl2->met_sx - 1; ix2++) {
          ix3 = ix2;
          if (ix3 < 0)
            ix3 += met->nx;
          else if (ix3 >= met->nx)
            ix3 -= met->nx;

          for (iy2 = GSL_MAX(iy - ctl2->met_sy + 1, 0);
               iy2 <= GSL_MIN(iy + ctl2->met_sy - 1, met->ny - 1); iy2++)
            for (ip2 = GSL_MAX(ip - ctl2->met_sp + 1, 0);
                 ip2 <= GSL_MIN(ip + ctl2->met_sp - 1, met->np - 1); ip2++) {
              w = (1.0f - (float)abs(ix - ix2) /  (float)ctl2->met_sx)
                * (1.0f - (float)abs(iy - iy2) /  (float)ctl2->met_sy)
                * (1.0f - (float)abs(ip - ip2) /  (float)ctl2->met_sp);
              help->ps[ix][iy] += w * met->ps[ix3][iy2];
              help->pt[ix][iy] += w * met->pt[ix3][iy2];
              help->z[ix][iy][ip] += w * met->z[ix3][iy2][ip2];
              help->t[ix][iy][ip] += w * met->t[ix3][iy2][ip2];
              help->u[ix][iy][ip] += w * met->u[ix3][iy2][ip2];
              help->v[ix][iy][ip] += w * met->v[ix3][iy2][ip2];
              help->w[ix][iy][ip] += w * met->w[ix3][iy2][ip2];
              help->pv[ix][iy][ip] += w * met->pv[ix3][iy2][ip2];
              help->h2o[ix][iy][ip] += w * met->h2o[ix3][iy2][ip2];
              help->o3[ix][iy][ip] += w * met->o3[ix3][iy2][ip2];
              wsum += w;
            }
        }
        help->ps[ix][iy] /= wsum;
        help->pt[ix][iy] /= wsum;
        help->t[ix][iy][ip] /= wsum;
        help->z[ix][iy][ip] /= wsum;
        help->u[ix][iy][ip] /= wsum;
        help->v[ix][iy][ip] /= wsum;
        help->w[ix][iy][ip] /= wsum;
        help->pv[ix][iy][ip] /= wsum;
        help->h2o[ix][iy][ip] /= wsum;
        help->o3[ix][iy][ip] /= wsum;
      }
    }
  }

  /* Downsampling... */
  met->nx = 0;
  for (ix = 0; ix < help->nx; ix += ctl2->met_dx) {
    met->lon[met->nx] = help->lon[ix];
    met->ny = 0;
    for (iy = 0; iy < help->ny; iy += ctl2->met_dy) {
      met->lat[met->ny] = help->lat[iy];
      met->ps[met->nx][met->ny] = help->ps[ix][iy];
      met->pt[met->nx][met->ny] = help->pt[ix][iy];
      met->np = 0;
      for (ip = 0; ip < help->np; ip += ctl2->met_dp) {
        met->p[met->np] = help->p[ip];
        met->z[met->nx][met->ny][met->np] = help->z[ix][iy][ip];
        met->t[met->nx][met->ny][met->np] = help->t[ix][iy][ip];
        met->u[met->nx][met->ny][met->np] = help->u[ix][iy][ip];
        met->v[met->nx][met->ny][met->np] = help->v[ix][iy][ip];
        met->w[met->nx][met->ny][met->np] = help->w[ix][iy][ip];
        met->pv[met->nx][met->ny][met->np] = help->pv[ix][iy][ip];
        met->h2o[met->nx][met->ny][met->np] = help->h2o[ix][iy][ip];
        met->o3[met->nx][met->ny][met->np] = help->o3[ix][iy][ip];
        met->np++;
```

```
      }
      met->ny++;
    }
    met->nx++;
  }

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

5.5 jurassic.c File Reference

JURASSIC library definitions.

Functions

- size_t atm2x (const ctl_t *ctl, const atm_t *atm, gsl_vector *x, int *iqa, int *ipa)

  Compose state vector or parameter vector.
- void atm2x_help (const double value, const int value_iqa, const int value_ip, gsl_vector *x, int *iqa, int *ipa, size_t *n)

  Add element to state vector.
- void cart2geo (const double *x, double *z, double *lon, double *lat)

  Convert Cartesian coordinates to geolocation.
- void climatology (const ctl_t *ctl, atm_t *atm)

  Interpolate climatological data.
- double ctmco2 (const double nu, const double p, const double t, const double u)

  Compute carbon dioxide continuum (optical depth).
- double ctmh2o (const double nu, const double p, const double t, const double q, const double u)

  Compute water vapor continuum (optical depth).
- double ctmn2 (const double nu, const double p, const double t)

  Compute nitrogen continuum (absorption coefficient).
- double ctmo2 (const double nu, const double p, const double t)

  Compute oxygen continuum (absorption coefficient).
- void copy_atm (const ctl_t *ctl, atm_t *atm_dest, const atm_t *atm_src, const int init)

  Copy and initialize atmospheric data.
- void copy_obs (const ctl_t *ctl, obs_t *obs_dest, const obs_t *obs_src, const int init)

  Copy and initialize observation data.
- int find_emitter (const ctl_t *ctl, const char *emitter)

  Find index of an emitter.
- void formod (const ctl_t *ctl, atm_t *atm, obs_t *obs)

  Determine ray paths and compute radiative transfer.
- void formod_continua (const ctl_t *ctl, const los_t *los, const int ip, double *beta)

  Compute absorption coefficient of continua.
- void formod_fov (const ctl_t *ctl, obs_t *obs)

  Apply field of view convolution.
- void formod_pencil (const ctl_t *ctl, const atm_t *atm, obs_t *obs, const int ir)

  Compute radiative transfer for a pencil beam.
- void formod_rfm (const ctl_t *ctl, const atm_t *atm, obs_t *obs)

  Apply RFM for radiative transfer calculations.
- void formod_srcfunc (const ctl_t *ctl, const tbl_t *tbl, const double t, double *src)

  Compute Planck source function.
- void geo2cart (const double z, const double lon, const double lat, double *x)

Convert geolocation to Cartesian coordinates.
- void hydrostatic (const ctl_t *ctl, atm_t *atm)

  Set hydrostatic equilibrium.
- void idx2name (const ctl_t *ctl, const int idx, char *quantity)

  Determine name of state vector quantity for given index.
- void init_srcfunc (const ctl_t *ctl, tbl_t *tbl)

  Initialize source function table.
- void intpol_atm (const ctl_t *ctl, const atm_t *atm, const double z, double *p, double *t, double *q, double *k)

  Interpolate atmospheric data.
- void intpol_tbl_cga (const ctl_t *ctl, const tbl_t *tbl, const los_t *los, const int ip, double tau_path[ND][NG], double tau_seg[ND])

  Get transmittance from look-up tables (CGA method).
- void intpol_tbl_ega (const ctl_t *ctl, const tbl_t *tbl, const los_t *los, const int ip, double tau_path[ND][NG], double tau_seg[ND])

  Get transmittance from look-up tables (EGA method).
- double intpol_tbl_eps (const tbl_t *tbl, const int ig, const int id, const int ip, const int it, const double u)

  Interpolate emissivity from look-up tables.
- double intpol_tbl_u (const tbl_t *tbl, const int ig, const int id, const int ip, const int it, const double eps)

  Interpolate column density from look-up tables.
- void jsec2time (const 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 (const double *xx, const int n, const double x)

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

  Find array index for regular grid.
- int locate_tbl (const float *xx, const int n, const double x)

  Find array index in float array.
- size_t obs2y (const ctl_t *ctl, const obs_t *obs, gsl_vector *y, int *ida, int *ira)

  Compose measurement vector.
- void raytrace (const ctl_t *ctl, const atm_t *atm, obs_t *obs, los_t *los, const int ir)

  Do ray-tracing to determine LOS.
- void read_atm (const char *dirname, const char *filename, const 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, const ctl_t *ctl, obs_t *obs)

  Read observation data.
- double read_obs_rfm (const char *basename, const double z, double *nu, double *f, int n)

  Read observation data in RFM format.
- void read_rfm_spec (const char *filename, double *nu, double *rad, int *npts)

  Read RFM spectrum.
- void read_shape (const char *filename, double *x, double *y, int *n)

  Read shape function.
- void read_tbl (const ctl_t *ctl, tbl_t *tbl)

  Read look-up table data.
- double scan_ctl (int argc, char *argv[ ], const char *varname, int arridx, const char *defvalue, char *value)

Search control parameter file for variable entry.
- double sza (const double sec, const double lon, const double lat)

  Calculate solar zenith angle.
- void tangent_point (const los_t ∗los, double ∗tpz, double ∗tplon, double ∗tplat)

  Find tangent point of a given LOS.
- void time2jsec (const int year, const int mon, const int day, const int hour, const int min, const int sec, const 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, const ctl_t ∗ctl, const atm_t ∗atm)

  Write atmospheric data.
- void write_atm_rfm (const char ∗filename, const ctl_t ∗ctl, const atm_t ∗atm)

  Write atmospheric data in RFM format.
- void write_matrix (const char ∗dirname, const char ∗filename, const ctl_t ∗ctl, const gsl_matrix ∗matrix, const atm_t ∗atm, const obs_t ∗obs, const char ∗rowspace, const char ∗colspace, const char ∗sort)

  Write matrix.
- void write_obs (const char ∗dirname, const char ∗filename, const ctl_t ∗ctl, const obs_t ∗obs)

  Write observation data.
- void write_shape (const char ∗filename, const double ∗x, const double ∗y, const int n)

  Write shape function.
- void write_tbl (const ctl_t ∗ctl, const tbl_t ∗tbl)

  Write look-up table data.
- void x2atm (const ctl_t ∗ctl, const gsl_vector ∗x, atm_t ∗atm)

  Decompose parameter vector or state vector.
- void x2atm_help (double ∗value, const gsl_vector ∗x, size_t ∗n)

  Get element from state vector.
- void y2obs (const ctl_t ∗ctl, const gsl_vector ∗y, obs_t ∗obs)

  Decompose measurement vector.

5.5.1 Detailed Description

JURASSIC library definitions.

Definition in file jurassic.c.

5.5.2 Function Documentation

atm2x()

```
size_t atm2x (
            const ctl_t * ctl,
            const atm_t * atm,
            gsl_vector * x,
            int * iqa,
            int * ipa )
```

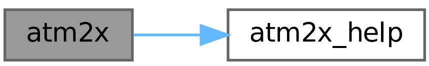
Compose state vector or parameter vector.

Definition at line 29 of file jurassic.c.

```
00034           {
```

```

  size_t n = 0;

  /* Add pressure... */
  for (int ip = 0; ip < atm->np; ip++)
    if (atm->z[ip] >= ctl->retp_zmin && atm->z[ip] <= ctl->retp_zmax)
      atm2x_help(atm->p[ip], IDXP, ip, x, iqa, ipa, &n);

  /* Add temperature... */
  for (int ip = 0; ip < atm->np; ip++)
    if (atm->z[ip] >= ctl->rett_zmin && atm->z[ip] <= ctl->rett_zmax)
      atm2x_help(atm->t[ip], IDXT, ip, x, iqa, ipa, &n);

  /* Add volume mixing ratios... */
  for (int ig = 0; ig < ctl->ng; ig++)
    for (int ip = 0; ip < atm->np; ip++)
      if (atm->z[ip] >= ctl->retq_zmin[ig]
          && atm->z[ip] <= ctl->retq_zmax[ig])
        atm2x_help(atm->q[ig][ip], IDXQ(ig), ip, x, iqa, ipa, &n);

  /* Add extinction... */
  for (int iw = 0; iw < ctl->nw; iw++)
    for (int ip = 0; ip < atm->np; ip++)
      if (atm->z[ip] >= ctl->retk_zmin[iw]
          && atm->z[ip] <= ctl->retk_zmax[iw])
        atm2x_help(atm->k[iw][ip], IDXK(iw), ip, x, iqa, ipa, &n);

  /* Add cloud parameters... */
  if (ctl->ret_clz)
    atm2x_help(atm->clz, IDXCLZ, 0, x, iqa, ipa, &n);
  if (ctl->ret_cldz)
    atm2x_help(atm->cldz, IDXCLDZ, 0, x, iqa, ipa, &n);
  if (ctl->ret_clk)
    for (int icl = 0; icl < ctl->ncl; icl++)
      atm2x_help(atm->clk[icl], IDXCLK(icl), 0, x, iqa, ipa, &n);

  /* Add surface parameters... */
  if (ctl->ret_sfz)
    atm2x_help(atm->sfz, IDXSFZ, 0, x, iqa, ipa, &n);
  if (ctl->ret_sfp)
    atm2x_help(atm->sfp, IDXSFP, 0, x, iqa, ipa, &n);
  if (ctl->ret_sft)
    atm2x_help(atm->sft, IDXSFT, 0, x, iqa, ipa, &n);
  if (ctl->ret_sfeps)
    for (int isf = 0; isf < ctl->nsf; isf++)
      atm2x_help(atm->sfeps[isf], IDXSFEPS(isf), 0, x, iqa, ipa, &n);

  return n;
}
```

Here is the call graph for this function:



atm2x_help()

```
void atm2x_help (
            const double value,
            const int value_iqa,
            const int value_ip,
            gsl_vector * x,
            int * iqa,
            int * ipa,
            size_t * n )
```

Add element to state vector.

Definition at line 87 of file jurassic.c.

```
                {

  /* Add element to state vector... */
  if (x != NULL)
    gsl_vector_set(x, *n, value);
  if (iqa != NULL)
    iqa[*n] = value_iqa;
  if (ipa != NULL)
    ipa[*n] = value_ip;
  (*n)++;
}
```

cart2geo()

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

Convert Cartesian coordinates to geolocation.

Definition at line 108 of file jurassic.c.

```
                  {

  const double radius = NORM(x);

  *lat = RAD2DEG(asin(x[2] / radius));
  *lon = RAD2DEG(atan2(x[1], x[0]));
  *z = radius - RE;
}
```

climatology()

```
void climatology (
            const ctl_t * ctl,
            atm_t * atm )
```

Interpolate climatological data.

Definition at line 123 of file jurassic.c.

```
                 {

  static double z[121] = {
    0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19,
    20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37,
    38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55,
    56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73,
    74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91,
    92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107,
    108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 120
  };

  static double pre[121] = {
    1017, 901.083, 796.45, 702.227, 617.614, 541.644, 473.437, 412.288,
    357.603, 308.96, 265.994, 228.348, 195.619, 167.351, 143.039, 122.198,
    104.369, 89.141, 76.1528, 65.0804, 55.641, 47.591, 40.7233, 34.8637,
    29.8633, 25.5956, 21.9534, 18.8445, 16.1909, 13.9258, 11.9913,
    10.34, 8.92988, 7.72454, 6.6924, 5.80701, 5.04654, 4.39238, 3.82902,
    3.34337, 2.92413, 2.56128, 2.2464, 1.97258, 1.73384, 1.52519, 1.34242,
    1.18197, 1.04086, 0.916546, 0.806832, 0.709875, 0.624101, 0.548176,
    0.480974, 0.421507, 0.368904, 0.322408, 0.281386, 0.245249, 0.213465,
    0.185549, 0.161072, 0.139644, 0.120913, 0.104568, 0.0903249, 0.0779269,
    0.0671493, 0.0577962, 0.0496902, 0.0426736, 0.0366093, 0.0313743,
    0.0268598, 0.0229699, 0.0196206, 0.0167399, 0.0142646, 0.0121397,
    0.0103181, 0.00875775, 0.00742226, 0.00628076, 0.00530519, 0.00447183,
    0.00376124, 0.00315632, 0.00264248, 0.00220738, 0.00184003, 0.00153095,
```

```
    0.00127204, 0.00105608, 0.000876652, 0.00072798, 0.00060492,
    0.000503201, 0.000419226, 0.000349896, 0.000292659, 0.000245421,
    0.000206394, 0.000174125, 0.000147441, 0.000125333, 0.000106985,
    9.173e-05, 7.90172e-05, 6.84172e-05, 5.95574e-05, 5.21183e-05,
    4.58348e-05, 4.05127e-05, 3.59987e-05, 3.21583e-05, 2.88718e-05,
    2.60322e-05, 2.35687e-05, 2.14263e-05, 1.95489e-05
  };

  static double tem[121] = {
    285.14, 279.34, 273.91, 268.3, 263.24, 256.55, 250.2, 242.82, 236.17,
    229.87, 225.04, 221.19, 218.85, 217.19, 216.2, 215.68, 215.42, 215.55,
    215.92, 216.4, 216.93, 217.45, 218, 218.68, 219.39, 220.25, 221.3,
    222.41, 223.88, 225.42, 227.2, 229.52, 231.89, 234.51, 236.85, 239.42,
    241.94, 244.57, 247.36, 250.32, 253.34, 255.82, 258.27, 260.39,
    262.03, 263.45, 264.2, 264.78, 264.67, 264.38, 263.24, 262.03, 260.02,
    258.09, 255.63, 253.28, 250.43, 247.81, 245.26, 242.77, 240.38,
    237.94, 235.79, 233.53, 231.5, 229.53, 227.6, 225.62, 223.77, 222.06,
    220.33, 218.69, 217.18, 215.64, 214.13, 212.52, 210.86, 209.25,
    207.49, 205.81, 204.11, 202.22, 200.32, 198.39, 195.92, 193.46,
    190.94, 188.31, 185.82, 183.57, 181.43, 179.74, 178.64, 178.1, 178.25,
    178.7, 179.41, 180.67, 182.31, 184.18, 186.6, 189.53, 192.66, 196.54,
    201.13, 205.93, 211.73, 217.86, 225, 233.53, 242.57, 252.14, 261.48,
    272.97, 285.26, 299.12, 312.2, 324.17, 338.34, 352.56, 365.28
  };

  static double c2h2[121] = {
    1.352e-09, 2.83e-10, 1.269e-10, 6.926e-11, 4.346e-11, 2.909e-11,
    2.014e-11, 1.363e-11, 8.71e-12, 5.237e-12, 2.718e-12, 1.375e-12,
    5.786e-13, 2.16e-13, 7.317e-14, 2.551e-14, 1.055e-14, 4.758e-15,
    2.056e-15, 7.703e-16, 2.82e-16, 1.035e-16, 4.382e-17, 1.946e-17,
    9.638e-18, 5.2e-18, 2.811e-18, 1.494e-18, 7.925e-19, 4.213e-19,
    1.998e-19, 8.78e-20, 3.877e-20, 1.728e-20, 7.743e-21, 3.536e-21,
    1.623e-21, 7.508e-22, 3.508e-22, 1.65e-22, 7.837e-23, 3.733e-23,
    1.808e-23, 8.77e-24, 4.285e-24, 2.095e-24, 1.032e-24, 5.082e-25,
    2.506e-25, 1.236e-25, 6.088e-26, 2.996e-26, 1.465e-26, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
  };

  static double c2h6[121] = {
    2.667e-09, 2.02e-09, 1.658e-09, 1.404e-09, 1.234e-09, 1.109e-09,
    1.012e-09, 9.262e-10, 8.472e-10, 7.71e-10, 6.932e-10, 6.216e-10,
    5.503e-10, 4.87e-10, 4.342e-10, 3.861e-10, 3.347e-10, 2.772e-10,
    2.209e-10, 1.672e-10, 1.197e-10, 8.536e-11, 5.783e-11, 3.846e-11,
    2.495e-11, 1.592e-11, 1.017e-11, 6.327e-12, 3.895e-12, 2.403e-12,
    1.416e-12, 8.101e-13, 4.649e-13, 2.686e-13, 1.557e-13, 9.14e-14,
    5.386e-14, 3.19e-14, 1.903e-14, 1.14e-14, 6.875e-15, 4.154e-15,
    2.538e-15, 1.553e-15, 9.548e-16, 5.872e-16, 3.63e-16, 2.244e-16,
    1.388e-16, 8.587e-17, 5.308e-17, 3.279e-17, 2.017e-17, 1.238e-17,
    7.542e-18, 4.585e-18, 2.776e-18, 1.671e-18, 9.985e-19, 5.937e-19,
    3.518e-19, 2.07e-19, 1.215e-19, 7.06e-20, 4.097e-20, 2.37e-20,
    1.363e-20, 7.802e-21, 4.441e-21, 2.523e-21, 1.424e-21, 8.015e-22,
    4.497e-22, 2.505e-22, 1.391e-22, 7.691e-23, 4.238e-23, 2.331e-23,
    1.274e-23, 6.929e-24, 3.752e-24, 2.02e-24, 1.083e-24, 5.774e-25,
    3.041e-25, 1.593e-25, 8.308e-26, 4.299e-26, 2.195e-26, 1.112e-26,
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0
  };

  static double ccl4[121] = {
    1.075e-10, 1.075e-10, 1.075e-10, 1.075e-10, 1.075e-10, 1.075e-10,
    1.075e-10, 1.075e-10, 1.075e-10, 1.06e-10, 1.024e-10, 9.69e-11,
    8.93e-11, 8.078e-11, 7.213e-11, 6.307e-11, 5.383e-11, 4.49e-11,
    3.609e-11, 2.705e-11, 1.935e-11, 1.385e-11, 8.35e-12, 5.485e-12,
    3.853e-12, 2.22e-12, 5.875e-13, 3.445e-13, 1.015e-13, 6.075e-14,
    4.383e-14, 2.692e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14,
    1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14,
    1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14,
    1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14,
    1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14,
    1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14,
    1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14,
    1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14,
    1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14,
    1e-14, 1e-14, 1e-14
  };

  static double ch4[121] = {
    1.864e-06, 1.835e-06, 1.819e-06, 1.805e-06, 1.796e-06, 1.788e-06,
    1.782e-06, 1.776e-06, 1.769e-06, 1.761e-06, 1.749e-06, 1.734e-06,
    1.716e-06, 1.692e-06, 1.654e-06, 1.61e-06, 1.567e-06, 1.502e-06,
    1.433e-06, 1.371e-06, 1.323e-06, 1.277e-06, 1.232e-06, 1.188e-06,
    1.147e-06, 1.108e-06, 1.07e-06, 1.027e-06, 9.854e-07, 9.416e-07,
    8.933e-07, 8.478e-07, 7.988e-07, 7.515e-07, 7.07e-07, 6.64e-07,
    6.239e-07, 5.864e-07, 5.512e-07, 5.184e-07, 4.87e-07, 4.571e-07,
    4.296e-07, 4.04e-07, 3.802e-07, 3.578e-07, 3.383e-07, 3.203e-07,
```

```
  3.032e-07, 2.889e-07, 2.76e-07, 2.635e-07, 2.519e-07, 2.409e-07,
  2.302e-07, 2.219e-07, 2.144e-07, 2.071e-07, 1.999e-07, 1.93e-07,
  1.862e-07, 1.795e-07, 1.731e-07, 1.668e-07, 1.607e-07, 1.548e-07,
  1.49e-07, 1.434e-07, 1.38e-07, 1.328e-07, 1.277e-07, 1.227e-07,
  1.18e-07, 1.134e-07, 1.089e-07, 1.046e-07, 1.004e-07, 9.635e-08,
  9.245e-08, 8.867e-08, 8.502e-08, 8.15e-08, 7.809e-08, 7.48e-08,
  7.159e-08, 6.849e-08, 6.55e-08, 6.262e-08, 5.98e-08, 5.708e-08,
  5.448e-08, 5.194e-08, 4.951e-08, 4.72e-08, 4.5e-08, 4.291e-08,
  4.093e-08, 3.905e-08, 3.729e-08, 3.563e-08, 3.408e-08, 3.265e-08,
  3.128e-08, 2.996e-08, 2.87e-08, 2.76e-08, 2.657e-08, 2.558e-08,
  2.467e-08, 2.385e-08, 2.307e-08, 2.234e-08, 2.168e-08, 2.108e-08,
  2.05e-08, 1.998e-08, 1.947e-08, 1.902e-08, 1.86e-08, 1.819e-08,
  1.782e-08
};

static double clo[121] = {
  7.419e-15, 1.061e-14, 1.518e-14, 2.195e-14, 3.175e-14, 4.666e-14,
  6.872e-14, 1.03e-13, 1.553e-13, 2.375e-13, 3.664e-13, 5.684e-13,
  8.915e-13, 1.402e-12, 2.269e-12, 4.125e-12, 7.501e-12, 1.257e-11,
  2.048e-11, 3.338e-11, 5.44e-11, 8.846e-11, 1.008e-10, 1.082e-10,
  1.157e-10, 1.232e-10, 1.312e-10, 1.539e-10, 1.822e-10, 2.118e-10,
  2.387e-10, 2.687e-10, 2.875e-10, 3.031e-10, 3.23e-10, 3.648e-10,
  4.117e-10, 4.477e-10, 4.633e-10, 4.794e-10, 4.95e-10, 5.104e-10,
  5.259e-10, 5.062e-10, 4.742e-10, 4.443e-10, 4.051e-10, 3.659e-10,
  3.305e-10, 2.911e-10, 2.54e-10, 2.215e-10, 1.927e-10, 1.675e-10,
  1.452e-10, 1.259e-10, 1.09e-10, 9.416e-11, 8.119e-11, 6.991e-11,
  6.015e-11, 5.163e-11, 4.43e-11, 3.789e-11, 3.24e-11, 2.769e-11,
  2.361e-11, 2.011e-11, 1.71e-11, 1.453e-11, 1.233e-11, 1.045e-11,
  8.851e-12, 7.48e-12, 6.316e-12, 5.326e-12, 4.487e-12, 3.778e-12,
  3.176e-12, 2.665e-12, 2.234e-12, 1.87e-12, 1.563e-12, 1.304e-12,
  1.085e-12, 9.007e-13, 7.468e-13, 6.179e-13, 5.092e-13, 4.188e-13,
  3.442e-13, 2.816e-13, 2.304e-13, 1.885e-13, 1.542e-13, 1.263e-13,
  1.035e-13, 8.5e-14, 7.004e-14, 5.783e-14, 4.795e-14, 4.007e-14,
  3.345e-14, 2.792e-14, 2.33e-14, 1.978e-14, 1.686e-14, 1.438e-14,
  1.234e-14, 1.07e-14, 9.312e-15, 8.131e-15, 7.164e-15, 6.367e-15,
  5.67e-15, 5.088e-15, 4.565e-15, 4.138e-15, 3.769e-15, 3.432e-15,
  3.148e-15
};

static double clono2[121] = {
  1.011e-13, 1.515e-13, 2.272e-13, 3.446e-13, 5.231e-13, 8.085e-13,
  1.253e-12, 1.979e-12, 3.149e-12, 5.092e-12, 8.312e-12, 1.366e-11,
  2.272e-11, 3.791e-11, 6.209e-11, 9.101e-11, 1.334e-10, 1.951e-10,
  2.853e-10, 3.94e-10, 4.771e-10, 5.771e-10, 6.675e-10, 7.665e-10,
  8.504e-10, 8.924e-10, 9.363e-10, 8.923e-10, 8.411e-10, 7.646e-10,
  6.525e-10, 5.576e-10, 4.398e-10, 3.403e-10, 2.612e-10, 1.915e-10,
  1.407e-10, 1.028e-10, 7.455e-11, 5.42e-11, 3.708e-11, 2.438e-11,
  1.618e-11, 1.075e-11, 7.17e-12, 4.784e-12, 3.205e-12, 2.147e-12,
  1.44e-12, 9.654e-13, 6.469e-13, 4.332e-13, 2.891e-13, 1.926e-13,
  1.274e-13, 8.422e-14, 5.547e-14, 3.636e-14, 2.368e-14, 1.536e-14,
  9.937e-15, 6.39e-15, 4.101e-15, 2.61e-15, 1.659e-15, 1.052e-15,
  6.638e-16, 4.172e-16, 2.61e-16, 1.63e-16, 1.013e-16, 6.275e-17,
  3.879e-17, 2.383e-17, 1.461e-17, 8.918e-18, 5.43e-18, 3.301e-18,
  1.997e-18, 1.203e-18, 7.216e-19, 4.311e-19, 2.564e-19, 1.519e-19,
  8.911e-20, 5.203e-20, 3.026e-20, 1.748e-20, 9.99e-21, 5.673e-21,
  3.215e-21, 1.799e-21, 1.006e-21, 5.628e-22, 3.146e-22, 1.766e-22,
  9.94e-23, 5.614e-23, 3.206e-23, 1.841e-23, 1.071e-23, 6.366e-24,
  3.776e-24, 2.238e-24, 1.326e-24, 8.253e-25, 5.201e-25, 3.279e-25,
  2.108e-25, 1.395e-25, 9.326e-26, 6.299e-26, 4.365e-26, 3.104e-26,
  2.219e-26, 1.621e-26, 1.185e-26, 8.92e-27, 6.804e-27, 5.191e-27,
  4.041e-27
};

static double co[121] = {
  1.907e-07, 1.553e-07, 1.362e-07, 1.216e-07, 1.114e-07, 1.036e-07,
  9.737e-08, 9.152e-08, 8.559e-08, 7.966e-08, 7.277e-08, 6.615e-08,
  5.884e-08, 5.22e-08, 4.699e-08, 4.284e-08, 3.776e-08, 3.274e-08,
  2.845e-08, 2.479e-08, 2.246e-08, 2.054e-08, 1.991e-08, 1.951e-08,
  1.94e-08, 2.009e-08, 2.1e-08, 2.201e-08, 2.322e-08, 2.45e-08,
  2.602e-08, 2.73e-08, 2.867e-08, 2.998e-08, 3.135e-08, 3.255e-08,
  3.352e-08, 3.426e-08, 3.484e-08, 3.53e-08, 3.593e-08, 3.671e-08,
  3.759e-08, 3.945e-08, 4.192e-08, 4.49e-08, 5.03e-08, 5.703e-08,
  6.538e-08, 7.878e-08, 9.644e-08, 1.196e-07, 1.498e-07, 1.904e-07,
  2.422e-07, 3.055e-07, 3.804e-07, 4.747e-07, 5.899e-07, 7.272e-07,
  8.91e-07, 1.071e-06, 1.296e-06, 1.546e-06, 1.823e-06, 2.135e-06,
  2.44e-06, 2.714e-06, 2.967e-06, 3.189e-06, 3.391e-06, 3.58e-06,
  3.773e-06, 4.022e-06, 4.346e-06, 4.749e-06, 5.199e-06, 5.668e-06,
  6.157e-06, 6.688e-06, 7.254e-06, 7.867e-06, 8.539e-06, 9.26e-06,
  1.009e-05, 1.119e-05, 1.228e-05, 1.365e-05, 1.506e-05, 1.641e-05,
  1.784e-05, 1.952e-05, 2.132e-05, 2.323e-05, 2.531e-05, 2.754e-05,
  3.047e-05, 3.459e-05, 3.922e-05, 4.439e-05, 4.825e-05, 5.077e-05,
  5.34e-05, 5.618e-05, 5.909e-05, 6.207e-05, 6.519e-05, 6.845e-05,
  6.819e-05, 6.726e-05, 6.622e-05, 6.512e-05, 6.671e-05, 6.862e-05,
  7.048e-05, 7.264e-05, 7.3e-05, 7.3e-05, 7.3e-05, 7.3e-05, 7.3e-05
};

static double cof2[121] = {
```

```
  7.5e-14, 1.055e-13, 1.485e-13, 2.111e-13, 3.001e-13, 4.333e-13,
  6.269e-13, 9.221e-13, 1.364e-12, 2.046e-12, 3.093e-12, 4.703e-12,
  7.225e-12, 1.113e-11, 1.66e-11, 2.088e-11, 2.626e-11, 3.433e-11,
  4.549e-11, 5.886e-11, 7.21e-11, 8.824e-11, 1.015e-10, 1.155e-10,
  1.288e-10, 1.388e-10, 1.497e-10, 1.554e-10, 1.606e-10, 1.639e-10,
  1.64e-10, 1.64e-10, 1.596e-10, 1.542e-10, 1.482e-10, 1.382e-10,
  1.289e-10, 1.198e-10, 1.109e-10, 1.026e-10, 9.484e-11, 8.75e-11,
  8.086e-11, 7.49e-11, 6.948e-11, 6.446e-11, 5.961e-11, 5.505e-11,
  5.085e-11, 4.586e-11, 4.1e-11, 3.665e-11, 3.235e-11, 2.842e-11,
  2.491e-11, 2.11e-11, 1.769e-11, 1.479e-11, 1.197e-11, 9.631e-12,
  7.74e-12, 6.201e-12, 4.963e-12, 3.956e-12, 3.151e-12, 2.507e-12,
  1.99e-12, 1.576e-12, 1.245e-12, 9.83e-13, 7.742e-13, 6.088e-13,
  4.782e-13, 3.745e-13, 2.929e-13, 2.286e-13, 1.782e-13, 1.388e-13,
  1.079e-13, 8.362e-14, 6.471e-14, 4.996e-14, 3.85e-14, 2.96e-14,
  2.265e-14, 1.729e-14, 1.317e-14, 9.998e-15, 7.549e-15, 5.683e-15,
  4.273e-15, 3.193e-15, 2.385e-15, 1.782e-15, 1.331e-15, 9.957e-16,
  7.461e-16, 5.601e-16, 4.228e-16, 3.201e-16, 2.438e-16, 1.878e-16,
  1.445e-16, 1.111e-16, 8.544e-17, 6.734e-17, 5.341e-17, 4.237e-17,
  3.394e-17, 2.759e-17, 2.254e-17, 1.851e-17, 1.54e-17, 1.297e-17,
  1.096e-17, 9.365e-18, 8e-18, 6.938e-18, 6.056e-18, 5.287e-18,
  4.662e-18
};

static double f11[121] = {
  2.65e-10, 2.65e-10, 2.65e-10, 2.65e-10, 2.65e-10, 2.65e-10, 2.65e-10,
  2.65e-10, 2.65e-10, 2.65e-10, 2.65e-10, 2.65e-10, 2.635e-10, 2.536e-10,
  2.44e-10, 2.348e-10, 2.258e-10, 2.153e-10, 2.046e-10, 1.929e-10,
  1.782e-10, 1.648e-10, 1.463e-10, 1.291e-10, 1.1e-10, 8.874e-11,
  7.165e-11, 5.201e-11, 3.744e-11, 2.577e-11, 1.64e-11, 1.048e-11,
  5.993e-12, 3.345e-12, 1.839e-12, 9.264e-13, 4.688e-13, 2.329e-13,
  1.129e-13, 5.505e-14, 2.825e-14, 1.492e-14, 7.997e-15, 5.384e-15,
  3.988e-15, 2.955e-15, 2.196e-15, 1.632e-15, 1.214e-15, 9.025e-16,
  6.708e-16, 4.984e-16, 3.693e-16, 2.733e-16, 2.013e-16, 1.481e-16,
  1.087e-16, 7.945e-17, 5.782e-17, 4.195e-17, 3.038e-17, 2.19e-17,
  1.577e-17, 1.128e-17, 8.063e-18, 5.753e-18, 4.09e-18, 2.899e-18,
  2.048e-18, 1.444e-18, 1.015e-18, 7.12e-19, 4.985e-19, 3.474e-19,
  2.417e-19, 1.677e-19, 1.161e-19, 8.029e-20, 5.533e-20, 3.799e-20,
  2.602e-20, 1.776e-20, 1.209e-20, 8.202e-21, 5.522e-21, 3.707e-21,
  2.48e-21, 1.652e-21, 1.091e-21, 7.174e-22, 4.709e-22, 3.063e-22,
  1.991e-22, 1.294e-22, 8.412e-23, 5.483e-23, 3.581e-23, 2.345e-23,
  1.548e-23, 1.027e-23, 6.869e-24, 4.673e-24, 3.173e-24, 2.153e-24,
  1.461e-24, 1.028e-24, 7.302e-25, 5.188e-25, 3.739e-25, 2.753e-25,
  2.043e-25, 1.528e-25, 1.164e-25, 9.041e-26, 7.051e-26, 5.587e-26,
  4.428e-26, 3.588e-26, 2.936e-26, 2.402e-26, 1.995e-26
};

static double f12[121] = {
  5.45e-10, 5.45e-10, 5.45e-10, 5.45e-10, 5.45e-10, 5.45e-10, 5.45e-10,
  5.45e-10, 5.45e-10, 5.45e-10, 5.45e-10, 5.45e-10, 5.429e-10, 5.291e-10,
  5.155e-10, 5.022e-10, 4.893e-10, 4.772e-10, 4.655e-10, 4.497e-10,
  4.249e-10, 4.015e-10, 3.632e-10, 3.261e-10, 2.858e-10, 2.408e-10,
  2.03e-10, 1.685e-10, 1.4e-10, 1.163e-10, 9.65e-11, 8.02e-11, 6.705e-11,
  5.624e-11, 4.764e-11, 4.249e-11, 3.792e-11, 3.315e-11, 2.819e-11,
  2.4e-11, 1.999e-11, 1.64e-11, 1.352e-11, 1.14e-11, 9.714e-12,
  8.28e-12, 7.176e-12, 6.251e-12, 5.446e-12, 4.72e-12, 4.081e-12,
  3.528e-12, 3.08e-12, 2.699e-12, 2.359e-12, 2.111e-12, 1.901e-12,
  1.709e-12, 1.534e-12, 1.376e-12, 1.233e-12, 1.103e-12, 9.869e-13,
  8.808e-13, 7.859e-13, 7.008e-13, 6.241e-13, 5.553e-13, 4.935e-13,
  4.383e-13, 3.889e-13, 3.447e-13, 3.054e-13, 2.702e-13, 2.389e-13,
  2.11e-13, 1.862e-13, 1.643e-13, 1.448e-13, 1.274e-13, 1.121e-13,
  9.844e-14, 8.638e-14, 7.572e-14, 6.62e-14, 5.782e-14, 5.045e-14,
  4.394e-14, 3.817e-14, 3.311e-14, 2.87e-14, 2.48e-14, 2.142e-14,
  1.851e-14, 1.599e-14, 1.383e-14, 1.196e-14, 1.036e-14, 9e-15,
  7.828e-15, 6.829e-15, 5.992e-15, 5.254e-15, 4.606e-15, 4.037e-15,
  3.583e-15, 3.19e-15, 2.841e-15, 2.542e-15, 2.291e-15, 2.07e-15,
  1.875e-15, 1.71e-15, 1.57e-15, 1.442e-15, 1.333e-15, 1.232e-15,
  1.147e-15, 1.071e-15, 1.001e-15, 9.396e-16
};

static double f14[121] = {
  9e-11, 9e-11, 9e-11, 9e-11, 9e-11, 9e-11, 9e-11, 9e-11, 9e-11, 9e-11,
  9e-11, 9e-11, 9e-11, 9e-11, 9e-11, 8.91e-11, 8.73e-11, 8.46e-11,
  8.19e-11, 7.92e-11, 7.74e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11,
  7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11,
  7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11,
  7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11,
  7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11,
  7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11,
  7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11,
  7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11,
  7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11,
  7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11,
  7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11,
  7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11,
  7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11,
  7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11,
  7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11
```

```
};

static double f22[121] = {
  1.4e-10, 1.4e-10, 1.4e-10, 1.4e-10, 1.4e-10, 1.4e-10, 1.4e-10,
  1.4e-10, 1.4e-10, 1.4e-10, 1.372e-10, 1.317e-10, 1.235e-10, 1.153e-10,
  1.075e-10, 1.002e-10, 9.332e-11, 8.738e-11, 8.194e-11, 7.7e-11,
  7.165e-11, 6.753e-11, 6.341e-11, 5.971e-11, 5.6e-11, 5.229e-11,
  4.859e-11, 4.488e-11, 4.118e-11, 3.83e-11, 3.568e-11, 3.308e-11,
  3.047e-11, 2.82e-11, 2.594e-11, 2.409e-11, 2.237e-11, 2.065e-11,
  1.894e-11, 1.771e-11, 1.647e-11, 1.532e-11, 1.416e-11, 1.332e-11,
  1.246e-11, 1.161e-11, 1.087e-11, 1.017e-11, 9.471e-12, 8.853e-12,
  8.235e-12, 7.741e-12, 7.247e-12, 6.836e-12, 6.506e-12, 6.176e-12,
  5.913e-12, 5.65e-12, 5.419e-12, 5.221e-12, 5.024e-12, 4.859e-12,
  4.694e-12, 4.546e-12, 4.414e-12, 4.282e-12, 4.15e-12, 4.019e-12,
  3.903e-12, 3.805e-12, 3.706e-12, 3.607e-12, 3.508e-12, 3.41e-12,
  3.31e-12, 3.212e-12, 3.129e-12, 3.047e-12, 2.964e-12, 2.882e-12,
  2.8e-12, 2.734e-12, 2.668e-12, 2.602e-12, 2.537e-12, 2.471e-12,
  2.421e-12, 2.372e-12, 2.322e-12, 2.273e-12, 2.224e-12, 2.182e-12,
  2.141e-12, 2.1e-12, 2.059e-12, 2.018e-12, 1.977e-12, 1.935e-12,
  1.894e-12, 1.853e-12, 1.812e-12, 1.77e-12, 1.73e-12, 1.688e-12,
  1.647e-12, 1.606e-12, 1.565e-12, 1.524e-12, 1.483e-12, 1.441e-12,
  1.4e-12, 1.359e-12, 1.317e-12, 1.276e-12, 1.235e-12, 1.194e-12,
  1.153e-12, 1.112e-12, 1.071e-12, 1.029e-12, 9.883e-13
};

static double h2o[121] = {
  0.01166, 0.008269, 0.005742, 0.003845, 0.00277, 0.001897, 0.001272,
  0.000827, 0.000539, 0.0003469, 0.0001579, 3.134e-05, 1.341e-05,
  6.764e-06, 4.498e-06, 3.703e-06, 3.724e-06, 3.899e-06, 4.002e-06,
  4.122e-06, 4.277e-06, 4.438e-06, 4.558e-06, 4.673e-06, 4.763e-06,
  4.809e-06, 4.856e-06, 4.936e-06, 5.021e-06, 5.114e-06, 5.222e-06,
  5.331e-06, 5.414e-06, 5.488e-06, 5.563e-06, 5.633e-06, 5.704e-06,
  5.767e-06, 5.819e-06, 5.872e-06, 5.914e-06, 5.949e-06, 5.984e-06,
  6.015e-06, 6.044e-06, 6.073e-06, 6.104e-06, 6.136e-06, 6.167e-06,
  6.189e-06, 6.208e-06, 6.226e-06, 6.212e-06, 6.185e-06, 6.158e-06,
  6.114e-06, 6.066e-06, 6.018e-06, 5.877e-06, 5.728e-06, 5.582e-06,
  5.437e-06, 5.296e-06, 5.156e-06, 5.02e-06, 4.886e-06, 4.754e-06,
  4.625e-06, 4.498e-06, 4.374e-06, 4.242e-06, 4.096e-06, 3.955e-06,
  3.817e-06, 3.683e-06, 3.491e-06, 3.204e-06, 2.94e-06, 2.696e-06,
  2.47e-06, 2.252e-06, 2.019e-06, 1.808e-06, 1.618e-06, 1.445e-06,
  1.285e-06, 1.105e-06, 9.489e-07, 8.121e-07, 6.938e-07, 5.924e-07,
  5.04e-07, 4.288e-07, 3.648e-07, 3.103e-07, 2.642e-07, 2.252e-07,
  1.921e-07, 1.643e-07, 1.408e-07, 1.211e-07, 1.048e-07, 9.063e-08,
  7.835e-08, 6.774e-08, 5.936e-08, 5.221e-08, 4.592e-08, 4.061e-08,
  3.62e-08, 3.236e-08, 2.902e-08, 2.62e-08, 2.383e-08, 2.171e-08,
  1.989e-08, 1.823e-08, 1.684e-08, 1.562e-08, 1.449e-08, 1.351e-08
};

static double h2o2[121] = {
  1.779e-10, 7.938e-10, 8.953e-10, 8.032e-10, 6.564e-10, 5.159e-10,
  4.003e-10, 3.026e-10, 2.222e-10, 1.58e-10, 1.044e-10, 6.605e-11,
  3.413e-11, 1.453e-11, 1.062e-11, 1.009e-11, 9.597e-12, 1.175e-11,
  1.572e-11, 2.091e-11, 2.746e-11, 3.603e-11, 4.791e-11, 6.387e-11,
  8.239e-11, 1.007e-10, 1.23e-10, 1.363e-10, 1.489e-10, 1.585e-10,
  1.608e-10, 1.632e-10, 1.576e-10, 1.502e-10, 1.423e-10, 1.302e-10,
  1.192e-10, 1.085e-10, 9.795e-11, 8.854e-11, 8.057e-11, 7.36e-11,
  6.736e-11, 6.362e-11, 6.087e-11, 5.825e-11, 5.623e-11, 5.443e-11,
  5.27e-11, 5.098e-11, 4.931e-11, 4.769e-11, 4.611e-11, 4.458e-11,
  4.308e-11, 4.102e-11, 3.887e-11, 3.682e-11, 3.521e-11, 3.369e-11,
  3.224e-11, 3.082e-11, 2.946e-11, 2.814e-11, 2.687e-11, 2.566e-11,
  2.449e-11, 2.336e-11, 2.227e-11, 2.123e-11, 2.023e-11, 1.927e-11,
  1.835e-11, 1.746e-11, 1.661e-11, 1.58e-11, 1.502e-11, 1.428e-11,
  1.357e-11, 1.289e-11, 1.224e-11, 1.161e-11, 1.102e-11, 1.045e-11,
  9.895e-12, 9.369e-12, 8.866e-12, 8.386e-12, 7.922e-12, 7.479e-12,
  7.06e-12, 6.656e-12, 6.274e-12, 5.914e-12, 5.575e-12, 5.257e-12,
  4.959e-12, 4.679e-12, 4.42e-12, 4.178e-12, 3.954e-12, 3.75e-12,
  3.557e-12, 3.372e-12, 3.198e-12, 3.047e-12, 2.908e-12, 2.775e-12,
  2.653e-12, 2.544e-12, 2.442e-12, 2.346e-12, 2.26e-12, 2.183e-12,
  2.11e-12, 2.044e-12, 1.98e-12, 1.924e-12, 1.871e-12, 1.821e-12,
  1.775e-12
};

static double hcn[121] = {
  5.5e-10, 5.5e-10, 5.5e-10, 5.5e-10, 5.5e-10, 5.5e-10, 5.5e-10,
  5.5e-10, 5.5e-10, 5.5e-10, 5.5e-10, 5.498e-10, 5.495e-10, 5.493e-10,
  5.49e-10, 5.488e-10, 4.717e-10, 3.946e-10, 3.174e-10, 2.4e-10,
  1.626e-10, 1.619e-10, 1.612e-10, 1.602e-10, 1.593e-10, 1.582e-10,
  1.572e-10, 1.56e-10, 1.549e-10, 1.539e-10, 1.53e-10, 1.519e-10,
  1.506e-10, 1.487e-10, 1.467e-10, 1.449e-10, 1.43e-10, 1.413e-10,
  1.397e-10, 1.382e-10, 1.368e-10, 1.354e-10, 1.337e-10, 1.315e-10,
  1.292e-10, 1.267e-10, 1.241e-10, 1.215e-10, 1.19e-10, 1.165e-10,
  1.141e-10, 1.118e-10, 1.096e-10, 1.072e-10, 1.047e-10, 1.021e-10,
  9.968e-11, 9.739e-11, 9.539e-11, 9.339e-11, 9.135e-11, 8.898e-11,
  8.664e-11, 8.439e-11, 8.249e-11, 8.075e-11, 7.904e-11, 7.735e-11,
  7.565e-11, 7.399e-11, 7.245e-11, 7.109e-11, 6.982e-11, 6.863e-11,
  6.755e-11, 6.657e-11, 6.587e-11, 6.527e-11, 6.476e-11, 6.428e-11,
  6.382e-11, 6.343e-11, 6.307e-11, 6.272e-11, 6.238e-11, 6.205e-11,
```

```
  6.17e-11, 6.137e-11, 6.102e-11, 6.072e-11, 6.046e-11, 6.03e-11,
  6.018e-11, 6.01e-11, 6.001e-11, 5.992e-11, 5.984e-11, 5.975e-11,
  5.967e-11, 5.958e-11, 5.95e-11, 5.941e-11, 5.933e-11, 5.925e-11,
  5.916e-11, 5.908e-11, 5.899e-11, 5.891e-11, 5.883e-11, 5.874e-11,
  5.866e-11, 5.858e-11, 5.85e-11, 5.841e-11, 5.833e-11, 5.825e-11,
  5.817e-11, 5.808e-11, 5.8e-11, 5.792e-11, 5.784e-11
};

static double hno3[121] = {
  1.809e-10, 7.234e-10, 5.899e-10, 4.342e-10, 3.277e-10, 2.661e-10,
  2.35e-10, 2.267e-10, 2.389e-10, 2.651e-10, 3.255e-10, 4.099e-10,
  5.42e-10, 6.978e-10, 8.807e-10, 1.112e-09, 1.405e-09, 2.04e-09,
  3.111e-09, 4.5e-09, 5.762e-09, 7.37e-09, 7.852e-09, 8.109e-09,
  8.067e-09, 7.554e-09, 7.076e-09, 6.268e-09, 5.524e-09, 4.749e-09,
  3.909e-09, 3.223e-09, 2.517e-09, 1.942e-09, 1.493e-09, 1.122e-09,
  8.449e-10, 6.361e-10, 4.787e-10, 3.611e-10, 2.804e-10, 2.215e-10,
  1.758e-10, 1.441e-10, 1.197e-10, 9.953e-11, 8.505e-11, 7.334e-11,
  6.325e-11, 5.625e-11, 5.058e-11, 4.548e-11, 4.122e-11, 3.748e-11,
  3.402e-11, 3.088e-11, 2.8e-11, 2.536e-11, 2.293e-11, 2.072e-11,
  1.871e-11, 1.687e-11, 1.52e-11, 1.368e-11, 1.23e-11, 1.105e-11,
  9.922e-12, 8.898e-12, 7.972e-12, 7.139e-12, 6.385e-12, 5.708e-12,
  5.099e-12, 4.549e-12, 4.056e-12, 3.613e-12, 3.216e-12, 2.862e-12,
  2.544e-12, 2.259e-12, 2.004e-12, 1.776e-12, 1.572e-12, 1.391e-12,
  1.227e-12, 1.082e-12, 9.528e-13, 8.379e-13, 7.349e-13, 6.436e-13,
  5.634e-13, 4.917e-13, 4.291e-13, 3.745e-13, 3.267e-13, 2.854e-13,
  2.494e-13, 2.181e-13, 1.913e-13, 1.68e-13, 1.479e-13, 1.31e-13,
  1.159e-13, 1.025e-13, 9.067e-14, 8.113e-14, 7.281e-14, 6.535e-14,
  5.892e-14, 5.348e-14, 4.867e-14, 4.439e-14, 4.073e-14, 3.76e-14,
  3.476e-14, 3.229e-14, 3e-14, 2.807e-14, 2.635e-14, 2.473e-14,
  2.332e-14
};

static double hno4[121] = {
  6.118e-12, 3.594e-12, 2.807e-12, 3.04e-12, 4.458e-12, 7.986e-12,
  1.509e-11, 2.661e-11, 3.738e-11, 4.652e-11, 4.429e-11, 3.992e-11,
  3.347e-11, 3.005e-11, 3.173e-11, 4.055e-11, 5.812e-11, 8.489e-11,
  1.19e-10, 1.482e-10, 1.766e-10, 2.103e-10, 2.35e-10, 2.598e-10,
  2.801e-10, 2.899e-10, 3e-10, 2.817e-10, 2.617e-10, 2.332e-10,
  1.933e-10, 1.605e-10, 1.232e-10, 9.285e-11, 6.941e-11, 4.951e-11,
  3.539e-11, 2.402e-11, 1.522e-11, 9.676e-12, 6.056e-12, 3.745e-12,
  2.34e-12, 1.463e-12, 9.186e-13, 5.769e-13, 3.322e-13, 1.853e-13,
  1.035e-13, 7.173e-14, 5.382e-14, 4.036e-14, 3.401e-14, 2.997e-14,
  2.635e-14, 2.316e-14, 2.034e-14, 1.783e-14, 1.56e-14, 1.363e-14,
  1.19e-14, 1.037e-14, 9.032e-15, 7.846e-15, 6.813e-15, 5.912e-15,
  5.121e-15, 4.431e-15, 3.829e-15, 3.306e-15, 2.851e-15, 2.456e-15,
  2.114e-15, 1.816e-15, 1.559e-15, 1.337e-15, 1.146e-15, 9.811e-16,
  8.389e-16, 7.162e-16, 6.109e-16, 5.203e-16, 4.425e-16, 3.76e-16,
  3.184e-16, 2.692e-16, 2.274e-16, 1.917e-16, 1.61e-16, 1.35e-16,
  1.131e-16, 9.437e-17, 7.874e-17, 6.57e-17, 5.481e-17, 4.579e-17,
  3.828e-17, 3.204e-17, 2.691e-17, 2.264e-17, 1.912e-17, 1.626e-17,
  1.382e-17, 1.174e-17, 9.972e-18, 8.603e-18, 7.45e-18, 6.453e-18,
  5.623e-18, 4.944e-18, 4.361e-18, 3.859e-18, 3.443e-18, 3.096e-18,
  2.788e-18, 2.528e-18, 2.293e-18, 2.099e-18, 1.929e-18, 1.773e-18,
  1.64e-18
};

static double hocl[121] = {
  1.056e-12, 1.194e-12, 1.35e-12, 1.531e-12, 1.737e-12, 1.982e-12,
  2.263e-12, 2.599e-12, 2.991e-12, 3.459e-12, 4.012e-12, 4.662e-12,
  5.438e-12, 6.35e-12, 7.425e-12, 8.686e-12, 1.016e-11, 1.188e-11,
  1.389e-11, 1.659e-11, 2.087e-11, 2.621e-11, 3.265e-11, 4.064e-11,
  4.859e-11, 5.441e-11, 6.09e-11, 6.373e-11, 6.611e-11, 6.94e-11,
  7.44e-11, 7.97e-11, 8.775e-11, 9.722e-11, 1.064e-10, 1.089e-10,
  1.114e-10, 1.106e-10, 1.053e-10, 1.004e-10, 9.006e-11, 7.778e-11,
  6.739e-11, 5.636e-11, 4.655e-11, 3.845e-11, 3.042e-11, 2.368e-11,
  1.845e-11, 1.442e-11, 1.127e-11, 8.814e-12, 6.544e-12, 4.763e-12,
  3.449e-12, 2.612e-12, 1.999e-12, 1.526e-12, 1.16e-12, 8.793e-13,
  6.655e-13, 5.017e-13, 3.778e-13, 2.829e-13, 2.117e-13, 1.582e-13,
  1.178e-13, 8.755e-14, 6.486e-14, 4.799e-14, 3.54e-14, 2.606e-14,
  1.916e-14, 1.403e-14, 1.026e-14, 7.48e-15, 5.446e-15, 3.961e-15,
  2.872e-15, 2.076e-15, 1.498e-15, 1.077e-15, 7.726e-16, 5.528e-16,
  3.929e-16, 2.785e-16, 1.969e-16, 1.386e-16, 9.69e-17, 6.747e-17,
  4.692e-17, 3.236e-17, 2.232e-17, 1.539e-17, 1.061e-17, 7.332e-18,
  5.076e-18, 3.522e-18, 2.461e-18, 1.726e-18, 1.22e-18, 8.75e-19,
  6.264e-19, 4.482e-19, 3.207e-19, 2.368e-19, 1.762e-19, 1.312e-19,
  9.891e-20, 7.595e-20, 5.87e-20, 4.567e-20, 3.612e-20, 2.904e-20,
  2.343e-20, 1.917e-20, 1.568e-20, 1.308e-20, 1.1e-20, 9.25e-21,
  7.881e-21
};

static double n2o[121] = {
  3.17e-07, 3.17e-07, 3.17e-07, 3.17e-07, 3.17e-07, 3.17e-07, 3.17e-07,
  3.17e-07, 3.17e-07, 3.17e-07, 3.124e-07, 3.077e-07, 3.03e-07,
  2.984e-07, 2.938e-07, 2.892e-07, 2.847e-07, 2.779e-07, 2.705e-07,
  2.631e-07, 2.557e-07, 2.484e-07, 2.345e-07, 2.201e-07, 2.01e-07,
  1.754e-07, 1.532e-07, 1.329e-07, 1.154e-07, 1.003e-07, 8.735e-08,
  7.617e-08, 6.512e-08, 5.547e-08, 4.709e-08, 3.915e-08, 3.259e-08,
```

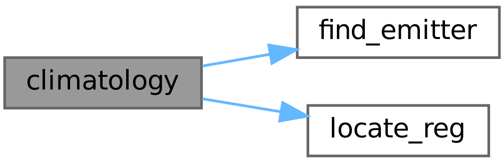
```
  2.738e-08, 2.327e-08, 1.98e-08, 1.711e-08, 1.493e-08, 1.306e-08,
  1.165e-08, 1.049e-08, 9.439e-09, 8.375e-09, 7.391e-09, 6.525e-09,
  5.759e-09, 5.083e-09, 4.485e-09, 3.953e-09, 3.601e-09, 3.27e-09,
  2.975e-09, 2.757e-09, 2.556e-09, 2.37e-09, 2.195e-09, 2.032e-09,
  1.912e-09, 1.79e-09, 1.679e-09, 1.572e-09, 1.482e-09, 1.402e-09,
  1.326e-09, 1.254e-09, 1.187e-09, 1.127e-09, 1.071e-09, 1.02e-09,
  9.673e-10, 9.193e-10, 8.752e-10, 8.379e-10, 8.017e-10, 7.66e-10,
  7.319e-10, 7.004e-10, 6.721e-10, 6.459e-10, 6.199e-10, 5.942e-10,
  5.703e-10, 5.488e-10, 5.283e-10, 5.082e-10, 4.877e-10, 4.696e-10,
  4.52e-10, 4.355e-10, 4.198e-10, 4.039e-10, 3.888e-10, 3.754e-10,
  3.624e-10, 3.499e-10, 3.381e-10, 3.267e-10, 3.163e-10, 3.058e-10,
  2.959e-10, 2.864e-10, 2.77e-10, 2.686e-10, 2.604e-10, 2.534e-10,
  2.462e-10, 2.386e-10, 2.318e-10, 2.247e-10, 2.189e-10, 2.133e-10,
  2.071e-10, 2.014e-10, 1.955e-10, 1.908e-10, 1.86e-10, 1.817e-10
};

static double n2o5[121] = {
  1.231e-11, 3.035e-12, 1.702e-12, 9.877e-13, 8.081e-13, 9.039e-13,
  1.169e-12, 1.474e-12, 1.651e-12, 1.795e-12, 1.998e-12, 2.543e-12,
  4.398e-12, 7.698e-12, 1.28e-11, 2.131e-11, 3.548e-11, 5.894e-11,
  7.645e-11, 1.089e-10, 1.391e-10, 1.886e-10, 2.386e-10, 2.986e-10,
  3.487e-10, 3.994e-10, 4.5e-10, 4.6e-10, 4.591e-10, 4.1e-10, 3.488e-10,
  2.846e-10, 2.287e-10, 1.696e-10, 1.011e-10, 6.428e-11, 4.324e-11,
  2.225e-11, 6.214e-12, 3.608e-12, 8.793e-13, 4.491e-13, 1.04e-13,
  6.1e-14, 3.436e-14, 6.671e-15, 1.171e-15, 5.848e-16, 1.212e-16,
  1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16,
  1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16,
  1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16,
  1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16,
  1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16,
  1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16,
  1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16,
  1e-16, 1e-16
};

static double nh3[121] = {
  1e-10, 1e-10, 1e-10, 1e-10, 1e-10, 1e-10, 1e-10, 1e-10, 1e-10, 1e-10,
  1e-10, 1e-10, 1e-10, 1e-10, 9.444e-11, 8.488e-11, 7.241e-11, 5.785e-11,
  4.178e-11, 3.018e-11, 2.18e-11, 1.574e-11, 1.137e-11, 8.211e-12,
  5.973e-12, 4.327e-12, 3.118e-12, 2.234e-12, 1.573e-12, 1.04e-12,
  6.762e-13, 4.202e-13, 2.406e-13, 1.335e-13, 6.938e-14, 3.105e-14,
  1.609e-14, 1.033e-14, 6.432e-15, 4.031e-15, 2.555e-15, 1.656e-15,
  1.115e-15, 7.904e-16, 5.63e-16, 4.048e-16, 2.876e-16, 2.004e-16,
  1.356e-16, 9.237e-17, 6.235e-17, 4.223e-17, 3.009e-17, 2.328e-17,
  2.002e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17,
  1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17,
  1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17,
  1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17,
  1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17,
  1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17,
  1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17,
  1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17,
  1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17,
  1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17,
  1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17,
  1.914e-17
};

static double no[121] = {
  2.586e-10, 4.143e-11, 1.566e-11, 9.591e-12, 8.088e-12, 8.462e-12,
  1.013e-11, 1.328e-11, 1.855e-11, 2.678e-11, 3.926e-11, 5.464e-11,
  7.012e-11, 8.912e-11, 1.127e-10, 1.347e-10, 1.498e-10, 1.544e-10,
  1.602e-10, 1.824e-10, 2.078e-10, 2.366e-10, 2.691e-10, 5.141e-10,
  8.259e-10, 1.254e-09, 1.849e-09, 2.473e-09, 3.294e-09, 4.16e-09,
  5.095e-09, 6.11e-09, 6.93e-09, 7.888e-09, 8.903e-09, 9.713e-09,
  1.052e-08, 1.115e-08, 1.173e-08, 1.21e-08, 1.228e-08, 1.239e-08,
  1.231e-08, 1.213e-08, 1.192e-08, 1.138e-08, 1.085e-08, 1.008e-08,
  9.224e-09, 8.389e-09, 7.262e-09, 6.278e-09, 5.335e-09, 4.388e-09,
  3.589e-09, 2.761e-09, 2.129e-09, 1.633e-09, 1.243e-09, 9.681e-10,
  8.355e-10, 7.665e-10, 7.442e-10, 8.584e-10, 9.732e-10, 1.063e-09,
  1.163e-09, 1.286e-09, 1.472e-09, 1.707e-09, 2.032e-09, 2.474e-09,
  2.977e-09, 3.506e-09, 4.102e-09, 5.013e-09, 6.493e-09, 8.414e-09,
  1.077e-08, 1.367e-08, 1.777e-08, 2.625e-08, 3.926e-08, 5.545e-08,
  7.195e-08, 9.464e-08, 1.404e-07, 2.183e-07, 3.329e-07, 4.535e-07,
  6.158e-07, 8.187e-07, 1.075e-06, 1.422e-06, 1.979e-06, 2.71e-06,
  3.58e-06, 4.573e-06, 5.951e-06, 7.999e-06, 1.072e-05, 1.372e-05,
  1.697e-05, 2.112e-05, 2.643e-05, 3.288e-05, 3.994e-05, 4.794e-05,
  5.606e-05, 6.383e-05, 7.286e-05, 8.156e-05, 8.883e-05, 9.469e-05,
  9.848e-05, 0.0001023, 0.0001066, 0.0001115, 0.0001145, 0.0001142,
  0.0001133
};

static double no2[121] = {
  3.036e-09, 2.945e-10, 9.982e-11, 5.069e-11, 3.485e-11, 2.982e-11,
  2.947e-11, 3.164e-11, 3.714e-11, 4.586e-11, 6.164e-11, 8.041e-11,
  9.982e-11, 1.283e-10, 1.73e-10, 2.56e-10, 3.909e-10, 5.959e-10,
  9.081e-10, 1.384e-09, 1.788e-09, 2.189e-09, 2.686e-09, 3.091e-09,
```

```
  3.49e-09, 3.796e-09, 4.2e-09, 5.103e-09, 6.005e-09, 6.3e-09, 6.706e-09,
  7.07e-09, 7.434e-09, 7.663e-09, 7.788e-09, 7.8e-09, 7.597e-09,
  7.482e-09, 7.227e-09, 6.403e-09, 5.585e-09, 4.606e-09, 3.703e-09,
  2.984e-09, 2.183e-09, 1.48e-09, 8.441e-10, 5.994e-10, 3.799e-10,
  2.751e-10, 1.927e-10, 1.507e-10, 1.102e-10, 6.971e-11, 5.839e-11,
  3.904e-11, 3.087e-11, 2.176e-11, 1.464e-11, 1.209e-11, 8.497e-12,
  6.477e-12, 4.371e-12, 2.914e-12, 2.424e-12, 1.753e-12, 1.35e-12,
  9.417e-13, 6.622e-13, 5.148e-13, 3.841e-13, 3.446e-13, 3.01e-13,
  2.551e-13, 2.151e-13, 1.829e-13, 1.64e-13, 1.475e-13, 1.352e-13,
  1.155e-13, 9.963e-14, 9.771e-14, 9.577e-14, 9.384e-14, 9.186e-14,
  9e-14, 9e-14, 9e-14, 9e-14, 9e-14, 9e-14, 9e-14, 9e-14, 9e-14, 9e-14,
  9e-14, 9e-14, 9e-14, 9e-14, 9e-14, 9e-14, 9e-14, 9e-14, 9e-14, 9e-14,
  9e-14, 9e-14, 9e-14, 9e-14, 9e-14, 9e-14, 9e-14, 9e-14, 9e-14, 9e-14,
  9e-14, 9e-14, 9e-14, 9e-14, 9e-14, 9e-14
};

static double o3[121] = {
  2.218e-08, 3.394e-08, 3.869e-08, 4.219e-08, 4.501e-08, 4.778e-08,
  5.067e-08, 5.402e-08, 5.872e-08, 6.521e-08, 7.709e-08, 9.461e-08,
  1.269e-07, 1.853e-07, 2.723e-07, 3.964e-07, 5.773e-07, 8.2e-07,
  1.155e-06, 1.59e-06, 2.076e-06, 2.706e-06, 3.249e-06, 3.848e-06,
  4.459e-06, 4.986e-06, 5.573e-06, 5.958e-06, 6.328e-06, 6.661e-06,
  6.9e-06, 7.146e-06, 7.276e-06, 7.374e-06, 7.447e-06, 7.383e-06,
  7.321e-06, 7.161e-06, 6.879e-06, 6.611e-06, 6.216e-06, 5.765e-06,
  5.355e-06, 4.905e-06, 4.471e-06, 4.075e-06, 3.728e-06, 3.413e-06,
  3.125e-06, 2.856e-06, 2.607e-06, 2.379e-06, 2.17e-06, 1.978e-06,
  1.8e-06, 1.646e-06, 1.506e-06, 1.376e-06, 1.233e-06, 1.102e-06,
  9.839e-07, 8.771e-07, 7.814e-07, 6.947e-07, 6.102e-07, 5.228e-07,
  4.509e-07, 3.922e-07, 3.501e-07, 3.183e-07, 2.909e-07, 2.686e-07,
  2.476e-07, 2.284e-07, 2.109e-07, 2.003e-07, 2.013e-07, 2.022e-07,
  2.032e-07, 2.042e-07, 2.097e-07, 2.361e-07, 2.656e-07, 2.989e-07,
  3.37e-07, 3.826e-07, 4.489e-07, 5.26e-07, 6.189e-07, 7.312e-07,
  8.496e-07, 8.444e-07, 8.392e-07, 8.339e-07, 8.286e-07, 8.234e-07,
  8.181e-07, 8.129e-07, 8.077e-07, 8.026e-07, 6.918e-07, 5.176e-07,
  3.865e-07, 2.885e-07, 2.156e-07, 1.619e-07, 1.219e-07, 9.161e-08,
  6.972e-08, 5.399e-08, 3.498e-08, 2.111e-08, 1.322e-08, 8.482e-09,
  5.527e-09, 3.423e-09, 2.071e-09, 1.314e-09, 8.529e-10, 5.503e-10,
  3.665e-10
};

static double ocs[121] = {
  6e-10, 6e-10, 6e-10, 6e-10, 6e-10, 6e-10, 6e-10, 6e-10, 6e-10, 5.997e-10,
  5.989e-10, 5.881e-10, 5.765e-10, 5.433e-10, 5.074e-10, 4.567e-10,
  4.067e-10, 3.601e-10, 3.093e-10, 2.619e-10, 2.232e-10, 1.805e-10,
  1.46e-10, 1.187e-10, 8.03e-11, 5.435e-11, 3.686e-11, 2.217e-11,
  1.341e-11, 8.756e-12, 4.511e-12, 2.37e-12, 1.264e-12, 8.28e-13,
  5.263e-13, 3.209e-13, 1.717e-13, 9.068e-14, 4.709e-14, 2.389e-14,
  1.236e-14, 1.127e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14,
  1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14,
  1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14,
  1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14,
  1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14,
  1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14,
  1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14,
  1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14,
  1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14,
  1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14,
  1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14,
  1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14,
  1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14,
  1.091e-14, 1.091e-14, 1.091e-14
};

static double sf6[121] = {
  4.103e-12, 4.103e-12, 4.103e-12, 4.103e-12, 4.103e-12, 4.103e-12,
  4.103e-12, 4.103e-12, 4.103e-12, 4.087e-12, 4.064e-12, 4.023e-12,
  3.988e-12, 3.941e-12, 3.884e-12, 3.755e-12, 3.622e-12, 3.484e-12,
  3.32e-12, 3.144e-12, 2.978e-12, 2.811e-12, 2.653e-12, 2.489e-12,
  2.332e-12, 2.199e-12, 2.089e-12, 2.013e-12, 1.953e-12, 1.898e-12,
  1.859e-12, 1.826e-12, 1.798e-12, 1.776e-12, 1.757e-12, 1.742e-12,
  1.728e-12, 1.717e-12, 1.707e-12, 1.698e-12, 1.691e-12, 1.685e-12,
  1.679e-12, 1.675e-12, 1.671e-12, 1.668e-12, 1.665e-12, 1.663e-12,
  1.661e-12, 1.659e-12, 1.658e-12, 1.657e-12, 1.656e-12, 1.655e-12,
  1.654e-12, 1.653e-12, 1.653e-12, 1.652e-12, 1.652e-12, 1.652e-12,
  1.651e-12, 1.651e-12, 1.651e-12, 1.651e-12, 1.651e-12, 1.651e-12,
  1.651e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12,
  1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12,
  1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12,
  1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12,
  1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12,
  1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12,
  1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12,
  1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12
};

static double so2[121] = {
  1e-10, 1e-10, 1e-10, 1e-10, 1e-10, 1e-10, 1e-10, 1e-10, 1e-10, 1e-10,
```

```
    1e-10, 1e-10, 9.867e-11, 9.537e-11, 9e-11, 8.404e-11, 7.799e-11,
    7.205e-11, 6.616e-11, 6.036e-11, 5.475e-11, 5.007e-11, 4.638e-11,
    4.346e-11, 4.055e-11, 3.763e-11, 3.471e-11, 3.186e-11, 2.905e-11,
    2.631e-11, 2.358e-11, 2.415e-11, 2.949e-11, 3.952e-11, 5.155e-11,
    6.76e-11, 8.741e-11, 1.099e-10, 1.278e-10, 1.414e-10, 1.512e-10,
    1.607e-10, 1.699e-10, 1.774e-10, 1.832e-10, 1.871e-10, 1.907e-10,
    1.943e-10, 1.974e-10, 1.993e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10,
    2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10,
    2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10,
    2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10,
    2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10,
    2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10,
    2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10,
    2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10
  };

  static int ig_co2 = -999;

  double *q[NG] = { NULL };

  /* Find emitter index of CO2... */
  if (ig_co2 == -999)
    ig_co2 = find_emitter(ctl, "CO2");

  /* Identify variable... */
  for (int ig = 0; ig < ctl->ng; ig++) {
    q[ig] = NULL;
    if (strcasecmp(ctl->emitter[ig], "C2H2") == 0)
      q[ig] = c2h2;
    if (strcasecmp(ctl->emitter[ig], "C2H6") == 0)
      q[ig] = c2h6;
    if (strcasecmp(ctl->emitter[ig], "CCl4") == 0)
      q[ig] = ccl4;
    if (strcasecmp(ctl->emitter[ig], "CH4") == 0)
      q[ig] = ch4;
    if (strcasecmp(ctl->emitter[ig], "ClO") == 0)
      q[ig] = clo;
    if (strcasecmp(ctl->emitter[ig], "ClONO2") == 0)
      q[ig] = clono2;
    if (strcasecmp(ctl->emitter[ig], "CO") == 0)
      q[ig] = co;
    if (strcasecmp(ctl->emitter[ig], "COF2") == 0)
      q[ig] = cof2;
    if (strcasecmp(ctl->emitter[ig], "F11") == 0)
      q[ig] = f11;
    if (strcasecmp(ctl->emitter[ig], "F12") == 0)
      q[ig] = f12;
    if (strcasecmp(ctl->emitter[ig], "F14") == 0)
      q[ig] = f14;
    if (strcasecmp(ctl->emitter[ig], "F22") == 0)
      q[ig] = f22;
    if (strcasecmp(ctl->emitter[ig], "H2O") == 0)
      q[ig] = h2o;
    if (strcasecmp(ctl->emitter[ig], "H2O2") == 0)
      q[ig] = h2o2;
    if (strcasecmp(ctl->emitter[ig], "HCN") == 0)
      q[ig] = hcn;
    if (strcasecmp(ctl->emitter[ig], "HNO3") == 0)
      q[ig] = hno3;
    if (strcasecmp(ctl->emitter[ig], "HNO4") == 0)
      q[ig] = hno4;
    if (strcasecmp(ctl->emitter[ig], "HOCl") == 0)
      q[ig] = hocl;
    if (strcasecmp(ctl->emitter[ig], "N2O") == 0)
      q[ig] = n2o;
    if (strcasecmp(ctl->emitter[ig], "N2O5") == 0)
      q[ig] = n2o5;
    if (strcasecmp(ctl->emitter[ig], "NH3") == 0)
      q[ig] = nh3;
    if (strcasecmp(ctl->emitter[ig], "NO") == 0)
      q[ig] = no;
    if (strcasecmp(ctl->emitter[ig], "NO2") == 0)
      q[ig] = no2;
    if (strcasecmp(ctl->emitter[ig], "O3") == 0)
      q[ig] = o3;
    if (strcasecmp(ctl->emitter[ig], "OCS") == 0)
      q[ig] = ocs;
    if (strcasecmp(ctl->emitter[ig], "SF6") == 0)
      q[ig] = sf6;
    if (strcasecmp(ctl->emitter[ig], "SO2") == 0)
      q[ig] = so2;
  }

  /* Loop over atmospheric data points... */
  for (int ip = 0; ip < atm->np; ip++) {

    /* Get altitude index... */
```

```
    const int iz = locate_reg(z, 121, atm->z[ip]);

    /* Interpolate pressure... */
    atm->p[ip] = LOGY(z[iz], pre[iz], z[iz + 1], pre[iz + 1], atm->z[ip]);

    /* Interpolate temperature... */
    atm->t[ip] = LIN(z[iz], tem[iz], z[iz + 1], tem[iz + 1], atm->z[ip]);

    /* Interpolate trace gases... */
    for (int ig = 0; ig < ctl->ng; ig++)
      if (q[ig] != NULL)
        atm->q[ig][ip] =
          LIN(z[iz], q[ig][iz], z[iz + 1], q[ig][iz + 1], atm->z[ip]);
      else
        atm->q[ig][ip] = 0;

    /* Set CO2... */
    if (ig_co2 >= 0)
      atm->q[ig_co2][ip] =
        371.789948e-6 + 2.026214e-6 * (atm->time[ip] - 63158400.) / 31557600.;

    /* Set extinction to zero... */
    for (int iw = 0; iw < ctl->nw; iw++)
      atm->k[iw][ip] = 0;

    /* Set cloud layer... */
    atm->clz = atm->cldz = 0;
    for (int icl = 0; icl < ctl->ncl; icl++)
      atm->clk[icl] = 0;

    /* Set surface layer... */
    atm->sfz = atm->sfp = atm->sft = 0;
    for (int isf = 0; isf < ctl->nsf; isf++)
      atm->sfeps[isf] = 1;
  }
}
```

Here is the call graph for this function:



ctmco2()

```
double ctmco2 (
            const double nu,
            const double p,
            const double t,
            const double u )
```

Compute carbon dioxide continuum (optical depth).

Definition at line 886 of file jurassic.c.

```
                    {

  static double co2296[2001] = { 9.3388e-5, 9.7711e-5, 1.0224e-4, 1.0697e-4,
    1.1193e-4, 1.1712e-4, 1.2255e-4, 1.2824e-4, 1.3419e-4, 1.4043e-4,
    1.4695e-4, 1.5378e-4, 1.6094e-4, 1.6842e-4, 1.7626e-4, 1.8447e-4,
    1.9307e-4, 2.0207e-4, 2.1149e-4, 2.2136e-4, 2.3169e-4, 2.4251e-4,
```

```
2.5384e-4, 2.657e-4, 2.7813e-4, 2.9114e-4, 3.0477e-4, 3.1904e-4,
3.3399e-4, 3.4965e-4, 3.6604e-4, 3.8322e-4, 4.0121e-4, 4.2006e-4,
4.398e-4, 4.6047e-4, 4.8214e-4, 5.0483e-4, 5.286e-4, 5.535e-4,
5.7959e-4, 6.0693e-4, 6.3557e-4, 6.6558e-4, 6.9702e-4, 7.2996e-4,
7.6449e-4, 8.0066e-4, 8.3856e-4, 8.7829e-4, 9.1991e-4, 9.6354e-4,
.0010093, .0010572, .0011074, .00116, .0012152, .001273,
.0013336, .0013972, .0014638, .0015336, .0016068, .0016835,
.001764, .0018483, .0019367, .0020295, .0021267, .0022286,
.0023355, .0024476, .0025652, .0026885, .0028178, .0029534,
.0030956, .0032448, .0034012, .0035654, .0037375, .0039181,
.0041076, .0043063, .0045148, .0047336, .0049632, .005204,
.0054567, .0057219, .0060002, .0062923, .0065988, .0069204,
.007258, .0076123, .0079842, .0083746, .0087844, .0092146,
.0096663, .01014, .010638, .011161, .01171, .012286, .012891,
.013527, .014194, .014895, .015631, .016404, .017217, .01807,
.018966, .019908, .020897, .021936, .023028, .024176, .025382,
.026649, .027981, .02938, .030851, .032397, .034023, .035732,
.037528, .039416, .041402, .04349, .045685, .047994, .050422,
.052975, .055661, .058486, .061458, .064584, .067873, .071334,
.074975, .078807, .082839, .087082, .091549, .096249, .1012,
.10641, .11189, .11767, .12375, .13015, .13689, .14399, .15147,
.15935, .16765, .17639, .18561, .19531, .20554, .21632, .22769,
.23967, .25229, .2656, .27964, .29443, .31004, .3265, .34386,
.36218, .3815, .40188, .42339, .44609, .47004, .49533, .52202,
.5502, .57995, .61137, .64455, .6796, .71663, .75574, .79707,
.84075, .88691, .9357, .98728, 1.0418, 1.0995, 1.1605, 1.225,
1.2932, 1.3654, 1.4418, 1.5227, 1.6083, 1.6989, 1.7948, 1.8964,
2.004, 2.118, 2.2388, 2.3668, 2.5025, 2.6463, 2.7988, 2.9606,
3.1321, 3.314, 3.5071, 3.712, 3.9296, 4.1605, 4.4058, 4.6663,
4.9431, 5.2374, 5.5501, 5.8818, 6.2353, 6.6114, 7.0115, 7.4372,
7.8905, 8.3731, 8.8871, 9.4349, 10.019, 10.641, 11.305, 12.013,
12.769, 13.576, 14.437, 15.358, 16.342, 17.39, 18.513, 19.716,
21.003, 22.379, 23.854, 25.436, 27.126, 28.942, 30.89, 32.973,
35.219, 37.634, 40.224, 43.021, 46.037, 49.29, 52.803, 56.447,
60.418, 64.792, 69.526, 74.637, 80.182, 86.193, 92.713, 99.786,
107.47, 115.84, 124.94, 134.86, 145.69, 157.49, 170.3, 184.39,
199.83, 216.4, 234.55, 254.72, 276.82, 299.85, 326.16, 354.99,
386.51, 416.68, 449.89, 490.12, 534.35, 578.25, 632.26, 692.61,
756.43, 834.75, 924.11, 1016.9, 996.96, 1102.7, 1219.2, 1351.9,
1494.3, 1654.1, 1826.5, 2027.9, 2249., 2453.8, 2714.4, 2999.4,
3209.5, 3509., 3840.4, 3907.5, 4190.7, 4533.5, 4648.3, 5059.1,
5561.6, 6191.4, 6820.8, 7905.9, 9362.2, 2431.3, 2211.3, 2046.8,
2023.8, 1985.9, 1905.9, 1491.1, 1369.8, 1262.2, 1200.7, 887.74,
820.25, 885.23, 887.21, 816.73, 1126.9, 1216.2, 1272.4, 1579.5,
1634.2, 1656.3, 1657.9, 1789.5, 1670.8, 1509.5, 8474.6, 7489.2,
6793.6, 6117., 5574.1, 5141.2, 5084.6, 4745.1, 4413.2, 4102.8,
4024.7, 3715., 3398.6, 3100.8, 2900.4, 2629.2, 2374., 2144.7,
1955.8, 1760.8, 1591.2, 1435.2, 1296.2, 1174., 1065.1, 967.76,
999.48, 897.45, 809.23, 732.77, 670.26, 611.93, 560.11, 518.77,
476.84, 438.8, 408.48, 380.21, 349.24, 322.71, 296.65, 272.85,
251.96, 232.04, 213.88, 197.69, 182.41, 168.41, 155.79, 144.05,
133.31, 123.48, 114.5, 106.21, 98.591, 91.612, 85.156, 79.204,
73.719, 68.666, 63.975, 59.637, 56.35, 52.545, 49.042, 45.788,
42.78, 39.992, 37.441, 35.037, 32.8, 30.744, 28.801, 26.986,
25.297, 23.731, 22.258, 20.883, 19.603, 18.403, 17.295, 16.249,
15.271, 14.356, 13.501, 12.701, 11.954, 11.254, 10.6, 9.9864,
9.4118, 8.8745, 8.3714, 7.8997, 7.4578, 7.0446, 6.6573, 6.2949,
5.9577, 5.6395, 5.3419, 5.063, 4.8037, 4.5608, 4.3452, 4.1364,
3.9413, 3.7394, 3.562, 3.3932, 3.2325, 3.0789, 2.9318, 2.7898,
2.6537, 2.5225, 2.3958, 2.2305, 2.1215, 2.0245, 1.9427, 1.8795,
1.8336, 1.7604, 1.7016, 1.6419, 1.5282, 1.4611, 1.3443, 1.27,
1.1675, 1.0824, 1.0534, .99833, .95854, .92981, .90887, .89346,
.88113, .87068, .86102, .85096, .88262, .86151, .83565, .80518,
.77045, .73736, .74744, .74954, .75773, .82267, .83493, .89402,
.89725, .93426, .95564, .94045, .94174, .93404, .92035, .90456,
.88621, .86673, .78117, .7515, .72056, .68822, .65658, .62764,
.55984, .55598, .57407, .60963, .63763, .66198, .61132, .60972,
.52496, .50649, .41872, .3964, .32422, .27276, .24048, .23772,
.2286, .22711, .23999, .32038, .34371, .36621, .38561, .39953,
.40636, .44913, .42716, .3919, .35477, .33935, .3351, .39746,
.40993, .49398, .49956, .56157, .54742, .57295, .57386, .55417,
.50745, .471, .43446, .39102, .34993, .31269, .27888, .24912,
.22291, .19994, .17972, .16197, .14633, .13252, .12029, .10942,
.099745, .091118, .083404, .076494, .070292, .064716, .059697,
.055173, .051093, .047411, .044089, .041092, .038392, .035965,
.033789, .031846, .030122, .028607, .02729, .026169, .025209,
.024405, .023766, .023288, .022925, .022716, .022681, .022685,
.022768, .023133, .023325, .023486, .024004, .024126, .024083,
.023785, .024023, .023029, .021649, .021108, .019454, .017809,
.017292, .016635, .017037, .018068, .018977, .018756, .017847,
.016557, .016142, .014459, .012869, .012381, .010875, .0098701,
.009285, .0091698, .0091701, .0096145, .010553, .01106, .012613,
.014362, .015017, .016507, .017741, .01768, .017784, .0171,
.016357, .016172, .017257, .018978, .020935, .021741, .023567,
.025183, .025589, .026732, .027648, .028278, .028215, .02856,
.029015, .029062, .028851, .028497, .027825, .027801, .026523,
.02487, .022967, .022168, .020194, .018605, .017903, .018439,
```

```
.019697, .020311, .020855, .020057, .018608, .016738, .015963,
.013844, .011801, .011134, .0097573, .0086007, .0086226,
.0083721, .0090978, .0097616, .0098426, .011317, .012853, .01447,
.014657, .015771, .016351, .016079, .014829, .013431, .013185,
.013207, .01448, .016176, .017971, .018265, .019526, .020455,
.019797, .019802, .0194, .018176, .017505, .016197, .015339,
.014401, .013213, .012203, .011186, .010236, .0093288, .0084854,
.0076837, .0069375, .0062614, .0056628, .0051153, .0046015,
.0041501, .003752, .0033996, .0030865, .0028077, .0025586,
.0023355, .0021353, .0019553, .0017931, .0016466, .0015141,
.0013941, .0012852, .0011862, .0010962, .0010142, 9.3935e-4,
8.71e-4, 8.0851e-4, 7.5132e-4, 6.9894e-4, 6.5093e-4, 6.0689e-4,
5.6647e-4, 5.2935e-4, 4.9525e-4, 4.6391e-4, 4.3509e-4, 4.086e-4,
3.8424e-4, 3.6185e-4, 3.4126e-4, 3.2235e-4, 3.0498e-4, 2.8904e-4,
2.7444e-4, 2.6106e-4, 2.4883e-4, 2.3766e-4, 2.275e-4, 2.1827e-4,
2.0992e-4, 2.0239e-4, 1.9563e-4, 1.896e-4, 1.8427e-4, 1.796e-4,
1.7555e-4, 1.7209e-4, 1.692e-4, 1.6687e-4, 1.6505e-4, 1.6375e-4,
1.6294e-4, 1.6261e-4, 1.6274e-4, 1.6334e-4, 1.6438e-4, 1.6587e-4,
1.678e-4, 1.7017e-4, 1.7297e-4, 1.762e-4, 1.7988e-4, 1.8399e-4,
1.8855e-4, 1.9355e-4, 1.9902e-4, 2.0494e-4, 2.1134e-4, 2.1823e-4,
2.2561e-4, 2.335e-4, 2.4192e-4, 2.5088e-4, 2.604e-4, 2.705e-4,
2.8119e-4, 2.9251e-4, 3.0447e-4, 3.171e-4, 3.3042e-4, 3.4447e-4,
3.5927e-4, 3.7486e-4, 3.9127e-4, 4.0854e-4, 4.267e-4, 4.4579e-4,
4.6586e-4, 4.8696e-4, 5.0912e-4, 5.324e-4, 5.5685e-4, 5.8253e-4,
6.0949e-4, 6.378e-4, 6.6753e-4, 6.9873e-4, 7.3149e-4, 7.6588e-4,
8.0198e-4, 8.3987e-4, 8.7964e-4, 9.2139e-4, 9.6522e-4, .0010112,
.0010595, .0011102, .0011634, .0012193, .001278, .0013396,
.0014043, .0014722, .0015436, .0016185, .0016972, .0017799,
.0018668, .001958, .0020539, .0021547, .0022606, .0023719,
.002489, .002612, .0027414, .0028775, .0030206, .0031712,
.0033295, .0034962, .0036716, .0038563, .0040506, .0042553,
.0044709, .004698, .0049373, .0051894, .0054552, .0057354,
.006031, .0063427, .0066717, .0070188, .0073854, .0077726,
.0081816, .0086138, .0090709, .0095543, .010066, .010607,
.011181, .011789, .012433, .013116, .013842, .014613, .015432,
.016304, .017233, .018224, .019281, .020394, .021574, .022836,
.024181, .025594, .027088, .028707, .030401, .032245, .034219,
.036262, .038539, .040987, .043578, .04641, .04949, .052726,
.056326, .0602, .064093, .068521, .073278, .077734, .083064,
.088731, .093885, .1003, .1072, .11365, .12187, .13078, .13989,
.15095, .16299, .17634, .19116, .20628, .22419, .24386, .26587,
.28811, .31399, .34321, .36606, .39675, .42742, .44243, .47197,
.49993, .49027, .51147, .52803, .48931, .49729, .5026, .43854,
.441, .44766, .43414, .46151, .50029, .55247, .43855, .32115,
.32607, .3431, .36119, .38029, .41179, .43996, .47144, .51853,
.55362, .59122, .66338, .69877, .74001, .82923, .86907, .90361,
1.0025, 1.031, 1.0559, 1.104, 1.1178, 1.1341, 1.1547, 1.351,
1.4772, 1.4812, 1.4907, 1.512, 1.5442, 1.5853, 1.6358, 1.6963,
1.7674, 1.8474, 1.9353, 2.0335, 2.143, 2.2592, 2.3853, 2.5217,
2.6686, 2.8273, 2.9998, 3.183, 3.3868, 3.6109, 3.8564, 4.1159,
4.4079, 4.7278, 5.0497, 5.3695, 5.758, 6.0834, 6.4976, 6.9312,
7.38, 7.5746, 7.9833, 8.3791, 8.3956, 8.7501, 9.1067, 9.072,
9.4649, 9.9112, 10.402, 10.829, 11.605, 12.54, 12.713, 10.443,
10.825, 11.375, 11.955, 12.623, 13.326, 14.101, 15.041, 15.547,
16.461, 17.439, 18.716, 19.84, 21.036, 22.642, 23.901, 25.244,
27.03, 28.411, 29.871, 31.403, 33.147, 34.744, 36.456, 39.239,
43.605, 45.162, 47.004, 49.093, 51.391, 53.946, 56.673, 59.629,
63.167, 66.576, 70.254, 74.222, 78.477, 83.034, 87.914, 93.18,
98.77, 104.74, 111.15, 117.95, 125.23, 133.01, 141.33, 150.21,
159.71, 169.89, 180.93, 192.54, 204.99, 218.34, 232.65, 248.,
264.47, 282.14, 301.13, 321.53, 343.48, 367.08, 392.5, 419.88,
449.4, 481.26, 515.64, 552.79, 592.99, 636.48, 683.61, 734.65,
789.99, 850.02, 915.14, 985.81, 1062.5, 1147.1, 1237.8, 1336.4,
1443.2, 1558.9, 1684.2, 1819.2, 1965.2, 2122.6, 2291.7, 2470.8,
2665.7, 2874.9, 3099.4, 3337.9, 3541., 3813.3, 4111.9, 4439.3,
4798.9, 5196., 5639.2, 6087.5, 6657.7, 7306.7, 8040.7, 8845.5,
9702.2, 10670., 11739., 12842., 14141., 15498., 17068., 18729.,
20557., 22559., 25248., 27664., 30207., 32915., 35611., 38081.,
40715., 43191., 41651., 42750., 43785., 44353., 44366., 44189.,
43618., 42862., 41878., 35133., 35215., 36383., 39420., 44055.,
44155., 45850., 46853., 39197., 38274., 29942., 28553., 21792.,
21228., 17106., 14955., 18181., 19557., 21427., 23728., 26301.,
28584., 30775., 32536., 33867., 40089., 39204., 37329., 34452.,
31373., 33921., 34800., 36043., 44415., 45162., 52181., 50895.,
54140., 50840., 50468., 48302., 44915., 40910., 36754., 32755.,
29093., 25860., 22962., 20448., 18247., 16326., 14645., 13165.,
11861., 10708., 9686.9, 8779.7, 7971.9, 7250.8, 6605.7, 6027.2,
5507.3, 5039.1, 4616.6, 4234.8, 3889., 3575.4, 3290.5, 3031.3,
2795.2, 2579.9, 2383.1, 2203.3, 2038.6, 1887.6, 1749.1, 1621.9,
1505., 1397.4, 1298.3, 1207., 1122.8, 1045., 973.1, 906.64,
845.16, 788.22, 735.48, 686.57, 641.21, 599.1, 559.99, 523.64,
489.85, 458.42, 429.16, 401.92, 376.54, 352.88, 330.82, 310.24,
291.03, 273.09, 256.34, 240.69, 226.05, 212.37, 199.57, 187.59,
176.37, 165.87, 156.03, 146.82, 138.17, 130.07, 122.47, 115.34,
108.65, 102.37, 96.473, 90.934, 85.73, 80.84, 76.243, 71.922,
67.858, 64.034, 60.438, 57.052, 53.866, 50.866, 48.04, 45.379,
42.872, 40.51, 38.285, 36.188, 34.211, 32.347, 30.588, 28.929,
```

```
27.362, 25.884, 24.489, 23.171, 21.929, 20.755, 19.646, 18.599,
17.61, 16.677, 15.795, 14.961, 14.174, 13.43, 12.725, 12.06,
11.431, 10.834, 10.27, 9.7361, 9.2302, 8.7518, 8.2997, 7.8724,
7.4674, 7.0848, 6.7226, 6.3794, 6.054, 5.745, 5.4525, 5.1752,
4.9121, 4.6625, 4.4259, 4.2015, 3.9888, 3.7872, 3.5961, 3.4149,
3.2431, 3.0802, 2.9257, 2.7792, 2.6402, 2.5084, 2.3834, 2.2648,
2.1522, 2.0455, 1.9441, 1.848, 1.7567, 1.6701, 1.5878, 1.5097,
1.4356, 1.3651, 1.2981, 1.2345, 1.174, 1.1167, 1.062, 1.0101,
.96087, .91414, .86986, .82781, .78777, .74971, .71339, .67882,
.64604, .61473, .58507, .55676, .52987, .5044, .48014, .45715,
.43527, .41453, .3948, .37609, .35831, .34142, .32524, .30995,
.29536, .28142, .26807, .25527, .24311, .23166, .22077, .21053,
.20081, .19143, .18261, .17407, .16603, .15833, .15089, .14385,
.13707, .13065, .12449, .11865, .11306, .10774, .10266, .097818,
.093203, .088815, .084641, .080671, .076892, .073296, .069873,
.066613, .06351, .060555, .05774, .055058, .052504, .050071,
.047752, .045543, .043438, .041432, .039521, .037699, .035962,
.034307, .032729, .031225, .029791, .028423, .02712, .025877,
.024692, .023563, .022485, .021458, .020478, .019543, .018652,
.017802, .016992, .016219, .015481, .014778, .014107, .013467,
.012856, .012274, .011718, .011188, .010682, .0102, .0097393,
.0093001, .008881, .0084812, .0080997, .0077358, .0073885,
.0070571, .0067409, .0064393, .0061514, .0058768, .0056147,
.0053647, .0051262, .0048987, .0046816, .0044745, .0042769,
.0040884, .0039088, .0037373, .0035739, .003418, .0032693,
.0031277, .0029926, .0028639, .0027413, .0026245, .0025133,
.0024074, .0023066, .0022108, .0021196, .002033, .0019507,
.0018726, .0017985, .0017282, .0016617, .0015988, .0015394,
.0014834, .0014306, .0013811, .0013346, .0012911, .0012506,
.0012131, .0011784, .0011465, .0011175, .0010912, .0010678,
.0010472, .0010295, .0010147, .001003, 9.9428e-4, 9.8883e-4,
9.8673e-4, 9.8821e-4, 9.9343e-4, .0010027, .0010164, .0010348,
.0010586, .0010882, .0011245, .0011685, .0012145, .0012666,
.0013095, .0013688, .0014048, .0014663, .0015309, .0015499,
.0016144, .0016312, .001705, .0017892, .0018499, .0019715,
.0021102, .0022442, .0024284, .0025893, .0027703, .0029445,
.0031193, .003346, .0034552, .0036906, .0037584, .0040084,
.0041934, .0044587, .0047093, .0049759, .0053421, .0055134,
.0059048, .0058663, .0061036, .0063259, .0059657, .0060653,
.0060972, .0055539, .0055653, .0055772, .005331, .0054953,
.0055919, .0058684, .006183, .0066675, .0069808, .0075142,
.0078536, .0084282, .0089454, .0094625, .0093703, .0095857,
.0099283, .010063, .010521, .0097778, .0098175, .010379, .010447,
.0105, .010617, .010706, .01078, .011177, .011212, .011304,
.011446, .011603, .011816, .012165, .012545, .013069, .013539,
.01411, .014776, .016103, .017016, .017994, .018978, .01998,
.021799, .022745, .023681, .024627, .025562, .026992, .027958,
.029013, .030154, .031402, .03228, .033651, .035272, .037088,
.039021, .041213, .043597, .045977, .04877, .051809, .054943,
.058064, .061528, .06537, .069309, .071928, .075752, .079589,
.083352, .084096, .087497, .090817, .091198, .094966, .099045,
.10429, .10867, .11518, .12269, .13126, .14087, .15161, .16388,
.16423, .1759, .18721, .19994, .21275, .22513, .23041, .24231,
.25299, .25396, .26396, .27696, .27929, .2908, .30595, .31433,
.3282, .3429, .35944, .37467, .39277, .41245, .43326, .45649,
.48152, .51897, .54686, .57877, .61263, .64962, .68983, .73945,
.78619, .83537, .89622, .95002, 1.0067, 1.0742, 1.1355, 1.2007,
1.2738, 1.347, 1.4254, 1.5094, 1.6009, 1.6976, 1.8019, 1.9148,
2.0357, 2.166, 2.3066, 2.4579, 2.6208, 2.7966, 2.986, 3.188,
3.4081, 3.6456, 3.9, 4.1747, 4.4712, 4.7931, 5.1359, 5.5097,
5.9117, 6.3435, 6.8003, 7.3001, 7.8385, 8.3945, 9.011, 9.6869,
10.392, 11.18, 12.036, 12.938, 13.944, 14.881, 16.029, 17.255,
18.574, 19.945, 21.38, 22.9, 24.477, 26.128, 27.87, 29.037,
30.988, 33.145, 35.506, 37.76, 40.885, 44.487, 48.505, 52.911,
57.56, 61.964, 67.217, 72.26, 78.343, 85.08, 91.867, 99.435,
107.68, 116.97, 127.12, 138.32, 150.26, 163.04, 174.81, 189.26,
205.61, 224.68, 240.98, 261.88, 285.1, 307.58, 334.35, 363.53,
394.68, 427.85, 458.85, 489.25, 472.87, 486.93, 496.27, 501.52,
501.57, 497.14, 488.09, 476.32, 393.76, 388.51, 393.42, 414.45,
455.12, 514.62, 520.38, 547.42, 562.6, 487.47, 480.83, 391.06,
376.92, 303.7, 295.91, 256.03, 236.73, 280.38, 310.71, 335.53,
367.88, 401.94, 435.52, 469.13, 497.94, 588.82, 597.94, 597.2,
588.28, 571.2, 555.75, 603.56, 638.15, 680.75, 801.72, 848.01,
962.15, 990.06, 1068.1, 1076.2, 1115.3, 1134.2, 1136.6, 1119.1,
1108.9, 1090.6, 1068.7, 1041.9, 1005.4, 967.98, 927.08, 780.1,
751.41, 733.12, 742.65, 785.56, 855.16, 852.45, 878.1, 784.59,
777.81, 765.13, 622.93, 498.09, 474.89, 386.9, 378.48, 336.17,
322.04, 329.57, 350.5, 383.38, 420.02, 462.39, 499.71, 531.98,
654.99, 653.43, 639.99, 605.16, 554.16, 504.42, 540.64, 552.33,
679.46, 699.51, 713.91, 832.17, 919.91, 884.96, 907.57, 846.56,
818.56, 768.93, 706.71, 642.17, 575.95, 515.38, 459.07, 409.02,
364.61, 325.46, 291.1, 260.89, 234.39, 211.01, 190.38, 172.11,
155.91, 141.49, 128.63, 117.13, 106.84, 97.584, 89.262, 81.756,
74.975, 68.842, 63.28, 58.232, 53.641, 49.46, 45.649, 42.168,
38.991, 36.078, 33.409, 30.96, 28.71, 26.642, 24.737, 22.985,
21.37, 19.882, 18.512, 17.242, 16.073, 14.987, 13.984, 13.05,
12.186, 11.384, 10.637, 9.9436, 9.2988, 8.6991, 8.141, 7.6215,
```

```
  7.1378, 6.6872, 6.2671, 5.8754, 5.51, 5.1691, 4.851, 4.5539,
  4.2764, 4.0169, 3.7742, 3.5472, 3.3348, 3.1359, 2.9495, 2.7749,
  2.6113, 2.4578, 2.3139, 2.1789, 2.0523, 1.9334, 1.8219, 1.7171,
  1.6188, 1.5263, 1.4395, 1.3579, 1.2812, 1.209, 1.1411, 1.0773,
  1.0171, .96048, .90713, .85684, .80959, .76495, .72282, .68309,
  .64563, .61035, .57707, .54573, .51622, .48834, .46199, .43709,
  .41359, .39129, .37034, .35064, .33198, .31442, .29784, .28218,
  .26732, .25337, .24017, .22774, .21601, .20479, .19426
};

static double co2260[2001] = { 5.7971e-5, 6.0733e-5, 6.3628e-5, 6.6662e-5,
  6.9843e-5, 7.3176e-5, 7.6671e-5, 8.0334e-5, 8.4175e-5, 8.8201e-5,
  9.2421e-5, 9.6846e-5, 1.0149e-4, 1.0635e-4, 1.1145e-4, 1.1679e-4,
  1.224e-4, 1.2828e-4, 1.3444e-4, 1.409e-4, 1.4768e-4, 1.5479e-4,
  1.6224e-4, 1.7006e-4, 1.7826e-4, 1.8685e-4, 1.9587e-4, 2.0532e-4,
  2.1524e-4, 2.2565e-4, 2.3656e-4, 2.48e-4, 2.6001e-4, 2.7261e-4,
  2.8582e-4, 2.9968e-4, 3.1422e-4, 3.2948e-4, 3.4548e-4, 3.6228e-4,
  3.799e-4, 3.9838e-4, 4.1778e-4, 4.3814e-4, 4.595e-4, 4.8191e-4,
  5.0543e-4, 5.3012e-4, 5.5603e-4, 5.8321e-4, 6.1175e-4, 6.417e-4,
  6.7314e-4, 7.0614e-4, 7.4078e-4, 7.7714e-4, 8.1531e-4, 8.5538e-4,
  8.9745e-4, 9.4162e-4, 9.8798e-4, .0010367, .0010878, .0011415,
  .0011978, .001257, .0013191, .0013844, .001453, .0015249,
  .0016006, .00168, .0017634, .001851, .001943, .0020397, .0021412,
  .0022479, .00236, .0024778, .0026015, .0027316, .0028682,
  .0030117, .0031626, .0033211, .0034877, .0036628, .0038469,
  .0040403, .0042436, .0044574, .004682, .0049182, .0051665,
  .0054276, .0057021, .0059907, .0062942, .0066133, .0069489,
  .0073018, .0076729, .0080632, .0084738, .0089056, .0093599,
  .0098377, .01034, .010869, .011426, .012011, .012627, .013276,
  .013958, .014676, .015431, .016226, .017063, .017944, .018872,
  .019848, .020876, .021958, .023098, .024298, .025561, .026892,
  .028293, .029769, .031323, .032961, .034686, .036503, .038418,
  .040435, .042561, .044801, .047161, .049649, .052271, .055035,
  .057948, .061019, .064256, .06767, .07127, .075066, .079069,
  .083291, .087744, .092441, .097396, .10262, .10814, .11396,
  .1201, .12658, .13342, .14064, .14826, .1563, .1648, .17376,
  .18323, .19324, .2038, .21496, .22674, .23919, .25234, .26624,
  .28093, .29646, .31287, .33021, .34855, .36794, .38844, .41012,
  .43305, .45731, .48297, .51011, .53884, .56924, .60141, .63547,
  .67152, .70969, .75012, .79292, .83826, .8863, .93718, .99111,
  1.0482, 1.1088, 1.173, 1.2411, 1.3133, 1.3898, 1.471, 1.5571,
  1.6485, 1.7455, 1.8485, 1.9577, 2.0737, 2.197, 2.3278, 2.4668,
  2.6145, 2.7715, 2.9383, 3.1156, 3.3042, 3.5047, 3.7181, 3.9451,
  4.1866, 4.4437, 4.7174, 5.0089, 5.3192, 5.65, 6.0025, 6.3782,
  6.7787, 7.206, 7.6617, 8.1479, 8.6669, 9.221, 9.8128, 10.445,
  11.12, 11.843, 12.615, 13.441, 14.325, 15.271, 16.283, 17.367,
  18.529, 19.776, 21.111, 22.544, 24.082, 25.731, 27.504, 29.409,
  31.452, 33.654, 36.024, 38.573, 41.323, 44.29, 47.492, 50.951,
  54.608, 58.588, 62.929, 67.629, 72.712, 78.226, 84.207, 90.699,
  97.749, 105.42, 113.77, 122.86, 132.78, 143.61, 155.44, 168.33,
  182.48, 198.01, 214.87, 233.39, 253.86, 276.34, 300.3, 327.28,
  356.89, 389.48, 422.29, 458.99, 501.39, 548.13, 595.62, 652.74,
  716.54, 784.57, 866.78, 960.59, 1062.8, 1072.5, 1189.5, 1319.4,
  1467.6, 1630.2, 1813.7, 2016.9, 2253., 2515.3, 2773.5, 3092.8,
  3444.4, 3720.4, 4104.3, 4527.5, 4645.9, 5021.7, 5462.2, 5597.,
  6110.6, 6732.5, 7513.8, 8270.6, 9640.6, 11487., 2796.1, 2680.1,
  2441.6, 2404.2, 2334.8, 2215.2, 1642.5, 1477.9, 1328.1, 1223.5,
  843.34, 766.96, 831.65, 834.84, 774.85, 1156.3, 1275.6, 1366.1,
  1795.6, 1885., 1936.5, 1953.4, 2154.4, 2002.7, 1789.8, 10381.,
  9040., 8216.5, 7384.7, 6721.9, 6187.7, 6143.8, 5703.9, 5276.6,
  4873.1, 4736., 4325.3, 3927., 3554.1, 3286.1, 2950.1, 2642.4,
  2368.7, 2138.9, 1914., 1719.6, 1543.9, 1388.6, 1252.1, 1132.2,
  1024.1, 1025.4, 920.58, 829.59, 750.54, 685.01, 624.25, 570.14,
  525.81, 481.85, 441.95, 408.71, 377.23, 345.86, 318.51, 292.26,
  268.34, 247.04, 227.14, 209.02, 192.69, 177.59, 163.78, 151.26,
  139.73, 129.19, 119.53, 110.7, 102.57, 95.109, 88.264, 81.948,
  76.13, 70.768, 65.827, 61.251, 57.022, 53.495, 49.824, 46.443,
  43.307, 40.405, 37.716, 35.241, 32.923, 30.77, 28.78, 26.915,
  25.177, 23.56, 22.059, 20.654, 19.345, 18.126, 16.988, 15.93,
  14.939, 14.014, 13.149, 12.343, 11.589, 10.884, 10.225, 9.6093,
  9.0327, 8.4934, 7.9889, 7.5166, 7.0744, 6.6604, 6.2727, 5.9098,
  5.5701, 5.2529, 4.955, 4.676, 4.4148, 4.171, 3.9426, 3.7332,
  3.5347, 3.3493, 3.1677, 3.0025, 2.8466, 2.6994, 2.5601, 2.4277,
  2.3016, 2.1814, 2.0664, 1.9564, 1.8279, 1.7311, 1.6427, 1.5645,
  1.4982, 1.443, 1.374, 1.3146, 1.2562, 1.17, 1.1105, 1.0272,
  .96863, .89718, .83654, .80226, .75908, .72431, .69573, .67174,
  .65126, .63315, .61693, .60182, .58715, .59554, .57649, .55526,
  .53177, .50622, .48176, .4813, .47642, .47492, .50273, .50293,
  .52687, .52239, .53419, .53814, .52626, .52211, .51492, .50622,
  .49746, .48841, .4792, .43534, .41999, .40349, .38586, .36799,
  .35108, .31089, .30803, .3171, .33599, .35041, .36149, .32924,
  .32462, .27309, .25961, .20922, .19504, .15683, .13098, .11588,
  .11478, .11204, .11363, .12135, .16423, .17785, .19094, .20236,
  .21084, .2154, .24108, .22848, .20871, .18797, .17963, .17834,
  .21552, .22284, .26945, .27052, .30108, .28977, .29772, .29224,
  .27658, .24956, .22777, .20654, .18392, .16338, .1452, .12916,
  .1152, .10304, .092437, .083163, .075031, .067878, .061564,
```

```
.055976, .051018, .046609, .042679, .03917, .036032, .033223,
.030706, .02845, .026428, .024617, .022998, .021554, .02027,
.019136, .018141, .017278, .016541, .015926, .015432, .015058,
.014807, .014666, .014635, .014728, .014947, .01527, .015728,
.016345, .017026, .017798, .018839, .019752, .020636, .021886,
.022695, .02327, .023478, .024292, .023544, .022222, .021932,
.020052, .018143, .017722, .017031, .017782, .01938, .020734,
.020476, .019255, .017477, .016878, .014617, .012489, .011765,
.0099077, .0086446, .0079446, .0078644, .0079763, .008671,
.01001, .0108, .012933, .015349, .016341, .018484, .020254,
.020254, .020478, .019591, .018595, .018385, .019913, .022254,
.024847, .025809, .028053, .029924, .030212, .031367, .03222,
.032739, .032537, .03286, .033344, .033507, .033499, .033339,
.032809, .033041, .031723, .029837, .027511, .026603, .024032,
.021914, .020948, .021701, .023425, .024259, .024987, .023818,
.021768, .019223, .018144, .015282, .012604, .01163, .0097907,
.008336, .0082473, .0079582, .0088077, .009779, .010129, .012145,
.014378, .016761, .01726, .018997, .019998, .019809, .01819,
.016358, .016099, .01617, .017939, .020223, .022521, .02277,
.024279, .025247, .024222, .023989, .023224, .021493, .020362,
.018596, .017309, .015975, .014466, .013171, .011921, .01078,
.0097229, .0087612, .0078729, .0070682, .0063494, .0057156,
.0051459, .0046273, .0041712, .0037686, .0034119, .003095,
.0028126, .0025603, .0023342, .0021314, .0019489, .0017845,
.001636, .0015017, .00138, .0012697, .0011694, .0010782,
9.9507e-4, 9.1931e-4, 8.5013e-4, 7.869e-4, 7.2907e-4, 6.7611e-4,
6.2758e-4, 5.8308e-4, 5.4223e-4, 5.0473e-4, 4.7027e-4, 4.3859e-4,
4.0946e-4, 3.8265e-4, 3.5798e-4, 3.3526e-4, 3.1436e-4, 2.9511e-4,
2.7739e-4, 2.6109e-4, 2.4609e-4, 2.3229e-4, 2.1961e-4, 2.0797e-4,
1.9729e-4, 1.875e-4, 1.7855e-4, 1.7038e-4, 1.6294e-4, 1.5619e-4,
1.5007e-4, 1.4456e-4, 1.3961e-4, 1.3521e-4, 1.3131e-4, 1.2789e-4,
1.2494e-4, 1.2242e-4, 1.2032e-4, 1.1863e-4, 1.1733e-4, 1.1641e-4,
1.1585e-4, 1.1565e-4, 1.158e-4, 1.1629e-4, 1.1712e-4, 1.1827e-4,
1.1976e-4, 1.2158e-4, 1.2373e-4, 1.262e-4, 1.2901e-4, 1.3214e-4,
1.3562e-4, 1.3944e-4, 1.4361e-4, 1.4814e-4, 1.5303e-4, 1.5829e-4,
1.6394e-4, 1.6999e-4, 1.7644e-4, 1.8332e-4, 1.9063e-4, 1.984e-4,
2.0663e-4, 2.1536e-4, 2.246e-4, 2.3436e-4, 2.4468e-4, 2.5558e-4,
2.6708e-4, 2.7921e-4, 2.92e-4, 3.0548e-4, 3.1968e-4, 3.3464e-4,
3.5039e-4, 3.6698e-4, 3.8443e-4, 4.0281e-4, 4.2214e-4, 4.4248e-4,
4.6389e-4, 4.864e-4, 5.1009e-4, 5.3501e-4, 5.6123e-4, 5.888e-4,
6.1781e-4, 6.4833e-4, 6.8043e-4, 7.142e-4, 7.4973e-4, 7.8711e-4,
8.2644e-4, 8.6783e-4, 9.1137e-4, 9.5721e-4, .0010054, .0010562,
.0011096, .0011659, .0012251, .0012875, .0013532, .0014224,
.0014953, .001572, .0016529, .0017381, .0018279, .0019226,
.0020224, .0021277, .0022386, .0023557, .0024792, .0026095,
.002747, .0028921, .0030453, .0032071, .003378, .0035586,
.0037494, .003951, .0041642, .0043897, .0046282, .0048805,
.0051476, .0054304, .00573, .0060473, .0063837, .0067404,
.0071188, .0075203, .0079466, .0083994, .0088806, .0093922,
.0099366, .010516, .011134, .011792, .012494, .013244, .014046,
.014898, .015808, .016781, .017822, .018929, .020108, .02138,
.022729, .02419, .02576, .027412, .029233, .031198, .033301,
.035594, .038092, .040767, .04372, .046918, .050246, .053974,
.058009, .061976, .066586, .071537, .076209, .081856, .087998,
.093821, .10113, .10913, .11731, .12724, .13821, .15025, .1639,
.17807, .19472, .21356, .23496, .25758, .28387, .31389, .34104,
.37469, .40989, .43309, .46845, .5042, .5023, .52981, .55275,
.51075, .51976, .52457, .44779, .44721, .4503, .4243, .45244,
.49491, .55399, .39021, .24802, .2501, .2618, .27475, .28879,
.31317, .33643, .36257, .4018, .43275, .46525, .53333, .56599,
.60557, .70142, .74194, .77736, .88567, .91182, .93294, .98407,
.98772, .99176, .9995, 1.2405, 1.3602, 1.338, 1.3255, 1.3267,
1.3404, 1.3634, 1.3967, 1.4407, 1.4961, 1.5603, 1.6328, 1.7153,
1.8094, 1.9091, 2.018, 2.1367, 2.264, 2.4035, 2.5562, 2.7179,
2.9017, 3.1052, 3.3304, 3.5731, 3.8488, 4.1553, 4.4769, 4.7818,
5.1711, 5.5204, 5.9516, 6.4097, 6.8899, 7.1118, 7.5469, 7.9735,
7.9511, 8.3014, 8.6418, 8.4757, 8.8256, 9.2294, 9.6923, 10.033,
10.842, 11.851, 11.78, 8.8435, 9.1381, 9.5956, 10.076, 10.629,
11.22, 11.883, 12.69, 13.163, 13.974, 14.846, 16.027, 17.053,
18.148, 19.715, 20.907, 22.163, 23.956, 25.235, 26.566, 27.94,
29.576, 30.956, 32.432, 35.337, 39.911, 41.128, 42.625, 44.386,
46.369, 48.619, 51.031, 53.674, 56.825, 59.921, 63.286, 66.929,
70.859, 75.081, 79.618, 84.513, 89.739, 95.335, 101.35, 107.76,
114.63, 121.98, 129.87, 138.3, 147.34, 157.04, 167.56, 178.67,
190.61, 203.43, 217.19, 231.99, 247.88, 264.98, 283.37, 303.17,
324.49, 347.47, 372.25, 398.98, 427.85, 459.06, 492.8, 529.31,
568.89, 611.79, 658.35, 708.91, 763.87, 823.65, 888.72, 959.58,
1036.8, 1121.8, 1213.9, 1314.3, 1423.8, 1543., 1672.8, 1813.4,
1966.1, 2131.4, 2309.5, 2499.3, 2705., 2925.7, 3161.6, 3411.3,
3611.5, 3889.2, 4191.1, 4519.3, 4877.9, 5272.9, 5712.9, 6142.7,
6719.6, 7385., 8145., 8977.7, 9831.9, 10827., 11934., 13063.,
14434., 15878., 17591., 19435., 21510., 23835., 26835., 29740.,
32878., 36305., 39830., 43273., 46931., 50499., 49586., 51598.,
53429., 54619., 55081., 55102., 54485., 53487., 52042., 42689.,
42607., 44020., 47994., 54169., 53916., 55808., 56642., 46049.,
44243., 32929., 30658., 21963., 20835., 15962., 13679., 17652.,
19680., 22388., 25625., 29184., 32520., 35720., 38414., 40523.,
```

```
    49228., 48173., 45678., 41768., 37600., 41313., 42654., 44465.,
    55736., 56630., 65409., 63308., 66572., 61845., 60379., 56777.,
    51920., 46601., 41367., 36529., 32219., 28470., 25192., 22362.,
    19907., 17772., 15907., 14273., 12835., 11567., 10445., 9450.2,
    8565.1, 7776., 7070.8, 6439.2, 5872.3, 5362.4, 4903., 4488.3,
    4113.4, 3773.8, 3465.8, 3186.1, 2931.7, 2700.1, 2488.8, 2296.,
    2119.8, 1958.6, 1810.9, 1675.6, 1551.4, 1437.3, 1332.4, 1236.,
    1147.2, 1065.3, 989.86, 920.22, 855.91, 796.48, 741.53, 690.69,
    643.62, 600.02, 559.6, 522.13, 487.35, 455.06, 425.08, 397.21,
    371.3, 347.2, 324.78, 303.9, 284.46, 266.34, 249.45, 233.7,
    219.01, 205.3, 192.5, 180.55, 169.38, 158.95, 149.2, 140.07,
    131.54, 123.56, 116.09, 109.09, 102.54, 96.405, 90.655, 85.266,
    80.213, 75.475, 71.031, 66.861, 62.948, 59.275, 55.827, 52.587,
    49.544, 46.686, 43.998, 41.473, 39.099, 36.867, 34.768, 32.795,
    30.939, 29.192, 27.546, 25.998, 24.539, 23.164, 21.869, 20.65,
    19.501, 18.419, 17.399, 16.438, 15.532, 14.678, 13.874, 13.115,
    12.4, 11.726, 11.088, 10.488, 9.921, 9.3846, 8.8784, 8.3996,
    7.9469, 7.5197, 7.1174, 6.738, 6.379, 6.0409, 5.7213, 5.419,
    5.1327, 4.8611, 4.6046, 4.3617, 4.1316, 3.9138, 3.7077, 3.5125,
    3.3281, 3.1536, 2.9885, 2.8323, 2.6846, 2.5447, 2.4124, 2.2871,
    2.1686, 2.0564, 1.9501, 1.8495, 1.7543, 1.6641, 1.5787, 1.4978,
    1.4212, 1.3486, 1.2799, 1.2147, 1.1529, 1.0943, 1.0388, .98602,
    .93596, .8886, .84352, .80078, .76029, .722, .68585, .65161,
    .61901, .58808, .55854, .53044, .5039, .47853, .45459, .43173,
    .41008, .38965, .37021, .35186, .33444, .31797, .30234, .28758,
    .2736, .26036, .24764, .2357, .22431, .21342, .20295, .19288,
    .18334, .17444, .166, .15815, .15072, .14348, .13674, .13015,
    .12399, .11807, .11231, .10689, .10164, .096696, .091955,
    .087476, .083183, .079113, .075229, .071536, .068026, .064698,
    .06154, .058544, .055699, .052997, .050431, .047993, .045676,
    .043475, .041382, .039392, .037501, .035702, .033991, .032364,
    .030817, .029345, .027945, .026613, .025345, .024139, .022991,
    .021899, .02086, .019871, .018929, .018033, .01718, .016368,
    .015595, .014859, .014158, .013491, .012856, .012251, .011675,
    .011126, .010604, .010107, .0096331, .009182, .0087523, .0083431,
    .0079533, .0075821, .0072284, .0068915, .0065706, .0062649,
    .0059737, .0056963, .005432, .0051802, .0049404, .0047118,
    .0044941, .0042867, .0040891, .0039009, .0037216, .0035507,
    .003388, .0032329, .0030852, .0029445, .0028105, .0026829,
    .0025613, .0024455, .0023353, .0022303, .0021304, .0020353,
    .0019448, .0018587, .0017767, .0016988, .0016247, .0015543,
    .0014874, .0014238, .0013635, .0013062, .0012519, .0012005,
    .0011517, .0011057, .0010621, .001021, 9.8233e-4, 9.4589e-4,
    9.1167e-4, 8.7961e-4, 8.4964e-4, 8.2173e-4, 7.9582e-4, 7.7189e-4,
    7.499e-4, 7.2983e-4, 7.1167e-4, 6.9542e-4, 6.8108e-4, 6.6866e-4,
    6.5819e-4, 6.4971e-4, 6.4328e-4, 6.3895e-4, 6.3681e-4, 6.3697e-4,
    6.3956e-4, 6.4472e-4, 6.5266e-4, 6.6359e-4, 6.778e-4, 6.9563e-4,
    7.1749e-4, 7.4392e-4, 7.7556e-4, 8.1028e-4, 8.4994e-4, 8.8709e-4,
    9.3413e-4, 9.6953e-4, .0010202, .0010738, .0010976, .0011507,
    .0011686, .0012264, .001291, .0013346, .0014246, .0015293,
    .0016359, .0017824, .0019255, .0020854, .002247, .0024148,
    .0026199, .0027523, .0029704, .0030702, .0033047, .0035013,
    .0037576, .0040275, .0043089, .0046927, .0049307, .0053486,
    .0053809, .0056699, .0059325, .0055488, .005634, .0056392,
    .004946, .0048855, .0048208, .0044386, .0045498, .0046377,
    .0048939, .0052396, .0057324, .0060859, .0066906, .0071148,
    .0077224, .0082687, .008769, .0084471, .008572, .0087729,
    .008775, .0090742, .0080704, .0080288, .0085747, .0086087,
    .0086408, .0088752, .0089381, .0089757, .0093532, .0092824,
    .0092566, .0092645, .0092735, .009342, .0095806, .0097991,
    .010213, .010611, .011129, .011756, .013237, .01412, .015034,
    .015936, .01682, .018597, .019315, .019995, .020658, .021289,
    .022363, .022996, .023716, .024512, .025434, .026067, .027118,
    .028396, .029865, .031442, .033253, .03525, .037296, .039701,
    .042356, .045154, .048059, .051294, .054893, .058636, .061407,
    .065172, .068974, .072676, .073379, .076547, .079556, .079134,
    .082308, .085739, .090192, .09359, .099599, .10669, .11496,
    .1244, .13512, .14752, .14494, .15647, .1668, .17863, .19029,
    .20124, .20254, .21179, .21982, .21625, .22364, .23405, .23382,
    .2434, .25708, .26406, .27621, .28909, .30395, .31717, .33271,
    .3496, .36765, .38774, .40949, .446, .46985, .49846, .5287, .562,
    .59841, .64598, .68834, .7327, .78978, .8373, .88708, .94744,
    1.0006, 1.0574, 1.1215, 1.1856, 1.2546, 1.3292, 1.4107, 1.4974,
    1.5913, 1.6931, 1.8028, 1.9212, 2.0492, 2.1874, 2.3365, 2.4978,
    2.6718, 2.8588, 3.062, 3.2818, 3.5188, 3.7752, 4.0527, 4.3542,
    4.6782, 5.0312, 5.4123, 5.8246, 6.2639, 6.7435, 7.2636, 7.8064,
    8.4091, 9.0696, 9.7677, 10.548, 11.4, 12.309, 13.324, 14.284,
    15.445, 16.687, 18.019, 19.403, 20.847, 22.366, 23.925, 25.537,
    27.213, 28.069, 29.864, 31.829, 33.988, 35.856, 38.829, 42.321,
    46.319, 50.606, 55.126, 59.126, 64.162, 68.708, 74.615, 81.176,
    87.739, 95.494, 103.83, 113.38, 123.99, 135.8, 148.7, 162.58,
    176.32, 192.6, 211.47, 232.7, 252.64, 277.41, 305.38, 333.44,
    366.42, 402.66, 442.14, 484.53, 526.42, 568.15, 558.78, 582.6,
    600.98, 613.94, 619.44, 618.24, 609.84, 595.96, 484.86, 475.59,
    478.49, 501.56, 552.19, 628.44, 630.39, 658.92, 671.96, 562.7,
    545.88, 423.43, 400.14, 306.59, 294.13, 246.8, 226.51, 278.21,
    314.39, 347.22, 389.13, 433.16, 477.48, 521.67, 560.54, 683.6,
```

```
  696.37, 695.91, 683.1, 658.24, 634.89, 698.85, 742.87, 796.66,
  954.49, 1009.5, 1150.5, 1179.1, 1267.9, 1272.4, 1312.7, 1330.4,
  1331.6, 1315.8, 1308.3, 1293.3, 1274.6, 1249.5, 1213.2, 1172.1,
  1124.4, 930.33, 893.36, 871.27, 883.54, 940.76, 1036., 1025.6,
  1053.1, 914.51, 894.15, 865.03, 670.63, 508.41, 475.15, 370.85,
  361.06, 319.38, 312.75, 331.87, 367.13, 415., 467.94, 525.49,
  578.41, 624.66, 794.82, 796.97, 780.29, 736.49, 670.18, 603.75,
  659.67, 679.8, 857.12, 884.05, 900.65, 1046.1, 1141.9, 1083.,
  1089.2, 1e3, 947.08, 872.31, 787.91, 704.75, 624.93, 553.68,
  489.91, 434.21, 385.64, 343.3, 306.42, 274.18, 245.94, 221.11,
  199.23, 179.88, 162.73, 147.48, 133.88, 121.73, 110.86, 101.1,
  92.323, 84.417, 77.281, 70.831, 64.991, 59.694, 54.884, 50.509,
  46.526, 42.893, 39.58, 36.549, 33.776, 31.236, 28.907, 26.77,
  24.805, 23., 21.339, 19.81, 18.404, 17.105, 15.909, 14.801,
  13.778, 12.83, 11.954, 11.142, 10.389, 9.691, 9.0434, 8.4423,
  7.8842, 7.3657, 6.8838, 6.4357, 6.0189, 5.6308, 5.2696, 4.9332,
  4.6198, 4.3277, 4.0553, 3.8012, 3.5639, 3.3424, 3.1355, 2.9422,
  2.7614, 2.5924, 2.4343, 2.2864, 2.148, 2.0184, 1.8971, 1.7835,
  1.677, 1.5773, 1.4838, 1.3961, 1.3139, 1.2369, 1.1645, 1.0966,
  1.0329, .97309, .91686, .86406, .81439, .76767, .72381, .68252,
  .64359, .60695, .57247, .54008, .50957, .48092, .45401, .42862,
  .40465, .38202, .36072, .34052, .3216, .30386, .28711, .27135,
  .25651, .24252, .2293, .21689, .20517, .19416, .18381, .17396,
  .16469
};

static double co2230[2001] = { 2.743e-5, 2.8815e-5, 3.027e-5, 3.1798e-5,
  3.3405e-5, 3.5094e-5, 3.6869e-5, 3.8734e-5, 4.0694e-5, 4.2754e-5,
  4.492e-5, 4.7196e-5, 4.9588e-5, 5.2103e-5, 5.4747e-5, 5.7525e-5,
  6.0446e-5, 6.3516e-5, 6.6744e-5, 7.0137e-5, 7.3704e-5, 7.7455e-5,
  8.1397e-5, 8.5543e-5, 8.9901e-5, 9.4484e-5, 9.9302e-5, 1.0437e-4,
  1.097e-4, 1.153e-4, 1.2119e-4, 1.2738e-4, 1.3389e-4, 1.4074e-4,
  1.4795e-4, 1.5552e-4, 1.6349e-4, 1.7187e-4, 1.8068e-4, 1.8995e-4,
  1.997e-4, 2.0996e-4, 2.2075e-4, 2.321e-4, 2.4403e-4, 2.5659e-4,
  2.698e-4, 2.837e-4, 2.9832e-4, 3.137e-4, 3.2988e-4, 3.4691e-4,
  3.6483e-4, 3.8368e-4, 4.0351e-4, 4.2439e-4, 4.4635e-4, 4.6947e-4,
  4.9379e-4, 5.1939e-4, 5.4633e-4, 5.7468e-4, 6.0452e-4, 6.3593e-4,
  6.69e-4, 7.038e-4, 7.4043e-4, 7.79e-4, 8.1959e-4, 8.6233e-4,
  9.0732e-4, 9.5469e-4, .0010046, .0010571, .0011124, .0011706,
  .0012319, .0012964, .0013644, .001436, .0015114, .0015908,
  .0016745, .0017625, .0018553, .0019531, .002056, .0021645,
  .0022788, .0023992, .002526, .0026596, .0028004, .0029488,
  .0031052, .0032699, .0034436, .0036265, .0038194, .0040227,
  .0042369, .0044628, .0047008, .0049518, .0052164, .0054953,
  .0057894, .0060995, .0064265, .0067713, .007135, .0075184,
  .0079228, .0083494, .0087993, .0092738, .0097745, .010303,
  .01086, .011448, .012068, .012722, .013413, .014142, .014911,
  .015723, .01658, .017484, .018439, .019447, .020511, .021635,
  .022821, .024074, .025397, .026794, .02827, .029829, .031475,
  .033215, .035052, .036994, .039045, .041213, .043504, .045926,
  .048485, .05119, .05405, .057074, .060271, .063651, .067225,
  .071006, .075004, .079233, .083708, .088441, .093449, .098749,
  .10436, .11029, .11657, .12322, .13026, .13772, .14561, .15397,
  .16282, .1722, .18214, .19266, .20381, .21563, .22816, .24143,
  .2555, .27043, .28625, .30303, .32082, .3397, .35972, .38097,
  .40352, .42746, .45286, .47983, .50847, .53888, .57119, .6055,
  .64196, .6807, .72187, .76564, .81217, .86165, .91427, .97025,
  1.0298, 1.0932, 1.1606, 1.2324, 1.3088, 1.3902, 1.477, 1.5693,
  1.6678, 1.7727, 1.8845, 2.0038, 2.131, 2.2666, 2.4114, 2.5659,
  2.7309, 2.907, 3.0951, 3.2961, 3.5109, 3.7405, 3.986, 4.2485,
  4.5293, 4.8299, 5.1516, 5.4961, 5.8651, 6.2605, 6.6842, 7.1385,
  7.6256, 8.1481, 8.7089, 9.3109, 9.9573, 10.652, 11.398, 12.2,
  13.063, 13.992, 14.99, 16.064, 17.222, 18.469, 19.813, 21.263,
  22.828, 24.516, 26.34, 28.31, 30.437, 32.738, 35.226, 37.914,
  40.824, 43.974, 47.377, 51.061, 55.011, 59.299, 63.961, 69.013,
  74.492, 80.444, 86.919, 93.836, 101.23, 109.25, 117.98, 127.47,
  137.81, 149.07, 161.35, 174.75, 189.42, 205.49, 223.02, 242.26,
  263.45, 286.75, 311.94, 340.01, 370.86, 404.92, 440.44, 480.27,
  525.17, 574.71, 626.22, 686.8, 754.38, 827.07, 913.38, 1011.7,
  1121.5, 1161.6, 1289.5, 1432.2, 1595.4, 1777., 1983.3, 2216.1,
  2485.7, 2788.3, 3101.5, 3481., 3902.1, 4257.1, 4740., 5272.8,
  5457.9, 5946.2, 6505.3, 6668.4, 7302.4, 8061.6, 9015.8, 9908.3,
  11613., 13956., 3249.6, 3243., 2901.5, 2841.3, 2729.6, 2558.2,
  1797.8, 1583.2, 1386., 1233.5, 787.74, 701.46, 761.66, 767.21,
  722.83, 1180.6, 1332.1, 1461.6, 2032.9, 2166., 2255.9, 2294.7,
  2587.2, 2396.5, 2122.4, 12553., 10784., 9832.5, 8827.3, 8029.1,
  7377.9, 7347.1, 6783.8, 6239.1, 5721.1, 5503., 4975.1, 4477.8,
  4021.3, 3676.8, 3275.3, 2914.9, 2597.4, 2328.2, 2075.4, 1857.6,
  1663.6, 1493.3, 1343.8, 1213.3, 1095.6, 1066.5, 958.91, 865.15,
  783.31, 714.35, 650.77, 593.98, 546.2, 499.9, 457.87, 421.75,
  387.61, 355.25, 326.62, 299.7, 275.21, 253.17, 232.83, 214.31,
  197.5, 182.08, 167.98, 155.12, 143.32, 132.5, 122.58, 113.48,
  105.11, 97.415, 90.182, 83.463, 77.281, 71.587, 66.341, 61.493,
  57.014, 53.062, 49.21, 45.663, 42.38, 39.348, 36.547, 33.967,
  31.573, 29.357, 27.314, 25.415, 23.658, 22.03, 20.524, 19.125,
  17.829, 16.627, 15.511, 14.476, 13.514, 12.618, 11.786, 11.013,
  10.294, 9.6246, 9.0018, 8.4218, 7.8816, 7.3783, 6.9092, 6.4719,
```

```
6.0641, 5.6838, 5.3289, 4.998, 4.6893, 4.4014, 4.1325, 3.8813,
3.6469, 3.4283, 3.2241, 3.035, 2.8576, 2.6922, 2.5348, 2.3896,
2.2535, 2.1258, 2.0059, 1.8929, 1.7862, 1.6854, 1.5898, 1.4992,
1.4017, 1.3218, 1.2479, 1.1809, 1.1215, 1.0693, 1.0116, .96016,
.9105, .84859, .80105, .74381, .69982, .65127, .60899, .57843,
.54592, .51792, .49336, .47155, .45201, .43426, .41807, .40303,
.38876, .3863, .37098, .35492, .33801, .32032, .30341, .29874,
.29193, .28689, .29584, .29155, .29826, .29195, .29287, .2904,
.28199, .27709, .27162, .26622, .26133, .25676, .25235, .23137,
.22365, .21519, .20597, .19636, .18699, .16485, .16262, .16643,
.17542, .18198, .18631, .16759, .16338, .13505, .1267, .10053,
.092554, .074093, .062159, .055523, .054849, .05401, .05528,
.058982, .07952, .08647, .093244, .099285, .10393, .10661,
.12072, .11417, .10396, .093265, .089137, .088909, .10902,
.11277, .13625, .13565, .14907, .14167, .1428, .13744, .12768,
.11382, .10244, .091686, .08109, .071739, .063616, .056579,
.050504, .045251, .040689, .036715, .033237, .030181, .027488,
.025107, .022998, .021125, .01946, .017979, .016661, .015489,
.014448, .013526, .012712, .011998, .011375, .010839, .010384,
.010007, .0097053, .0094783, .0093257, .0092489, .0092504,
.0093346, .0095077, .0097676, .01012, .01058, .011157, .011844,
.012672, .013665, .014766, .015999, .017509, .018972, .020444,
.022311, .023742, .0249, .025599, .026981, .026462, .025143,
.025066, .022814, .020458, .020026, .019142, .020189, .022371,
.024163, .023728, .02199, .019506, .018591, .015576, .012784,
.011744, .0094777, .0079148, .0070652, .006986, .0071758,
.008086, .0098025, .01087, .013609, .016764, .018137, .021061,
.023498, .023576, .023965, .022828, .021519, .021283, .023364,
.026457, .029782, .030856, .033486, .035515, .035543, .036558,
.037198, .037472, .037045, .037284, .03777, .038085, .038366,
.038526, .038282, .038915, .037697, .035667, .032941, .031959,
.028692, .025918, .024596, .025592, .027873, .028935, .02984,
.028148, .025305, .021912, .020454, .016732, .013357, .01205,
.009731, .0079881, .0077704, .0074387, .0083895, .0096776,
.010326, .01293, .015955, .019247, .020145, .02267, .024231,
.024184, .022131, .019784, .01955, .01971, .022119, .025116,
.027978, .028107, .029808, .030701, .029164, .028551, .027286,
.024946, .023259, .020982, .019221, .017471, .015643, .014074,
.01261, .011301, .010116, .0090582, .0081036, .0072542, .0065034,
.0058436, .0052571, .0047321, .0042697, .0038607, .0034977,
.0031747, .0028864, .0026284, .002397, .002189, .0020017,
.0018326, .0016798, .0015414, .0014159, .0013019, .0011983,
.0011039, .0010177, 9.391e-4, 8.6717e-4, 8.0131e-4, 7.4093e-4,
6.8553e-4, 6.3464e-4, 5.8787e-4, 5.4487e-4, 5.0533e-4, 4.69e-4,
4.3556e-4, 4.0474e-4, 3.7629e-4, 3.5e-4, 3.2569e-4, 3.032e-4,
2.8239e-4, 2.6314e-4, 2.4535e-4, 2.2891e-4, 2.1374e-4, 1.9975e-4,
1.8685e-4, 1.7498e-4, 1.6406e-4, 1.5401e-4, 1.4479e-4, 1.3633e-4,
1.2858e-4, 1.2148e-4, 1.1499e-4, 1.0907e-4, 1.0369e-4, 9.8791e-5,
9.4359e-5, 9.0359e-5, 8.6766e-5, 8.3555e-5, 8.0703e-5, 7.8192e-5,
7.6003e-5, 7.4119e-5, 7.2528e-5, 7.1216e-5, 7.0171e-5, 6.9385e-5,
6.8848e-5, 6.8554e-5, 6.8496e-5, 6.8669e-5, 6.9069e-5, 6.9694e-5,
7.054e-5, 7.1608e-5, 7.2896e-5, 7.4406e-5, 7.6139e-5, 7.8097e-5,
8.0283e-5, 8.2702e-5, 8.5357e-5, 8.8255e-5, 9.1402e-5, 9.4806e-5,
9.8473e-5, 1.0241e-4, 1.0664e-4, 1.1115e-4, 1.1598e-4, 1.2112e-4,
1.2659e-4, 1.3241e-4, 1.3859e-4, 1.4515e-4, 1.521e-4, 1.5947e-4,
1.6728e-4, 1.7555e-4, 1.8429e-4, 1.9355e-4, 2.0334e-4, 2.1369e-4,
2.2463e-4, 2.3619e-4, 2.4841e-4, 2.6132e-4, 2.7497e-4, 2.8938e-4,
3.0462e-4, 3.2071e-4, 3.3771e-4, 3.5567e-4, 3.7465e-4, 3.947e-4,
4.1588e-4, 4.3828e-4, 4.6194e-4, 4.8695e-4, 5.1338e-4, 5.4133e-4,
5.7087e-4, 6.0211e-4, 6.3515e-4, 6.701e-4, 7.0706e-4, 7.4617e-4,
7.8756e-4, 8.3136e-4, 8.7772e-4, 9.2681e-4, 9.788e-4, .0010339,
.0010922, .001154, .0012195, .0012889, .0013626, .0014407,
.0015235, .0016114, .0017048, .0018038, .001909, .0020207,
.0021395, .0022657, .0023998, .0025426, .0026944, .002856,
.0030281, .0032114, .0034068, .003615, .0038371, .004074,
.004327, .0045971, .0048857, .0051942, .0055239, .0058766,
.0062538, .0066573, .0070891, .007551, .0080455, .0085747,
.0091412, .0097481, .010397, .011092, .011837, .012638, .013495,
.014415, .01541, .016475, .017621, .018857, .020175, .02162,
.023185, .024876, .02672, .028732, .030916, .033319, .035939,
.038736, .041847, .04524, .048715, .052678, .056977, .061203,
.066184, .07164, .076952, .083477, .090674, .098049, .10697,
.1169, .1277, .14011, .15323, .1684, .18601, .20626, .22831,
.25417, .28407, .31405, .34957, .38823, .41923, .46026, .50409,
.51227, .54805, .57976, .53818, .55056, .557, .46741, .46403,
.4636, .42265, .45166, .49852, .56663, .34306, .17779, .17697,
.18346, .19129, .20014, .21778, .23604, .25649, .28676, .31238,
.33856, .39998, .4288, .46568, .56654, .60786, .64473, .76466,
.7897, .80778, .86443, .85736, .84798, .84157, 1.1385, 1.2446,
1.1923, 1.1552, 1.1338, 1.1266, 1.1292, 1.1431, 1.1683, 1.2059,
1.2521, 1.3069, 1.3712, 1.4471, 1.5275, 1.6165, 1.7145, 1.8189,
1.9359, 2.065, 2.2007, 2.3591, 2.5362, 2.7346, 2.9515, 3.2021,
3.4851, 3.7935, 4.0694, 4.4463, 4.807, 5.2443, 5.7178, 6.2231,
6.4796, 6.9461, 7.4099, 7.3652, 7.7182, 8.048, 7.7373, 8.0363,
8.3855, 8.8044, 9.0257, 9.8574, 10.948, 10.563, 6.8979, 7.0744,
7.4121, 7.7663, 8.1768, 8.6243, 9.1437, 9.7847, 10.182, 10.849,
11.572, 12.602, 13.482, 14.431, 15.907, 16.983, 18.11, 19.884,
```

```
21.02, 22.18, 23.355, 24.848, 25.954, 27.13, 30.186, 34.893,
35.682, 36.755, 38.111, 39.703, 41.58, 43.606, 45.868, 48.573,
51.298, 54.291, 57.559, 61.116, 64.964, 69.124, 73.628, 78.471,
83.683, 89.307, 95.341, 101.84, 108.83, 116.36, 124.46, 133.18,
142.57, 152.79, 163.69, 175.43, 188.11, 201.79, 216.55, 232.51,
249.74, 268.38, 288.54, 310.35, 333.97, 359.55, 387.26, 417.3,
449.88, 485.2, 523.54, 565.14, 610.28, 659.31, 712.56, 770.43,
833.36, 901.82, 976.36, 1057.6, 1146.8, 1243.8, 1350., 1466.3,
1593.6, 1732.7, 1884.1, 2049.1, 2228.2, 2421.9, 2629.4, 2853.7,
3094.4, 3351.1, 3622.3, 3829.8, 4123.1, 4438.3, 4777.2, 5144.1,
5545.4, 5990.5, 6404.5, 6996.8, 7687.6, 8482.9, 9349.4, 10203.,
11223., 12358., 13493., 14916., 16416., 18236., 20222., 22501.,
25102., 28358., 31707., 35404., 39538., 43911., 48391., 53193.,
58028., 58082., 61276., 64193., 66294., 67480., 67921., 67423.,
66254., 64341., 51737., 51420., 53072., 58145., 66195., 65358.,
67377., 67869., 53509., 50553., 35737., 32425., 21704., 19974.,
14457., 12142., 16798., 19489., 23049., 27270., 31910., 36457.,
40877., 44748., 47876., 59793., 58626., 55454., 50337., 44893.,
50228., 52216., 54747., 69541., 70455., 81014., 77694., 80533.,
73953., 70927., 65539., 59002., 52281., 45953., 40292., 35360.,
31124., 27478., 24346., 21647., 19308., 17271., 15491., 13927.,
12550., 11331., 10250., 9288.8, 8431.4, 7664.9, 6978.3, 6361.8,
5807.4, 5307.7, 4856.8, 4449., 4079.8, 3744.9, 3440.8, 3164.2,
2912.3, 2682.7, 2473., 2281.4, 2106., 1945.3, 1797.9, 1662.5,
1538.1, 1423.6, 1318.1, 1221., 1131.5, 1049., 972.99, 902.87,
838.01, 777.95, 722.2, 670.44, 622.35, 577.68, 536.21, 497.76,
462.12, 429.13, 398.61, 370.39, 344.29, 320.16, 297.85, 277.2,
258.08, 240.38, 223.97, 208.77, 194.66, 181.58, 169.43, 158.15,
147.67, 137.92, 128.86, 120.44, 112.6, 105.3, 98.499, 92.166,
86.264, 80.763, 75.632, 70.846, 66.381, 62.213, 58.321, 54.685,
51.288, 48.114, 45.145, 42.368, 39.772, 37.341, 35.065, 32.937,
30.943, 29.077, 27.33, 25.693, 24.158, 22.717, 21.367, 20.099,
18.909, 17.792, 16.744, 15.761, 14.838, 13.971, 13.157, 12.393,
11.676, 11.003, 10.369, 9.775, 9.2165, 8.6902, 8.1963, 7.7314,
7.2923, 6.8794, 6.4898, 6.122, 5.7764, 5.4525, 5.1484, 4.8611,
4.5918, 4.3379, 4.0982, 3.8716, 3.6567, 3.4545, 3.2634, 3.0828,
2.9122, 2.7512, 2.5993, 2.4561, 2.3211, 2.1938, 2.0737, 1.9603,
1.8534, 1.7525, 1.6572, 1.5673, 1.4824, 1.4022, 1.3265, 1.2551,
1.1876, 1.1239, 1.0637, 1.0069, .9532, .90248, .85454, .80921,
.76631, .72569, .6872, .65072, .61635, .5836, .55261, .52336,
.49581, .46998, .44559, .42236, .40036, .37929, .35924, .34043,
.32238, .30547, .28931, .27405, .25975, .24616, .23341, .22133,
.20997, .19924, .18917, .17967, .17075, .16211, .15411, .14646,
.13912, .13201, .12509, .11857, .11261, .10698, .10186, .097039,
.092236, .087844, .083443, .07938, .075452, .071564, .067931,
.064389, .061078, .057901, .054921, .052061, .049364, .046789,
.04435, .042044, .039866, .037808, .035863, .034023, .032282,
.030634, .029073, .027595, .026194, .024866, .023608, .022415,
.021283, .02021, .019193, .018228, .017312, .016443, .015619,
.014837, .014094, .01339, .012721, .012086, .011483, .010911,
.010368, .009852, .0093623, .0088972, .0084556, .0080362,
.0076379, .0072596, .0069003, .006559, .0062349, .0059269,
.0056344, .0053565, .0050925, .0048417, .0046034, .004377,
.0041618, .0039575, .0037633, .0035788, .0034034, .0032368,
.0030785, .002928, .0027851, .0026492, .0025201, .0023975,
.0022809, .0021701, .0020649, .0019649, .0018699, .0017796,
.0016938, .0016122, .0015348, .0014612, .0013913, .001325,
.0012619, .0012021, .0011452, .0010913, .0010401, 9.9149e-4,
9.454e-4, 9.0169e-4, 8.6024e-4, 8.2097e-4, 7.8377e-4, 7.4854e-4,
7.1522e-4, 6.8371e-4, 6.5393e-4, 6.2582e-4, 5.9932e-4, 5.7435e-4,
5.5087e-4, 5.2882e-4, 5.0814e-4, 4.8881e-4, 4.7076e-4, 4.5398e-4,
4.3843e-4, 4.2407e-4, 4.109e-4, 3.9888e-4, 3.88e-4, 3.7826e-4,
3.6963e-4, 3.6213e-4, 3.5575e-4, 3.505e-4, 3.464e-4, 3.4346e-4,
3.4173e-4, 3.4125e-4, 3.4206e-4, 3.4424e-4, 3.4787e-4, 3.5303e-4,
3.5986e-4, 3.6847e-4, 3.7903e-4, 3.9174e-4, 4.0681e-4, 4.2455e-4,
4.4527e-4, 4.6942e-4, 4.9637e-4, 5.2698e-4, 5.5808e-4, 5.9514e-4,
6.2757e-4, 6.689e-4, 7.1298e-4, 7.3955e-4, 7.8403e-4, 8.0449e-4,
8.5131e-4, 9.0256e-4, 9.3692e-4, .0010051, .0010846, .0011678,
.001282, .0014016, .0015355, .0016764, .0018272, .0020055,
.0021455, .0023421, .0024615, .0026786, .0028787, .0031259,
.0034046, .0036985, .0040917, .0043902, .0048349, .0049531,
.0052989, .0056148, .0052452, .0053357, .005333, .0045069,
.0043851, .004253, .003738, .0038084, .0039013, .0041505,
.0045372, .0050569, .0054507, .0061267, .0066122, .0072449,
.0078012, .0082651, .0076538, .0076573, .0076806, .0075227,
.0076269, .0063758, .006254, .0067749, .0067909, .0068231,
.0072143, .0072762, .0072954, .007679, .0075107, .0073658,
.0072441, .0071074, .0070378, .007176, .0072472, .0075844,
.0079291, .008412, .0090165, .010688, .011535, .012375, .013166,
.013895, .015567, .016011, .016392, .016737, .017043, .017731,
.018031, .018419, .018877, .019474, .019868, .020604, .021538,
.022653, .023869, .025288, .026879, .028547, .030524, .03274,
.035132, .03769, .040567, .043793, .047188, .049962, .053542,
.057205, .060776, .061489, .064419, .067124, .065945, .068487,
.071209, .074783, .077039, .082444, .08902, .09692, .10617,
.11687, .12952, .12362, .13498, .14412, .15492, .16519, .1744,
.17096, .17714, .18208, .17363, .17813, .18564, .18295, .19045,
```

```
    .20252, .20815, .21844, .22929, .24229, .25321, .26588, .2797,
    .29465, .31136, .32961, .36529, .38486, .41027, .43694, .4667,
    .49943, .54542, .58348, .62303, .67633, .71755, .76054, .81371,
    .85934, .90841, .96438, 1.0207, 1.0821, 1.1491, 1.2226, 1.3018,
    1.388, 1.4818, 1.5835, 1.6939, 1.8137, 1.9435, 2.0843, 2.237,
    2.4026, 2.5818, 2.7767, 2.9885, 3.2182, 3.4679, 3.7391, 4.0349,
    4.3554, 4.7053, 5.0849, 5.4986, 5.9436, 6.4294, 6.9598, 7.5203,
    8.143, 8.8253, 9.5568, 10.371, 11.267, 12.233, 13.31, 14.357,
    15.598, 16.93, 18.358, 19.849, 21.408, 23.04, 24.706, 26.409,
    28.153, 28.795, 30.549, 32.43, 34.49, 36.027, 38.955, 42.465,
    46.565, 50.875, 55.378, 59.002, 63.882, 67.949, 73.693, 80.095,
    86.403, 94.264, 102.65, 112.37, 123.3, 135.54, 149.14, 163.83,
    179.17, 196.89, 217.91, 240.94, 264.13, 292.39, 324.83, 358.21,
    397.16, 440.5, 488.6, 541.04, 595.3, 650.43, 652.03, 688.74,
    719.47, 743.54, 757.68, 762.35, 756.43, 741.42, 595.43, 580.97,
    580.83, 605.68, 667.88, 764.49, 759.93, 789.12, 798.17, 645.66,
    615.65, 455.05, 421.09, 306.45, 289.14, 235.7, 215.52, 274.57,
    316.53, 357.73, 409.89, 465.06, 521.84, 579.02, 630.64, 794.46,
    813., 813.56, 796.25, 761.57, 727.97, 812.14, 866.75, 932.5,
    1132.8, 1194.8, 1362.2, 1387.2, 1482.3, 1479.7, 1517.9, 1533.1,
    1534.2, 1523.3, 1522.5, 1515.5, 1505.2, 1486.5, 1454., 1412.,
    1358.8, 1107.8, 1060.9, 1033.5, 1048.2, 1122.4, 1248.9, 1227.1,
    1255.4, 1058.9, 1020.7, 970.59, 715.24, 512.56, 468.47, 349.3,
    338.26, 299.22, 301.26, 332.38, 382.08, 445.49, 515.87, 590.85,
    662.3, 726.05, 955.59, 964.11, 945.17, 891.48, 807.11, 720.9,
    803.36, 834.46, 1073.9, 1107.1, 1123.6, 1296., 1393.7, 1303.1,
    1284.3, 1161.8, 1078.8, 976.13, 868.72, 767.4, 674.72, 593.73,
    523.12, 462.24, 409.75, 364.34, 325., 290.73, 260.76, 234.46,
    211.28, 190.78, 172.61, 156.44, 142.01, 129.12, 117.57, 107.2,
    97.877, 89.47, 81.882, 75.021, 68.807, 63.171, 58.052, 53.396,
    49.155, 45.288, 41.759, 38.531, 35.576, 32.868, 30.384, 28.102,
    26.003, 24.071, 22.293, 20.655, 19.147, 17.756, 16.476, 15.292,
    14.198, 13.183, 12.241, 11.367, 10.554, 9.7989, 9.0978, 8.4475,
    7.845, 7.2868, 6.7704, 6.2927, 5.8508, 5.4421, 5.064, 4.714,
    4.3902, 4.0902, 3.8121, 3.5543, 3.315, 3.093, 2.8869, 2.6953,
    2.5172, 2.3517, 2.1977, 2.0544, 1.9211, 1.7969, 1.6812, 1.5735,
    1.4731, 1.3794, 1.2921, 1.2107, 1.1346, 1.0637, .99744, .93554,
    .87771, .82368, .77313, .72587, .6816, .64014, .60134, .565,
    .53086, .49883, .46881, .44074, .4144, .38979, .36679, .34513,
    .32474, .30552, .28751, .27045, .25458, .23976, .22584, .21278,
    .20051, .18899, .17815, .16801, .15846, .14954, .14117, .13328,
    .12584
  };

  /* Get CO2 continuum absorption... */
  const double xw = nu / 2 + 1;
  if (xw >= 1 && xw < 2001) {
    const int iw = (int) xw;
    const double dw = xw - iw;
    const double ew = 1 - dw;
    const double cw296 = ew * co2296[iw - 1] + dw * co2296[iw];
    const double cw260 = ew * co2260[iw - 1] + dw * co2260[iw];
    const double cw230 = ew * co2230[iw - 1] + dw * co2230[iw];
    const double dt230 = t - 230;
    const double dt260 = t - 260;
    const double dt296 = t - 296;
    const double ctw =
      dt260 * 5.050505e-4 * dt296 * cw230 -
      dt230 * 9.259259e-4 * dt296 * cw260 +
      dt230 * 4.208754e-4 * dt260 * cw296;
    return u / NA / 1000 * p / P0 * ctw;
  } else
    return 0;
}
```

ctmh2o()

```
double ctmh2o (
        const double nu,
        const double p,
        const double t,
        const double q,
        const double u )
```

Compute water vapor continuum (optical depth).

Definition at line 1746 of file jurassic.c.

```
                   {
```

```
static double h2o296[2001] = { .17, .1695, .172, .168, .1687, .1624, .1606,
  .1508, .1447, .1344, .1214, .1133, .1009, .09217, .08297, .06989,
  .06513, .05469, .05056, .04417, .03779, .03484, .02994, .0272,
  .02325, .02063, .01818, .01592, .01405, .01251, .0108, .009647,
  .008424, .007519, .006555, .00588, .005136, .004511, .003989,
  .003509, .003114, .00274, .002446, .002144, .001895, .001676,
  .001486, .001312, .001164, .001031, 9.129e-4, 8.106e-4, 7.213e-4,
  6.4e-4, 5.687e-4, 5.063e-4, 4.511e-4, 4.029e-4, 3.596e-4,
  3.22e-4, 2.889e-4, 2.597e-4, 2.337e-4, 2.108e-4, 1.907e-4,
  1.728e-4, 1.57e-4, 1.43e-4, 1.305e-4, 1.195e-4, 1.097e-4,
  1.009e-4, 9.307e-5, 8.604e-5, 7.971e-5, 7.407e-5, 6.896e-5,
  6.433e-5, 6.013e-5, 5.631e-5, 5.283e-5, 4.963e-5, 4.669e-5,
  4.398e-5, 4.148e-5, 3.917e-5, 3.702e-5, 3.502e-5, 3.316e-5,
  3.142e-5, 2.978e-5, 2.825e-5, 2.681e-5, 2.546e-5, 2.419e-5,
  2.299e-5, 2.186e-5, 2.079e-5, 1.979e-5, 1.884e-5, 1.795e-5,
  1.711e-5, 1.633e-5, 1.559e-5, 1.49e-5, 1.426e-5, 1.367e-5,
  1.312e-5, 1.263e-5, 1.218e-5, 1.178e-5, 1.143e-5, 1.112e-5,
  1.088e-5, 1.07e-5, 1.057e-5, 1.05e-5, 1.051e-5, 1.059e-5,
  1.076e-5, 1.1e-5, 1.133e-5, 1.18e-5, 1.237e-5, 1.308e-5,
  1.393e-5, 1.483e-5, 1.614e-5, 1.758e-5, 1.93e-5, 2.123e-5,
  2.346e-5, 2.647e-5, 2.93e-5, 3.279e-5, 3.745e-5, 4.152e-5,
  4.813e-5, 5.477e-5, 6.203e-5, 7.331e-5, 8.056e-5, 9.882e-5,
  1.05e-4, 1.21e-4, 1.341e-4, 1.572e-4, 1.698e-4, 1.968e-4,
  2.175e-4, 2.431e-4, 2.735e-4, 2.867e-4, 3.19e-4, 3.371e-4,
  3.554e-4, 3.726e-4, 3.837e-4, 3.878e-4, 3.864e-4, 3.858e-4,
  3.841e-4, 3.852e-4, 3.815e-4, 3.762e-4, 3.618e-4, 3.579e-4,
  3.45e-4, 3.202e-4, 3.018e-4, 2.785e-4, 2.602e-4, 2.416e-4,
  2.097e-4, 1.939e-4, 1.689e-4, 1.498e-4, 1.308e-4, 1.17e-4,
  1.011e-4, 9.237e-5, 7.909e-5, 7.006e-5, 6.112e-5, 5.401e-5,
  4.914e-5, 4.266e-5, 3.963e-5, 3.316e-5, 3.037e-5, 2.598e-5,
  2.294e-5, 2.066e-5, 1.813e-5, 1.583e-5, 1.423e-5, 1.247e-5,
  1.116e-5, 9.76e-6, 8.596e-6, 7.72e-6, 6.825e-6, 6.108e-6,
  5.366e-6, 4.733e-6, 4.229e-6, 3.731e-6, 3.346e-6, 2.972e-6,
  2.628e-6, 2.356e-6, 2.102e-6, 1.878e-6, 1.678e-6, 1.507e-6,
  1.348e-6, 1.21e-6, 1.089e-6, 9.806e-7, 8.857e-7, 8.004e-7,
  7.261e-7, 6.599e-7, 6.005e-7, 5.479e-7, 5.011e-7, 4.595e-7,
  4.219e-7, 3.885e-7, 3.583e-7, 3.314e-7, 3.071e-7, 2.852e-7,
  2.654e-7, 2.474e-7, 2.311e-7, 2.162e-7, 2.026e-7, 1.902e-7,
  1.788e-7, 1.683e-7, 1.587e-7, 1.497e-7, 1.415e-7, 1.338e-7,
  1.266e-7, 1.2e-7, 1.138e-7, 1.08e-7, 1.027e-7, 9.764e-8,
  9.296e-8, 8.862e-8, 8.458e-8, 8.087e-8, 7.744e-8, 7.429e-8,
  7.145e-8, 6.893e-8, 6.664e-8, 6.468e-8, 6.322e-8, 6.162e-8,
  6.07e-8, 5.992e-8, 5.913e-8, 5.841e-8, 5.796e-8, 5.757e-8,
  5.746e-8, 5.731e-8, 5.679e-8, 5.577e-8, 5.671e-8, 5.656e-8,
  5.594e-8, 5.593e-8, 5.602e-8, 5.62e-8, 5.693e-8, 5.725e-8,
  5.858e-8, 6.037e-8, 6.249e-8, 6.535e-8, 6.899e-8, 7.356e-8,
  7.918e-8, 8.618e-8, 9.385e-8, 1.039e-7, 1.158e-7, 1.29e-7,
  1.437e-7, 1.65e-7, 1.871e-7, 2.121e-7, 2.427e-7, 2.773e-7,
  3.247e-7, 3.677e-7, 4.037e-7, 4.776e-7, 5.101e-7, 6.214e-7,
  6.936e-7, 7.581e-7, 8.486e-7, 9.355e-7, 9.942e-7, 1.063e-6,
  1.123e-6, 1.191e-6, 1.215e-6, 1.247e-6, 1.26e-6, 1.271e-6,
  1.284e-6, 1.317e-6, 1.323e-6, 1.349e-6, 1.353e-6, 1.362e-6,
  1.344e-6, 1.329e-6, 1.336e-6, 1.327e-6, 1.325e-6, 1.359e-6,
  1.374e-6, 1.415e-6, 1.462e-6, 1.526e-6, 1.619e-6, 1.735e-6,
  1.863e-6, 2.034e-6, 2.265e-6, 2.482e-6, 2.756e-6, 3.103e-6,
  3.466e-6, 3.832e-6, 4.378e-6, 4.913e-6, 5.651e-6, 6.311e-6,
  7.169e-6, 8.057e-6, 9.253e-6, 1.047e-5, 1.212e-5, 1.36e-5,
  1.569e-5, 1.776e-5, 2.02e-5, 2.281e-5, 2.683e-5, 2.994e-5,
  3.488e-5, 3.896e-5, 4.499e-5, 5.175e-5, 6.035e-5, 6.34e-5,
  7.281e-5, 7.923e-5, 8.348e-5, 9.631e-5, 1.044e-4, 1.102e-4,
  1.176e-4, 1.244e-4, 1.283e-4, 1.326e-4, 1.4e-4, 1.395e-4,
  1.387e-4, 1.363e-4, 1.314e-4, 1.241e-4, 1.228e-4, 1.148e-4,
  1.086e-4, 1.018e-4, 8.89e-5, 8.316e-5, 7.292e-5, 6.452e-5,
  5.625e-5, 5.045e-5, 4.38e-5, 3.762e-5, 3.29e-5, 2.836e-5,
  2.485e-5, 2.168e-5, 1.895e-5, 1.659e-5, 1.453e-5, 1.282e-5,
  1.132e-5, 1.001e-5, 8.836e-6, 7.804e-6, 6.922e-6, 6.116e-6,
  5.429e-6, 4.824e-6, 4.278e-6, 3.788e-6, 3.371e-6, 2.985e-6,
  2.649e-6, 2.357e-6, 2.09e-6, 1.858e-6, 1.647e-6, 1.462e-6,
  1.299e-6, 1.155e-6, 1.028e-6, 9.142e-7, 8.132e-7, 7.246e-7,
  6.451e-7, 5.764e-7, 5.151e-7, 4.603e-7, 4.121e-7, 3.694e-7,
  3.318e-7, 2.985e-7, 2.69e-7, 2.428e-7, 2.197e-7, 1.992e-7,
  1.81e-7, 1.649e-7, 1.506e-7, 1.378e-7, 1.265e-7, 1.163e-7,
  1.073e-7, 9.918e-8, 9.191e-8, 8.538e-8, 7.949e-8, 7.419e-8,
  6.94e-8, 6.508e-8, 6.114e-8, 5.761e-8, 5.437e-8, 5.146e-8,
  4.89e-8, 4.636e-8, 4.406e-8, 4.201e-8, 4.015e-8, 3.84e-8,
  3.661e-8, 3.51e-8, 3.377e-8, 3.242e-8, 3.13e-8, 3.015e-8,
  2.918e-8, 2.83e-8, 2.758e-8, 2.707e-8, 2.656e-8, 2.619e-8,
  2.609e-8, 2.615e-8, 2.63e-8, 2.675e-8, 2.745e-8, 2.842e-8,
  2.966e-8, 3.125e-8, 3.318e-8, 3.565e-8, 3.85e-8, 4.191e-8,
  4.59e-8, 5.059e-8, 5.607e-8, 6.239e-8, 6.958e-8, 7.796e-8,
  8.773e-8, 9.88e-8, 1.114e-7, 1.258e-7, 1.422e-7, 1.61e-7,
  1.822e-7, 2.06e-7, 2.337e-7, 2.645e-7, 2.996e-7, 3.393e-7,
  3.843e-7, 4.363e-7, 4.935e-7, 5.607e-7, 6.363e-7, 7.242e-7,
  8.23e-7, 9.411e-7, 1.071e-6, 1.232e-6, 1.402e-6, 1.6e-6, 1.82e-6,
  2.128e-6, 2.386e-6, 2.781e-6, 3.242e-6, 3.653e-6, 4.323e-6,
  4.747e-6, 5.321e-6, 5.919e-6, 6.681e-6, 7.101e-6, 7.983e-6,
```

```
8.342e-6, 8.741e-6, 9.431e-6, 9.952e-6, 1.026e-5, 1.055e-5,
1.095e-5, 1.095e-5, 1.087e-5, 1.056e-5, 1.026e-5, 9.715e-6,
9.252e-6, 8.452e-6, 7.958e-6, 7.268e-6, 6.295e-6, 6.003e-6, 5e-6,
4.591e-6, 3.983e-6, 3.479e-6, 3.058e-6, 2.667e-6, 2.293e-6,
1.995e-6, 1.747e-6, 1.517e-6, 1.335e-6, 1.165e-6, 1.028e-6,
9.007e-7, 7.956e-7, 7.015e-7, 6.192e-7, 5.491e-7, 4.859e-7,
4.297e-7, 3.799e-7, 3.38e-7, 3.002e-7, 2.659e-7, 2.366e-7,
2.103e-7, 1.861e-7, 1.655e-7, 1.469e-7, 1.309e-7, 1.162e-7,
1.032e-7, 9.198e-8, 8.181e-8, 7.294e-8, 6.516e-8, 5.787e-8,
5.163e-8, 4.612e-8, 4.119e-8, 3.695e-8, 3.308e-8, 2.976e-8,
2.67e-8, 2.407e-8, 2.171e-8, 1.965e-8, 1.78e-8, 1.617e-8,
1.47e-8, 1.341e-8, 1.227e-8, 1.125e-8, 1.033e-8, 9.524e-9,
8.797e-9, 8.162e-9, 7.565e-9, 7.04e-9, 6.56e-9, 6.129e-9,
5.733e-9, 5.376e-9, 5.043e-9, 4.75e-9, 4.466e-9, 4.211e-9,
3.977e-9, 3.759e-9, 3.558e-9, 3.373e-9, 3.201e-9, 3.043e-9,
2.895e-9, 2.76e-9, 2.635e-9, 2.518e-9, 2.411e-9, 2.314e-9,
2.23e-9, 2.151e-9, 2.087e-9, 2.035e-9, 1.988e-9, 1.946e-9,
1.927e-9, 1.916e-9, 1.916e-9, 1.933e-9, 1.966e-9, 2.018e-9,
2.09e-9, 2.182e-9, 2.299e-9, 2.442e-9, 2.623e-9, 2.832e-9,
3.079e-9, 3.368e-9, 3.714e-9, 4.104e-9, 4.567e-9, 5.091e-9,
5.701e-9, 6.398e-9, 7.194e-9, 8.127e-9, 9.141e-9, 1.035e-8,
1.177e-8, 1.338e-8, 1.508e-8, 1.711e-8, 1.955e-8, 2.216e-8,
2.534e-8, 2.871e-8, 3.291e-8, 3.711e-8, 4.285e-8, 4.868e-8,
5.509e-8, 6.276e-8, 7.262e-8, 8.252e-8, 9.4e-8, 1.064e-7,
1.247e-7, 1.411e-7, 1.626e-7, 1.827e-7, 2.044e-7, 2.284e-7,
2.452e-7, 2.854e-7, 3.026e-7, 3.278e-7, 3.474e-7, 3.693e-7,
3.93e-7, 4.104e-7, 4.22e-7, 4.439e-7, 4.545e-7, 4.778e-7,
4.812e-7, 5.018e-7, 4.899e-7, 5.075e-7, 5.073e-7, 5.171e-7,
5.131e-7, 5.25e-7, 5.617e-7, 5.846e-7, 6.239e-7, 6.696e-7,
7.398e-7, 8.073e-7, 9.15e-7, 1.009e-6, 1.116e-6, 1.264e-6,
1.439e-6, 1.644e-6, 1.856e-6, 2.147e-6, 2.317e-6, 2.713e-6,
2.882e-6, 2.99e-6, 3.489e-6, 3.581e-6, 4.033e-6, 4.26e-6,
4.543e-6, 4.84e-6, 4.826e-6, 5.013e-6, 5.252e-6, 5.277e-6,
5.306e-6, 5.236e-6, 5.123e-6, 5.171e-6, 4.843e-6, 4.615e-6,
4.385e-6, 3.97e-6, 3.693e-6, 3.231e-6, 2.915e-6, 2.495e-6,
2.144e-6, 1.91e-6, 1.639e-6, 1.417e-6, 1.226e-6, 1.065e-6,
9.29e-7, 8.142e-7, 7.161e-7, 6.318e-7, 5.581e-7, 4.943e-7,
4.376e-7, 3.884e-7, 3.449e-7, 3.06e-7, 2.712e-7, 2.412e-7,
2.139e-7, 1.903e-7, 1.689e-7, 1.499e-7, 1.331e-7, 1.183e-7,
1.05e-7, 9.362e-8, 8.306e-8, 7.403e-8, 6.578e-8, 5.853e-8,
5.216e-8, 4.632e-8, 4.127e-8, 3.678e-8, 3.279e-8, 2.923e-8,
2.612e-8, 2.339e-8, 2.094e-8, 1.877e-8, 1.686e-8, 1.516e-8,
1.366e-8, 1.234e-8, 1.114e-8, 1.012e-8, 9.182e-9, 8.362e-9,
7.634e-9, 6.981e-9, 6.406e-9, 5.888e-9, 5.428e-9, 5.021e-9,
4.65e-9, 4.326e-9, 4.033e-9, 3.77e-9, 3.536e-9, 3.327e-9,
3.141e-9, 2.974e-9, 2.825e-9, 2.697e-9, 2.584e-9, 2.488e-9,
2.406e-9, 2.34e-9, 2.292e-9, 2.259e-9, 2.244e-9, 2.243e-9,
2.272e-9, 2.31e-9, 2.378e-9, 2.454e-9, 2.618e-9, 2.672e-9,
2.831e-9, 3.05e-9, 3.225e-9, 3.425e-9, 3.677e-9, 3.968e-9,
4.221e-9, 4.639e-9, 4.96e-9, 5.359e-9, 5.649e-9, 6.23e-9,
6.716e-9, 7.218e-9, 7.746e-9, 7.988e-9, 8.627e-9, 8.999e-9,
9.442e-9, 9.82e-9, 1.015e-8, 1.06e-8, 1.079e-8, 1.109e-8,
1.137e-8, 1.186e-8, 1.18e-8, 1.187e-8, 1.194e-8, 1.192e-8,
1.224e-8, 1.245e-8, 1.246e-8, 1.318e-8, 1.377e-8, 1.471e-8,
1.582e-8, 1.713e-8, 1.853e-8, 2.063e-8, 2.27e-8, 2.567e-8,
2.891e-8, 3.264e-8, 3.744e-8, 4.286e-8, 4.915e-8, 5.623e-8,
6.336e-8, 7.293e-8, 8.309e-8, 9.319e-8, 1.091e-7, 1.243e-7,
1.348e-7, 1.449e-7, 1.62e-7, 1.846e-7, 1.937e-7, 2.04e-7,
2.179e-7, 2.298e-7, 2.433e-7, 2.439e-7, 2.464e-7, 2.611e-7,
2.617e-7, 2.582e-7, 2.453e-7, 2.401e-7, 2.349e-7, 2.203e-7,
2.066e-7, 1.939e-7, 1.78e-7, 1.558e-7, 1.391e-7, 1.203e-7,
1.048e-7, 9.464e-8, 8.306e-8, 7.239e-8, 6.317e-8, 5.52e-8,
4.847e-8, 4.282e-8, 3.796e-8, 3.377e-8, 2.996e-8, 2.678e-8,
2.4e-8, 2.134e-8, 1.904e-8, 1.705e-8, 1.523e-8, 1.35e-8,
1.204e-8, 1.07e-8, 9.408e-9, 8.476e-9, 7.47e-9, 6.679e-9,
5.929e-9, 5.267e-9, 4.711e-9, 4.172e-9, 3.761e-9, 3.288e-9,
2.929e-9, 2.609e-9, 2.315e-9, 2.042e-9, 1.844e-9, 1.64e-9,
1.47e-9, 1.31e-9, 1.176e-9, 1.049e-9, 9.377e-10, 8.462e-10,
7.616e-10, 6.854e-10, 6.191e-10, 5.596e-10, 5.078e-10, 4.611e-10,
4.197e-10, 3.83e-10, 3.505e-10, 3.215e-10, 2.956e-10, 2.726e-10,
2.521e-10, 2.338e-10, 2.173e-10, 2.026e-10, 1.895e-10, 1.777e-10,
1.672e-10, 1.579e-10, 1.496e-10, 1.423e-10, 1.358e-10, 1.302e-10,
1.254e-10, 1.216e-10, 1.187e-10, 1.163e-10, 1.147e-10, 1.145e-10,
1.15e-10, 1.17e-10, 1.192e-10, 1.25e-10, 1.298e-10, 1.345e-10,
1.405e-10, 1.538e-10, 1.648e-10, 1.721e-10, 1.872e-10, 1.968e-10,
2.089e-10, 2.172e-10, 2.317e-10, 2.389e-10, 2.503e-10, 2.585e-10,
2.686e-10, 2.8e-10, 2.895e-10, 3.019e-10, 3.037e-10, 3.076e-10,
3.146e-10, 3.198e-10, 3.332e-10, 3.397e-10, 3.54e-10, 3.667e-10,
3.895e-10, 4.071e-10, 4.565e-10, 4.983e-10, 5.439e-10, 5.968e-10,
6.676e-10, 7.456e-10, 8.405e-10, 9.478e-10, 1.064e-9, 1.218e-9,
1.386e-9, 1.581e-9, 1.787e-9, 2.032e-9, 2.347e-9, 2.677e-9,
3.008e-9, 3.544e-9, 4.056e-9, 4.687e-9, 5.331e-9, 6.227e-9,
6.854e-9, 8.139e-9, 8.945e-9, 9.865e-9, 1.125e-8, 1.178e-8,
1.364e-8, 1.436e-8, 1.54e-8, 1.672e-8, 1.793e-8, 1.906e-8,
2.036e-8, 2.144e-8, 2.292e-8, 2.371e-8, 2.493e-8, 2.606e-8,
2.706e-8, 2.866e-8, 3.036e-8, 3.136e-8, 3.405e-8, 3.665e-8,
3.837e-8, 4.229e-8, 4.748e-8, 5.32e-8, 5.763e-8, 6.677e-8,
```

```
7.216e-8, 7.716e-8, 8.958e-8, 9.419e-8, 1.036e-7, 1.108e-7,
1.189e-7, 1.246e-7, 1.348e-7, 1.31e-7, 1.361e-7, 1.364e-7,
1.363e-7, 1.343e-7, 1.293e-7, 1.254e-7, 1.235e-7, 1.158e-7,
1.107e-7, 9.961e-8, 9.011e-8, 7.91e-8, 6.916e-8, 6.338e-8,
5.564e-8, 4.827e-8, 4.198e-8, 3.695e-8, 3.276e-8, 2.929e-8,
2.633e-8, 2.391e-8, 2.192e-8, 2.021e-8, 1.89e-8, 1.772e-8,
1.667e-8, 1.603e-8, 1.547e-8, 1.537e-8, 1.492e-8, 1.515e-8,
1.479e-8, 1.45e-8, 1.513e-8, 1.495e-8, 1.529e-8, 1.565e-8,
1.564e-8, 1.553e-8, 1.569e-8, 1.584e-8, 1.57e-8, 1.538e-8,
1.513e-8, 1.472e-8, 1.425e-8, 1.349e-8, 1.328e-8, 1.249e-8,
1.17e-8, 1.077e-8, 9.514e-9, 8.614e-9, 7.46e-9, 6.621e-9,
5.775e-9, 5.006e-9, 4.308e-9, 3.747e-9, 3.24e-9, 2.84e-9,
2.481e-9, 2.184e-9, 1.923e-9, 1.71e-9, 1.504e-9, 1.334e-9,
1.187e-9, 1.053e-9, 9.367e-10, 8.306e-10, 7.419e-10, 6.63e-10,
5.918e-10, 5.277e-10, 4.717e-10, 4.222e-10, 3.783e-10, 3.39e-10,
3.036e-10, 2.729e-10, 2.455e-10, 2.211e-10, 1.995e-10, 1.804e-10,
1.635e-10, 1.485e-10, 1.355e-10, 1.24e-10, 1.139e-10, 1.051e-10,
9.757e-11, 9.114e-11, 8.577e-11, 8.139e-11, 7.792e-11, 7.52e-11,
7.39e-11, 7.311e-11, 7.277e-11, 7.482e-11, 7.698e-11, 8.162e-11,
8.517e-11, 8.968e-11, 9.905e-11, 1.075e-10, 1.187e-10, 1.291e-10,
1.426e-10, 1.573e-10, 1.734e-10, 1.905e-10, 2.097e-10, 2.28e-10,
2.473e-10, 2.718e-10, 2.922e-10, 3.128e-10, 3.361e-10, 3.641e-10,
3.91e-10, 4.196e-10, 4.501e-10, 4.932e-10, 5.258e-10, 5.755e-10,
6.253e-10, 6.664e-10, 7.344e-10, 7.985e-10, 8.877e-10, 1.005e-9,
1.118e-9, 1.251e-9, 1.428e-9, 1.61e-9, 1.888e-9, 2.077e-9,
2.331e-9, 2.751e-9, 3.061e-9, 3.522e-9, 3.805e-9, 4.181e-9,
4.575e-9, 5.167e-9, 5.634e-9, 6.007e-9, 6.501e-9, 6.829e-9,
7.211e-9, 7.262e-9, 7.696e-9, 7.832e-9, 7.799e-9, 7.651e-9,
7.304e-9, 7.15e-9, 6.977e-9, 6.603e-9, 6.209e-9, 5.69e-9,
5.432e-9, 4.764e-9, 4.189e-9, 3.64e-9, 3.203e-9, 2.848e-9,
2.51e-9, 2.194e-9, 1.946e-9, 1.75e-9, 1.567e-9, 1.426e-9,
1.302e-9, 1.197e-9, 1.109e-9, 1.035e-9, 9.719e-10, 9.207e-10,
8.957e-10, 8.578e-10, 8.262e-10, 8.117e-10, 7.987e-10, 7.875e-10,
7.741e-10, 7.762e-10, 7.537e-10, 7.424e-10, 7.474e-10, 7.294e-10,
7.216e-10, 7.233e-10, 7.075e-10, 6.892e-10, 6.618e-10, 6.314e-10,
6.208e-10, 5.689e-10, 5.55e-10, 4.984e-10, 4.6e-10, 4.078e-10,
3.879e-10, 3.459e-10, 2.982e-10, 2.626e-10, 2.329e-10, 1.988e-10,
1.735e-10, 1.487e-10, 1.297e-10, 1.133e-10, 9.943e-11, 8.736e-11,
7.726e-11, 6.836e-11, 6.053e-11, 5.384e-11, 4.789e-11, 4.267e-11,
3.804e-11, 3.398e-11, 3.034e-11, 2.71e-11, 2.425e-11, 2.173e-11,
1.95e-11, 1.752e-11, 1.574e-11, 1.418e-11, 1.278e-11, 1.154e-11,
1.044e-11, 9.463e-12, 8.602e-12, 7.841e-12, 7.171e-12, 6.584e-12,
6.073e-12, 5.631e-12, 5.254e-12, 4.937e-12, 4.679e-12, 4.476e-12,
4.328e-12, 4.233e-12, 4.194e-12, 4.211e-12, 4.286e-12, 4.424e-12,
4.628e-12, 4.906e-12, 5.262e-12, 5.708e-12, 6.254e-12, 6.914e-12,
7.714e-12, 8.677e-12, 9.747e-12, 1.101e-11, 1.256e-11, 1.409e-11,
1.597e-11, 1.807e-11, 2.034e-11, 2.316e-11, 2.622e-11, 2.962e-11,
3.369e-11, 3.819e-11, 4.329e-11, 4.932e-11, 5.589e-11, 6.364e-11,
7.284e-11, 8.236e-11, 9.447e-11, 1.078e-10, 1.229e-10, 1.417e-10,
1.614e-10, 1.843e-10, 2.107e-10, 2.406e-10, 2.728e-10, 3.195e-10,
3.595e-10, 4.153e-10, 4.736e-10, 5.41e-10, 6.088e-10, 6.769e-10,
7.691e-10, 8.545e-10, 9.621e-10, 1.047e-9, 1.161e-9, 1.296e-9,
1.424e-9, 1.576e-9, 1.739e-9, 1.893e-9, 2.08e-9, 2.336e-9,
2.604e-9, 2.76e-9, 3.001e-9, 3.365e-9, 3.55e-9, 3.895e-9,
4.183e-9, 4.614e-9, 4.846e-9, 5.068e-9, 5.427e-9, 5.541e-9,
5.864e-9, 5.997e-9, 5.997e-9, 6.061e-9, 5.944e-9, 5.855e-9,
5.661e-9, 5.523e-9, 5.374e-9, 4.94e-9, 4.688e-9, 4.17e-9,
3.913e-9, 3.423e-9, 2.997e-9, 2.598e-9, 2.253e-9, 1.946e-9,
1.71e-9, 1.507e-9, 1.336e-9, 1.19e-9, 1.068e-9, 9.623e-10,
8.772e-10, 8.007e-10, 7.42e-10, 6.884e-10, 6.483e-10, 6.162e-10,
5.922e-10, 5.688e-10, 5.654e-10, 5.637e-10, 5.701e-10, 5.781e-10,
5.874e-10, 6.268e-10, 6.357e-10, 6.525e-10, 7.137e-10, 7.441e-10,
8.024e-10, 8.485e-10, 9.143e-10, 9.536e-10, 9.717e-10, 1.018e-9,
1.042e-9, 1.054e-9, 1.092e-9, 1.079e-9, 1.064e-9, 1.043e-9,
1.02e-9, 9.687e-10, 9.273e-10, 9.208e-10, 9.068e-10, 7.687e-10,
7.385e-10, 6.595e-10, 5.87e-10, 5.144e-10, 4.417e-10, 3.804e-10,
3.301e-10, 2.866e-10, 2.509e-10, 2.202e-10, 1.947e-10, 1.719e-10,
1.525e-10, 1.361e-10, 1.21e-10, 1.084e-10, 9.8e-11, 8.801e-11,
7.954e-11, 7.124e-11, 6.335e-11, 5.76e-11, 5.132e-11, 4.601e-11,
4.096e-11, 3.657e-11, 3.25e-11, 2.909e-11, 2.587e-11, 2.297e-11,
2.05e-11, 1.828e-11, 1.632e-11, 1.462e-11, 1.314e-11, 1.185e-11,
1.073e-11, 9.76e-12, 8.922e-12, 8.206e-12, 7.602e-12, 7.1e-12,
6.694e-12, 6.378e-12, 6.149e-12, 6.004e-12, 5.941e-12, 5.962e-12,
6.069e-12, 6.265e-12, 6.551e-12, 6.935e-12, 7.457e-12, 8.074e-12,
8.811e-12, 9.852e-12, 1.086e-11, 1.207e-11, 1.361e-11, 1.553e-11,
1.737e-11, 1.93e-11, 2.175e-11, 2.41e-11, 2.706e-11, 3.023e-11,
3.313e-11, 3.657e-11, 4.118e-11, 4.569e-11, 5.025e-11, 5.66e-11,
6.231e-11, 6.881e-11, 7.996e-11, 8.526e-11, 9.694e-11, 1.106e-10,
1.222e-10, 1.355e-10, 1.525e-10, 1.775e-10, 1.924e-10, 2.181e-10,
2.379e-10, 2.662e-10, 2.907e-10, 3.154e-10, 3.366e-10, 3.579e-10,
3.858e-10, 4.046e-10, 4.196e-10, 4.166e-10, 4.457e-10, 4.466e-10,
4.404e-10, 4.337e-10, 4.15e-10, 4.083e-10, 3.91e-10, 3.723e-10,
3.514e-10, 3.303e-10, 2.847e-10, 2.546e-10, 2.23e-10, 1.994e-10,
1.733e-10, 1.488e-10, 1.297e-10, 1.144e-10, 1.004e-10, 8.741e-11,
7.928e-11, 7.034e-11, 6.323e-11, 5.754e-11, 5.25e-11, 4.85e-11,
4.502e-11, 4.286e-11, 4.028e-11, 3.899e-11, 3.824e-11, 3.761e-11,
3.804e-11, 3.839e-11, 3.845e-11, 4.244e-11, 4.382e-11, 4.582e-11,
```

```
    4.847e-11, 5.209e-11, 5.384e-11, 5.887e-11, 6.371e-11, 6.737e-11,
    7.168e-11, 7.415e-11, 7.827e-11, 8.037e-11, 8.12e-11, 8.071e-11,
    8.008e-11, 7.851e-11, 7.544e-11, 7.377e-11, 7.173e-11, 6.801e-11,
    6.267e-11, 5.727e-11, 5.288e-11, 4.853e-11, 4.082e-11, 3.645e-11,
    3.136e-11, 2.672e-11, 2.304e-11, 1.986e-11, 1.725e-11, 1.503e-11,
    1.315e-11, 1.153e-11, 1.014e-11, 8.942e-12, 7.901e-12, 6.993e-12,
    6.199e-12, 5.502e-12, 4.89e-12, 4.351e-12, 3.878e-12, 3.461e-12,
    3.094e-12, 2.771e-12, 2.488e-12, 2.241e-12, 2.025e-12, 1.838e-12,
    1.677e-12, 1.541e-12, 1.427e-12, 1.335e-12, 1.262e-12, 1.209e-12,
    1.176e-12, 1.161e-12, 1.165e-12, 1.189e-12, 1.234e-12, 1.3e-12,
    1.389e-12, 1.503e-12, 1.644e-12, 1.814e-12, 2.017e-12, 2.255e-12,
    2.534e-12, 2.858e-12, 3.231e-12, 3.661e-12, 4.153e-12, 4.717e-12,
    5.36e-12, 6.094e-12, 6.93e-12, 7.882e-12, 8.966e-12, 1.02e-11,
    1.162e-11, 1.324e-11, 1.51e-11, 1.72e-11, 1.965e-11, 2.237e-11,
    2.56e-11, 2.927e-11, 3.371e-11, 3.842e-11, 4.429e-11, 5.139e-11,
    5.798e-11, 6.697e-11, 7.626e-11, 8.647e-11, 1.022e-10, 1.136e-10,
    1.3e-10, 1.481e-10, 1.672e-10, 1.871e-10, 2.126e-10, 2.357e-10,
    2.583e-10, 2.997e-10, 3.289e-10, 3.702e-10, 4.012e-10, 4.319e-10,
    4.527e-10, 5.001e-10, 5.448e-10, 5.611e-10, 5.76e-10, 5.965e-10,
    6.079e-10, 6.207e-10, 6.276e-10, 6.222e-10, 6.137e-10, 6e-10,
    5.814e-10, 5.393e-10, 5.35e-10, 4.947e-10, 4.629e-10, 4.117e-10,
    3.712e-10, 3.372e-10, 2.923e-10, 2.55e-10, 2.232e-10, 1.929e-10,
    1.679e-10, 1.46e-10, 1.289e-10, 1.13e-10, 9.953e-11, 8.763e-11,
    7.76e-11, 6.9e-11, 6.16e-11, 5.525e-11, 4.958e-11, 4.489e-11,
    4.072e-11, 3.728e-11, 3.438e-11, 3.205e-11, 3.006e-11, 2.848e-11,
    2.766e-11, 2.688e-11, 2.664e-11, 2.67e-11, 2.696e-11, 2.786e-11,
    2.861e-11, 3.009e-11, 3.178e-11, 3.389e-11, 3.587e-11, 3.819e-11,
    4.054e-11, 4.417e-11, 4.703e-11, 5.137e-11, 5.46e-11, 6.055e-11,
    6.333e-11, 6.773e-11, 7.219e-11, 7.717e-11, 8.131e-11, 8.491e-11,
    8.574e-11, 9.01e-11, 9.017e-11, 8.999e-11, 8.959e-11, 8.838e-11,
    8.579e-11, 8.162e-11, 8.098e-11, 7.472e-11, 7.108e-11, 6.559e-11,
    5.994e-11, 5.172e-11, 4.424e-11, 3.951e-11, 3.34e-11, 2.902e-11,
    2.541e-11, 2.215e-11, 1.945e-11, 1.716e-11, 1.503e-11, 1.339e-11,
    1.185e-11, 1.05e-11, 9.336e-12, 8.307e-12, 7.312e-12, 6.55e-12,
    5.836e-12, 5.178e-12, 4.6e-12, 4.086e-12, 3.639e-12, 3.247e-12,
    2.904e-12, 2.604e-12, 2.341e-12, 2.112e-12, 1.914e-12, 1.744e-12,
    1.598e-12, 1.476e-12, 1.374e-12, 1.293e-12, 1.23e-12, 1.185e-12,
    1.158e-12, 1.147e-12, 1.154e-12, 1.177e-12, 1.219e-12, 1.28e-12,
    1.36e-12, 1.463e-12, 1.591e-12, 1.75e-12, 1.94e-12, 2.156e-12,
    2.43e-12, 2.748e-12, 3.052e-12, 3.533e-12, 3.967e-12, 4.471e-12,
    5.041e-12, 5.86e-12, 6.664e-12, 7.522e-12, 8.342e-12, 9.412e-12,
    1.072e-11, 1.213e-11, 1.343e-11, 1.496e-11, 1.664e-11, 1.822e-11,
    2.029e-11, 2.233e-11, 2.457e-11, 2.709e-11, 2.928e-11, 3.115e-11,
    3.356e-11, 3.592e-11, 3.818e-11, 3.936e-11, 4.061e-11, 4.149e-11,
    4.299e-11, 4.223e-11, 4.251e-11, 4.287e-11, 4.177e-11, 4.094e-11,
    3.942e-11, 3.772e-11, 3.614e-11, 3.394e-11, 3.222e-11, 2.791e-11,
    2.665e-11, 2.309e-11, 2.032e-11, 1.74e-11, 1.535e-11, 1.323e-11,
    1.151e-11, 9.803e-12, 8.65e-12, 7.54e-12, 6.619e-12, 5.832e-12,
    5.113e-12, 4.503e-12, 3.975e-12, 3.52e-12, 3.112e-12, 2.797e-12,
    2.5e-12, 2.24e-12, 2.013e-12, 1.819e-12, 1.653e-12, 1.513e-12,
    1.395e-12, 1.299e-12, 1.225e-12, 1.168e-12, 1.124e-12, 1.148e-12,
    1.107e-12, 1.128e-12, 1.169e-12, 1.233e-12, 1.307e-12, 1.359e-12,
    1.543e-12, 1.686e-12, 1.794e-12, 2.028e-12, 2.21e-12, 2.441e-12,
    2.653e-12, 2.828e-12, 3.093e-12, 3.28e-12, 3.551e-12, 3.677e-12,
    3.803e-12, 3.844e-12, 4.068e-12, 4.093e-12, 4.002e-12, 3.904e-12,
    3.624e-12, 3.633e-12, 3.622e-12, 3.443e-12, 3.184e-12, 2.934e-12,
    2.476e-12, 2.212e-12, 1.867e-12, 1.594e-12, 1.37e-12, 1.192e-12,
    1.045e-12, 9.211e-13, 8.17e-13, 7.29e-13, 6.55e-13, 5.929e-13,
    5.415e-13, 4.995e-13, 4.661e-13, 4.406e-13, 4.225e-13, 4.116e-13,
    4.075e-13, 4.102e-13, 4.198e-13, 4.365e-13, 4.606e-13, 4.925e-13,
    5.326e-13, 5.818e-13, 6.407e-13, 7.104e-13, 7.92e-13, 8.868e-13,
    9.964e-13, 1.123e-12, 1.268e-12, 1.434e-12, 1.626e-12, 1.848e-12,
    2.107e-12, 2.422e-12, 2.772e-12, 3.145e-12, 3.704e-12, 4.27e-12,
    4.721e-12, 5.361e-12, 6.083e-12, 7.095e-12, 7.968e-12, 9.228e-12,
    1.048e-11, 1.187e-11, 1.336e-11, 1.577e-11, 1.772e-11, 2.017e-11,
    2.25e-11, 2.63e-11, 2.911e-11, 3.356e-11, 3.82e-11, 4.173e-11,
    4.811e-11, 5.254e-11, 5.839e-11, 6.187e-11, 6.805e-11, 7.118e-11,
    7.369e-11, 7.664e-11, 7.794e-11, 7.947e-11, 8.036e-11, 7.954e-11,
    7.849e-11, 7.518e-11, 7.462e-11, 6.926e-11, 6.531e-11, 6.197e-11,
    5.421e-11, 4.777e-11, 4.111e-11, 3.679e-11, 3.166e-11, 2.786e-11,
    2.436e-11, 2.144e-11, 1.859e-11, 1.628e-11, 1.414e-11, 1.237e-11,
    1.093e-11, 9.558e-12
  };

  static double h2o260[2001] = { .2752, .2732, .2749, .2676, .2667, .2545,
    .2497, .2327, .2218, .2036, .1825, .1694, .1497, .1353, .121,
    .1014, .09405, .07848, .07195, .06246, .05306, .04853, .04138,
    .03735, .03171, .02785, .02431, .02111, .01845, .0164, .01405,
    .01255, .01098, .009797, .008646, .007779, .006898, .006099,
    .005453, .004909, .004413, .003959, .003581, .003199, .002871,
    .002583, .00233, .002086, .001874, .001684, .001512, .001361,
    .001225, .0011, 9.89e-4, 8.916e-4, 8.039e-4, 7.256e-4, 6.545e-4,
    5.918e-4, 5.359e-4, 4.867e-4, 4.426e-4, 4.033e-4, 3.682e-4,
    3.366e-4, 3.085e-4, 2.833e-4, 2.605e-4, 2.403e-4, 2.221e-4,
    2.055e-4, 1.908e-4, 1.774e-4, 1.653e-4, 1.544e-4, 1.443e-4,
    1.351e-4, 1.267e-4, 1.19e-4, 1.119e-4, 1.053e-4, 9.922e-5,
    9.355e-5, 8.831e-5, 8.339e-5, 7.878e-5, 7.449e-5, 7.043e-5,
```

```
6.664e-5, 6.307e-5, 5.969e-5, 5.654e-5, 5.357e-5, 5.075e-5,
4.81e-5, 4.56e-5, 4.322e-5, 4.102e-5, 3.892e-5, 3.696e-5,
3.511e-5, 3.339e-5, 3.177e-5, 3.026e-5, 2.886e-5, 2.756e-5,
2.636e-5, 2.527e-5, 2.427e-5, 2.337e-5, 2.257e-5, 2.185e-5,
2.127e-5, 2.08e-5, 2.041e-5, 2.013e-5, 2e-5, 1.997e-5, 2.009e-5,
2.031e-5, 2.068e-5, 2.124e-5, 2.189e-5, 2.267e-5, 2.364e-5,
2.463e-5, 2.618e-5, 2.774e-5, 2.937e-5, 3.144e-5, 3.359e-5,
3.695e-5, 4.002e-5, 4.374e-5, 4.947e-5, 5.431e-5, 6.281e-5,
7.169e-5, 8.157e-5, 9.728e-5, 1.079e-4, 1.337e-4, 1.442e-4,
1.683e-4, 1.879e-4, 2.223e-4, 2.425e-4, 2.838e-4, 3.143e-4,
3.527e-4, 4.012e-4, 4.237e-4, 4.747e-4, 5.057e-4, 5.409e-4,
5.734e-4, 5.944e-4, 6.077e-4, 6.175e-4, 6.238e-4, 6.226e-4,
6.248e-4, 6.192e-4, 6.098e-4, 5.818e-4, 5.709e-4, 5.465e-4,
5.043e-4, 4.699e-4, 4.294e-4, 3.984e-4, 3.672e-4, 3.152e-4,
2.883e-4, 2.503e-4, 2.211e-4, 1.92e-4, 1.714e-4, 1.485e-4,
1.358e-4, 1.156e-4, 1.021e-4, 8.887e-5, 7.842e-5, 7.12e-5,
6.186e-5, 5.73e-5, 4.792e-5, 4.364e-5, 3.72e-5, 3.28e-5,
2.946e-5, 2.591e-5, 2.261e-5, 2.048e-5, 1.813e-5, 1.63e-5,
1.447e-5, 1.282e-5, 1.167e-5, 1.041e-5, 9.449e-6, 8.51e-6,
7.596e-6, 6.961e-6, 6.272e-6, 5.728e-6, 5.198e-6, 4.667e-6,
4.288e-6, 3.897e-6, 3.551e-6, 3.235e-6, 2.952e-6, 2.688e-6,
2.449e-6, 2.241e-6, 2.05e-6, 1.879e-6, 1.722e-6, 1.582e-6,
1.456e-6, 1.339e-6, 1.236e-6, 1.144e-6, 1.06e-6, 9.83e-7,
9.149e-7, 8.535e-7, 7.973e-7, 7.466e-7, 6.999e-7, 6.574e-7,
6.18e-7, 5.821e-7, 5.487e-7, 5.18e-7, 4.896e-7, 4.631e-7,
4.386e-7, 4.16e-7, 3.945e-7, 3.748e-7, 3.562e-7, 3.385e-7,
3.222e-7, 3.068e-7, 2.922e-7, 2.788e-7, 2.659e-7, 2.539e-7,
2.425e-7, 2.318e-7, 2.219e-7, 2.127e-7, 2.039e-7, 1.958e-7,
1.885e-7, 1.818e-7, 1.758e-7, 1.711e-7, 1.662e-7, 1.63e-7,
1.605e-7, 1.58e-7, 1.559e-7, 1.545e-7, 1.532e-7, 1.522e-7,
1.51e-7, 1.495e-7, 1.465e-7, 1.483e-7, 1.469e-7, 1.448e-7,
1.444e-7, 1.436e-7, 1.426e-7, 1.431e-7, 1.425e-7, 1.445e-7,
1.477e-7, 1.515e-7, 1.567e-7, 1.634e-7, 1.712e-7, 1.802e-7,
1.914e-7, 2.024e-7, 2.159e-7, 2.295e-7, 2.461e-7, 2.621e-7,
2.868e-7, 3.102e-7, 3.394e-7, 3.784e-7, 4.223e-7, 4.864e-7,
5.501e-7, 6.039e-7, 7.193e-7, 7.728e-7, 9.514e-7, 1.073e-6,
1.18e-6, 1.333e-6, 1.472e-6, 1.566e-6, 1.677e-6, 1.784e-6,
1.904e-6, 1.953e-6, 2.02e-6, 2.074e-6, 2.128e-6, 2.162e-6,
2.219e-6, 2.221e-6, 2.249e-6, 2.239e-6, 2.235e-6, 2.185e-6,
2.141e-6, 2.124e-6, 2.09e-6, 2.068e-6, 2.1e-6, 2.104e-6,
2.142e-6, 2.181e-6, 2.257e-6, 2.362e-6, 2.5e-6, 2.664e-6,
2.884e-6, 3.189e-6, 3.48e-6, 3.847e-6, 4.313e-6, 4.79e-6,
5.25e-6, 5.989e-6, 6.692e-6, 7.668e-6, 8.52e-6, 9.606e-6,
1.073e-5, 1.225e-5, 1.377e-5, 1.582e-5, 1.761e-5, 2.029e-5,
2.284e-5, 2.602e-5, 2.94e-5, 3.483e-5, 3.928e-5, 4.618e-5,
5.24e-5, 6.132e-5, 7.183e-5, 8.521e-5, 9.111e-5, 1.07e-4,
1.184e-4, 1.264e-4, 1.475e-4, 1.612e-4, 1.704e-4, 1.818e-4,
1.924e-4, 1.994e-4, 2.061e-4, 2.18e-4, 2.187e-4, 2.2e-4,
2.196e-4, 2.131e-4, 2.015e-4, 1.988e-4, 1.847e-4, 1.729e-4,
1.597e-4, 1.373e-4, 1.262e-4, 1.087e-4, 9.439e-5, 8.061e-5,
7.093e-5, 6.049e-5, 5.12e-5, 4.435e-5, 3.817e-5, 3.34e-5,
2.927e-5, 2.573e-5, 2.291e-5, 2.04e-5, 1.827e-5, 1.636e-5,
1.463e-5, 1.309e-5, 1.17e-5, 1.047e-5, 9.315e-6, 8.328e-6,
7.458e-6, 6.665e-6, 5.94e-6, 5.316e-6, 4.752e-6, 4.252e-6,
3.825e-6, 3.421e-6, 3.064e-6, 2.746e-6, 2.465e-6, 2.216e-6,
1.99e-6, 1.79e-6, 1.609e-6, 1.449e-6, 1.306e-6, 1.177e-6,
1.063e-6, 9.607e-7, 8.672e-7, 7.855e-7, 7.118e-7, 6.46e-7,
5.871e-7, 5.34e-7, 4.868e-7, 4.447e-7, 4.068e-7, 3.729e-7,
3.423e-7, 3.151e-7, 2.905e-7, 2.686e-7, 2.484e-7, 2.306e-7,
2.142e-7, 1.995e-7, 1.86e-7, 1.738e-7, 1.626e-7, 1.522e-7,
1.427e-7, 1.338e-7, 1.258e-7, 1.183e-7, 1.116e-7, 1.056e-7,
9.972e-8, 9.46e-8, 9.007e-8, 8.592e-8, 8.195e-8, 7.816e-8,
7.483e-8, 7.193e-8, 6.892e-8, 6.642e-8, 6.386e-8, 6.154e-8,
5.949e-8, 5.764e-8, 5.622e-8, 5.479e-8, 5.364e-8, 5.301e-8,
5.267e-8, 5.263e-8, 5.313e-8, 5.41e-8, 5.55e-8, 5.745e-8,
6.003e-8, 6.311e-8, 6.713e-8, 7.173e-8, 7.724e-8, 8.368e-8,
9.121e-8, 9.986e-8, 1.097e-7, 1.209e-7, 1.338e-7, 1.486e-7,
1.651e-7, 1.837e-7, 2.048e-7, 2.289e-7, 2.557e-7, 2.857e-7,
3.195e-7, 3.587e-7, 4.015e-7, 4.497e-7, 5.049e-7, 5.665e-7,
6.366e-7, 7.121e-7, 7.996e-7, 8.946e-7, 1.002e-6, 1.117e-6,
1.262e-6, 1.416e-6, 1.611e-6, 1.807e-6, 2.056e-6, 2.351e-6,
2.769e-6, 3.138e-6, 3.699e-6, 4.386e-6, 5.041e-6, 6.074e-6,
6.812e-6, 7.79e-6, 8.855e-6, 1.014e-5, 1.095e-5, 1.245e-5,
1.316e-5, 1.39e-5, 1.504e-5, 1.583e-5, 1.617e-5, 1.652e-5,
1.713e-5, 1.724e-5, 1.715e-5, 1.668e-5, 1.629e-5, 1.552e-5,
1.478e-5, 1.34e-5, 1.245e-5, 1.121e-5, 9.575e-6, 8.956e-6,
7.345e-6, 6.597e-6, 5.612e-6, 4.818e-6, 4.165e-6, 3.579e-6,
3.041e-6, 2.623e-6, 2.29e-6, 1.984e-6, 1.748e-6, 1.534e-6,
1.369e-6, 1.219e-6, 1.092e-6, 9.8e-7, 8.762e-7, 7.896e-7,
7.104e-7, 6.364e-7, 5.691e-7, 5.107e-7, 4.575e-7, 4.09e-7,
3.667e-7, 3.287e-7, 2.931e-7, 2.633e-7, 2.356e-7, 2.111e-7,
1.895e-7, 1.697e-7, 1.525e-7, 1.369e-7, 1.233e-7, 1.114e-7,
9.988e-8, 9.004e-8, 8.149e-8, 7.352e-8, 6.662e-8, 6.03e-8,
5.479e-8, 4.974e-8, 4.532e-8, 4.129e-8, 3.781e-8, 3.462e-8,
3.176e-8, 2.919e-8, 2.687e-8, 2.481e-8, 2.292e-8, 2.119e-8,
1.967e-8, 1.828e-8, 1.706e-8, 1.589e-8, 1.487e-8, 1.393e-8,
1.307e-8, 1.228e-8, 1.156e-8, 1.089e-8, 1.028e-8, 9.696e-9,
```

```
9.159e-9, 8.658e-9, 8.187e-9, 7.746e-9, 7.34e-9, 6.953e-9,
6.594e-9, 6.259e-9, 5.948e-9, 5.66e-9, 5.386e-9, 5.135e-9,
4.903e-9, 4.703e-9, 4.515e-9, 4.362e-9, 4.233e-9, 4.117e-9,
4.017e-9, 3.962e-9, 3.924e-9, 3.905e-9, 3.922e-9, 3.967e-9,
4.046e-9, 4.165e-9, 4.32e-9, 4.522e-9, 4.769e-9, 5.083e-9,
5.443e-9, 5.872e-9, 6.366e-9, 6.949e-9, 7.601e-9, 8.371e-9,
9.22e-9, 1.02e-8, 1.129e-8, 1.251e-8, 1.393e-8, 1.542e-8,
1.72e-8, 1.926e-8, 2.152e-8, 2.392e-8, 2.678e-8, 3.028e-8,
3.39e-8, 3.836e-8, 4.309e-8, 4.9e-8, 5.481e-8, 6.252e-8,
7.039e-8, 7.883e-8, 8.849e-8, 1.012e-7, 1.142e-7, 1.3e-7,
1.475e-7, 1.732e-7, 1.978e-7, 2.304e-7, 2.631e-7, 2.988e-7,
3.392e-7, 3.69e-7, 4.355e-7, 4.672e-7, 5.11e-7, 5.461e-7,
5.828e-7, 6.233e-7, 6.509e-7, 6.672e-7, 6.969e-7, 7.104e-7,
7.439e-7, 7.463e-7, 7.708e-7, 7.466e-7, 7.668e-7, 7.549e-7,
7.586e-7, 7.384e-7, 7.439e-7, 7.785e-7, 7.915e-7, 8.31e-7,
8.745e-7, 9.558e-7, 1.038e-6, 1.173e-6, 1.304e-6, 1.452e-6,
1.671e-6, 1.931e-6, 2.239e-6, 2.578e-6, 3.032e-6, 3.334e-6,
3.98e-6, 4.3e-6, 4.518e-6, 5.321e-6, 5.508e-6, 6.211e-6, 6.59e-6,
7.046e-6, 7.555e-6, 7.558e-6, 7.875e-6, 8.319e-6, 8.433e-6,
8.59e-6, 8.503e-6, 8.304e-6, 8.336e-6, 7.739e-6, 7.301e-6,
6.827e-6, 6.078e-6, 5.551e-6, 4.762e-6, 4.224e-6, 3.538e-6,
2.984e-6, 2.619e-6, 2.227e-6, 1.923e-6, 1.669e-6, 1.462e-6,
1.294e-6, 1.155e-6, 1.033e-6, 9.231e-7, 8.238e-7, 7.36e-7,
6.564e-7, 5.869e-7, 5.236e-7, 4.673e-7, 4.174e-7, 3.736e-7,
3.33e-7, 2.976e-7, 2.657e-7, 2.367e-7, 2.106e-7, 1.877e-7,
1.671e-7, 1.494e-7, 1.332e-7, 1.192e-7, 1.065e-7, 9.558e-8,
8.586e-8, 7.717e-8, 6.958e-8, 6.278e-8, 5.666e-8, 5.121e-8,
4.647e-8, 4.213e-8, 3.815e-8, 3.459e-8, 3.146e-8, 2.862e-8,
2.604e-8, 2.375e-8, 2.162e-8, 1.981e-8, 1.817e-8, 1.67e-8,
1.537e-8, 1.417e-8, 1.31e-8, 1.215e-8, 1.128e-8, 1.05e-8,
9.793e-9, 9.158e-9, 8.586e-9, 8.068e-9, 7.595e-9, 7.166e-9,
6.778e-9, 6.427e-9, 6.108e-9, 5.826e-9, 5.571e-9, 5.347e-9,
5.144e-9, 4.968e-9, 4.822e-9, 4.692e-9, 4.589e-9, 4.506e-9,
4.467e-9, 4.44e-9, 4.466e-9, 4.515e-9, 4.718e-9, 4.729e-9,
4.937e-9, 5.249e-9, 5.466e-9, 5.713e-9, 6.03e-9, 6.436e-9,
6.741e-9, 7.33e-9, 7.787e-9, 8.414e-9, 8.908e-9, 9.868e-9,
1.069e-8, 1.158e-8, 1.253e-8, 1.3e-8, 1.409e-8, 1.47e-8,
1.548e-8, 1.612e-8, 1.666e-8, 1.736e-8, 1.763e-8, 1.812e-8,
1.852e-8, 1.923e-8, 1.897e-8, 1.893e-8, 1.888e-8, 1.868e-8,
1.895e-8, 1.899e-8, 1.876e-8, 1.96e-8, 2.02e-8, 2.121e-8,
2.239e-8, 2.379e-8, 2.526e-8, 2.766e-8, 2.994e-8, 3.332e-8,
3.703e-8, 4.158e-8, 4.774e-8, 5.499e-8, 6.355e-8, 7.349e-8,
8.414e-8, 9.846e-8, 1.143e-7, 1.307e-7, 1.562e-7, 1.817e-7,
2.011e-7, 2.192e-7, 2.485e-7, 2.867e-7, 3.035e-7, 3.223e-7,
3.443e-7, 3.617e-7, 3.793e-7, 3.793e-7, 3.839e-7, 4.081e-7,
4.117e-7, 4.085e-7, 3.92e-7, 3.851e-7, 3.754e-7, 3.49e-7,
3.229e-7, 2.978e-7, 2.691e-7, 2.312e-7, 2.029e-7, 1.721e-7,
1.472e-7, 1.308e-7, 1.132e-7, 9.736e-8, 8.458e-8, 7.402e-8,
6.534e-8, 5.811e-8, 5.235e-8, 4.762e-8, 4.293e-8, 3.896e-8,
3.526e-8, 3.165e-8, 2.833e-8, 2.551e-8, 2.288e-8, 2.036e-8,
1.82e-8, 1.626e-8, 1.438e-8, 1.299e-8, 1.149e-8, 1.03e-8,
9.148e-9, 8.122e-9, 7.264e-9, 6.425e-9, 5.777e-9, 5.06e-9,
4.502e-9, 4.013e-9, 3.567e-9, 3.145e-9, 2.864e-9, 2.553e-9,
2.311e-9, 2.087e-9, 1.886e-9, 1.716e-9, 1.556e-9, 1.432e-9,
1.311e-9, 1.202e-9, 1.104e-9, 1.013e-9, 9.293e-10, 8.493e-10,
7.79e-10, 7.185e-10, 6.642e-10, 6.141e-10, 5.684e-10, 5.346e-10,
5.032e-10, 4.725e-10, 4.439e-10, 4.176e-10, 3.93e-10, 3.714e-10,
3.515e-10, 3.332e-10, 3.167e-10, 3.02e-10, 2.887e-10, 2.769e-10,
2.665e-10, 2.578e-10, 2.503e-10, 2.436e-10, 2.377e-10, 2.342e-10,
2.305e-10, 2.296e-10, 2.278e-10, 2.321e-10, 2.355e-10, 2.402e-10,
2.478e-10, 2.67e-10, 2.848e-10, 2.982e-10, 3.263e-10, 3.438e-10,
3.649e-10, 3.829e-10, 4.115e-10, 4.264e-10, 4.473e-10, 4.63e-10,
4.808e-10, 4.995e-10, 5.142e-10, 5.313e-10, 5.318e-10, 5.358e-10,
5.452e-10, 5.507e-10, 5.698e-10, 5.782e-10, 5.983e-10, 6.164e-10,
6.532e-10, 6.811e-10, 7.624e-10, 8.302e-10, 9.067e-10, 9.937e-10,
1.104e-9, 1.221e-9, 1.361e-9, 1.516e-9, 1.675e-9, 1.883e-9,
2.101e-9, 2.349e-9, 2.614e-9, 2.92e-9, 3.305e-9, 3.724e-9,
4.142e-9, 4.887e-9, 5.614e-9, 6.506e-9, 7.463e-9, 8.817e-9,
9.849e-9, 1.187e-8, 1.321e-8, 1.474e-8, 1.698e-8, 1.794e-8,
2.09e-8, 2.211e-8, 2.362e-8, 2.556e-8, 2.729e-8, 2.88e-8,
3.046e-8, 3.167e-8, 3.367e-8, 3.457e-8, 3.59e-8, 3.711e-8,
3.826e-8, 4.001e-8, 4.211e-8, 4.315e-8, 4.661e-8, 5.01e-8,
5.249e-8, 5.84e-8, 6.628e-8, 7.512e-8, 8.253e-8, 9.722e-8,
1.067e-7, 1.153e-7, 1.347e-7, 1.428e-7, 1.577e-7, 1.694e-7,
1.833e-7, 1.938e-7, 2.108e-7, 2.059e-7, 2.157e-7, 2.185e-7,
2.208e-7, 2.182e-7, 2.093e-7, 2.014e-7, 1.962e-7, 1.819e-7,
1.713e-7, 1.51e-7, 1.34e-7, 1.154e-7, 9.89e-8, 8.88e-8, 7.673e-8,
6.599e-8, 5.73e-8, 5.081e-8, 4.567e-8, 4.147e-8, 3.773e-8,
3.46e-8, 3.194e-8, 2.953e-8, 2.759e-8, 2.594e-8, 2.442e-8,
2.355e-8, 2.283e-8, 2.279e-8, 2.231e-8, 2.279e-8, 2.239e-8,
2.21e-8, 2.309e-8, 2.293e-8, 2.352e-8, 2.415e-8, 2.43e-8,
2.426e-8, 2.465e-8, 2.5e-8, 2.496e-8, 2.465e-8, 2.445e-8,
2.383e-8, 2.299e-8, 2.165e-8, 2.113e-8, 1.968e-8, 1.819e-8,
1.644e-8, 1.427e-8, 1.27e-8, 1.082e-8, 9.428e-9, 8.091e-9,
6.958e-9, 5.988e-9, 5.246e-9, 4.601e-9, 4.098e-9, 3.664e-9,
3.287e-9, 2.942e-9, 2.656e-9, 2.364e-9, 2.118e-9, 1.903e-9,
1.703e-9, 1.525e-9, 1.365e-9, 1.229e-9, 1.107e-9, 9.96e-10,
```

```
8.945e-10, 8.08e-10, 7.308e-10, 6.616e-10, 5.994e-10, 5.422e-10,
4.929e-10, 4.478e-10, 4.07e-10, 3.707e-10, 3.379e-10, 3.087e-10,
2.823e-10, 2.592e-10, 2.385e-10, 2.201e-10, 2.038e-10, 1.897e-10,
1.774e-10, 1.667e-10, 1.577e-10, 1.502e-10, 1.437e-10, 1.394e-10,
1.358e-10, 1.324e-10, 1.329e-10, 1.324e-10, 1.36e-10, 1.39e-10,
1.424e-10, 1.544e-10, 1.651e-10, 1.817e-10, 1.984e-10, 2.195e-10,
2.438e-10, 2.7e-10, 2.991e-10, 3.322e-10, 3.632e-10, 3.957e-10,
4.36e-10, 4.701e-10, 5.03e-10, 5.381e-10, 5.793e-10, 6.19e-10,
6.596e-10, 7.004e-10, 7.561e-10, 7.934e-10, 8.552e-10, 9.142e-10,
9.57e-10, 1.027e-9, 1.097e-9, 1.193e-9, 1.334e-9, 1.47e-9,
1.636e-9, 1.871e-9, 2.122e-9, 2.519e-9, 2.806e-9, 3.203e-9,
3.846e-9, 4.362e-9, 5.114e-9, 5.643e-9, 6.305e-9, 6.981e-9,
7.983e-9, 8.783e-9, 9.419e-9, 1.017e-8, 1.063e-8, 1.121e-8,
1.13e-8, 1.201e-8, 1.225e-8, 1.232e-8, 1.223e-8, 1.177e-8,
1.151e-8, 1.116e-8, 1.047e-8, 9.698e-9, 8.734e-9, 8.202e-9,
7.041e-9, 6.074e-9, 5.172e-9, 4.468e-9, 3.913e-9, 3.414e-9,
2.975e-9, 2.65e-9, 2.406e-9, 2.173e-9, 2.009e-9, 1.861e-9,
1.727e-9, 1.612e-9, 1.514e-9, 1.43e-9, 1.362e-9, 1.333e-9,
1.288e-9, 1.249e-9, 1.238e-9, 1.228e-9, 1.217e-9, 1.202e-9,
1.209e-9, 1.177e-9, 1.157e-9, 1.165e-9, 1.142e-9, 1.131e-9,
1.138e-9, 1.117e-9, 1.1e-9, 1.069e-9, 1.023e-9, 1.005e-9,
9.159e-10, 8.863e-10, 7.865e-10, 7.153e-10, 6.247e-10, 5.846e-10,
5.133e-10, 4.36e-10, 3.789e-10, 3.335e-10, 2.833e-10, 2.483e-10,
2.155e-10, 1.918e-10, 1.709e-10, 1.529e-10, 1.374e-10, 1.235e-10,
1.108e-10, 9.933e-11, 8.932e-11, 8.022e-11, 7.224e-11, 6.52e-11,
5.896e-11, 5.328e-11, 4.813e-11, 4.365e-11, 3.961e-11, 3.594e-11,
3.266e-11, 2.967e-11, 2.701e-11, 2.464e-11, 2.248e-11, 2.054e-11,
1.878e-11, 1.721e-11, 1.579e-11, 1.453e-11, 1.341e-11, 1.241e-11,
1.154e-11, 1.078e-11, 1.014e-11, 9.601e-12, 9.167e-12, 8.838e-12,
8.614e-12, 8.493e-12, 8.481e-12, 8.581e-12, 8.795e-12, 9.131e-12,
9.601e-12, 1.021e-11, 1.097e-11, 1.191e-11, 1.303e-11, 1.439e-11,
1.601e-11, 1.778e-11, 1.984e-11, 2.234e-11, 2.474e-11, 2.766e-11,
3.085e-11, 3.415e-11, 3.821e-11, 4.261e-11, 4.748e-11, 5.323e-11,
5.935e-11, 6.619e-11, 7.418e-11, 8.294e-11, 9.26e-11, 1.039e-10,
1.156e-10, 1.297e-10, 1.46e-10, 1.641e-10, 1.858e-10, 2.1e-10,
2.383e-10, 2.724e-10, 3.116e-10, 3.538e-10, 4.173e-10, 4.727e-10,
5.503e-10, 6.337e-10, 7.32e-10, 8.298e-10, 9.328e-10, 1.059e-9,
1.176e-9, 1.328e-9, 1.445e-9, 1.593e-9, 1.77e-9, 1.954e-9,
2.175e-9, 2.405e-9, 2.622e-9, 2.906e-9, 3.294e-9, 3.713e-9,
3.98e-9, 4.384e-9, 4.987e-9, 5.311e-9, 5.874e-9, 6.337e-9,
7.027e-9, 7.39e-9, 7.769e-9, 8.374e-9, 8.605e-9, 9.165e-9,
9.415e-9, 9.511e-9, 9.704e-9, 9.588e-9, 9.45e-9, 9.086e-9,
8.798e-9, 8.469e-9, 7.697e-9, 7.168e-9, 6.255e-9, 5.772e-9,
4.97e-9, 4.271e-9, 3.653e-9, 3.154e-9, 2.742e-9, 2.435e-9,
2.166e-9, 1.936e-9, 1.731e-9, 1.556e-9, 1.399e-9, 1.272e-9,
1.157e-9, 1.066e-9, 9.844e-10, 9.258e-10, 8.787e-10, 8.421e-10,
8.083e-10, 8.046e-10, 8.067e-10, 8.181e-10, 8.325e-10, 8.517e-10,
9.151e-10, 9.351e-10, 9.677e-10, 1.071e-9, 1.126e-9, 1.219e-9,
1.297e-9, 1.408e-9, 1.476e-9, 1.517e-9, 1.6e-9, 1.649e-9,
1.678e-9, 1.746e-9, 1.742e-9, 1.728e-9, 1.699e-9, 1.655e-9,
1.561e-9, 1.48e-9, 1.451e-9, 1.411e-9, 1.171e-9, 1.106e-9,
9.714e-10, 8.523e-10, 7.346e-10, 6.241e-10, 5.371e-10, 4.704e-10,
4.144e-10, 3.683e-10, 3.292e-10, 2.942e-10, 2.62e-10, 2.341e-10,
2.104e-10, 1.884e-10, 1.7e-10, 1.546e-10, 1.394e-10, 1.265e-10,
1.14e-10, 1.019e-10, 9.279e-11, 8.283e-11, 7.458e-11, 6.668e-11,
5.976e-11, 5.33e-11, 4.794e-11, 4.289e-11, 3.841e-11, 3.467e-11,
3.13e-11, 2.832e-11, 2.582e-11, 2.356e-11, 2.152e-11, 1.97e-11,
1.808e-11, 1.664e-11, 1.539e-11, 1.434e-11, 1.344e-11, 1.269e-11,
1.209e-11, 1.162e-11, 1.129e-11, 1.108e-11, 1.099e-11, 1.103e-11,
1.119e-11, 1.148e-11, 1.193e-11, 1.252e-11, 1.329e-11, 1.421e-11,
1.555e-11, 1.685e-11, 1.839e-11, 2.054e-11, 2.317e-11, 2.571e-11,
2.839e-11, 3.171e-11, 3.49e-11, 3.886e-11, 4.287e-11, 4.645e-11,
5.047e-11, 5.592e-11, 6.109e-11, 6.628e-11, 7.381e-11, 8.088e-11,
8.966e-11, 1.045e-10, 1.12e-10, 1.287e-10, 1.486e-10, 1.662e-10,
1.866e-10, 2.133e-10, 2.524e-10, 2.776e-10, 3.204e-10, 3.559e-10,
4.028e-10, 4.448e-10, 4.882e-10, 5.244e-10, 5.605e-10, 6.018e-10,
6.328e-10, 6.579e-10, 6.541e-10, 7.024e-10, 7.074e-10, 7.068e-10,
7.009e-10, 6.698e-10, 6.545e-10, 6.209e-10, 5.834e-10, 5.412e-10,
5.001e-10, 4.231e-10, 3.727e-10, 3.211e-10, 2.833e-10, 2.447e-10,
2.097e-10, 1.843e-10, 1.639e-10, 1.449e-10, 1.27e-10, 1.161e-10,
1.033e-10, 9.282e-11, 8.407e-11, 7.639e-11, 7.023e-11, 6.474e-11,
6.142e-11, 5.76e-11, 5.568e-11, 5.472e-11, 5.39e-11, 5.455e-11,
5.54e-11, 5.587e-11, 6.23e-11, 6.49e-11, 6.868e-11, 7.382e-11,
8.022e-11, 8.372e-11, 9.243e-11, 1.004e-10, 1.062e-10, 1.13e-10,
1.176e-10, 1.244e-10, 1.279e-10, 1.298e-10, 1.302e-10, 1.312e-10,
1.295e-10, 1.244e-10, 1.211e-10, 1.167e-10, 1.098e-10, 9.927e-11,
8.854e-11, 8.011e-11, 7.182e-11, 5.923e-11, 5.212e-11, 4.453e-11,
3.832e-11, 3.371e-11, 2.987e-11, 2.651e-11, 2.354e-11, 2.093e-11,
1.863e-11, 1.662e-11, 1.486e-11, 1.331e-11, 1.193e-11, 1.071e-11,
9.628e-12, 8.66e-12, 7.801e-12, 7.031e-12, 6.347e-12, 5.733e-12,
5.182e-12, 4.695e-12, 4.26e-12, 3.874e-12, 3.533e-12, 3.235e-12,
2.979e-12, 2.76e-12, 2.579e-12, 2.432e-12, 2.321e-12, 2.246e-12,
2.205e-12, 2.196e-12, 2.223e-12, 2.288e-12, 2.387e-12, 2.525e-12,
2.704e-12, 2.925e-12, 3.191e-12, 3.508e-12, 3.876e-12, 4.303e-12,
4.793e-12, 5.347e-12, 5.978e-12, 6.682e-12, 7.467e-12, 8.34e-12,
9.293e-12, 1.035e-11, 1.152e-11, 1.285e-11, 1.428e-11, 1.586e-11,
1.764e-11, 1.972e-11, 2.214e-11, 2.478e-11, 2.776e-11, 3.151e-11,
```

```
    3.591e-11, 4.103e-11, 4.66e-11, 5.395e-11, 6.306e-11, 7.172e-11,
    8.358e-11, 9.67e-11, 1.11e-10, 1.325e-10, 1.494e-10, 1.736e-10,
    2.007e-10, 2.296e-10, 2.608e-10, 3.004e-10, 3.361e-10, 3.727e-10,
    4.373e-10, 4.838e-10, 5.483e-10, 6.006e-10, 6.535e-10, 6.899e-10,
    7.687e-10, 8.444e-10, 8.798e-10, 9.135e-10, 9.532e-10, 9.757e-10,
    9.968e-10, 1.006e-9, 9.949e-10, 9.789e-10, 9.564e-10, 9.215e-10,
    8.51e-10, 8.394e-10, 7.707e-10, 7.152e-10, 6.274e-10, 5.598e-10,
    5.028e-10, 4.3e-10, 3.71e-10, 3.245e-10, 2.809e-10, 2.461e-10,
    2.154e-10, 1.91e-10, 1.685e-10, 1.487e-10, 1.313e-10, 1.163e-10,
    1.031e-10, 9.172e-11, 8.221e-11, 7.382e-11, 6.693e-11, 6.079e-11,
    5.581e-11, 5.167e-11, 4.811e-11, 4.506e-11, 4.255e-11, 4.083e-11,
    3.949e-11, 3.881e-11, 3.861e-11, 3.858e-11, 3.951e-11, 4.045e-11,
    4.24e-11, 4.487e-11, 4.806e-11, 5.133e-11, 5.518e-11, 5.919e-11,
    6.533e-11, 7.031e-11, 7.762e-11, 8.305e-11, 9.252e-11, 9.727e-11,
    1.045e-10, 1.117e-10, 1.2e-10, 1.275e-10, 1.341e-10, 1.362e-10,
    1.438e-10, 1.45e-10, 1.455e-10, 1.455e-10, 1.434e-10, 1.381e-10,
    1.301e-10, 1.276e-10, 1.163e-10, 1.089e-10, 9.911e-11, 8.943e-11,
    7.618e-11, 6.424e-11, 5.717e-11, 4.866e-11, 4.257e-11, 3.773e-11,
    3.331e-11, 2.958e-11, 2.629e-11, 2.316e-11, 2.073e-11, 1.841e-11,
    1.635e-11, 1.464e-11, 1.31e-11, 1.16e-11, 1.047e-11, 9.408e-12,
    8.414e-12, 7.521e-12, 6.705e-12, 5.993e-12, 5.371e-12, 4.815e-12,
    4.338e-12, 3.921e-12, 3.567e-12, 3.265e-12, 3.01e-12, 2.795e-12,
    2.613e-12, 2.464e-12, 2.346e-12, 2.256e-12, 2.195e-12, 2.165e-12,
    2.166e-12, 2.198e-12, 2.262e-12, 2.364e-12, 2.502e-12, 2.682e-12,
    2.908e-12, 3.187e-12, 3.533e-12, 3.946e-12, 4.418e-12, 5.013e-12,
    5.708e-12, 6.379e-12, 7.43e-12, 8.39e-12, 9.51e-12, 1.078e-11,
    1.259e-11, 1.438e-11, 1.63e-11, 1.814e-11, 2.055e-11, 2.348e-11,
    2.664e-11, 2.956e-11, 3.3e-11, 3.677e-11, 4.032e-11, 4.494e-11,
    4.951e-11, 5.452e-11, 6.014e-11, 6.5e-11, 6.915e-11, 7.45e-11,
    7.971e-11, 8.468e-11, 8.726e-11, 8.995e-11, 9.182e-11, 9.509e-11,
    9.333e-11, 9.386e-11, 9.457e-11, 9.21e-11, 9.019e-11, 8.68e-11,
    8.298e-11, 7.947e-11, 7.46e-11, 7.082e-11, 6.132e-11, 5.855e-11,
    5.073e-11, 4.464e-11, 3.825e-11, 3.375e-11, 2.911e-11, 2.535e-11,
    2.16e-11, 1.907e-11, 1.665e-11, 1.463e-11, 1.291e-11, 1.133e-11,
    9.997e-12, 8.836e-12, 7.839e-12, 6.943e-12, 6.254e-12, 5.6e-12,
    5.029e-12, 4.529e-12, 4.102e-12, 3.737e-12, 3.428e-12, 3.169e-12,
    2.959e-12, 2.798e-12, 2.675e-12, 2.582e-12, 2.644e-12, 2.557e-12,
    2.614e-12, 2.717e-12, 2.874e-12, 3.056e-12, 3.187e-12, 3.631e-12,
    3.979e-12, 4.248e-12, 4.817e-12, 5.266e-12, 5.836e-12, 6.365e-12,
    6.807e-12, 7.47e-12, 7.951e-12, 8.636e-12, 8.972e-12, 9.314e-12,
    9.445e-12, 1.003e-11, 1.013e-11, 9.937e-12, 9.729e-12, 9.064e-12,
    9.119e-12, 9.124e-12, 8.704e-12, 8.078e-12, 7.47e-12, 6.329e-12,
    5.674e-12, 4.808e-12, 4.119e-12, 3.554e-12, 3.103e-12, 2.731e-12,
    2.415e-12, 2.15e-12, 1.926e-12, 1.737e-12, 1.578e-12, 1.447e-12,
    1.34e-12, 1.255e-12, 1.191e-12, 1.146e-12, 1.121e-12, 1.114e-12,
    1.126e-12, 1.156e-12, 1.207e-12, 1.278e-12, 1.372e-12, 1.49e-12,
    1.633e-12, 1.805e-12, 2.01e-12, 2.249e-12, 2.528e-12, 2.852e-12,
    3.228e-12, 3.658e-12, 4.153e-12, 4.728e-12, 5.394e-12, 6.176e-12,
    7.126e-12, 8.188e-12, 9.328e-12, 1.103e-11, 1.276e-11, 1.417e-11,
    1.615e-11, 1.84e-11, 2.155e-11, 2.429e-11, 2.826e-11, 3.222e-11,
    3.664e-11, 4.14e-11, 4.906e-11, 5.536e-11, 6.327e-11, 7.088e-11,
    8.316e-11, 9.242e-11, 1.07e-10, 1.223e-10, 1.341e-10, 1.553e-10,
    1.703e-10, 1.9e-10, 2.022e-10, 2.233e-10, 2.345e-10, 2.438e-10,
    2.546e-10, 2.599e-10, 2.661e-10, 2.703e-10, 2.686e-10, 2.662e-10,
    2.56e-10, 2.552e-10, 2.378e-10, 2.252e-10, 2.146e-10, 1.885e-10,
    1.668e-10, 1.441e-10, 1.295e-10, 1.119e-10, 9.893e-11, 8.687e-11,
    7.678e-11, 6.685e-11, 5.879e-11, 5.127e-11, 4.505e-11, 3.997e-11,
    3.511e-11
  };

  static double h2ofrn[2001] = { .01095, .01126, .01205, .01322, .0143,
    .01506, .01548, .01534, .01486, .01373, .01262, .01134, .01001,
    .008702, .007475, .006481, .00548, .0046, .003833, .00311,
    .002543, .002049, .00168, .001374, .001046, 8.193e-4, 6.267e-4,
    4.968e-4, 3.924e-4, 2.983e-4, 2.477e-4, 1.997e-4, 1.596e-4,
    1.331e-4, 1.061e-4, 8.942e-5, 7.168e-5, 5.887e-5, 4.848e-5,
    3.817e-5, 3.17e-5, 2.579e-5, 2.162e-5, 1.768e-5, 1.49e-5,
    1.231e-5, 1.013e-5, 8.555e-6, 7.328e-6, 6.148e-6, 5.207e-6,
    4.387e-6, 3.741e-6, 3.22e-6, 2.753e-6, 2.346e-6, 1.985e-6,
    1.716e-6, 1.475e-6, 1.286e-6, 1.122e-6, 9.661e-7, 8.284e-7,
    7.057e-7, 6.119e-7, 5.29e-7, 4.571e-7, 3.948e-7, 3.432e-7,
    2.983e-7, 2.589e-7, 2.265e-7, 1.976e-7, 1.704e-7, 1.456e-7,
    1.26e-7, 1.101e-7, 9.648e-8, 8.415e-8, 7.34e-8, 6.441e-8,
    5.643e-8, 4.94e-8, 4.276e-8, 3.703e-8, 3.227e-8, 2.825e-8,
    2.478e-8, 2.174e-8, 1.898e-8, 1.664e-8, 1.458e-8, 1.278e-8,
    1.126e-8, 9.891e-9, 8.709e-9, 7.652e-9, 6.759e-9, 5.975e-9,
    5.31e-9, 4.728e-9, 4.214e-9, 3.792e-9, 3.463e-9, 3.226e-9,
    2.992e-9, 2.813e-9, 2.749e-9, 2.809e-9, 2.913e-9, 3.037e-9,
    3.413e-9, 3.738e-9, 4.189e-9, 4.808e-9, 5.978e-9, 7.088e-9,
    8.071e-9, 9.61e-9, 1.21e-8, 1.5e-8, 1.764e-8, 2.221e-8, 2.898e-8,
    3.948e-8, 5.068e-8, 6.227e-8, 7.898e-8, 1.033e-7, 1.437e-7,
    1.889e-7, 2.589e-7, 3.59e-7, 4.971e-7, 7.156e-7, 9.983e-7,
    1.381e-6, 1.929e-6, 2.591e-6, 3.453e-6, 4.57e-6, 5.93e-6,
    7.552e-6, 9.556e-6, 1.183e-5, 1.425e-5, 1.681e-5, 1.978e-5,
    2.335e-5, 2.668e-5, 3.022e-5, 3.371e-5, 3.715e-5, 3.967e-5,
    4.06e-5, 4.01e-5, 3.809e-5, 3.491e-5, 3.155e-5, 2.848e-5,
    2.678e-5, 2.66e-5, 2.811e-5, 3.071e-5, 3.294e-5, 3.459e-5,
```

```
3.569e-5, 3.56e-5, 3.434e-5, 3.186e-5, 2.916e-5, 2.622e-5,
2.275e-5, 1.918e-5, 1.62e-5, 1.373e-5, 1.182e-5, 1.006e-5,
8.556e-6, 7.26e-6, 6.107e-6, 5.034e-6, 4.211e-6, 3.426e-6,
2.865e-6, 2.446e-6, 1.998e-6, 1.628e-6, 1.242e-6, 1.005e-6,
7.853e-7, 6.21e-7, 5.071e-7, 4.156e-7, 3.548e-7, 2.825e-7,
2.261e-7, 1.916e-7, 1.51e-7, 1.279e-7, 1.059e-7, 9.14e-8,
7.707e-8, 6.17e-8, 5.311e-8, 4.263e-8, 3.518e-8, 2.961e-8,
2.457e-8, 2.119e-8, 1.712e-8, 1.439e-8, 1.201e-8, 1.003e-8,
8.564e-9, 7.199e-9, 6.184e-9, 5.206e-9, 4.376e-9, 3.708e-9,
3.157e-9, 2.725e-9, 2.361e-9, 2.074e-9, 1.797e-9, 1.562e-9,
1.364e-9, 1.196e-9, 1.042e-9, 8.862e-10, 7.648e-10, 6.544e-10,
5.609e-10, 4.791e-10, 4.108e-10, 3.531e-10, 3.038e-10, 2.618e-10,
2.268e-10, 1.969e-10, 1.715e-10, 1.496e-10, 1.308e-10, 1.147e-10,
1.008e-10, 8.894e-11, 7.885e-11, 7.031e-11, 6.355e-11, 5.854e-11,
5.534e-11, 5.466e-11, 5.725e-11, 6.447e-11, 7.943e-11, 1.038e-10,
1.437e-10, 2.04e-10, 2.901e-10, 4.051e-10, 5.556e-10, 7.314e-10,
9.291e-10, 1.134e-9, 1.321e-9, 1.482e-9, 1.596e-9, 1.669e-9,
1.715e-9, 1.762e-9, 1.817e-9, 1.828e-9, 1.848e-9, 1.873e-9,
1.902e-9, 1.894e-9, 1.864e-9, 1.841e-9, 1.797e-9, 1.704e-9,
1.559e-9, 1.382e-9, 1.187e-9, 1.001e-9, 8.468e-10, 7.265e-10,
6.521e-10, 6.381e-10, 6.66e-10, 7.637e-10, 9.705e-10, 1.368e-9,
1.856e-9, 2.656e-9, 3.954e-9, 5.96e-9, 8.72e-9, 1.247e-8,
1.781e-8, 2.491e-8, 3.311e-8, 4.272e-8, 5.205e-8, 6.268e-8,
7.337e-8, 8.277e-8, 9.185e-8, 1.004e-7, 1.091e-7, 1.159e-7,
1.188e-7, 1.175e-7, 1.124e-7, 1.033e-7, 9.381e-8, 8.501e-8,
7.956e-8, 7.894e-8, 8.331e-8, 9.102e-8, 9.836e-8, 1.035e-7,
1.064e-7, 1.06e-7, 1.032e-7, 9.808e-8, 9.139e-8, 8.442e-8,
7.641e-8, 6.881e-8, 6.161e-8, 5.404e-8, 4.804e-8, 4.446e-8,
4.328e-8, 4.259e-8, 4.421e-8, 4.673e-8, 4.985e-8, 5.335e-8,
5.796e-8, 6.542e-8, 7.714e-8, 8.827e-8, 1.04e-7, 1.238e-7,
1.499e-7, 1.829e-7, 2.222e-7, 2.689e-7, 3.303e-7, 3.981e-7,
4.84e-7, 5.91e-7, 7.363e-7, 9.087e-7, 1.139e-6, 1.455e-6,
1.866e-6, 2.44e-6, 3.115e-6, 3.941e-6, 4.891e-6, 5.992e-6,
7.111e-6, 8.296e-6, 9.21e-6, 9.987e-6, 1.044e-5, 1.073e-5,
1.092e-5, 1.106e-5, 1.138e-5, 1.171e-5, 1.186e-5, 1.186e-5,
1.179e-5, 1.166e-5, 1.151e-5, 1.16e-5, 1.197e-5, 1.241e-5,
1.268e-5, 1.26e-5, 1.184e-5, 1.063e-5, 9.204e-6, 7.584e-6,
6.053e-6, 4.482e-6, 3.252e-6, 2.337e-6, 1.662e-6, 1.18e-6,
8.15e-7, 5.95e-7, 4.354e-7, 3.302e-7, 2.494e-7, 1.93e-7,
1.545e-7, 1.25e-7, 1.039e-7, 8.602e-8, 7.127e-8, 5.897e-8,
4.838e-8, 4.018e-8, 3.28e-8, 2.72e-8, 2.307e-8, 1.972e-8,
1.654e-8, 1.421e-8, 1.174e-8, 1.004e-8, 8.739e-9, 7.358e-9,
6.242e-9, 5.303e-9, 4.567e-9, 3.94e-9, 3.375e-9, 2.864e-9,
2.422e-9, 2.057e-9, 1.75e-9, 1.505e-9, 1.294e-9, 1.101e-9,
9.401e-10, 8.018e-10, 6.903e-10, 5.965e-10, 5.087e-10, 4.364e-10,
3.759e-10, 3.247e-10, 2.809e-10, 2.438e-10, 2.123e-10, 1.853e-10,
1.622e-10, 1.426e-10, 1.26e-10, 1.125e-10, 1.022e-10, 9.582e-11,
9.388e-11, 9.801e-11, 1.08e-10, 1.276e-10, 1.551e-10, 1.903e-10,
2.291e-10, 2.724e-10, 3.117e-10, 3.4e-10, 3.562e-10, 3.625e-10,
3.619e-10, 3.429e-10, 3.221e-10, 2.943e-10, 2.645e-10, 2.338e-10,
2.062e-10, 1.901e-10, 1.814e-10, 1.827e-10, 1.906e-10, 1.984e-10,
2.04e-10, 2.068e-10, 2.075e-10, 2.018e-10, 1.959e-10, 1.897e-10,
1.852e-10, 1.791e-10, 1.696e-10, 1.634e-10, 1.598e-10, 1.561e-10,
1.518e-10, 1.443e-10, 1.377e-10, 1.346e-10, 1.342e-10, 1.375e-10,
1.525e-10, 1.767e-10, 2.108e-10, 2.524e-10, 2.981e-10, 3.477e-10,
4.262e-10, 5.326e-10, 6.646e-10, 8.321e-10, 1.069e-9, 1.386e-9,
1.743e-9, 2.216e-9, 2.808e-9, 3.585e-9, 4.552e-9, 5.907e-9,
7.611e-9, 9.774e-9, 1.255e-8, 1.666e-8, 2.279e-8, 3.221e-8,
4.531e-8, 6.4e-8, 9.187e-8, 1.295e-7, 1.825e-7, 2.431e-7,
3.181e-7, 4.009e-7, 4.941e-7, 5.88e-7, 6.623e-7, 7.155e-7,
7.451e-7, 7.594e-7, 7.541e-7, 7.467e-7, 7.527e-7, 7.935e-7,
8.461e-7, 8.954e-7, 9.364e-7, 9.843e-7, 1.024e-6, 1.05e-6,
1.059e-6, 1.074e-6, 1.072e-6, 1.043e-6, 9.789e-7, 8.803e-7,
7.662e-7, 6.378e-7, 5.133e-7, 3.958e-7, 2.914e-7, 2.144e-7,
1.57e-7, 1.14e-7, 8.47e-8, 6.2e-8, 4.657e-8, 3.559e-8, 2.813e-8,
2.222e-8, 1.769e-8, 1.391e-8, 1.125e-8, 9.186e-9, 7.704e-9,
6.447e-9, 5.381e-9, 4.442e-9, 3.669e-9, 3.057e-9, 2.564e-9,
2.153e-9, 1.784e-9, 1.499e-9, 1.281e-9, 1.082e-9, 9.304e-10,
8.169e-10, 6.856e-10, 5.866e-10, 5.043e-10, 4.336e-10, 3.731e-10,
3.175e-10, 2.745e-10, 2.374e-10, 2.007e-10, 1.737e-10, 1.508e-10,
1.302e-10, 1.13e-10, 9.672e-11, 8.375e-11, 7.265e-11, 6.244e-11,
5.343e-11, 4.654e-11, 3.975e-11, 3.488e-11, 3.097e-11, 2.834e-11,
2.649e-11, 2.519e-11, 2.462e-11, 2.443e-11, 2.44e-11, 2.398e-11,
2.306e-11, 2.183e-11, 2.021e-11, 1.821e-11, 1.599e-11, 1.403e-11,
1.196e-11, 1.023e-11, 8.728e-12, 7.606e-12, 6.941e-12, 6.545e-12,
6.484e-12, 6.6e-12, 6.718e-12, 6.785e-12, 6.746e-12, 6.724e-12,
6.764e-12, 6.995e-12, 7.144e-12, 7.32e-12, 7.33e-12, 7.208e-12,
6.789e-12, 6.09e-12, 5.337e-12, 4.62e-12, 4.037e-12, 3.574e-12,
3.311e-12, 3.346e-12, 3.566e-12, 3.836e-12, 4.076e-12, 4.351e-12,
4.691e-12, 5.114e-12, 5.427e-12, 6.167e-12, 7.436e-12, 8.842e-12,
1.038e-11, 1.249e-11, 1.54e-11, 1.915e-11, 2.48e-11, 3.256e-11,
4.339e-11, 5.611e-11, 7.519e-11, 1.037e-10, 1.409e-10, 1.883e-10,
2.503e-10, 3.38e-10, 4.468e-10, 5.801e-10, 7.335e-10, 8.98e-10,
1.11e-9, 1.363e-9, 1.677e-9, 2.104e-9, 2.681e-9, 3.531e-9,
4.621e-9, 6.106e-9, 8.154e-9, 1.046e-8, 1.312e-8, 1.607e-8,
1.948e-8, 2.266e-8, 2.495e-8, 2.655e-8, 2.739e-8, 2.739e-8,
2.662e-8, 2.589e-8, 2.59e-8, 2.664e-8, 2.833e-8, 3.023e-8,
```

```
3.305e-8, 3.558e-8, 3.793e-8, 3.961e-8, 4.056e-8, 4.102e-8,
4.025e-8, 3.917e-8, 3.706e-8, 3.493e-8, 3.249e-8, 3.096e-8,
3.011e-8, 3.111e-8, 3.395e-8, 3.958e-8, 4.875e-8, 6.066e-8,
7.915e-8, 1.011e-7, 1.3e-7, 1.622e-7, 2.003e-7, 2.448e-7,
2.863e-7, 3.317e-7, 3.655e-7, 3.96e-7, 4.098e-7, 4.168e-7,
4.198e-7, 4.207e-7, 4.289e-7, 4.384e-7, 4.471e-7, 4.524e-7,
4.574e-7, 4.633e-7, 4.785e-7, 5.028e-7, 5.371e-7, 5.727e-7,
5.955e-7, 5.998e-7, 5.669e-7, 5.082e-7, 4.397e-7, 3.596e-7,
2.814e-7, 2.074e-7, 1.486e-7, 1.057e-7, 7.25e-8, 4.946e-8,
3.43e-8, 2.447e-8, 1.793e-8, 1.375e-8, 1.096e-8, 9.091e-9,
7.709e-9, 6.631e-9, 5.714e-9, 4.886e-9, 4.205e-9, 3.575e-9,
3.07e-9, 2.631e-9, 2.284e-9, 2.002e-9, 1.745e-9, 1.509e-9,
1.284e-9, 1.084e-9, 9.163e-10, 7.663e-10, 6.346e-10, 5.283e-10,
4.354e-10, 3.59e-10, 2.982e-10, 2.455e-10, 2.033e-10, 1.696e-10,
1.432e-10, 1.211e-10, 1.02e-10, 8.702e-11, 7.38e-11, 6.293e-11,
5.343e-11, 4.532e-11, 3.907e-11, 3.365e-11, 2.945e-11, 2.558e-11,
2.192e-11, 1.895e-11, 1.636e-11, 1.42e-11, 1.228e-11, 1.063e-11,
9.348e-12, 8.2e-12, 7.231e-12, 6.43e-12, 5.702e-12, 5.052e-12,
4.469e-12, 4e-12, 3.679e-12, 3.387e-12, 3.197e-12, 3.158e-12,
3.327e-12, 3.675e-12, 4.292e-12, 5.437e-12, 7.197e-12, 1.008e-11,
1.437e-11, 2.035e-11, 2.905e-11, 4.062e-11, 5.528e-11, 7.177e-11,
9.064e-11, 1.109e-10, 1.297e-10, 1.473e-10, 1.652e-10, 1.851e-10,
2.079e-10, 2.313e-10, 2.619e-10, 2.958e-10, 3.352e-10, 3.796e-10,
4.295e-10, 4.923e-10, 5.49e-10, 5.998e-10, 6.388e-10, 6.645e-10,
6.712e-10, 6.549e-10, 6.38e-10, 6.255e-10, 6.253e-10, 6.459e-10,
6.977e-10, 7.59e-10, 8.242e-10, 8.92e-10, 9.403e-10, 9.701e-10,
9.483e-10, 9.135e-10, 8.617e-10, 7.921e-10, 7.168e-10, 6.382e-10,
5.677e-10, 5.045e-10, 4.572e-10, 4.312e-10, 4.145e-10, 4.192e-10,
4.541e-10, 5.368e-10, 6.771e-10, 8.962e-10, 1.21e-9, 1.659e-9,
2.33e-9, 3.249e-9, 4.495e-9, 5.923e-9, 7.642e-9, 9.607e-9,
1.178e-8, 1.399e-8, 1.584e-8, 1.73e-8, 1.816e-8, 1.87e-8,
1.868e-8, 1.87e-8, 1.884e-8, 1.99e-8, 2.15e-8, 2.258e-8,
2.364e-8, 2.473e-8, 2.602e-8, 2.689e-8, 2.731e-8, 2.816e-8,
2.859e-8, 2.839e-8, 2.703e-8, 2.451e-8, 2.149e-8, 1.787e-8,
1.449e-8, 1.111e-8, 8.282e-9, 6.121e-9, 4.494e-9, 3.367e-9,
2.487e-9, 1.885e-9, 1.503e-9, 1.249e-9, 1.074e-9, 9.427e-10,
8.439e-10, 7.563e-10, 6.772e-10, 6.002e-10, 5.254e-10, 4.588e-10,
3.977e-10, 3.449e-10, 3.003e-10, 2.624e-10, 2.335e-10, 2.04e-10,
1.771e-10, 1.534e-10, 1.296e-10, 1.097e-10, 9.173e-11, 7.73e-11,
6.547e-11, 5.191e-11, 4.198e-11, 3.361e-11, 2.732e-11, 2.244e-11,
1.791e-11, 1.509e-11, 1.243e-11, 1.035e-11, 8.969e-12, 7.394e-12,
6.323e-12, 5.282e-12, 4.543e-12, 3.752e-12, 3.14e-12, 2.6e-12,
2.194e-12, 1.825e-12, 1.511e-12, 1.245e-12, 1.024e-12, 8.539e-13,
7.227e-13, 6.102e-13, 5.189e-13, 4.43e-13, 3.774e-13, 3.236e-13,
2.8e-13, 2.444e-13, 2.156e-13, 1.932e-13, 1.775e-13, 1.695e-13,
1.672e-13, 1.704e-13, 1.825e-13, 2.087e-13, 2.614e-13, 3.377e-13,
4.817e-13, 6.989e-13, 1.062e-12, 1.562e-12, 2.288e-12, 3.295e-12,
4.55e-12, 5.965e-12, 7.546e-12, 9.395e-12, 1.103e-11, 1.228e-11,
1.318e-11, 1.38e-11, 1.421e-11, 1.39e-11, 1.358e-11, 1.336e-11,
1.342e-11, 1.356e-11, 1.424e-11, 1.552e-11, 1.73e-11, 1.951e-11,
2.128e-11, 2.249e-11, 2.277e-11, 2.226e-11, 2.111e-11, 1.922e-11,
1.775e-11, 1.661e-11, 1.547e-11, 1.446e-11, 1.323e-11, 1.21e-11,
1.054e-11, 9.283e-12, 8.671e-12, 8.67e-12, 9.429e-12, 1.062e-11,
1.255e-11, 1.506e-11, 1.818e-11, 2.26e-11, 2.831e-11, 3.723e-11,
5.092e-11, 6.968e-11, 9.826e-11, 1.349e-10, 1.87e-10, 2.58e-10,
3.43e-10, 4.424e-10, 5.521e-10, 6.812e-10, 8.064e-10, 9.109e-10,
9.839e-10, 1.028e-9, 1.044e-9, 1.029e-9, 1.005e-9, 1.002e-9,
1.038e-9, 1.122e-9, 1.233e-9, 1.372e-9, 1.524e-9, 1.665e-9,
1.804e-9, 1.908e-9, 2.015e-9, 2.117e-9, 2.219e-9, 2.336e-9,
2.531e-9, 2.805e-9, 3.189e-9, 3.617e-9, 4.208e-9, 4.911e-9,
5.619e-9, 6.469e-9, 7.188e-9, 7.957e-9, 8.503e-9, 9.028e-9,
9.571e-9, 9.99e-9, 1.055e-8, 1.102e-8, 1.132e-8, 1.141e-8,
1.145e-8, 1.145e-8, 1.176e-8, 1.224e-8, 1.304e-8, 1.388e-8,
1.445e-8, 1.453e-8, 1.368e-8, 1.22e-8, 1.042e-8, 8.404e-9,
6.403e-9, 4.643e-9, 3.325e-9, 2.335e-9, 1.638e-9, 1.19e-9,
9.161e-10, 7.412e-10, 6.226e-10, 5.516e-10, 5.068e-10, 4.831e-10,
4.856e-10, 5.162e-10, 5.785e-10, 6.539e-10, 7.485e-10, 8.565e-10,
9.534e-10, 1.052e-9, 1.115e-9, 1.173e-9, 1.203e-9, 1.224e-9,
1.243e-9, 1.248e-9, 1.261e-9, 1.265e-9, 1.25e-9, 1.217e-9,
1.176e-9, 1.145e-9, 1.153e-9, 1.199e-9, 1.278e-9, 1.366e-9,
1.426e-9, 1.444e-9, 1.365e-9, 1.224e-9, 1.051e-9, 8.539e-10,
6.564e-10, 4.751e-10, 3.404e-10, 2.377e-10, 1.631e-10, 1.114e-10,
7.87e-11, 5.793e-11, 4.284e-11, 3.3e-11, 2.62e-11, 2.152e-11,
1.777e-11, 1.496e-11, 1.242e-11, 1.037e-11, 8.725e-12, 7.004e-12,
5.718e-12, 4.769e-12, 3.952e-12, 3.336e-12, 2.712e-12, 2.213e-12,
1.803e-12, 1.492e-12, 1.236e-12, 1.006e-12, 8.384e-13, 7.063e-13,
5.879e-13, 4.93e-13, 4.171e-13, 3.569e-13, 3.083e-13, 2.688e-13,
2.333e-13, 2.035e-13, 1.82e-13, 1.682e-13, 1.635e-13, 1.674e-13,
1.769e-13, 2.022e-13, 2.485e-13, 3.127e-13, 4.25e-13, 5.928e-13,
8.514e-13, 1.236e-12, 1.701e-12, 2.392e-12, 3.231e-12, 4.35e-12,
5.559e-12, 6.915e-12, 8.519e-12, 1.013e-11, 1.146e-11, 1.24e-11,
1.305e-11, 1.333e-11, 1.318e-11, 1.263e-11, 1.238e-11, 1.244e-11,
1.305e-11, 1.432e-11, 1.623e-11, 1.846e-11, 2.09e-11, 2.328e-11,
2.526e-11, 2.637e-11, 2.702e-11, 2.794e-11, 2.889e-11, 2.989e-11,
3.231e-11, 3.68e-11, 4.375e-11, 5.504e-11, 7.159e-11, 9.502e-11,
1.279e-10, 1.645e-10, 2.098e-10, 2.618e-10, 3.189e-10, 3.79e-10,
4.303e-10, 4.753e-10, 5.027e-10, 5.221e-10, 5.293e-10, 5.346e-10,
```

```
5.467e-10, 5.796e-10, 6.2e-10, 6.454e-10, 6.705e-10, 6.925e-10,
7.233e-10, 7.35e-10, 7.538e-10, 7.861e-10, 8.077e-10, 8.132e-10,
7.749e-10, 7.036e-10, 6.143e-10, 5.093e-10, 4.089e-10, 3.092e-10,
2.299e-10, 1.705e-10, 1.277e-10, 9.723e-11, 7.533e-11, 6.126e-11,
5.154e-11, 4.428e-11, 3.913e-11, 3.521e-11, 3.297e-11, 3.275e-11,
3.46e-11, 3.798e-11, 4.251e-11, 4.745e-11, 5.232e-11, 5.606e-11,
5.82e-11, 5.88e-11, 5.79e-11, 5.661e-11, 5.491e-11, 5.366e-11,
5.341e-11, 5.353e-11, 5.336e-11, 5.293e-11, 5.248e-11, 5.235e-11,
5.208e-11, 5.322e-11, 5.521e-11, 5.725e-11, 5.827e-11, 5.685e-11,
5.245e-11, 4.612e-11, 3.884e-11, 3.129e-11, 2.404e-11, 1.732e-11,
1.223e-11, 8.574e-12, 5.888e-12, 3.986e-12, 2.732e-12, 1.948e-12,
1.414e-12, 1.061e-12, 8.298e-13, 6.612e-13, 5.413e-13, 4.472e-13,
3.772e-13, 3.181e-13, 2.645e-13, 2.171e-13, 1.778e-13, 1.464e-13,
1.183e-13, 9.637e-14, 7.991e-14, 6.668e-14, 5.57e-14, 4.663e-14,
3.848e-14, 3.233e-14, 2.706e-14, 2.284e-14, 1.944e-14, 1.664e-14,
1.43e-14, 1.233e-14, 1.066e-14, 9.234e-15, 8.023e-15, 6.993e-15,
6.119e-15, 5.384e-15, 4.774e-15, 4.283e-15, 3.916e-15, 3.695e-15,
3.682e-15, 4.004e-15, 4.912e-15, 6.853e-15, 1.056e-14, 1.712e-14,
2.804e-14, 4.516e-14, 7.113e-14, 1.084e-13, 1.426e-13, 1.734e-13,
1.978e-13, 2.194e-13, 2.388e-13, 2.489e-13, 2.626e-13, 2.865e-13,
3.105e-13, 3.387e-13, 3.652e-13, 3.984e-13, 4.398e-13, 4.906e-13,
5.55e-13, 6.517e-13, 7.813e-13, 9.272e-13, 1.164e-12, 1.434e-12,
1.849e-12, 2.524e-12, 3.328e-12, 4.523e-12, 6.108e-12, 8.207e-12,
1.122e-11, 1.477e-11, 1.9e-11, 2.412e-11, 2.984e-11, 3.68e-11,
4.353e-11, 4.963e-11, 5.478e-11, 5.903e-11, 6.233e-11, 6.483e-11,
6.904e-11, 7.569e-11, 8.719e-11, 1.048e-10, 1.278e-10, 1.557e-10,
1.869e-10, 2.218e-10, 2.61e-10, 2.975e-10, 3.371e-10, 3.746e-10,
4.065e-10, 4.336e-10, 4.503e-10, 4.701e-10, 4.8e-10, 4.917e-10,
5.038e-10, 5.128e-10, 5.143e-10, 5.071e-10, 5.019e-10, 5.025e-10,
5.183e-10, 5.496e-10, 5.877e-10, 6.235e-10, 6.42e-10, 6.234e-10,
5.698e-10, 4.916e-10, 4.022e-10, 3.126e-10, 2.282e-10, 1.639e-10,
1.142e-10, 7.919e-11, 5.69e-11, 4.313e-11, 3.413e-11, 2.807e-11,
2.41e-11, 2.166e-11, 2.024e-11, 1.946e-11, 1.929e-11, 1.963e-11,
2.035e-11, 2.162e-11, 2.305e-11, 2.493e-11, 2.748e-11, 3.048e-11,
3.413e-11, 3.754e-11, 4.155e-11, 4.635e-11, 5.11e-11, 5.734e-11,
6.338e-11, 6.99e-11, 7.611e-11, 8.125e-11, 8.654e-11, 8.951e-11,
9.182e-11, 9.31e-11, 9.273e-11, 9.094e-11, 8.849e-11, 8.662e-11,
8.67e-11, 8.972e-11, 9.566e-11, 1.025e-10, 1.083e-10, 1.111e-10,
1.074e-10, 9.771e-11, 8.468e-11, 6.958e-11, 5.47e-11, 4.04e-11,
2.94e-11, 2.075e-11, 1.442e-11, 1.01e-11, 7.281e-12, 5.409e-12,
4.138e-12, 3.304e-12, 2.784e-12, 2.473e-12, 2.273e-12, 2.186e-12,
2.118e-12, 2.066e-12, 1.958e-12, 1.818e-12, 1.675e-12, 1.509e-12,
1.349e-12, 1.171e-12, 9.838e-13, 8.213e-13, 6.765e-13, 5.378e-13,
4.161e-13, 3.119e-13, 2.279e-13, 1.637e-13, 1.152e-13, 8.112e-14,
5.919e-14, 4.47e-14, 3.492e-14, 2.811e-14, 2.319e-14, 1.948e-14,
1.66e-14, 1.432e-14, 1.251e-14, 1.109e-14, 1.006e-14, 9.45e-15,
9.384e-15, 1.012e-14, 1.216e-14, 1.636e-14, 2.305e-14, 3.488e-14,
5.572e-14, 8.479e-14, 1.265e-13, 1.905e-13, 2.73e-13, 3.809e-13,
4.955e-13, 6.303e-13, 7.861e-13, 9.427e-13, 1.097e-12, 1.212e-12,
1.328e-12, 1.415e-12, 1.463e-12, 1.495e-12, 1.571e-12, 1.731e-12,
1.981e-12, 2.387e-12, 2.93e-12, 3.642e-12, 4.584e-12, 5.822e-12,
7.278e-12, 9.193e-12, 1.135e-11, 1.382e-11, 1.662e-11, 1.958e-11,
2.286e-11, 2.559e-11, 2.805e-11, 2.988e-11, 3.106e-11, 3.182e-11,
3.2e-11, 3.258e-11, 3.362e-11, 3.558e-11, 3.688e-11, 3.8e-11,
3.929e-11, 4.062e-11, 4.186e-11, 4.293e-11, 4.48e-11, 4.643e-11,
4.704e-11, 4.571e-11, 4.206e-11, 3.715e-11, 3.131e-11, 2.541e-11,
1.978e-11, 1.508e-11, 1.146e-11, 8.7e-12, 6.603e-12, 5.162e-12,
4.157e-12, 3.408e-12, 2.829e-12, 2.405e-12, 2.071e-12, 1.826e-12,
1.648e-12, 1.542e-12, 1.489e-12, 1.485e-12, 1.493e-12, 1.545e-12,
1.637e-12, 1.814e-12, 2.061e-12, 2.312e-12, 2.651e-12, 3.03e-12,
3.46e-12, 3.901e-12, 4.306e-12, 4.721e-12, 5.008e-12, 5.281e-12,
5.541e-12, 5.791e-12, 6.115e-12, 6.442e-12, 6.68e-12, 6.791e-12,
6.831e-12, 6.839e-12, 6.946e-12, 7.128e-12, 7.537e-12, 8.036e-12,
8.392e-12, 8.526e-12, 8.11e-12, 7.325e-12, 6.329e-12, 5.183e-12,
4.081e-12, 2.985e-12, 2.141e-12, 1.492e-12, 1.015e-12, 6.684e-13,
4.414e-13, 2.987e-13, 2.038e-13, 1.391e-13, 9.86e-14, 7.24e-14,
5.493e-14, 4.288e-14, 3.427e-14, 2.787e-14, 2.296e-14, 1.909e-14,
1.598e-14, 1.344e-14, 1.135e-14, 9.616e-15, 8.169e-15, 6.957e-15,
5.938e-15, 5.08e-15, 4.353e-15, 3.738e-15, 3.217e-15, 2.773e-15,
2.397e-15, 2.077e-15, 1.805e-15, 1.575e-15, 1.382e-15, 1.221e-15,
1.09e-15, 9.855e-16, 9.068e-16, 8.537e-16, 8.27e-16, 8.29e-16,
8.634e-16, 9.359e-16, 1.055e-15, 1.233e-15, 1.486e-15, 1.839e-15,
2.326e-15, 2.998e-15, 3.934e-15, 5.256e-15, 7.164e-15, 9.984e-15,
1.427e-14, 2.099e-14, 3.196e-14, 5.121e-14, 7.908e-14, 1.131e-13,
1.602e-13, 2.239e-13, 3.075e-13, 4.134e-13, 5.749e-13, 7.886e-13,
1.071e-12, 1.464e-12, 2.032e-12, 2.8e-12, 3.732e-12, 4.996e-12,
6.483e-12, 8.143e-12, 1.006e-11, 1.238e-11, 1.484e-11, 1.744e-11,
2.02e-11, 2.274e-11, 2.562e-11, 2.848e-11, 3.191e-11, 3.617e-11,
4.081e-11, 4.577e-11, 4.937e-11, 5.204e-11, 5.401e-11, 5.462e-11,
5.507e-11, 5.51e-11, 5.605e-11, 5.686e-11, 5.739e-11, 5.766e-11,
5.74e-11, 5.754e-11, 5.761e-11, 5.777e-11, 5.712e-11, 5.51e-11,
5.088e-11, 4.438e-11, 3.728e-11, 2.994e-11, 2.305e-11, 1.715e-11,
1.256e-11, 9.208e-12, 6.745e-12, 5.014e-12, 3.785e-12, 2.9e-12,
2.239e-12, 1.757e-12, 1.414e-12, 1.142e-12, 9.482e-13, 8.01e-13,
6.961e-13, 6.253e-13, 5.735e-13, 5.433e-13, 5.352e-13, 5.493e-13,
5.706e-13, 6.068e-13, 6.531e-13, 7.109e-13, 7.767e-13, 8.59e-13,
9.792e-13, 1.142e-12, 1.371e-12, 1.65e-12, 1.957e-12, 2.302e-12,
```

```
    2.705e-12, 3.145e-12, 3.608e-12, 4.071e-12, 4.602e-12, 5.133e-12,
    5.572e-12, 5.987e-12, 6.248e-12, 6.533e-12, 6.757e-12, 6.935e-12,
    7.224e-12, 7.422e-12, 7.538e-12, 7.547e-12, 7.495e-12, 7.543e-12,
    7.725e-12, 8.139e-12, 8.627e-12, 9.146e-12, 9.443e-12, 9.318e-12,
    8.649e-12, 7.512e-12, 6.261e-12, 4.915e-12, 3.647e-12, 2.597e-12,
    1.785e-12, 1.242e-12, 8.66e-13, 6.207e-13, 4.61e-13, 3.444e-13,
    2.634e-13, 2.1e-13, 1.725e-13, 1.455e-13, 1.237e-13, 1.085e-13,
    9.513e-14, 7.978e-14, 6.603e-14, 5.288e-14, 4.084e-14, 2.952e-14,
    2.157e-14, 1.593e-14, 1.199e-14, 9.267e-15, 7.365e-15, 6.004e-15,
    4.995e-15, 4.218e-15, 3.601e-15, 3.101e-15, 2.692e-15, 2.36e-15,
    2.094e-15, 1.891e-15, 1.755e-15, 1.699e-15, 1.755e-15, 1.987e-15,
    2.506e-15, 3.506e-15, 5.289e-15, 8.311e-15, 1.325e-14, 2.129e-14,
    3.237e-14, 4.595e-14, 6.441e-14, 8.433e-14, 1.074e-13, 1.383e-13,
    1.762e-13, 2.281e-13, 2.831e-13, 3.523e-13, 4.38e-13, 5.304e-13,
    6.29e-13, 7.142e-13, 8.032e-13, 8.934e-13, 9.888e-13, 1.109e-12,
    1.261e-12, 1.462e-12, 1.74e-12, 2.099e-12, 2.535e-12, 3.008e-12,
    3.462e-12, 3.856e-12, 4.098e-12, 4.239e-12, 4.234e-12, 4.132e-12,
    3.986e-12, 3.866e-12, 3.829e-12, 3.742e-12, 3.705e-12, 3.694e-12,
    3.765e-12, 3.849e-12, 3.929e-12, 4.056e-12, 4.092e-12, 4.047e-12,
    3.792e-12, 3.407e-12, 2.953e-12, 2.429e-12, 1.931e-12, 1.46e-12,
    1.099e-12, 8.199e-13, 6.077e-13, 4.449e-13, 3.359e-13, 2.524e-13,
    1.881e-13, 1.391e-13, 1.02e-13, 7.544e-14, 5.555e-14, 4.22e-14,
    3.321e-14, 2.686e-14, 2.212e-14, 1.78e-14, 1.369e-14, 1.094e-14,
    9.13e-15, 8.101e-15, 7.828e-15, 8.393e-15, 1.012e-14, 1.259e-14,
    1.538e-14, 1.961e-14, 2.619e-14, 3.679e-14, 5.049e-14, 6.917e-14,
    8.88e-14, 1.115e-13, 1.373e-13, 1.619e-13, 1.878e-13, 2.111e-13,
    2.33e-13, 2.503e-13, 2.613e-13, 2.743e-13, 2.826e-13, 2.976e-13,
    3.162e-13, 3.36e-13, 3.491e-13, 3.541e-13, 3.595e-13, 3.608e-13,
    3.709e-13, 3.869e-13, 4.12e-13, 4.366e-13, 4.504e-13, 4.379e-13,
    3.955e-13, 3.385e-13, 2.741e-13, 2.089e-13, 1.427e-13, 9.294e-14,
    5.775e-14, 3.565e-14, 2.21e-14, 1.398e-14, 9.194e-15, 6.363e-15,
    4.644e-15, 3.55e-15, 2.808e-15, 2.274e-15, 1.871e-15, 1.557e-15,
    1.308e-15, 1.108e-15, 9.488e-16, 8.222e-16, 7.238e-16, 6.506e-16,
    6.008e-16, 5.742e-16, 5.724e-16, 5.991e-16, 6.625e-16, 7.775e-16,
    9.734e-16, 1.306e-15, 1.88e-15, 2.879e-15, 4.616e-15, 7.579e-15,
    1.248e-14, 2.03e-14, 3.244e-14, 5.171e-14, 7.394e-14, 9.676e-14,
    1.199e-13, 1.467e-13, 1.737e-13, 2.02e-13, 2.425e-13, 3.016e-13,
    3.7e-13, 4.617e-13, 5.949e-13, 7.473e-13, 9.378e-13, 1.191e-12,
    1.481e-12, 1.813e-12, 2.232e-12, 2.722e-12, 3.254e-12, 3.845e-12,
    4.458e-12, 5.048e-12, 5.511e-12, 5.898e-12, 6.204e-12, 6.293e-12,
    6.386e-12, 6.467e-12, 6.507e-12, 6.466e-12, 6.443e-12, 6.598e-12,
    6.873e-12, 7.3e-12, 7.816e-12, 8.368e-12, 8.643e-12, 8.466e-12,
    7.871e-12, 6.853e-12, 5.714e-12, 4.482e-12, 3.392e-12, 2.613e-12,
    2.008e-12, 1.562e-12, 1.228e-12, 9.888e-13, 7.646e-13, 5.769e-13,
    4.368e-13, 3.324e-13, 2.508e-13, 1.916e-13
  };

  static double xfcrev[15] =
    { 1.003, 1.009, 1.015, 1.023, 1.029, 1.033, 1.037,
    1.039, 1.04, 1.046, 1.036, 1.027, 1.01, 1.002, 1.
  };

  double sfac;

  /* Get H2O continuum absorption... */
  const double xw = nu / 10 + 1;
  if (xw >= 1 && xw < 2001) {
    const int iw = (int) xw;
    const double dw = xw - iw;
    const double ew = 1 - dw;
    const double cw296 = ew * h2o296[iw - 1] + dw * h2o296[iw];
    const double cw260 = ew * h2o260[iw - 1] + dw * h2o260[iw];
    const double cwfrn = ew * h2ofrn[iw - 1] + dw * h2ofrn[iw];
    if (nu <= 820 || nu >= 960) {
      sfac = 1;
    } else {
      const double xx = (nu - 820) / 10;
      const int ix = (int) xx;
      const double dx = xx - ix;
      sfac = (1 - dx) * xfcrev[ix] + dx * xfcrev[ix + 1];
    }
    const double ctwslf =
      sfac * cw296 * pow(cw260 / cw296, (296 - t) / (296 - 260));
    const double vf2 = POW2(nu - 370);
    const double vf6 = POW3(vf2);
    const double fscal = 36100 / (vf2 + vf6 * 1e-8 + 36100) * -.25 + 1;
    const double ctwfrn = cwfrn * fscal;
    const double a1 = nu * u * tanh(.7193876 / t * nu);
    const double a2 = 296 / t;
    const double a3 = p / P0 * (q * ctwslf + (1 - q) * ctwfrn) * 1e-20;
    return a1 * a2 * a3;
  } else
    return 0;
}
```

ctmn2()

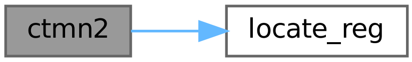
```
double ctmn2 (
            const double nu,
            const double p,
            const double t )
```

Compute nitrogen continuum (absorption coefficient).

Definition at line 2796 of file jurassic.c.

```
                     {

  static double ba[98] = { 0., 4.45e-8, 5.22e-8, 6.46e-8, 7.75e-8, 9.03e-8,
    1.06e-7, 1.21e-7, 1.37e-7, 1.57e-7, 1.75e-7, 2.01e-7, 2.3e-7,
    2.59e-7, 2.95e-7, 3.26e-7, 3.66e-7, 4.05e-7, 4.47e-7, 4.92e-7,
    5.34e-7, 5.84e-7, 6.24e-7, 6.67e-7, 7.14e-7, 7.26e-7, 7.54e-7,
    7.84e-7, 8.09e-7, 8.42e-7, 8.62e-7, 8.87e-7, 9.11e-7, 9.36e-7,
    9.76e-7, 1.03e-6, 1.11e-6, 1.23e-6, 1.39e-6, 1.61e-6, 1.76e-6,
    1.94e-6, 1.97e-6, 1.87e-6, 1.75e-6, 1.56e-6, 1.42e-6, 1.35e-6,
    1.32e-6, 1.29e-6, 1.29e-6, 1.29e-6, 1.3e-6, 1.32e-6, 1.33e-6,
    1.34e-6, 1.35e-6, 1.33e-6, 1.31e-6, 1.29e-6, 1.24e-6, 1.2e-6,
    1.16e-6, 1.1e-6, 1.04e-6, 9.96e-7, 9.38e-7, 8.63e-7, 7.98e-7,
    7.26e-7, 6.55e-7, 5.94e-7, 5.35e-7, 4.74e-7, 4.24e-7, 3.77e-7,
    3.33e-7, 2.96e-7, 2.63e-7, 2.34e-7, 2.08e-7, 1.85e-7, 1.67e-7,
    1.47e-7, 1.32e-7, 1.2e-7, 1.09e-7, 9.85e-8, 9.08e-8, 8.18e-8,
    7.56e-8, 6.85e-8, 6.14e-8, 5.83e-8, 5.77e-8, 5e-8, 4.32e-8, 0.
  };

  static double betaa[98] = { 802., 802., 761., 722., 679., 646., 609., 562.,
    511., 472., 436., 406., 377., 355., 338., 319., 299., 278., 255.,
    233., 208., 184., 149., 107., 66., 25., -13., -49., -82., -104.,
    -119., -130., -139., -144., -146., -146., -147., -148., -150.,
    -153., -160., -169., -181., -189., -195., -200., -205., -209.,
    -211., -210., -210., -209., -205., -199., -190., -180., -168.,
    -157., -143., -126., -108., -89., -63., -32., 1., 35., 65., 95.,
    121., 141., 152., 161., 164., 164., 161., 155., 148., 143., 137.,
    133., 131., 133., 139., 150., 165., 187., 213., 248., 284., 321.,
    372., 449., 514., 569., 609., 642., 673., 673.
  };

  static double nua[98] = { 2120., 2125., 2130., 2135., 2140., 2145., 2150.,
    2155., 2160., 2165., 2170., 2175., 2180., 2185., 2190., 2195.,
    2200., 2205., 2210., 2215., 2220., 2225., 2230., 2235., 2240.,
    2245., 2250., 2255., 2260., 2265., 2270., 2275., 2280., 2285.,
    2290., 2295., 2300., 2305., 2310., 2315., 2320., 2325., 2330.,
    2335., 2340., 2345., 2350., 2355., 2360., 2365., 2370., 2375.,
    2380., 2385., 2390., 2395., 2400., 2405., 2410., 2415., 2420.,
    2425., 2430., 2435., 2440., 2445., 2450., 2455., 2460., 2465.,
    2470., 2475., 2480., 2485., 2490., 2495., 2500., 2505., 2510.,
    2515., 2520., 2525., 2530., 2535., 2540., 2545., 2550., 2555.,
    2560., 2565., 2570., 2575., 2580., 2585., 2590., 2595., 2600., 2605.
  };

  const double q_n2 = 0.79, t0 = 273.0, tr = 296.0;

  /* Check wavenumber range... */
  if (nu < nua[0] || nu > nua[97])
    return 0;

  /* Interpolate B and beta... */
  const int idx = locate_reg(nua, 98, nu);
  const double b = LIN(nua[idx], ba[idx], nua[idx + 1], ba[idx + 1], nu);
  const double beta =
    LIN(nua[idx], betaa[idx], nua[idx + 1], betaa[idx + 1], nu);

  /* Compute absorption coefficient... */
  return 0.1 * POW2(p / P0 * t0 / t) * exp(beta * (1 / tr - 1 / t))
    * q_n2 * b * (q_n2 + (1 - q_n2) * (1.294 - 0.4545 * t / tr));
}
```

Here is the call graph for this function:



ctmo2()

```
double ctmo2 (
            const double nu,
            const double p,
            const double t )
```

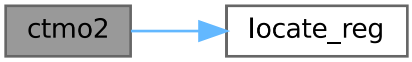
Compute oxygen continuum (absorption coefficient).

Definition at line 2861 of file jurassic.c.

```
                {

  static double ba[90] = { 0., .061, .074, .084, .096, .12, .162, .208, .246,
    .285, .314, .38, .444, .5, .571, .673, .768, .853, .966, 1.097,
    1.214, 1.333, 1.466, 1.591, 1.693, 1.796, 1.922, 2.037, 2.154,
    2.264, 2.375, 2.508, 2.671, 2.847, 3.066, 3.417, 3.828, 4.204,
    4.453, 4.599, 4.528, 4.284, 3.955, 3.678, 3.477, 3.346, 3.29,
    3.251, 3.231, 3.226, 3.212, 3.192, 3.108, 3.033, 2.911, 2.798,
    2.646, 2.508, 2.322, 2.13, 1.928, 1.757, 1.588, 1.417, 1.253,
    1.109, .99, .888, .791, .678, .587, .524, .464, .403, .357, .32,
    .29, .267, .242, .215, .182, .16, .146, .128, .103, .087, .081,
    .071, .064, 0.
  };

  static double betaa[90] = { 467., 467., 400., 315., 379., 368., 475., 521.,
    531., 512., 442., 444., 430., 381., 335., 324., 296., 248., 215.,
    193., 158., 127., 101., 71., 31., -6., -26., -47., -63., -79.,
    -88., -88., -87., -90., -98., -99., -109., -134., -160., -167.,
    -164., -158., -153., -151., -156., -166., -168., -173., -170.,
    -161., -145., -126., -108., -84., -59., -29., 4., 41., 73., 97.,
    123., 159., 198., 220., 242., 256., 281., 311., 334., 319., 313.,
    321., 323., 310., 315., 320., 335., 361., 378., 373., 338., 319.,
    346., 322., 291., 290., 350., 371., 504., 504.
  };

  static double nua[90] = { 1360., 1365., 1370., 1375., 1380., 1385., 1390.,
    1395., 1400., 1405., 1410., 1415., 1420., 1425., 1430., 1435.,
    1440., 1445., 1450., 1455., 1460., 1465., 1470., 1475., 1480.,
    1485., 1490., 1495., 1500., 1505., 1510., 1515., 1520., 1525.,
    1530., 1535., 1540., 1545., 1550., 1555., 1560., 1565., 1570.,
    1575., 1580., 1585., 1590., 1595., 1600., 1605., 1610., 1615.,
    1620., 1625., 1630., 1635., 1640., 1645., 1650., 1655., 1660.,
    1665., 1670., 1675., 1680., 1685., 1690., 1695., 1700., 1705.,
    1710., 1715., 1720., 1725., 1730., 1735., 1740., 1745., 1750.,
    1755., 1760., 1765., 1770., 1775., 1780., 1785., 1790., 1795.,
    1800., 1805.
  };

  const double q_o2 = 0.21, t0 = 273, tr = 296;

  /* Check wavenumber range... */
  if (nu < nua[0] || nu > nua[89])
    return 0;

  /* Interpolate B and beta... */
  const int idx = locate_reg(nua, 90, nu);
  const double b = LIN(nua[idx], ba[idx], nua[idx + 1], ba[idx + 1], nu);
  const double beta =
    LIN(nua[idx], betaa[idx], nua[idx + 1], betaa[idx + 1], nu);

```

```
  /* Compute absorption coefficient... */
  return 0.1 * POW2(p / P0 * t0 / t) * exp(beta * (1 / tr - 1 / t)) * q_o2 *
    b;
}
```

Here is the call graph for this function:



copy_atm()

```
void copy_atm (
            const ctl_t * ctl,
            atm_t * atm_dest,
            const atm_t * atm_src,
            const int init )
```

Copy and initialize atmospheric data.

Definition at line 2921 of file jurassic.c.

```
                         {

  /* Data size... */
  const size_t s = (size_t) atm_src->np * sizeof(double);

  /* Copy data... */
  atm_dest->np = atm_src->np;
  memcpy(atm_dest->time, atm_src->time, s);
  memcpy(atm_dest->z, atm_src->z, s);
  memcpy(atm_dest->lon, atm_src->lon, s);
  memcpy(atm_dest->lat, atm_src->lat, s);
  memcpy(atm_dest->p, atm_src->p, s);
  memcpy(atm_dest->t, atm_src->t, s);
  for (int ig = 0; ig < ctl->ng; ig++)
    memcpy(atm_dest->q[ig], atm_src->q[ig], s);
  for (int iw = 0; iw < ctl->nw; iw++)
    memcpy(atm_dest->k[iw], atm_src->k[iw], s);
  atm_dest->clz = atm_src->clz;
  atm_dest->cldz = atm_src->cldz;
  for (int icl = 0; icl < ctl->ncl; icl++)
    atm_dest->clk[icl] = atm_src->clk[icl];
  atm_dest->sfz = atm_src->sfz;
  atm_dest->sfp = atm_src->sfp;
  atm_dest->sft = atm_src->sft;
  for (int isf = 0; isf < ctl->nsf; isf++)
    atm_dest->sfeps[isf] = atm_src->sfeps[isf];

  /* Initialize... */
  if (init)
    for (int ip = 0; ip < atm_dest->np; ip++) {
      atm_dest->p[ip] = 0;
      atm_dest->t[ip] = 0;
      for (int ig = 0; ig < ctl->ng; ig++)
        atm_dest->q[ig][ip] = 0;
      for (int iw = 0; iw < ctl->nw; iw++)
        atm_dest->k[iw][ip] = 0;
      atm_dest->clz = 0;
      atm_dest->cldz = 0;
      for (int icl = 0; icl < ctl->ncl; icl++)
        atm_dest->clk[icl] = 0;
      atm_dest->sfz = 0;
      atm_dest->sfp = 0;
      atm_dest->sft = 0;
      for (int isf = 0; isf < ctl->nsf; isf++)
        atm_dest->sfeps[isf] = 1;
    }
}
```

copy_obs()

```
void copy_obs (
        const ctl_t * ctl,
        obs_t * obs_dest,
        const obs_t * obs_src,
        const int init )
```

Copy and initialize observation data.

Definition at line 2975 of file jurassic.c.

```
                     {

  /* Data size... */
  const size_t s = (size_t) obs_src->nr * sizeof(double);

  /* Copy data... */
  obs_dest->nr = obs_src->nr;
  memcpy(obs_dest->time, obs_src->time, s);
  memcpy(obs_dest->obsz, obs_src->obsz, s);
  memcpy(obs_dest->obslon, obs_src->obslon, s);
  memcpy(obs_dest->obslat, obs_src->obslat, s);
  memcpy(obs_dest->vpz, obs_src->vpz, s);
  memcpy(obs_dest->vplon, obs_src->vplon, s);
  memcpy(obs_dest->vplat, obs_src->vplat, s);
  memcpy(obs_dest->tpz, obs_src->tpz, s);
  memcpy(obs_dest->tplon, obs_src->tplon, s);
  memcpy(obs_dest->tplat, obs_src->tplat, s);
  for (int id = 0; id < ctl->nd; id++)
    memcpy(obs_dest->rad[id], obs_src->rad[id], s);
  for (int id = 0; id < ctl->nd; id++)
    memcpy(obs_dest->tau[id], obs_src->tau[id], s);

  /* Initialize... */
  if (init)
    for (int id = 0; id < ctl->nd; id++)
      for (int ir = 0; ir < obs_dest->nr; ir++)
        if (isfinite(obs_dest->rad[id][ir])) {
          obs_dest->rad[id][ir] = 0;
          obs_dest->tau[id][ir] = 0;
        }
}
```

find_emitter()

```
int find_emitter (
        const ctl_t * ctl,
        const char * emitter )
```

Find index of an emitter.

Definition at line 3013 of file jurassic.c.

```
                          {

  for (int ig = 0; ig < ctl->ng; ig++)
    if (strcasecmp(ctl->emitter[ig], emitter) == 0)
      return ig;

  return -1;
}
```

formod()

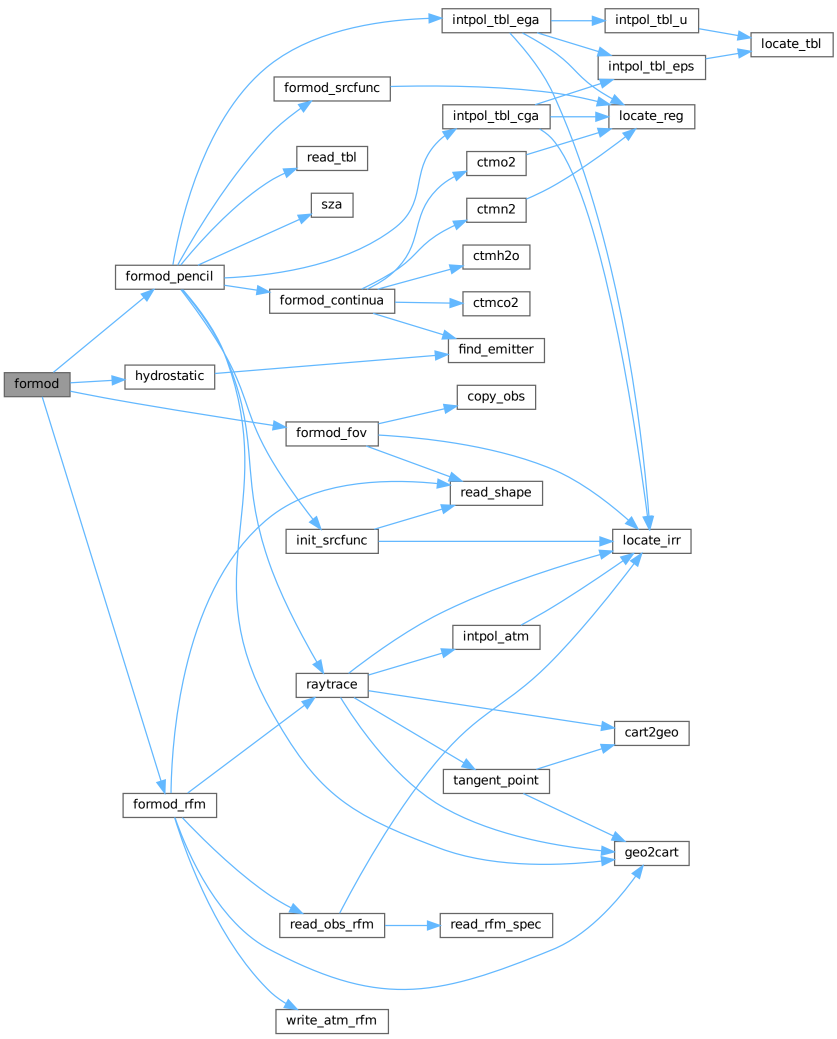
```
void formod (
        const ctl_t * ctl,
        atm_t * atm,
        obs_t * obs )
```

Determine ray paths and compute radiative transfer.

Definition at line 3026 of file jurassic.c.

```
                {

  int *mask;

  /* Allocate... */
  ALLOC(mask, int,
        ND * NR);

  /* Save observation mask... */
  for (int id = 0; id < ctl->nd; id++)
    for (int ir = 0; ir < obs->nr; ir++)
      mask[id * NR + ir] = !isfinite(obs->rad[id][ir]);

  /* Hydrostatic equilibrium... */
  hydrostatic(ctl, atm);

  /* CGA or EGA forward model... */
  if (ctl->formod == 0 || ctl->formod == 1)
    for (int ir = 0; ir < obs->nr; ir++)
      formod_pencil(ctl, atm, obs, ir);

  /* Call RFM... */
  else if (ctl->formod == 2)
    formod_rfm(ctl, atm, obs);

  /* Apply field-of-view convolution... */
  formod_fov(ctl, obs);

  /* Convert radiance to brightness temperature... */
  if (ctl->write_bbt)
    for (int id = 0; id < ctl->nd; id++)
      for (int ir = 0; ir < obs->nr; ir++)
        obs->rad[id][ir] = BRIGHT(obs->rad[id][ir], ctl->nu[id]);

  /* Apply observation mask... */
  for (int id = 0; id < ctl->nd; id++)
    for (int ir = 0; ir < obs->nr; ir++)
      if (mask[id * NR + ir])
        obs->rad[id][ir] = NAN;

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

Here is the call graph for this function:



formod_continua()

```
void formod_continua (
            const ctl_t * ctl,
            const los_t * los,
            const int ip,
            double * beta )
```

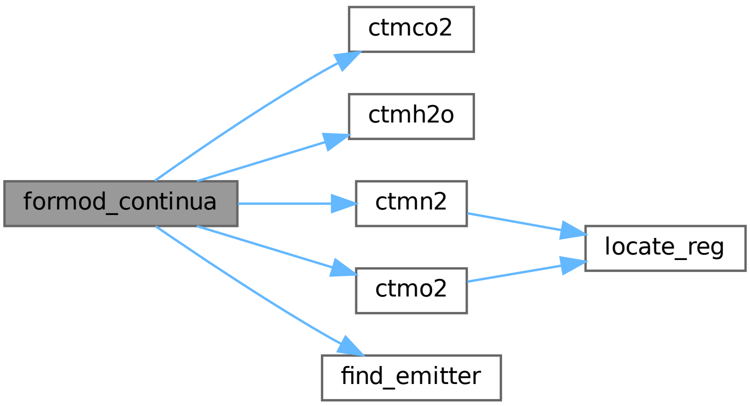
Compute absorption coefficient of continua.

Definition at line 3075 of file jurassic.c.

```
                 {

  static int ig_co2 = -999, ig_h2o = -999;

  /* Extinction... */
  for (int id = 0; id < ctl->nd; id++)
    beta[id] = los->k[ip][id];

```

```
  /* CO2 continuum... */
  if (ctl->ctm_co2) {
    if (ig_co2 == -999)
      ig_co2 = find_emitter(ctl, "CO2");
    if (ig_co2 >= 0)
      for (int id = 0; id < ctl->nd; id++)
        beta[id] += ctmco2(ctl->nu[id], los->p[ip], los->t[ip],
                           los->u[ip][ig_co2]) / los->ds[ip];
  }

  /* H2O continuum... */
  if (ctl->ctm_h2o) {
    if (ig_h2o == -999)
      ig_h2o = find_emitter(ctl, "H2O");
    if (ig_h2o >= 0)
      for (int id = 0; id < ctl->nd; id++)
        beta[id] += ctmh2o(ctl->nu[id], los->p[ip], los->t[ip],
                           los->q[ip][ig_h2o], los->u[ip][ig_h2o])
          / los->ds[ip];
  }

  /* N2 continuum... */
  if (ctl->ctm_n2)
    for (int id = 0; id < ctl->nd; id++)
      beta[id] += ctmn2(ctl->nu[id], los->p[ip], los->t[ip]);

  /* O2 continuum... */
  if (ctl->ctm_o2)
    for (int id = 0; id < ctl->nd; id++)
      beta[id] += ctmo2(ctl->nu[id], los->p[ip], los->t[ip]);
}
```

Here is the call graph for this function:



formod_fov()

```
void formod_fov (
               const ctl_t * ctl,
               obs_t * obs )
```

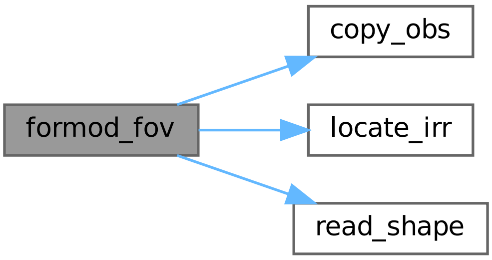
Apply field of view convolution.

Definition at line 3121 of file jurassic.c.

```
              {

  static double dz[NSHAPE], w[NSHAPE];

```

```
  static int init = 0, n;

  obs_t *obs2;

  double rad[ND][NR], tau[ND][NR], z[NR];

  /* Do not take into account FOV... */
  if (ctl->fov[0] == '-')
    return;

  /* Initialize FOV data... */
  if (!init) {
    init = 1;
    read_shape(ctl->fov, dz, w, &n);
  }

  /* Allocate... */
  ALLOC(obs2, obs_t, 1);

  /* Copy observation data... */
  copy_obs(ctl, obs2, obs, 0);

  /* Loop over ray paths... */
  for (int ir = 0; ir < obs->nr; ir++) {

    /* Get radiance and transmittance profiles... */
    int nz = 0;
    for (int ir2 = MAX(ir - NFOV, 0);
         ir2 < MIN(ir + 1 + NFOV, obs->nr); ir2++)
      if (obs->time[ir2] == obs->time[ir]) {
        z[nz] = obs2->vpz[ir2];
        for (int id = 0; id < ctl->nd; id++) {
          rad[id][nz] = obs2->rad[id][ir2];
          tau[id][nz] = obs2->tau[id][ir2];
        }
        nz++;
      }
    if (nz < 2)
      ERRMSG("Cannot apply FOV convolution!");

    /* Convolute profiles with FOV... */
    double wsum = 0;
    for (int id = 0; id < ctl->nd; id++) {
      obs->rad[id][ir] = 0;
      obs->tau[id][ir] = 0;
    }
    for (int i = 0; i < n; i++) {
      const double zfov = obs->vpz[ir] + dz[i];
      const int idx = locate_irr(z, nz, zfov);
      for (int id = 0; id < ctl->nd; id++) {
        obs->rad[id][ir] += w[i]
          * LIN(z[idx], rad[id][idx], z[idx + 1], rad[id][idx + 1], zfov);
        obs->tau[id][ir] += w[i]
          * LIN(z[idx], tau[id][idx], z[idx + 1], tau[id][idx + 1], zfov);
      }
      wsum += w[i];
    }
    for (int id = 0; id < ctl->nd; id++) {
      obs->rad[id][ir] /= wsum;
      obs->tau[id][ir] /= wsum;
    }
  }

  /* Free... */
  free(obs2);
}
```

Here is the call graph for this function:



formod_pencil()

```
void formod_pencil (
            const ctl_t * ctl,
            const atm_t * atm,
            obs_t * obs,
            const int ir )
```

Compute radiative transfer for a pencil beam.

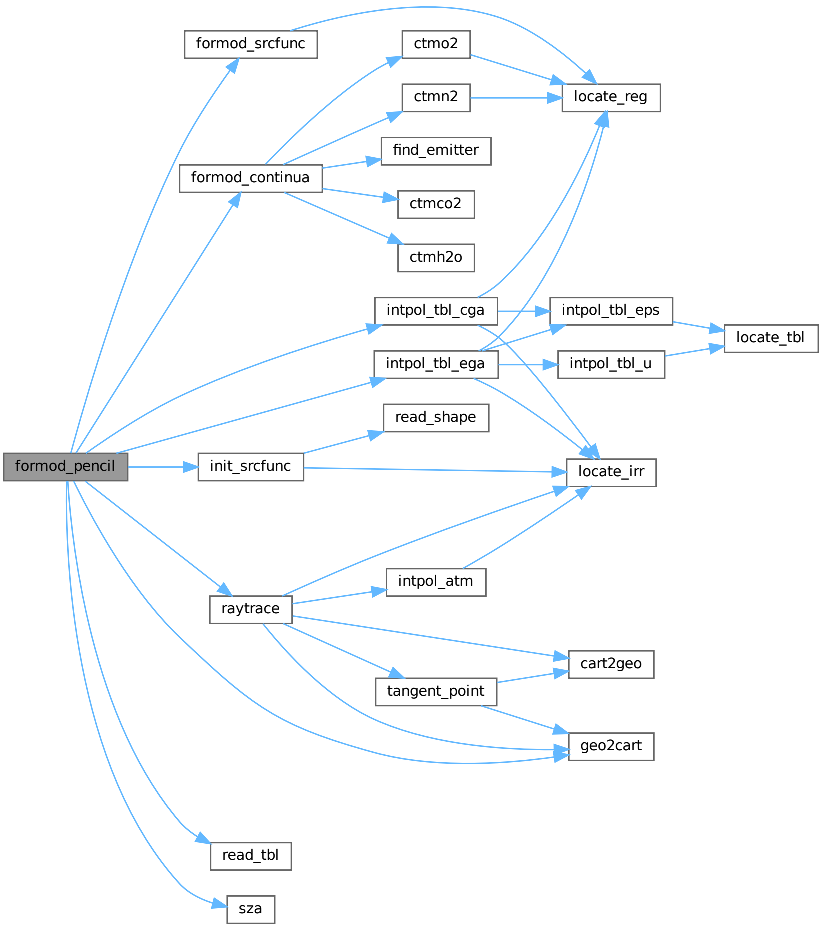
Definition at line 3196 of file jurassic.c.

```
                 {

  static tbl_t *tbl;

  static int init = 0;

  los_t *los;

  double beta_ctm[ND], rad[ND], tau[ND], tau_refl[ND],
    tau_path[ND][NG], tau_gas[ND], x0[3], x1[3];

  /* Initialize look-up tables... */
  if (!init) {
    init = 1;
    ALLOC(tbl, tbl_t, 1);
    read_tbl(ctl, tbl);
    init_srcfunc(ctl, tbl);
  }

  /* Allocate... */
  ALLOC(los, los_t, 1);

  /* Initialize... */
  for (int id = 0; id < ctl->nd; id++) {
    rad[id] = 0;
    tau[id] = 1;
    for (int ig = 0; ig < ctl->ng; ig++)
      tau_path[id][ig] = 1;
  }

  /* Raytracing... */
  raytrace(ctl, atm, obs, los, ir);

  /* Loop over LOS points... */
  for (int ip = 0; ip < los->np; ip++) {

    /* Get trace gas transmittance... */
    if (ctl->formod == 0)
      intpol_tbl_cga(ctl, tbl, los, ip, tau_path, tau_gas);
```

```
    else
      intpol_tbl_ega(ctl, tbl, los, ip, tau_path, tau_gas);

    /* Get continuum absorption... */
    formod_continua(ctl, los, ip, beta_ctm);

    /* Compute Planck function... */
    formod_srcfunc(ctl, tbl, los->t[ip], los->src[ip]);

    /* Loop over channels... */
    for (int id = 0; id < ctl->nd; id++)
      if (tau_gas[id] > 0) {

        /* Get segment emissivity... */
        los->eps[ip][id] = 1 - tau_gas[id] * exp(-beta_ctm[id] * los->ds[ip]);

        /* Compute radiance... */
        rad[id] += los->src[ip][id] * los->eps[ip][id] * tau[id];

        /* Compute path transmittance... */
        tau[id] *= (1 - los->eps[ip][id]);
      }
  }

  /* Check whether LOS hit the ground... */
  if (ctl->sftype >= 1 && los->sft > 0) {

    /* Add surface emissions... */
    double src_sf[ND];
    formod_srcfunc(ctl, tbl, los->sft, src_sf);
    for (int id = 0; id < ctl->nd; id++)
      rad[id] += los->sfeps[id] * src_sf[id] * tau[id];

    /* Check reflectivity... */
    int refl = 0;
    if (ctl->sftype >= 2)
      for (int id = 0; id < ctl->nd; id++)
        if (los->sfeps[id] < 1) {
          refl = 1;
          break;
        }

    /* Calculate reflection... */
    if (refl) {

      /* Initialize... */
      for (int id = 0; id < ctl->nd; id++)
        tau_refl[id] = 1;

      /* Add down-welling radiance... */
      for (int ip = los->np - 1; ip >= 0; ip--)
        for (int id = 0; id < ctl->nd; id++) {
          rad[id] += los->src[ip][id] * los->eps[ip][id] * tau_refl[id]
            * tau[id] * (1 - los->sfeps[id]);
          tau_refl[id] *= (1 - los->eps[ip][id]);
        }

      /* Add solar term... */
      if (ctl->sftype >= 3) {

        /* Get solar zenith angle... */
        double sza2;
        if (ctl->sfsza < 0)
          sza2 =
            sza(obs->time[ir], los->lon[los->np - 1], los->lat[los->np - 1]);
        else
          sza2 = ctl->sfsza;

        /* Check solar zenith angle... */
        if (sza2 < 89.999) {

          /* Get angle of incidence... */
          geo2cart(los->z[los->np - 1], los->lon[los->np - 1],
                   los->lat[los->np - 1], x0);
          geo2cart(los->z[0], los->lon[0], los->lat[0], x1);
          for (int i = 0; i < 3; i++)
            x1[i] -= x0[i];
          const double cosa = DOTP(x0, x1) / NORM(x0) / NORM(x1);

          /* Get ratio of SZA and incident radiation... */
          const double rcos = cosa / cos(DEG2RAD(sza2));

          /* Add solar radiation... */
          for (int id = 0; id < ctl->nd; id++)
            rad[id] += 6.764e-5 / (2. * M_PI) * PLANCK(TSUN, ctl->nu[id])
              * tau_refl[id] * (1 - los->sfeps[id]) * tau[id] * rcos;
        }
```

```
        }
      }
    }

    /* Copy results... */
    for (int id = 0; id < ctl->nd; id++) {
      obs->rad[id][ir] = rad[id];
      obs->tau[id][ir] = tau[id];
    }

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

Here is the call graph for this function:



formod_rfm()

```
void formod_rfm (
            const ctl_t * ctl,
            const atm_t * atm,
            obs_t * obs )
```

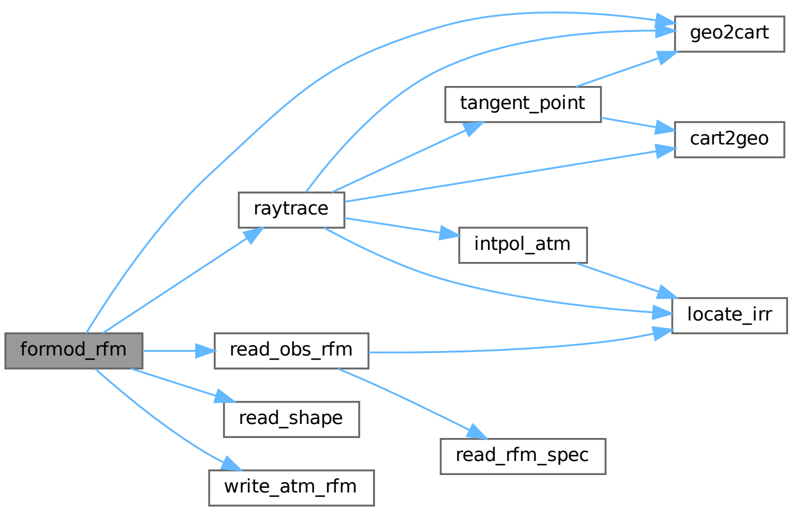
Apply RFM for radiative transfer calculations.

Definition at line 3342 of file jurassic.c.

```
		{

  los_t *los;

  FILE *out;

  char cmd[2 * LEN], filename[2 * LEN],
    rfmflg[LEN] = { "RAD TRA MIX LIN SFC" };

  double f[NSHAPE], nu[NSHAPE], nu0, nu1, obsz = -999, tsurf,
    xd[3], xo[3], xv[3], z[NR], zmin, zmax;

  int n, nadir = 0;

  /* Allocate... */
  ALLOC(los, los_t, 1);

  /* Check observer positions... */
  for (int ir = 1; ir < obs->nr; ir++)
    if (obs->obsz[ir] != obs->obsz[0]
	|| obs->obslon[ir] != obs->obslon[0]
	|| obs->obslat[ir] != obs->obslat[0])
      ERRMSG("RFM interface requires identical observer positions!");

  /* Check extinction data... */
  for (int iw = 0; iw < ctl->nw; iw++)
    for (int ip = 0; ip < atm->np; ip++)
      if (atm->k[iw][ip] != 0)
	ERRMSG("RFM interface cannot handle extinction data!");

  /* Get altitude range of atmospheric data... */
  gsl_stats_minmax(&zmin, &zmax, atm->z, 1, (size_t) atm->np);

  /* Observer within atmosphere? */
  if (obs->obsz[0] >= zmin && obs->obsz[0] <= zmax) {
    obsz = obs->obsz[0];
    strcat(rfmflg, " OBS");
  }

  /* Determine tangent altitude or air mass factor... */
  for (int ir = 0; ir < obs->nr; ir++) {

    /* Raytracing... */
    raytrace(ctl, atm, obs, los, ir);

    /* Nadir? */
    if (obs->tpz[ir] <= zmin) {
      geo2cart(obs->obsz[ir], obs->obslon[ir], obs->obslat[ir], xo);
      geo2cart(obs->vpz[ir], obs->vplon[ir], obs->vplat[ir], xv);
      for (int i = 0; i < 3; i++)
	xd[i] = xo[i] - xv[i];
      z[ir] = NORM(xo) * NORM(xd) / DOTP(xo, xd);
      nadir++;
    } else
      z[ir] = obs->tpz[ir];
  }
  if (nadir > 0 && nadir < obs->nr)
    ERRMSG("Limb and nadir not simultaneously possible!");

  /* Nadir? */
  if (nadir)
    strcat(rfmflg, " NAD");

  /* Get surface temperature... */
  tsurf = atm->t[gsl_stats_min_index(atm->z, 1, (size_t) atm->np)];

  /* Refraction? */
  if (!nadir && !ctl->refrac)
    strcat(rfmflg, " GEO");

  /* Continua? */
  if (ctl->ctm_co2 || ctl->ctm_h2o || ctl->ctm_n2 || ctl->ctm_o2)
    strcat(rfmflg, " CTM");

  /* Write atmospheric data file... */
  write_atm_rfm("rfm.atm", ctl, atm);

  /* Loop over channels... */
  for (int id = 0; id < ctl->nd; id++) {

    /* Read filter function... */
    sprintf(filename, "%s_%.4f.filt", ctl->tblbase, ctl->nu[id]);
    read_shape(filename, nu, f, &n);

    /* Set spectral range... */
    nu0 = nu[0];
    nu1 = nu[n - 1];
```

```

    /* Create RFM driver file... */
    if (!(out = fopen("rfm.drv", "w")))
      ERRMSG("Cannot create file!");
    fprintf(out, "*HDR\nRFM call by JURASSIC.\n");
    fprintf(out, "*FLG\n%s\n", rfmflg);
    fprintf(out, "*SPC\n%.4f %.4f 0.0005\n", nu0, nu1);
    fprintf(out, "*GAS\n");
    for (int ig = 0; ig < ctl->ng; ig++)
      fprintf(out, "%s\n", ctl->emitter[ig]);
    fprintf(out, "*ATM\nrfm.atm\n");
    fprintf(out, "*TAN\n");
    for (int ir = 0; ir < obs->nr; ir++)
      fprintf(out, "%g\n", z[ir]);
    fprintf(out, "*SFC\n%g 1.0\n", tsurf);
    if (obsz >= 0)
      fprintf(out, "*OBS\n%g\n", obsz);
    fprintf(out, "*HIT\n%s\n", ctl->rfmhit);
    fprintf(out, "*XSC\n");
    for (int ig = 0; ig < ctl->ng; ig++)
      if (ctl->rfmxsc[ig][0] != '-')
        fprintf(out, "%s\n", ctl->rfmxsc[ig]);
    fprintf(out, "*END\n");
    fclose(out);

    /* Remove temporary files... */
    if (system("rm -f rfm.runlog rad_*.asc tra_*.asc"))
      ERRMSG("Cannot remove temporary files!");

    /* Call RFM... */
    sprintf(cmd, "echo | %s", ctl->rfmbin);
    if (system(cmd))
      ERRMSG("Error while calling RFM!");

    /* Read data... */
    for (int ir = 0; ir < obs->nr; ir++) {
      obs->rad[id][ir] = read_obs_rfm("rad", z[ir], nu, f, n) * 1e-5;
      obs->tau[id][ir] = read_obs_rfm("tra", z[ir], nu, f, n);
    }
  }

  /* Remove temporary files... */
  if (system("rm -f rfm.drv rfm.atm rfm.runlog rad_*.asc tra_*.asc"))
    ERRMSG("Error while removing temporary files!");

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

Here is the call graph for this function:



formod_srcfunc()

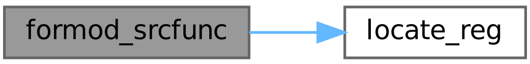
```
void formod_srcfunc (
            const ctl_t * ctl,
            const tbl_t * tbl,
            const double t,
            double * src )
```

Compute Planck source function.

Definition at line 3483 of file jurassic.c.

```
                  {

  /* Determine index in temperature array... */
  const int it = locate_reg(tbl->st, TBLNS, t);

  /* Interpolate Planck function value... */
  for (int id = 0; id < ctl->nd; id++)
    src[id] = LIN(tbl->st[it], tbl->sr[it][id],
                  tbl->st[it + 1], tbl->sr[it + 1][id], t);
}
```

Here is the call graph for this function:



geo2cart()

```
void geo2cart (
            const double z,
            const double lon,
            const double lat,
            double * x )
```

Convert geolocation to Cartesian coordinates.

Definition at line 3500 of file jurassic.c.

```
                 {

  const double radius = z + RE;

  const double latrad = lat / 180. * M_PI;
  const double lonrad = lon / 180. * M_PI;

  const double coslat = cos(latrad);

  x[0] = radius * coslat * cos(lonrad);
  x[1] = radius * coslat * sin(lonrad);
  x[2] = radius * sin(latrad);
}
```

hydrostatic()

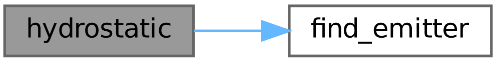
```
void hydrostatic (
            const ctl_t * ctl,
            atm_t * atm )
```

Set hydrostatic equilibrium.

Definition at line 3520 of file jurassic.c.

```
                   {

  const double mmair = 28.96456e-3, mmh2o = 18.0153e-3;

  const int ipts = 20;

  static int ig_h2o = -999;

  double dzmin = 1e99, e = 0;

  int ipref = 0;

  /* Check reference height... */
  if (ctl->hydz < 0)
    return;

  /* Determine emitter index of H2O... */
  if (ig_h2o == -999)
    ig_h2o = find_emitter(ctl, "H2O");

  /* Find air parcel next to reference height... */
  for (int ip = 0; ip < atm->np; ip++)
    if (fabs(atm->z[ip] - ctl->hydz) < dzmin) {
      dzmin = fabs(atm->z[ip] - ctl->hydz);
      ipref = ip;
    }

  /* Upper part of profile... */
  for (int ip = ipref + 1; ip < atm->np; ip++) {
    double mean = 0;
    for (int i = 0; i < ipts; i++) {
      if (ig_h2o >= 0)
        e = LIN(0.0, atm->q[ig_h2o][ip - 1],
                ipts - 1.0, atm->q[ig_h2o][ip], (double) i);
      mean += (e * mmh2o + (1 - e) * mmair)
        * G0 / RI
        / LIN(0.0, atm->t[ip - 1], ipts - 1.0, atm->t[ip], (double) i) / ipts;
    }

    /* Compute p(z,T)... */
    atm->p[ip] =
      exp(log(atm->p[ip - 1]) - mean * 1000 * (atm->z[ip] - atm->z[ip - 1]));
  }

  /* Lower part of profile... */
  for (int ip = ipref - 1; ip >= 0; ip--) {
    double mean = 0;
    for (int i = 0; i < ipts; i++) {
      if (ig_h2o >= 0)
        e = LIN(0.0, atm->q[ig_h2o][ip + 1],
                ipts - 1.0, atm->q[ig_h2o][ip], (double) i);
      mean += (e * mmh2o + (1 - e) * mmair)
        * G0 / RI
        / LIN(0.0, atm->t[ip + 1], ipts - 1.0, atm->t[ip], (double) i) / ipts;
    }

    /* Compute p(z,T)... */
    atm->p[ip] =
      exp(log(atm->p[ip + 1]) - mean * 1000 * (atm->z[ip] - atm->z[ip + 1]));
  }
}
```

Here is the call graph for this function:



idx2name()

```
void idx2name (
            const ctl_t * ctl,
            const int idx,
            char * quantity )
```

Determine name of state vector quantity for given index.

Definition at line 3586 of file jurassic.c.

```
                    {

  if (idx == IDXP)
    sprintf(quantity, "PRESSURE");

  if (idx == IDXT)
    sprintf(quantity, "TEMPERATURE");

  for (int ig = 0; ig < ctl->ng; ig++)
    if (idx == IDXQ(ig))
      sprintf(quantity, "%s", ctl->emitter[ig]);

  for (int iw = 0; iw < ctl->nw; iw++)
    if (idx == IDXK(iw))
      sprintf(quantity, "EXTINCT_WINDOW_%d", iw);

  if (idx == IDXCLZ)
    sprintf(quantity, "CLOUD_HEIGHT");

  if (idx == IDXCLDZ)
    sprintf(quantity, "CLOUD_DEPTH");

  for (int icl = 0; icl < ctl->ncl; icl++)
    if (idx == IDXCLK(icl))
      sprintf(quantity, "CLOUD_EXTINCT_%.4f", ctl->clnu[icl]);

  if (idx == IDXSFZ)
    sprintf(quantity, "SURFACE_HEIGHT");

  if (idx == IDXSFP)
    sprintf(quantity, "SURFACE_PRESSURE");

  if (idx == IDXSFT)
    sprintf(quantity, "SURFACE_TEMPERATURE");

  for (int isf = 0; isf < ctl->nsf; isf++)
    if (idx == IDXSFEPS(isf))
      sprintf(quantity, "SURFACE_EMISSIVITY_%.4f", ctl->sfnu[isf]);
}
```

init_srcfunc()

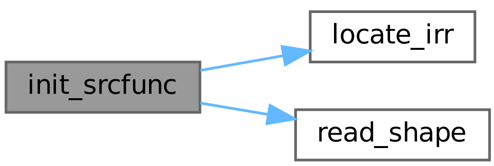
```
void init_srcfunc (
            const ctl_t * ctl,
            tbl_t * tbl )
```

Initialize source function table.

Definition at line 3631 of file jurassic.c.

```
{

  char filename[2 * LEN];

  double f[NSHAPE], nu[NSHAPE];

  int n;

  /* Write info... */
  LOG(1, "Initialize source function table...");
  LOG(2, "Number of data points: %d", TBLNS);

  /* Loop over channels... */
  for (int id = 0; id < ctl->nd; id++) {

    /* Read filter function... */
    sprintf(filename, "%s_%.4f.filt", ctl->tblbase, ctl->nu[id]);
    read_shape(filename, nu, f, &n);

    /* Get minimum grid spacing... */
    double dnu = 1.0;
    for (int i = 1; i < n; i++)
      dnu = MIN(dnu, nu[i] - nu[i - 1]);

    /* Compute source function table... */
#pragma omp parallel for default(none) shared(ctl,tbl,id,nu,f,n,dnu)
    for (int it = 0; it < TBLNS; it++) {

      /* Set temperature... */
      tbl->st[it] = LIN(0.0, TMIN, TBLNS - 1.0, TMAX, (double) it);

      /* Integrate Planck function... */
      double fsum = tbl->sr[it][id] = 0;
      for (double fnu = nu[0]; fnu <= nu[n - 1]; fnu += dnu) {
        const int i = locate_irr(nu, n, fnu);
        const double ff = LIN(nu[i], f[i], nu[i + 1], f[i + 1], fnu);
        fsum += ff;
        tbl->sr[it][id] += ff * PLANCK(tbl->st[it], fnu);
      }
      tbl->sr[it][id] /= fsum;
    }

    /* Write info... */
    LOG(2,
        "channel= %.4f cm^-1 | T= %g ... %g K | B= %g ... %g W/(m^2 sr cm^-1)",
        ctl->nu[id], tbl->st[0], tbl->st[TBLNS - 1], tbl->sr[0][id],
        tbl->sr[TBLNS - 1][id]);
  }
}
```

Here is the call graph for this function:



intpol_atm()

```
void intpol_atm (
            const ctl_t * ctl,
```

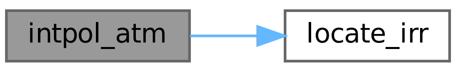
```
            const atm_t * atm,
            const double z,
            double * p,
            double * t,
            double * q,
            double * k )
```

Interpolate atmospheric data.

Definition at line 3685 of file jurassic.c.

```
                {

  /* Get array index... */
  const int ip = locate_irr(atm->z, atm->np, z);

  /* Interpolate... */
  *p = LOGY(atm->z[ip], atm->p[ip], atm->z[ip + 1], atm->p[ip + 1], z);
  *t = LIN(atm->z[ip], atm->t[ip], atm->z[ip + 1], atm->t[ip + 1], z);
  for (int ig = 0; ig < ctl->ng; ig++)
    q[ig] =
      LIN(atm->z[ip], atm->q[ig][ip], atm->z[ip + 1], atm->q[ig][ip + 1], z);
  for (int iw = 0; iw < ctl->nw; iw++)
    k[iw] =
      LIN(atm->z[ip], atm->k[iw][ip], atm->z[ip + 1], atm->k[iw][ip + 1], z);
}
```

Here is the call graph for this function:



intpol_tbl_cga()

```
void intpol_tbl_cga (
            const ctl_t * ctl,
            const tbl_t * tbl,
            const los_t * los,
            const int ip,
            double tau_path[ND][NG],
            double tau_seg[ND] )
```

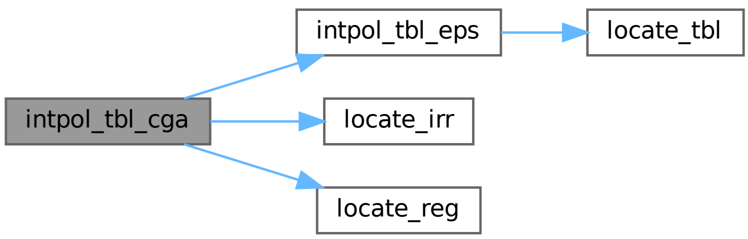
Get transmittance from look-up tables (CGA method).

Definition at line 3710 of file jurassic.c.

```
                          {

  double eps;

  /* Loop over channels... */
  for (int id = 0; id < ctl->nd; id++) {

    /* Initialize... */
    tau_seg[id] = 1;

    /* Loop over emitters.... */
    for (int ig = 0; ig < ctl->ng; ig++) {

      /* Check size of table (pressure)... */
```

```c
      if (tbl->np[id][ig] < 30)
        eps = 0;

      /* Check transmittance... */
      else if (tau_path[id][ig] < 1e-9)
        eps = 1;

      /* Interpolate... */
      else {

        /* Determine pressure and temperature indices... */
        const int ipr =
          locate_irr(tbl->p[id][ig], tbl->np[id][ig], los->cgp[ip][ig]);
        const int it0 = locate_reg(tbl->t[id][ig][ipr], tbl->nt[id][ig][ipr],
                                   los->cgt[ip][ig]);
        const int it1 =
          locate_reg(tbl->t[id][ig][ipr + 1], tbl->nt[id][ig][ipr + 1],
                     los->cgt[ip][ig]);

        /* Check size of table (temperature and column density)... */
        if (tbl->nt[id][ig][ipr] < 2 || tbl->nt[id][ig][ipr + 1] < 2
            || tbl->nu[id][ig][ipr][it0] < 2
            || tbl->nu[id][ig][ipr][it0 + 1] < 2
            || tbl->nu[id][ig][ipr + 1][it1] < 2
            || tbl->nu[id][ig][ipr + 1][it1 + 1] < 2)
          eps = 0;

        else {

          /* Get emissivities of extended path... */
          double eps00
            = intpol_tbl_eps(tbl, ig, id, ipr, it0, los->cgu[ip][ig]);
          double eps01 =
            intpol_tbl_eps(tbl, ig, id, ipr, it0 + 1, los->cgu[ip][ig]);
          double eps10 =
            intpol_tbl_eps(tbl, ig, id, ipr + 1, it1, los->cgu[ip][ig]);
          double eps11 =
            intpol_tbl_eps(tbl, ig, id, ipr + 1, it1 + 1, los->cgu[ip][ig]);

          /* Interpolate with respect to temperature... */
          eps00 = LIN(tbl->t[id][ig][ipr][it0], eps00,
                      tbl->t[id][ig][ipr][it0 + 1], eps01, los->cgt[ip][ig]);
          eps11 = LIN(tbl->t[id][ig][ipr + 1][it1], eps10,
                      tbl->t[id][ig][ipr + 1][it1 + 1],
                      eps11, los->cgt[ip][ig]);

          /* Interpolate with respect to pressure... */
          eps00 = LOGX(tbl->p[id][ig][ipr], eps00,
                       tbl->p[id][ig][ipr + 1], eps11, los->cgp[ip][ig]);

          /* Check emssivity range... */
          eps00 = MAX(MIN(eps00, 1), 0);

          /* Determine segment emissivity... */
          eps = 1 - (1 - eps00) / tau_path[id][ig];
        }
      }

      /* Get transmittance of extended path... */
      tau_path[id][ig] *= (1 - eps);

      /* Get segment transmittance... */
      tau_seg[id] *= (1 - eps);
    }
  }
}
```

Here is the call graph for this function:



intpol_tbl_ega()

```
void intpol_tbl_ega (
            const ctl_t * ctl,
            const tbl_t * tbl,
            const los_t * los,
            const int ip,
            double tau_path[ND][NG],
            double tau_seg[ND] )
```

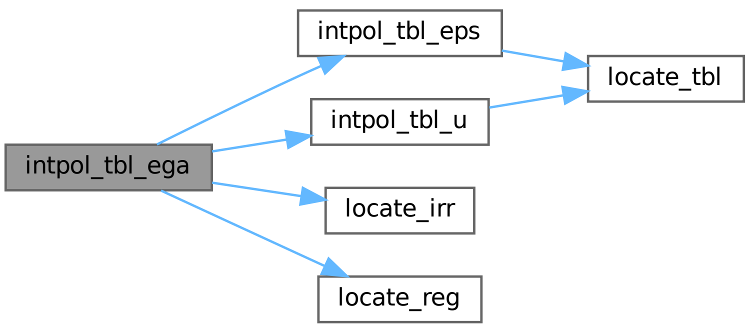
Get transmittance from look-up tables (EGA method).

Definition at line 3799 of file jurassic.c.

```
                   {

  double eps, u;

  /* Loop over channels... */
  for (int id = 0; id < ctl->nd; id++) {

    /* Initialize... */
    tau_seg[id] = 1;

    /* Loop over emitters.... */
    for (int ig = 0; ig < ctl->ng; ig++) {

      /* Check size of table (pressure)... */
      if (tbl->np[id][ig] < 30)
        eps = 0;

      /* Check transmittance... */
      else if (tau_path[id][ig] < 1e-9)
        eps = 1;

      /* Interpolate... */
      else {

        /* Determine pressure and temperature indices... */
        const int ipr =
          locate_irr(tbl->p[id][ig], tbl->np[id][ig], los->p[ip]);
        const int it0 =
          locate_reg(tbl->t[id][ig][ipr], tbl->nt[id][ig][ipr], los->t[ip]);
        const int it1 =
          locate_reg(tbl->t[id][ig][ipr + 1], tbl->nt[id][ig][ipr + 1],
                     los->t[ip]);

        /* Check size of table (temperature and column density)... */
        if (tbl->nt[id][ig][ipr] < 2 || tbl->nt[id][ig][ipr + 1] < 2
            || tbl->nu[id][ig][ipr][it0] < 2
            || tbl->nu[id][ig][ipr][it0 + 1] < 2
            || tbl->nu[id][ig][ipr + 1][it1] < 2
            || tbl->nu[id][ig][ipr + 1][it1 + 1] < 2)
```

```
          eps = 0;

        else {

          /* Get emissivities of extended path... */
          u = intpol_tbl_u(tbl, ig, id, ipr, it0, 1 - tau_path[id][ig]);
          double eps00
            = intpol_tbl_eps(tbl, ig, id, ipr, it0, u + los->u[ip][ig]);

          u = intpol_tbl_u(tbl, ig, id, ipr, it0 + 1, 1 - tau_path[id][ig]);
          double eps01 =
            intpol_tbl_eps(tbl, ig, id, ipr, it0 + 1, u + los->u[ip][ig]);

          u = intpol_tbl_u(tbl, ig, id, ipr + 1, it1, 1 - tau_path[id][ig]);
          double eps10 =
            intpol_tbl_eps(tbl, ig, id, ipr + 1, it1, u + los->u[ip][ig]);

          u =
            intpol_tbl_u(tbl, ig, id, ipr + 1, it1 + 1, 1 - tau_path[id][ig]);
          double eps11 =
            intpol_tbl_eps(tbl, ig, id, ipr + 1, it1 + 1, u + los->u[ip][ig]);

          /* Interpolate with respect to temperature... */
          eps00 = LIN(tbl->t[id][ig][ipr][it0], eps00,
                      tbl->t[id][ig][ipr][it0 + 1], eps01, los->t[ip]);
          eps11 = LIN(tbl->t[id][ig][ipr + 1][it1], eps10,
                      tbl->t[id][ig][ipr + 1][it1 + 1], eps11, los->t[ip]);

          /* Interpolate with respect to pressure... */
          eps00 = LIN(tbl->p[id][ig][ipr], eps00,
                      tbl->p[id][ig][ipr + 1], eps11, los->p[ip]);

          /* Check emssivity range... */
          eps00 = MAX(MIN(eps00, 1), 0);

          /* Determine segment emissivity... */
          eps = 1 - (1 - eps00) / tau_path[id][ig];
        }
      }

      /* Get transmittance of extended path... */
      tau_path[id][ig] *= (1 - eps);

      /* Get segment transmittance... */
      tau_seg[id] *= (1 - eps);
    }
  }
}
```

Here is the call graph for this function:



intpol_tbl_eps()

```
double intpol_tbl_eps (
            const tbl_t * tbl,
```

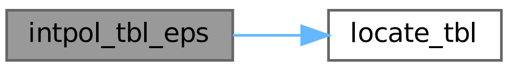
```
            const int ig,
            const int id,
            const int ip,
            const int it,
            const double u )
```

Interpolate emissivity from look-up tables.

Definition at line 3895 of file jurassic.c.

```
                    {

  /* Lower boundary... */
  if (u < tbl->u[id][ig][ip][it][0])
    return LIN(0, 0, tbl->u[id][ig][ip][it][0], tbl->eps[id][ig][ip][it][0],
               u);

  /* Upper boundary... */
  else if (u > tbl->u[id][ig][ip][it][tbl->nu[id][ig][ip][it] - 1]) {
    const double a =
      log(1 - tbl->eps[id][ig][ip][it][tbl->nu[id][ig][ip][it] - 1])
      / tbl->u[id][ig][ip][it][tbl->nu[id][ig][ip][it] - 1];
    return 1 - exp(a * u);
  }

  /* Interpolation... */
  else {

    /* Get index... */
    const int idx =
      locate_tbl(tbl->u[id][ig][ip][it], tbl->nu[id][ig][ip][it], u);

    /* Interpolate... */
    return
      LIN(tbl->u[id][ig][ip][it][idx], tbl->eps[id][ig][ip][it][idx],
          tbl->u[id][ig][ip][it][idx + 1], tbl->eps[id][ig][ip][it][idx + 1],
          u);
  }
}
```

Here is the call graph for this function:



intpol_tbl_u()

```
double intpol_tbl_u (
            const tbl_t * tbl,
            const int ig,
            const int id,
            const int ip,
            const int it,
            const double eps )
```

Interpolate column density from look-up tables.

Definition at line 3933 of file jurassic.c.

```
                       {

```

```
  /* Lower boundary... */
  if (eps < tbl->eps[id][ig][ip][it][0])
    return LIN(0, 0, tbl->eps[id][ig][ip][it][0], tbl->u[id][ig][ip][it][0],
               eps);

  /* Upper boundary... */
  else if (eps > tbl->eps[id][ig][ip][it][tbl->nu[id][ig][ip][it] - 1]) {
    const double a =
      log(1 - tbl->eps[id][ig][ip][it][tbl->nu[id][ig][ip][it] - 1])
      / tbl->u[id][ig][ip][it][tbl->nu[id][ig][ip][it] - 1];
    return log(1 - eps) / a;
  }

  /* Interpolation... */
  else {

    /* Get index... */
    const int idx =
      locate_tbl(tbl->eps[id][ig][ip][it], tbl->nu[id][ig][ip][it], eps);

    /* Interpolate... */
    return
      LIN(tbl->eps[id][ig][ip][it][idx], tbl->u[id][ig][ip][it][idx],
          tbl->eps[id][ig][ip][it][idx + 1], tbl->u[id][ig][ip][it][idx + 1],
          eps);
  }
}
```

Here is the call graph for this function:

jsec2time()

```
void jsec2time (
            const double jsec,
            int * year,
            int * mon,
            int * day,
            int * hour,
            int * min,
            int * sec,
            double * remain )
```

Convert seconds to date.

Definition at line 3971 of file jurassic.c.

```
                        {

  struct tm t0, *t1;

  t0.tm_year = 100;
  t0.tm_mon = 0;
  t0.tm_mday = 1;
  t0.tm_hour = 0;
  t0.tm_min = 0;
  t0.tm_sec = 0;

  time_t jsec0 = (time_t) jsec + timegm(&t0);
  t1 = gmtime(&jsec0);

```

```
  *year = t1->tm_year + 1900;
  *mon = t1->tm_mon + 1;
  *day = t1->tm_mday;
  *hour = t1->tm_hour;
  *min = t1->tm_min;
  *sec = t1->tm_sec;
  *remain = jsec - floor(jsec);
}
```

kernel()

```
void kernel (
              ctl_t * ctl,
              atm_t * atm,
              obs_t * obs,
              gsl_matrix * k )
```

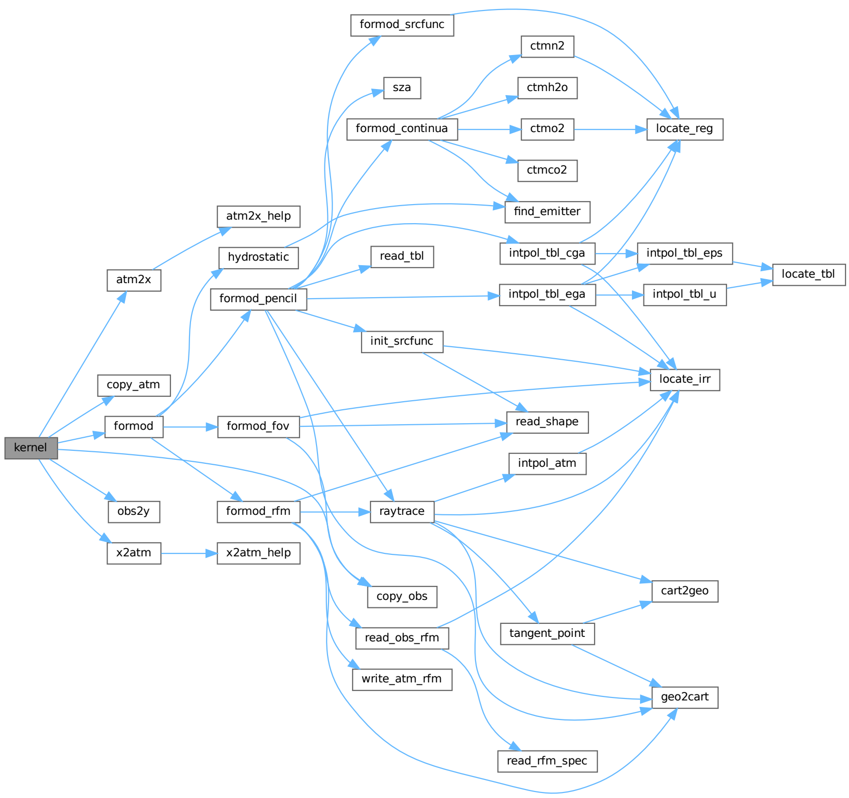
Compute Jacobians.

Definition at line 4004 of file jurassic.c.

```
                   {

  atm_t *atm1;
  obs_t *obs1;

  int *iqa;

  /* Get sizes... */
  const size_t m = k->size1;
  const size_t n = k->size2;

  /* Allocate... */
  gsl_vector *x0 = gsl_vector_alloc(n);
  gsl_vector *yy0 = gsl_vector_alloc(m);
  ALLOC(iqa, int,
        N);

  /* Compute radiance for undisturbed atmospheric data... */
  formod(ctl, atm, obs);

  /* Compose vectors... */
  atm2x(ctl, atm, x0, iqa, NULL);
  obs2y(ctl, obs, yy0, NULL, NULL);

  /* Initialize kernel matrix... */
  gsl_matrix_set_zero(k);

  /* Loop over state vector elements... */
#pragma omp parallel for default(none) shared(ctl,atm,obs,k,x0,yy0,n,m,iqa) private(atm1, obs1)
  for (size_t j = 0; j < n; j++) {

    /* Allocate... */
    gsl_vector *x1 = gsl_vector_alloc(n);
    gsl_vector *yy1 = gsl_vector_alloc(m);
    ALLOC(atm1, atm_t, 1);
    ALLOC(obs1, obs_t, 1);

    /* Set perturbation size... */
    double h;
    if (iqa[j] == IDXP)
      h = MAX(fabs(0.01 * gsl_vector_get(x0, j)), 1e-7);
    else if (iqa[j] == IDXT)
      h = 1.0;
    else if (iqa[j] >= IDXQ(0) && iqa[j] < IDXQ(ctl->ng))
      h = MAX(fabs(0.01 * gsl_vector_get(x0, j)), 1e-15);
    else if (iqa[j] >= IDXK(0) && iqa[j] < IDXK(ctl->nw))
      h = 1e-4;
    else if (iqa[j] == IDXCLZ || iqa[j] == IDXCLDZ)
      h = 1.0;
    else if (iqa[j] >= IDXCLK(0) && iqa[j] < IDXCLK(ctl->ncl))
      h = 1e-4;
    else if (iqa[j] == IDXSFZ)
      h = 0.1;
    else if (iqa[j] == IDXSFP)
      h = 10.0;
    else if (iqa[j] == IDXSFT)
      h = 1.0;
    else if (iqa[j] >= IDXSFEPS(0) && iqa[j] < IDXSFEPS(ctl->nsf))
      h = 1e-2;
```

```
    else
      ERRMSG("Cannot set perturbation size!");

    /* Disturb state vector element... */
    gsl_vector_memcpy(x1, x0);
    gsl_vector_set(x1, j, gsl_vector_get(x1, j) + h);
    copy_atm(ctl, atm1, atm, 0);
    copy_obs(ctl, obs1, obs, 0);
    x2atm(ctl, x1, atm1);

    /* Compute radiance for disturbed atmospheric data... */
    formod(ctl, atm1, obs1);

    /* Compose measurement vector for disturbed radiance data... */
    obs2y(ctl, obs1, yy1, NULL, NULL);

    /* Compute derivatives... */
    for (size_t i = 0; i < m; i++)
      gsl_matrix_set(k, i, j,
                     (gsl_vector_get(yy1, i) - gsl_vector_get(yy0, i)) / h);

    /* Free... */
    gsl_vector_free(x1);
    gsl_vector_free(yy1);
    free(atm1);
    free(obs1);
  }

  /* Free... */
  gsl_vector_free(x0);
  gsl_vector_free(yy0);
  free(iqa);
}
```

Here is the call graph for this function:



locate_irr()

```
int locate_irr (
```

```
const double * xx,
const int n,
const double x )
```

Find array index for irregular grid.

Definition at line 4103 of file jurassic.c.

```
                    {

  int ilo = 0;
  int ihi = n - 1;
  int i = (ihi + ilo) » 1;

  if (xx[i] < xx[i + 1])
    while (ihi > ilo + 1) {
      i = (ihi + ilo) » 1;
      if (xx[i] > x)
        ihi = i;
      else
        ilo = i;
  } else
    while (ihi > ilo + 1) {
      i = (ihi + ilo) » 1;
      if (xx[i] <= x)
        ihi = i;
      else
        ilo = i;
    }

  return ilo;
}
```

locate_reg()

```
int locate_reg (
            const double * xx,
            const int n,
            const double x )
```

Find array index for regular grid.

Definition at line 4133 of file jurassic.c.

```
                    {

  /* Calculate index... */
  const int i = (int) ((x - xx[0]) / (xx[1] - xx[0]));

  /* Check range... */
  if (i < 0)
    return 0;
  else if (i > n - 2)
    return n - 2;
  else
    return i;
}
```

locate_tbl()

```
int locate_tbl (
            const float * xx,
            const int n,
            const double x )
```

Find array index in float array.

Definition at line 4152 of file jurassic.c.

```
                    {

```

```
  int ilo = 0;
  int ihi = n - 1;
  int i = (ihi + ilo) » 1;

  while (ihi > ilo + 1) {
    i = (ihi + ilo) » 1;
    if (xx[i] > x)
      ihi = i;
    else
      ilo = i;
  }

  return ilo;
}
```

obs2y()

```
size_t obs2y (
            const ctl_t * ctl,
            const obs_t * obs,
            gsl_vector * y,
            int * ida,
            int * ira )
```

Compose measurement vector.

Definition at line 4174 of file jurassic.c.

```
                {

  size_t m = 0;

  /* Determine measurement vector... */
  for (int ir = 0; ir < obs->nr; ir++)
    for (int id = 0; id < ctl->nd; id++)
      if (isfinite(obs->rad[id][ir])) {
        if (y != NULL)
          gsl_vector_set(y, m, obs->rad[id][ir]);
        if (ida != NULL)
          ida[m] = id;
        if (ira != NULL)
          ira[m] = ir;
        m++;
      }

  return m;
}
```

raytrace()

```
void raytrace (
            const ctl_t * ctl,
            const atm_t * atm,
            obs_t * obs,
            los_t * los,
            const int ir )
```

Do ray-tracing to determine LOS.

Definition at line 4201 of file jurassic.c.

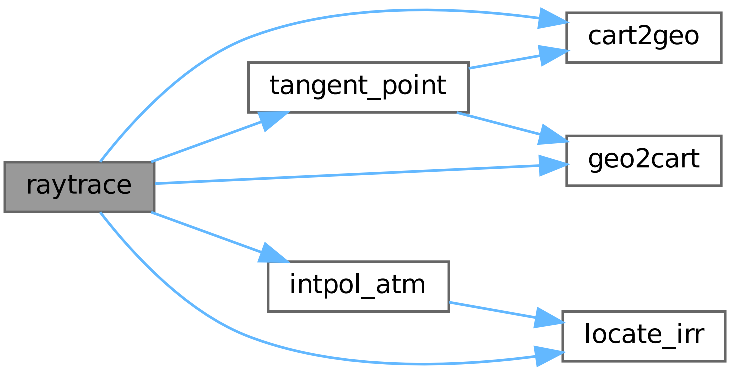
```
                    {

  const double h = 0.02, zrefrac = 60;

  double ex0[3], ex1[3], k[NW], lat, lon, n, ng[3], norm, p, q[NG], t,
    x[3], xh[3], xobs[3], xvp[3], z = 1e99, zmax, zmin;

  int stop = 0;

  /* Initialize... */
```

```
  los->np = 0;
  los->sft = -999;
  obs->tpz[ir] = obs->vpz[ir];
  obs->tplon[ir] = obs->vplon[ir];
  obs->tplat[ir] = obs->vplat[ir];

  /* Get altitude range of atmospheric data... */
  gsl_stats_minmax(&zmin, &zmax, atm->z, 1, (size_t) atm->np);
  if (ctl->nsf > 0) {
    zmin = MAX(atm->sfz, zmin);
    if (atm->sfp > 0) {
      const int ip = locate_irr(atm->p, atm->np, atm->sfp);
      const double zip =
        LIN(log(atm->p[ip]), atm->z[ip], log(atm->p[ip + 1]), atm->z[ip + 1],
            log(atm->sfp));
      zmin = MAX(zip, zmin);
    }
  }

  /* Check observer altitude... */
  if (obs->obsz[ir] < zmin)
    ERRMSG("Observer below surface!");

  /* Check view point altitude... */
  if (obs->vpz[ir] > zmax)
    return;

  /* Determine Cartesian coordinates for observer and view point... */
  geo2cart(obs->obsz[ir], obs->obslon[ir], obs->obslat[ir], xobs);
  geo2cart(obs->vpz[ir], obs->vplon[ir], obs->vplat[ir], xvp);

  /* Determine initial tangent vector... */
  for (int i = 0; i < 3; i++)
    ex0[i] = xvp[i] - xobs[i];
  norm = NORM(ex0);
  for (int i = 0; i < 3; i++)
    ex0[i] /= norm;

  /* Observer within atmosphere... */
  for (int i = 0; i < 3; i++)
    x[i] = xobs[i];

  /* Observer above atmosphere (search entry point)... */
  if (obs->obsz[ir] > zmax) {
    double dmax = norm, dmin = 0;
    while (fabs(dmin - dmax) > 0.001) {
      const double d = (dmax + dmin) / 2;
      for (int i = 0; i < 3; i++)
        x[i] = xobs[i] + d * ex0[i];
      cart2geo(x, &z, &lon, &lat);
      if (z <= zmax && z > zmax - 0.001)
        break;
      if (z < zmax - 0.0005)
        dmax = d;
      else
        dmin = d;
    }
  }

  /* Ray-tracing... */
  while (1) {

    /* Set step length... */
    double ds = ctl->rayds;
    if (ctl->raydz > 0) {
      norm = NORM(x);
      for (int i = 0; i < 3; i++)
        xh[i] = x[i] / norm;
      const double cosa = fabs(DOTP(ex0, xh));
      if (cosa != 0)
        ds = MIN(ctl->rayds, ctl->raydz / cosa);
    }

    /* Determine geolocation... */
    cart2geo(x, &z, &lon, &lat);

    /* Check if LOS hits the ground or has left atmosphere... */
    if (z < zmin || z > zmax) {
      stop = (z < zmin ? 2 : 1);
      const double frac =
        ((z <
          zmin ? zmin : zmax) - los->z[los->np - 1]) / (z - los->z[los->np -
                                                                    1]);
      geo2cart(los->z[los->np - 1], los->lon[los->np - 1],
               los->lat[los->np - 1], xh);
      for (int i = 0; i < 3; i++)
        x[i] = xh[i] + frac * (x[i] - xh[i]);
```

```
      cart2geo(x, &z, &lon, &lat);
      los->ds[los->np - 1] = ds * frac;
      ds = 0;
    }

    /* Interpolate atmospheric data... */
    intpol_atm(ctl, atm, z, &p, &t, q, k);

    /* Save data... */
    los->lon[los->np] = lon;
    los->lat[los->np] = lat;
    los->z[los->np] = z;
    los->p[los->np] = p;
    los->t[los->np] = t;
    for (int ig = 0; ig < ctl->ng; ig++)
      los->q[los->np][ig] = q[ig];
    for (int id = 0; id < ctl->nd; id++)
      los->k[los->np][id] = k[ctl->window[id]];
    los->ds[los->np] = ds;

    /* Add cloud extinction... */
    if (ctl->ncl > 0 && atm->cldz > 0) {
      const double aux = exp(-0.5 * POW2((z - atm->clz) / atm->cldz));
      for (int id = 0; id < ctl->nd; id++) {
        const int icl = locate_irr(ctl->clnu, ctl->ncl, ctl->nu[id]);
        los->k[los->np][id]
          += aux * LIN(ctl->clnu[icl], atm->clk[icl],
                       ctl->clnu[icl + 1], atm->clk[icl + 1], ctl->nu[id]);
      }
    }

    /* Increment and check number of LOS points... */
    if ((++los->np) > NLOS)
      ERRMSG("Too many LOS points!");

    /* Check stop flag... */
    if (stop) {

      /* Set surface temperature... */
      if (ctl->nsf > 0 && atm->sft > 0)
        t = atm->sft;
      los->sft = (stop == 2 ? t : -999);

      /* Set surface emissivity... */
      for (int id = 0; id < ctl->nd; id++) {
        los->sfeps[id] = 1.0;
        if (ctl->nsf > 0) {
          const int isf = locate_irr(ctl->sfnu, ctl->nsf, ctl->nu[id]);
          los->sfeps[id] = LIN(ctl->sfnu[isf], atm->sfeps[isf],
                               ctl->sfnu[isf + 1], atm->sfeps[isf + 1],
                               ctl->nu[id]);
        }
      }

      /* Leave raytracer... */
      break;
    }

    /* Determine refractivity... */
    if (ctl->refrac && z <= zrefrac)
      n = 1 + REFRAC(p, t);
    else
      n = 1;

    /* Construct new tangent vector (first term)... */
    for (int i = 0; i < 3; i++)
      ex1[i] = ex0[i] * n;

    /* Compute gradient of refractivity... */
    if (ctl->refrac && z <= zrefrac) {
      for (int i = 0; i < 3; i++)
        xh[i] = x[i] + 0.5 * ds * ex0[i];
      cart2geo(xh, &z, &lon, &lat);
      intpol_atm(ctl, atm, z, &p, &t, q, k);
      n = REFRAC(p, t);
      for (int i = 0; i < 3; i++) {
        xh[i] += h;
        cart2geo(xh, &z, &lon, &lat);
        intpol_atm(ctl, atm, z, &p, &t, q, k);
        ng[i] = (REFRAC(p, t) - n) / h;
        xh[i] -= h;
      }
    } else
      for (int i = 0; i < 3; i++)
        ng[i] = 0;

    /* Construct new tangent vector (second term)... */
```

```
    for (int i = 0; i < 3; i++)
      ex1[i] += ds * ng[i];

    /* Normalize new tangent vector... */
    norm = NORM(ex1);
    for (int i = 0; i < 3; i++)
      ex1[i] /= norm;

    /* Determine next point of LOS... */
    for (int i = 0; i < 3; i++)
      x[i] += 0.5 * ds * (ex0[i] + ex1[i]);

    /* Copy tangent vector... */
    for (int i = 0; i < 3; i++)
      ex0[i] = ex1[i];
  }

  /* Get tangent point (to be done before changing segment lengths!)... */
  tangent_point(los, &obs->tpz[ir], &obs->tplon[ir], &obs->tplat[ir]);

  /* Change segment lengths according to trapezoid rule... */
  for (int ip = los->np - 1; ip >= 1; ip--)
    los->ds[ip] = 0.5 * (los->ds[ip - 1] + los->ds[ip]);
  los->ds[0] *= 0.5;

  /* Compute column density... */
  for (int ip = 0; ip < los->np; ip++)
    for (int ig = 0; ig < ctl->ng; ig++)
      los->u[ip][ig] = 10 * los->q[ip][ig] * los->p[ip]
        / (KB * los->t[ip]) * los->ds[ip];

  /* Compute Curtis-Godson means... */
  for (int ig = 0; ig < ctl->ng; ig++) {
    los->cgu[0][ig] = los->u[0][ig];
    los->cgp[0][ig] = los->u[0][ig] * los->p[0];
    los->cgt[0][ig] = los->u[0][ig] * los->t[0];
  }
  for (int ip = 1; ip < los->np; ip++)
    for (int ig = 0; ig < ctl->ng; ig++) {
      los->cgu[ip][ig] = los->cgu[ip - 1][ig] + los->u[ip][ig];
      los->cgp[ip][ig] = los->cgp[ip - 1][ig] + los->u[ip][ig] * los->p[ip];
      los->cgt[ip][ig] = los->cgt[ip - 1][ig] + los->u[ip][ig] * los->t[ip];
    }
  for (int ip = 0; ip < los->np; ip++)
    for (int ig = 0; ig < ctl->ng; ig++) {
      los->cgp[ip][ig] /= los->cgu[ip][ig];
      los->cgt[ip][ig] /= los->cgu[ip][ig];
    }
}
```

Here is the call graph for this function:



read_atm()

```
void read_atm (
        const char * dirname,
        const char * filename,
        const ctl_t * ctl,
        atm_t * atm )
```

Read atmospheric data.

Definition at line 4442 of file jurassic.c.

```
                {

  FILE *in;

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

  /* Init... */
  atm->np = 0;

  /* Set filename... */
  if (dirname != NULL)
    sprintf(file, "%s/%s", dirname, filename);
  else
    sprintf(file, "%s", filename);

  /* Write info... */
  LOG(1, "Read atmospheric data: %s", file);

  /* Open file... */
  if (!(in = fopen(file, "r")))
    ERRMSG("Cannot open file!");

  /* Read line... */
  while (fgets(line, LEN, in)) {

    /* Read data... */
    TOK(line, tok, "%lg", atm->time[atm->np]);
    TOK(NULL, tok, "%lg", atm->z[atm->np]);
    TOK(NULL, tok, "%lg", atm->lon[atm->np]);
    TOK(NULL, tok, "%lg", atm->lat[atm->np]);
    TOK(NULL, tok, "%lg", atm->p[atm->np]);
    TOK(NULL, tok, "%lg", atm->t[atm->np]);
    for (int ig = 0; ig < ctl->ng; ig++)
      TOK(NULL, tok, "%lg", atm->q[ig][atm->np]);
    for (int iw = 0; iw < ctl->nw; iw++)
      TOK(NULL, tok, "%lg", atm->k[iw][atm->np]);
    if (ctl->ncl > 0 && atm->np == 0) {
      TOK(NULL, tok, "%lg", atm->clz);
      TOK(NULL, tok, "%lg", atm->cldz);
      for (int icl = 0; icl < ctl->ncl; icl++)
        TOK(NULL, tok, "%lg", atm->clk[icl]);
    }
    if (ctl->nsf > 0 && atm->np == 0) {
      TOK(NULL, tok, "%lg", atm->sfz);
      TOK(NULL, tok, "%lg", atm->sfp);
      TOK(NULL, tok, "%lg", atm->sft);
      for (int isf = 0; isf < ctl->nsf; isf++)
        TOK(NULL, tok, "%lg", atm->sfeps[isf]);
    }

    /* Increment data point counter... */
    if ((++atm->np) > NP)
      ERRMSG("Too many data points!");
  }

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

  /* Check number of points... */
  if (atm->np < 1)
    ERRMSG("Could not read any data!");

  /* Write info... */
  double mini, maxi;
  LOG(2, "Number of data points: %d", atm->np);
  gsl_stats_minmax(&mini, &maxi, atm->time, 1, (size_t) atm->np);
  LOG(2, "Time range: %.2f ... %.2f s", mini, maxi);
  gsl_stats_minmax(&mini, &maxi, atm->z, 1, (size_t) atm->np);
  LOG(2, "Altitude range: %g ... %g km", mini, maxi);
  gsl_stats_minmax(&mini, &maxi, atm->lon, 1, (size_t) atm->np);
```

```
  LOG(2, "Longitude range: %g ... %g deg", mini, maxi);
  gsl_stats_minmax(&mini, &maxi, atm->lat, 1, (size_t) atm->np);
  LOG(2, "Latitude range: %g ... %g deg", mini, maxi);
  gsl_stats_minmax(&mini, &maxi, atm->p, 1, (size_t) atm->np);
  LOG(2, "Pressure range: %g ... %g hPa", maxi, mini);
  gsl_stats_minmax(&mini, &maxi, atm->t, 1, (size_t) atm->np);
  LOG(2, "Temperature range: %g ... %g K", mini, maxi);
  for (int ig = 0; ig < ctl->ng; ig++) {
    gsl_stats_minmax(&mini, &maxi, atm->q[ig], 1, (size_t) atm->np);
    LOG(2, "Emitter %s range: %g ... %g ppv", ctl->emitter[ig], mini, maxi);
  }
  for (int iw = 0; iw < ctl->nw; iw++) {
    gsl_stats_minmax(&mini, &maxi, atm->k[iw], 1, (size_t) atm->np);
    LOG(2, "Extinction range (window %d): %g ... %g km^-1", iw, mini, maxi);
  }
  if (ctl->ncl > 0 && atm->np == 0) {
    LOG(2, "Cloud layer: z= %g km | dz= %g km | k= %g ... %g km^-1",
        atm->clz, atm->cldz, atm->clk[0], atm->clk[ctl->ncl - 1]);
  } else
    LOG(2, "Cloud layer: none");
  if (ctl->nsf > 0 && atm->np == 0) {
    LOG(2,
        "Surface layer: z_s= %g km | p_s= %g hPa | T_s = %g K | eps= %g ... %g",
        atm->sfz, atm->sfp, atm->sft, atm->sfeps[0],
        atm->sfeps[ctl->nsf - 1]);
  } else
    LOG(2, "Surface layer: none");
}
```

read_ctl()

```
void read_ctl (
              int argc,
              char * argv[],
              ctl_t * ctl )
```

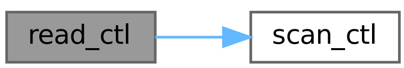
Read forward model control parameters.

Definition at line 4547 of file jurassic.c.

```
                {

  /* Write info... */
  LOG(1, "\nJuelich Rapid Spectral Simulation Code (JURASSIC)\n"
      "(executable: %s | version: %s | compiled: %s, %s)\n",
      argv[0], VERSION, __DATE__, __TIME__);

  /* Emitters... */
  ctl->ng = (int) scan_ctl(argc, argv, "NG", -1, "0", NULL);
  if (ctl->ng < 0 || ctl->ng > NG)
    ERRMSG("Set 0 <= NG <= MAX!");
  for (int ig = 0; ig < ctl->ng; ig++)
    scan_ctl(argc, argv, "EMITTER", ig, "", ctl->emitter[ig]);

  /* Radiance channels... */
  ctl->nd = (int) scan_ctl(argc, argv, "ND", -1, "0", NULL);
  if (ctl->nd < 0 || ctl->nd > ND)
    ERRMSG("Set 0 <= ND <= MAX!");
  for (int id = 0; id < ctl->nd; id++)
    ctl->nu[id] = scan_ctl(argc, argv, "NU", id, "", NULL);

  /* Spectral windows... */
  ctl->nw = (int) scan_ctl(argc, argv, "NW", -1, "1", NULL);
  if (ctl->nw < 0 || ctl->nw > NW)
    ERRMSG("Set 0 <= NW <= MAX!");
  for (int id = 0; id < ctl->nd; id++)
    ctl->window[id] = (int) scan_ctl(argc, argv, "WINDOW", id, "0", NULL);

  /* Cloud data... */
  ctl->ncl = (int) scan_ctl(argc, argv, "NCL", -1, "0", NULL);
  if (ctl->ncl < 0 || ctl->ncl > NCL)
    ERRMSG("Set 0 <= NCL <= MAX!");
  if (ctl->ncl == 1)
    ERRMSG("Set NCL > 1!");
  for (int icl = 0; icl < ctl->ncl; icl++)
    ctl->clnu[icl] = scan_ctl(argc, argv, "CLNU", icl, "", NULL);

  /* Surface data... */
  ctl->nsf = (int) scan_ctl(argc, argv, "NSF", -1, "0", NULL);
  if (ctl->nsf < 0 || ctl->nsf > NSF)
```

```
    ERRMSG("Set 0 <= NSF <= MAX!");
  if (ctl->nsf == 1)
    ERRMSG("Set NSF > 1!");
  for (int isf = 0; isf < ctl->nsf; isf++)
    ctl->sfnu[isf] = scan_ctl(argc, argv, "SFNU", isf, "", NULL);
  ctl->sftype = (int) scan_ctl(argc, argv, "SFTYPE", -1, "2", NULL);
  if (ctl->sftype < 0 || ctl->sftype > 3)
    ERRMSG("Set 0 <= SFTYPE <= 3!");
  ctl->sfsza = scan_ctl(argc, argv, "SFSZA", -1, "-999", NULL);

  /* Emissivity look-up tables... */
  scan_ctl(argc, argv, "TBLBASE", -1, "-", ctl->tblbase);
  ctl->tblfmt = (int) scan_ctl(argc, argv, "TBLFMT", -1, "1", NULL);

  /* Hydrostatic equilibrium... */
  ctl->hydz = scan_ctl(argc, argv, "HYDZ", -1, "-999", NULL);

  /* Continua... */
  ctl->ctm_co2 = (int) scan_ctl(argc, argv, "CTM_CO2", -1, "1", NULL);
  ctl->ctm_h2o = (int) scan_ctl(argc, argv, "CTM_H2O", -1, "1", NULL);
  ctl->ctm_n2 = (int) scan_ctl(argc, argv, "CTM_N2", -1, "1", NULL);
  ctl->ctm_o2 = (int) scan_ctl(argc, argv, "CTM_O2", -1, "1", NULL);

  /* Ray-tracing... */
  ctl->refrac = (int) scan_ctl(argc, argv, "REFRAC", -1, "1", NULL);
  ctl->rayds = scan_ctl(argc, argv, "RAYDS", -1, "10", NULL);
  ctl->raydz = scan_ctl(argc, argv, "RAYDZ", -1, "0.1", NULL);

  /* Field of view... */
  scan_ctl(argc, argv, "FOV", -1, "-", ctl->fov);

  /* Retrieval interface... */
  ctl->retp_zmin = scan_ctl(argc, argv, "RETP_ZMIN", -1, "-999", NULL);
  ctl->retp_zmax = scan_ctl(argc, argv, "RETP_ZMAX", -1, "-999", NULL);
  ctl->rett_zmin = scan_ctl(argc, argv, "RETT_ZMIN", -1, "-999", NULL);
  ctl->rett_zmax = scan_ctl(argc, argv, "RETT_ZMAX", -1, "-999", NULL);
  for (int ig = 0; ig < ctl->ng; ig++) {
    ctl->retq_zmin[ig] = scan_ctl(argc, argv, "RETQ_ZMIN", ig, "-999", NULL);
    ctl->retq_zmax[ig] = scan_ctl(argc, argv, "RETQ_ZMAX", ig, "-999", NULL);
  }
  for (int iw = 0; iw < ctl->nw; iw++) {
    ctl->retk_zmin[iw] = scan_ctl(argc, argv, "RETK_ZMIN", iw, "-999", NULL);
    ctl->retk_zmax[iw] = scan_ctl(argc, argv, "RETK_ZMAX", iw, "-999", NULL);
  }
  ctl->ret_clz = (int) scan_ctl(argc, argv, "RET_CLZ", -1, "0", NULL);
  ctl->ret_cldz = (int) scan_ctl(argc, argv, "RET_CLDZ", -1, "0", NULL);
  ctl->ret_clk = (int) scan_ctl(argc, argv, "RET_CLK", -1, "0", NULL);
  ctl->ret_sfz = (int) scan_ctl(argc, argv, "RET_SFZ", -1, "0", NULL);
  ctl->ret_sfp = (int) scan_ctl(argc, argv, "RET_SFP", -1, "0", NULL);
  ctl->ret_sft = (int) scan_ctl(argc, argv, "RET_SFT", -1, "0", NULL);
  ctl->ret_sfeps = (int) scan_ctl(argc, argv, "RET_SFEPS", -1, "0", NULL);

  /* Output flags... */
  ctl->write_bbt = (int) scan_ctl(argc, argv, "WRITE_BBT", -1, "0", NULL);
  ctl->write_matrix =
    (int) scan_ctl(argc, argv, "WRITE_MATRIX", -1, "0", NULL);

  /* External forward models... */
  ctl->formod = (int) scan_ctl(argc, argv, "FORMOD", -1, "1", NULL);
  scan_ctl(argc, argv, "RFMBIN", -1, "-", ctl->rfmbin);
  scan_ctl(argc, argv, "RFMHIT", -1, "-", ctl->rfmhit);
  for (int ig = 0; ig < ctl->ng; ig++)
    scan_ctl(argc, argv, "RFMXSC", ig, "-", ctl->rfmxsc[ig]);
}
```

Here is the call graph for this function:



read_matrix()

```
void read_matrix (
            const char * dirname,
            const char * filename,
            gsl_matrix * matrix )
```

Read matrix.

Definition at line 4657 of file jurassic.c.

```
                        {

  FILE *in;

  char dum[LEN], file[LEN], line[LEN];

  double value;

  int i, j;

  /* Set filename... */
  if (dirname != NULL)
    sprintf(file, "%s/%s", dirname, filename);
  else
    sprintf(file, "%s", filename);

  /* Write info... */
  LOG(1, "Read matrix: %s", file);

  /* Open file... */
  if (!(in = fopen(file, "r")))
    ERRMSG("Cannot open file!");

  /* Read data... */
  gsl_matrix_set_zero(matrix);
  while (fgets(line, LEN, in))
    if (sscanf(line, "%d %s %s %s %s %s %d %s %s %s %s %s %lg",
               &i, dum, dum, dum, dum, dum,
               &j, dum, dum, dum, dum, dum, &value) == 13)
      gsl_matrix_set(matrix, (size_t) i, (size_t) j, value);

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

read_obs()

```
void read_obs (
            const char * dirname,
            const char * filename,
            const ctl_t * ctl,
            obs_t * obs )
```

Read observation data.

Definition at line 4697 of file jurassic.c.

```
                {

  FILE *in;

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

  /* Init... */
  obs->nr = 0;

  /* Set filename... */
  if (dirname != NULL)
    sprintf(file, "%s/%s", dirname, filename);
  else
    sprintf(file, "%s", filename);

  /* Write info... */
  LOG(1, "Read observation data: %s", file);
```

```

  /* Open file... */
  if (!(in = fopen(file, "r")))
    ERRMSG("Cannot open file!");

  /* Read line... */
  while (fgets(line, LEN, in)) {

    /* Read data... */
    TOK(line, tok, "%lg", obs->time[obs->nr]);
    TOK(NULL, tok, "%lg", obs->obsz[obs->nr]);
    TOK(NULL, tok, "%lg", obs->obslon[obs->nr]);
    TOK(NULL, tok, "%lg", obs->obslat[obs->nr]);
    TOK(NULL, tok, "%lg", obs->vpz[obs->nr]);
    TOK(NULL, tok, "%lg", obs->vplon[obs->nr]);
    TOK(NULL, tok, "%lg", obs->vplat[obs->nr]);
    TOK(NULL, tok, "%lg", obs->tpz[obs->nr]);
    TOK(NULL, tok, "%lg", obs->tplon[obs->nr]);
    TOK(NULL, tok, "%lg", obs->tplat[obs->nr]);
    for (int id = 0; id < ctl->nd; id++)
      TOK(NULL, tok, "%lg", obs->rad[id][obs->nr]);
    for (int id = 0; id < ctl->nd; id++)
      TOK(NULL, tok, "%lg", obs->tau[id][obs->nr]);

    /* Increment counter... */
    if ((++obs->nr) > NR)
      ERRMSG("Too many rays!");
  }

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

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

  /* Write info... */
  double mini, maxi;
  LOG(2, "Number of ray paths: %d", obs->nr);
  gsl_stats_minmax(&mini, &maxi, obs->time, 1, (size_t) obs->nr);
  LOG(2, "Time range: %.2f ... %.2f s", mini, maxi);
  gsl_stats_minmax(&mini, &maxi, obs->obsz, 1, (size_t) obs->nr);
  LOG(2, "Observer altitude range: %g ... %g km", mini, maxi);
  gsl_stats_minmax(&mini, &maxi, obs->obslon, 1, (size_t) obs->nr);
  LOG(2, "Observer longitude range: %g ... %g deg", mini, maxi);
  gsl_stats_minmax(&mini, &maxi, obs->obslat, 1, (size_t) obs->nr);
  LOG(2, "Observer latitude range: %g ... %g deg", mini, maxi);
  gsl_stats_minmax(&mini, &maxi, obs->vpz, 1, (size_t) obs->nr);
  LOG(2, "View point altitude range: %g ... %g km", mini, maxi);
  gsl_stats_minmax(&mini, &maxi, obs->vplon, 1, (size_t) obs->nr);
  LOG(2, "View point longitude range: %g ... %g deg", mini, maxi);
  gsl_stats_minmax(&mini, &maxi, obs->vplat, 1, (size_t) obs->nr);
  LOG(2, "View point latitude range: %g ... %g deg", mini, maxi);
  gsl_stats_minmax(&mini, &maxi, obs->tpz, 1, (size_t) obs->nr);
  LOG(2, "Tangent point altitude range: %g ... %g km", mini, maxi);
  gsl_stats_minmax(&mini, &maxi, obs->tplon, 1, (size_t) obs->nr);
  LOG(2, "Tangent point longitude range: %g ... %g deg", mini, maxi);
  gsl_stats_minmax(&mini, &maxi, obs->tplat, 1, (size_t) obs->nr);
  LOG(2, "Tangent point latitude range: %g ... %g deg", mini, maxi);
  for (int id = 0; id < ctl->nd; id++) {
    gsl_stats_minmax(&mini, &maxi, obs->rad[id], 1, (size_t) obs->nr);
    if (ctl->write_bbt) {
      LOG(2, "Brightness temperature (%.4f cm^-1) range: %g ... %g K",
          ctl->nu[id], mini, maxi);
    } else {
      LOG(2, "Radiance (%.4f cm^-1) range: %g ... %g W/(m^2 sr cm^-1)",
          ctl->nu[id], mini, maxi);
    }
  }
  for (int id = 0; id < ctl->nd; id++) {
    gsl_stats_minmax(&mini, &maxi, obs->tau[id], 1, (size_t) obs->nr);
    if (ctl->write_bbt) {
      LOG(2, "Transmittance (%.4f cm^-1) range: %g ... %g",
          ctl->nu[id], mini, maxi);
    }
  }
}
```

read_obs_rfm()

```
double read_obs_rfm (
          const char * basename,
```

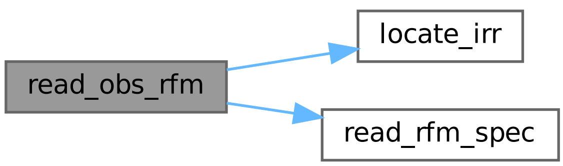
```
const double z,
double * nu,
double * f,
int n )
```

Read observation data in RFM format.

Definition at line 4798 of file jurassic.c.

```
        {

  FILE *in;

  char filename[LEN];

  double filt, fsum = 0, nu2[NSHAPE], *nurfm, *rad, radsum = 0;

  int npts;

  /* Allocate... */
  ALLOC(nurfm, double,
        RFMNPTS);
  ALLOC(rad, double,
        RFMNPTS);

  /* Search RFM spectrum... */
  sprintf(filename, "%s_%05d.asc", basename, (int) (z * 1000));
  if (!(in = fopen(filename, "r"))) {
    sprintf(filename, "%s_%05d.asc", basename, (int) (z * 1000) + 1);
    if (!(in = fopen(filename, "r")))
      ERRMSG("Cannot find RFM data file!");
  }
  fclose(in);

  /* Read RFM spectrum... */
  read_rfm_spec(filename, nurfm, rad, &npts);

  /* Set wavenumbers... */
  nu2[0] = nu[0];
  nu2[n - 1] = nu[n - 1];
  for (int i = 1; i < n - 1; i++)
    nu2[i] = LIN(0.0, nu2[0], n - 1.0, nu2[n - 1], i);

  /* Convolute... */
  for (int ipts = 0; ipts < npts; ipts++)
    if (nurfm[ipts] >= nu2[0] && nurfm[ipts] <= nu2[n - 1]) {
      const int idx = locate_irr(nu2, n, nurfm[ipts]);
      filt = LIN(nu2[idx], f[idx], nu2[idx + 1], f[idx + 1], nurfm[ipts]);
      fsum += filt;
      radsum += filt * rad[ipts];
    }

  /* Free... */
  free(nurfm);
  free(rad);

  /* Return radiance... */
  return radsum / fsum;
}
```

Here is the call graph for this function:



read_rfm_spec()

```
void read_rfm_spec (
            const char * filename,
            double * nu,
            double * rad,
            int * npts )
```

Read RFM spectrum.

Definition at line 4856 of file jurassic.c.

```
                   {

  FILE *in;

  char line[RFMLINE], *tok;

  double dnu, nu0, nu1;

  int ipts = 0;

  /* Write info... */
  LOG(1, "Read RFM data: %s", filename);

  /* Open file... */
  if (!(in = fopen(filename, "r")))
    ERRMSG("Cannot open file!");

  /* Read header...... */
  for (int i = 0; i < 4; i++)
    if (fgets(line, RFMLINE, in) == NULL)
      ERRMSG("Error while reading file header!");
  sscanf(line, "%d %lg %lg %lg", npts, &nu0, &dnu, &nu1);
  if (*npts > RFMNPTS)
    ERRMSG("Too many spectral grid points!");

  /* Read radiance data... */
  while (fgets(line, RFMLINE, in) && ipts < *npts) {
    if ((tok = strtok(line, " \t\n")) != NULL)
      if (sscanf(tok, "%lg", &rad[ipts]) == 1)
        ipts++;
    while ((tok = strtok(NULL, " \t\n")) != NULL)
      if (sscanf(tok, "%lg", &rad[ipts]) == 1)
        ipts++;
  }
  if (ipts != *npts)
    ERRMSG("Error while reading RFM data!");

  /* Compute wavenumbers... */
  for (ipts = 0; ipts < *npts; ipts++)
    nu[ipts] = LIN(0.0, nu0, (double) (*npts - 1), nu1, (double) ipts);

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

read_shape()

```
void read_shape (
            const char * filename,
            double * x,
            double * y,
            int * n )
```

Read shape function.

Definition at line 4907 of file jurassic.c.

```
              {

  FILE *in;

  char line[LEN];

```

```
  /* Write info... */
  LOG(1, "Read shape function: %s", filename);

  /* Open file... */
  if (!(in = fopen(filename, "r")))
    ERRMSG("Cannot open file!");

  /* Read data... */
  *n = 0;
  while (fgets(line, LEN, in))
    if (sscanf(line, "%lg %lg", &x[*n], &y[*n]) == 2)
      if ((++(*n)) > NSHAPE)
        ERRMSG("Too many data points!");

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

  /* Check number of data points... */
  if (*n < 2)
    ERRMSG("Could not read any data!");

  /* Write info... */
  double mini, maxi;
  LOG(2, "Number of data points: %d", *n);
  gsl_stats_minmax(&mini, &maxi, x, 1, (size_t) *n);
  LOG(2, "Range of x values: %.4f ... %.4f", mini, maxi);
  gsl_stats_minmax(&mini, &maxi, y, 1, (size_t) *n);
  LOG(2, "Range of y values: %g ... %g", mini, maxi);
}
```

read_tbl()

```
void read_tbl (
            const ctl_t * ctl,
            tbl_t * tbl )
```

Read look-up table data.

Definition at line 4949 of file jurassic.c.

```
                {

  FILE *in;

  char filename[2 * LEN], line[LEN];

  double eps, press, temp, u;

  /* Loop over trace gases and channels... */
  for (int id = 0; id < ctl->nd; id++)
    for (int ig = 0; ig < ctl->ng; ig++) {

      /* Initialize... */
      tbl->np[id][ig] = -1;
      double eps_old = -999;
      double press_old = -999;
      double temp_old = -999;
      double u_old = -999;
      int nrange = 0;

      /* Set filename... */
      sprintf(filename, "%s_%.4f_%s.%s", ctl->tblbase,
              ctl->nu[id], ctl->emitter[ig],
              ctl->tblfmt == 1 ? "tab" : "bin");

      /* Write info... */
      LOG(1, "Read emissivity table: %s", filename);

      /* Try to open file... */
      if (!(in = fopen(filename, "r"))) {
        WARN("Missing emissivity table: %s", filename);
        continue;
      }

      /* Read ASCII tables... */
      if (ctl->tblfmt == 1) {

        /* Read data... */
        while (fgets(line, LEN, in)) {

          /* Parse line... */
```

```
        if (sscanf(line, "%lg %lg %lg %lg", &press, &temp, &u, &eps) != 4)
          continue;

        /* Check ranges... */
        if (u < UMIN || u > UMAX || eps < EPSMIN || eps > EPSMAX) {
          nrange++;
          continue;
        }

        /* Determine pressure index... */
        if (press != press_old) {
          press_old = press;
          if ((++tbl->np[id][ig]) >= TBLNP)
            ERRMSG("Too many pressure levels!");
          tbl->nt[id][ig][tbl->np[id][ig]] = -1;
        }

        /* Determine temperature index... */
        if (temp != temp_old) {
          temp_old = temp;
          if ((++tbl->nt[id][ig][tbl->np[id][ig]]) >= TBLNT)
            ERRMSG("Too many temperatures!");
          tbl->nu[id][ig][tbl->np[id][ig]]
            [tbl->nt[id][ig][tbl->np[id][ig]]] = -1;
        }

        /* Determine column density index... */
        if ((eps > eps_old && u > u_old) || tbl->nu[id][ig][tbl->np[id][ig]]
            [tbl->nt[id][ig][tbl->np[id][ig]]] < 0) {
          eps_old = eps;
          u_old = u;
          if ((++tbl->nu[id][ig][tbl->np[id][ig]]
               [tbl->nt[id][ig][tbl->np[id][ig]]]) >= TBLNU)
            ERRMSG("Too many column densities!");
        }

        /* Store data... */
        tbl->p[id][ig][tbl->np[id][ig]] = press;
        tbl->t[id][ig][tbl->np[id][ig]][tbl->nt[id][ig][tbl->np[id][ig]]]
          = temp;
        tbl->u[id][ig][tbl->np[id][ig]][tbl->nt[id][ig][tbl->np[id][ig]]]
          [tbl->nu[id][ig][tbl->np[id][ig]]
           [tbl->nt[id][ig][tbl->np[id][ig]]]] = (float) u;
        tbl->eps[id][ig][tbl->np[id][ig]][tbl->nt[id][ig][tbl->np[id][ig]]]
          [tbl->nu[id][ig][tbl->np[id][ig]]
           [tbl->nt[id][ig][tbl->np[id][ig]]]] = (float) eps;
      }

      /* Increment counters... */
      tbl->np[id][ig]++;
      for (int ip = 0; ip < tbl->np[id][ig]; ip++) {
        tbl->nt[id][ig][ip]++;
        for (int it = 0; it < tbl->nt[id][ig][ip]; it++)
          tbl->nu[id][ig][ip][it]++;
      }
    }

    /* Read binary data... */
    else if (ctl->tblfmt == 2) {

      /* Read data... */
      FREAD(&tbl->np[id][ig], int,
            1,
            in);
      if (tbl->np[id][ig] > TBLNP)
        ERRMSG("Too many pressure levels!");
      FREAD(tbl->p[id][ig], double,
              (size_t) tbl->np[id][ig],
            in);
      for (int ip = 0; ip < tbl->np[id][ig]; ip++) {
        FREAD(&tbl->nt[id][ig][ip], int,
              1,
              in);
        if (tbl->nt[id][ig][ip] > TBLNT)
          ERRMSG("Too many temperatures!");
        FREAD(tbl->t[id][ig][ip], double,
                (size_t) tbl->nt[id][ig][ip],
              in);
        for (int it = 0; it < tbl->nt[id][ig][ip]; it++) {
          FREAD(&tbl->nu[id][ig][ip][it], int,
                1,
                in);
          if (tbl->nu[id][ig][ip][it] > TBLNU)
            ERRMSG("Too many column densities!");
          FREAD(tbl->u[id][ig][ip][it], float,
                  (size_t) tbl->nu[id][ig][ip][it],
                in);
```

```
            FREAD(tbl->eps[id][ig][ip][it], float,
                    (size_t) tbl->nu[id][ig][ip][it],
                  in);
          }
        }
      }

      /* Error message... */
      else
        ERRMSG("Unknown look-up table format!");

      /* Check ranges... */
      if (nrange > 0)
        WARN("Column density or emissivity out of range (%d data points)!",
             nrange);

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

      /* Write info... */
      for (int ip = 0; ip < tbl->np[id][ig]; ip++)
        LOG(2,
            "p[%2d]= %.5e hPa | T[0:%2d]= %.2f ... %.2f K | u[0:%3d]= %.5e ... %.5e molec/cm^2 |
      eps[0:%3d]= %.5e ... %.5e",
            ip, tbl->p[id][ig][ip], tbl->nt[id][ig][ip] - 1,
            tbl->t[id][ig][ip][0],
            tbl->t[id][ig][ip][tbl->nt[id][ig][ip] - 1],
            tbl->nu[id][ig][ip][0] - 1, tbl->u[id][ig][ip][0][0],
            tbl->u[id][ig][ip][0][tbl->nu[id][ig][ip][0] - 1],
            tbl->nu[id][ig][ip][0] - 1, tbl->eps[id][ig][ip][0][0],
            tbl->eps[id][ig][ip][0][tbl->nu[id][ig][ip][0] - 1]);
    }
}
```

scan_ctl()

```
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 5114 of file jurassic.c.

```
                   {

  FILE *in = NULL;

  char dummy[LEN], fullname1[LEN], fullname2[LEN], line[LEN],
    rvarname[LEN], rval[LEN];

  int contain = 0;

  /* Open file... */
  if (argv[1][0] != '-')
    if (!(in = fopen(argv[1], "r")))
      ERRMSG("Cannot open file!");

  /* Set full variable name... */
  if (arridx >= 0) {
    sprintf(fullname1, "%s[%d]", varname, arridx);
    sprintf(fullname2, "%s[*]", varname);
  } else {
    sprintf(fullname1, "%s", varname);
    sprintf(fullname2, "%s", varname);
  }

  /* Read data... */
  if (in != NULL)
    while (fgets(line, LEN, in))
      if (sscanf(line, "%s %s %s", rvarname, dummy, rval) == 3)
        if (strcasecmp(rvarname, fullname1) == 0 ||
            strcasecmp(rvarname, fullname2) == 0) {
          contain = 1;
          break;
```

```
      }
  for (int i = 1; i < argc - 1; i++)
    if (strcasecmp(argv[i], fullname1) == 0 ||
        strcasecmp(argv[i], fullname2) == 0) {
      sprintf(rval, "%s", argv[i + 1]);
      contain = 1;
      break;
    }

  /* Close file... */
  if (in != NULL)
    fclose(in);

  /* Check for missing variables... */
  if (!contain) {
    if (strlen(defvalue) > 0)
      sprintf(rval, "%s", defvalue);
    else
      ERRMSG("Missing variable %s!\n", fullname1);
  }

  /* Write info... */
  LOG(1, "%s = %s", fullname1, rval);

  /* Return values... */
  if (value != NULL)
    sprintf(value, "%s", rval);
  return atof(rval);
}
```

sza()

```
double sza (
            const double sec,
            const double lon,
            const double lat )
```

Calculate solar zenith angle.

Definition at line 5183 of file jurassic.c.

```
                          {

  /* Number of days and fraction with respect to 2000-01-01T12:00Z... */
  const double D = sec / 86400 - 0.5;

  /* Geocentric apparent ecliptic longitude [rad]... */
  const double g = DEG2RAD(357.529 + 0.98560028 * D);
  const double q = 280.459 + 0.98564736 * D;
  const double L = DEG2RAD(q + 1.915 * sin(g) + 0.020 * sin(2 * g));

  /* Mean obliquity of the ecliptic [rad]... */
  const double e = DEG2RAD(23.439 - 0.00000036 * D);

  /* Declination [rad]... */
  const double dec = asin(sin(e) * sin(L));

  /* Right ascension [rad]... */
  const double ra = atan2(cos(e) * sin(L), cos(L));

  /* Greenwich Mean Sidereal Time [h]... */
  const double GMST = 18.697374558 + 24.06570982441908 * D;

  /* Local Sidereal Time [h]... */
  const double LST = GMST + lon / 15;

  /* Hour angle [rad]... */
  const double h = LST / 12 * M_PI - ra;

  /* Convert latitude... */
  const double latr = DEG2RAD(lat);

  /* Return solar zenith angle [deg]... */
  return RAD2DEG(acos(sin(latr) * sin(dec) + cos(latr) * cos(dec) * cos(h)));
}
```

tangent_point()

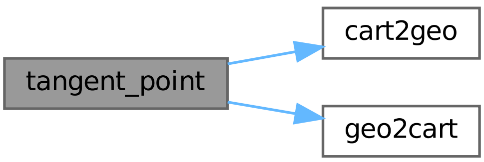
```
void tangent_point (
            const los_t * los,
            double * tpz,
            double * tplon,
            double * tplat )
```

Find tangent point of a given LOS.

Definition at line 5223 of file jurassic.c.

```
                   {

  double dummy, v[3], v0[3], v2[3];

  /* Find minimum altitude... */
  const size_t ip = gsl_stats_min_index(los->z, 1, (size_t) los->np);

  /* Nadir or zenith... */
  if (ip <= 0 || ip >= (size_t) los->np - 1) {
    *tpz = los->z[los->np - 1];
    *tplon = los->lon[los->np - 1];
    *tplat = los->lat[los->np - 1];
  }

  /* Limb... */
  else {

    /* Determine interpolating polynomial y=a*x^2+b*x+c... */
    const double yy0 = los->z[ip - 1];
    const double yy1 = los->z[ip];
    const double yy2 = los->z[ip + 1];
    const double x1 = sqrt(POW2(los->ds[ip]) - POW2(yy1 - yy0));
    const double x2 = x1 + sqrt(POW2(los->ds[ip + 1]) - POW2(yy2 - yy1));
    const double a = 1 / (x1 - x2) * (-(yy0 - yy1) / x1 + (yy0 - yy2) / x2);
    const double b = -(yy0 - yy1) / x1 - a * x1;
    const double c = yy0;

    /* Get tangent point location... */
    const double x = -b / (2 * a);
    *tpz = a * x * x + b * x + c;
    geo2cart(los->z[ip - 1], los->lon[ip - 1], los->lat[ip - 1], v0);
    geo2cart(los->z[ip + 1], los->lon[ip + 1], los->lat[ip + 1], v2);
    for (int i = 0; i < 3; i++)
      v[i] = LIN(0.0, v0[i], x2, v2[i], x);
    cart2geo(v, &dummy, tplon, tplat);
  }
}
```

Here is the call graph for this function:



time2jsec()

```
void time2jsec (
            const int year,
```

```
            const int mon,
            const int day,
            const int hour,
            const int min,
            const int sec,
            const double remain,
            double * jsec )
```

Convert date to seconds.

Definition at line 5267 of file jurassic.c.

```
                  {

  struct tm t0, t1;

  t0.tm_year = 100;
  t0.tm_mon = 0;
  t0.tm_mday = 1;
  t0.tm_hour = 0;
  t0.tm_min = 0;
  t0.tm_sec = 0;

  t1.tm_year = year - 1900;
  t1.tm_mon = mon - 1;
  t1.tm_mday = day;
  t1.tm_hour = hour;
  t1.tm_min = min;
  t1.tm_sec = sec;

  *jsec = (double) timegm(&t1) - (double) timegm(&t0) + remain;
}
```

timer()

```
void timer (
            const char * name,
            const char * file,
            const char * func,
            int line,
            int mode )
```

Measure wall-clock time.

Definition at line 5298 of file jurassic.c.

```
              {

  static double w0[10];

  static int l0[10], nt;

  /* Start new timer... */
  if (mode == 1) {
    w0[nt] = omp_get_wtime();
    l0[nt] = line;
    if ((++nt) >= 10)
      ERRMSG("Too many timers!");
  }

  /* Write elapsed time... */
  else {

    /* Check timer index... */
    if (nt - 1 < 0)
      ERRMSG("Coding error!");

    /* Write elapsed time... */
    LOG(1, "Timer '%s' (%s, %s, l%d-%d): %.3f sec",
        name, file, func, l0[nt - 1], line, omp_get_wtime() - w0[nt - 1]);
  }

  /* Stop timer... */
  if (mode == 3)
    nt--;
}
```

write_atm()

```
void write_atm (
            const char * dirname,
            const char * filename,
            const ctl_t * ctl,
            const atm_t * atm )
```

Write atmospheric data.

Definition at line 5336 of file jurassic.c.

```
                     {

  FILE *out;

  char file[LEN];

  int n = 6;

  /* Set filename... */
  if (dirname != NULL)
    sprintf(file, "%s/%s", dirname, filename);
  else
    sprintf(file, "%s", filename);

  /* Write info... */
  LOG(1, "Write atmospheric data: %s", file);

  /* Create file... */
  if (!(out = fopen(file, "w")))
    ERRMSG("Cannot create file!");

  /* Write header... */
  fprintf(out,
          "# $1 = time (seconds since 2000-01-01T00:00Z)\n"
          "# $2 = altitude [km]\n"
          "# $3 = longitude [deg]\n"
          "# $4 = latitude [deg]\n"
          "# $5 = pressure [hPa]\n" "# $6 = temperature [K]\n");
  for (int ig = 0; ig < ctl->ng; ig++)
    fprintf(out, "# $%d = %s volume mixing ratio [ppv]\n",
            ++n, ctl->emitter[ig]);
  for (int iw = 0; iw < ctl->nw; iw++)
    fprintf(out, "# $%d = extinction (window %d) [km^-1]\n", ++n, iw);
  if (ctl->ncl > 0) {
    fprintf(out, "# $%d = cloud layer height [km]\n", ++n);
    fprintf(out, "# $%d = cloud layer depth [km]\n", ++n);
    for (int icl = 0; icl < ctl->ncl; icl++)
      fprintf(out, "# $%d = cloud layer extinction (%.4f cm^-1) [km^-1]\n",
              ++n, ctl->clnu[icl]);
  }
  if (ctl->nsf > 0) {
    fprintf(out, "# $%d = surface layer height [km]\n", ++n);
    fprintf(out, "# $%d = surface layer pressure [hPa]\n", ++n);
    fprintf(out, "# $%d = surface layer temperature [K]\n", ++n);
    for (int isf = 0; isf < ctl->nsf; isf++)
      fprintf(out, "# $%d = surface layer emissivity (%.4f cm^-1)\n",
              ++n, ctl->sfnu[isf]);
  }

  /* Write data... */
  for (int ip = 0; ip < atm->np; ip++) {
    if (ip == 0 || atm->time[ip] != atm->time[ip - 1])
      fprintf(out, "\n");
    fprintf(out, "%.2f %g %g %g %g %g", atm->time[ip], atm->z[ip],
            atm->lon[ip], atm->lat[ip], atm->p[ip], atm->t[ip]);
    for (int ig = 0; ig < ctl->ng; ig++)
      fprintf(out, " %g", atm->q[ig][ip]);
    for (int iw = 0; iw < ctl->nw; iw++)
      fprintf(out, " %g", atm->k[iw][ip]);
    if (ctl->ncl > 0) {
      fprintf(out, " %g %g", atm->clz, atm->cldz);
      for (int icl = 0; icl < ctl->ncl; icl++)
        fprintf(out, " %g", atm->clk[icl]);
    }
    if (ctl->nsf > 0) {
      fprintf(out, " %g %g %g", atm->sfz, atm->sfp, atm->sft);
      for (int isf = 0; isf < ctl->nsf; isf++)
        fprintf(out, " %g", atm->sfeps[isf]);
    }
    fprintf(out, "\n");
```

```
  }

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

  /* Write info... */
  double mini, maxi;
  LOG(2, "Number of data points: %d", atm->np);
  gsl_stats_minmax(&mini, &maxi, atm->time, 1, (size_t) atm->np);
  LOG(2, "Time range: %.2f ... %.2f s", mini, maxi);
  gsl_stats_minmax(&mini, &maxi, atm->z, 1, (size_t) atm->np);
  LOG(2, "Altitude range: %g ... %g km", mini, maxi);
  gsl_stats_minmax(&mini, &maxi, atm->lon, 1, (size_t) atm->np);
  LOG(2, "Longitude range: %g ... %g deg", mini, maxi);
  gsl_stats_minmax(&mini, &maxi, atm->lat, 1, (size_t) atm->np);
  LOG(2, "Latitude range: %g ... %g deg", mini, maxi);
  gsl_stats_minmax(&mini, &maxi, atm->p, 1, (size_t) atm->np);
  LOG(2, "Pressure range: %g ... %g hPa", maxi, mini);
  gsl_stats_minmax(&mini, &maxi, atm->t, 1, (size_t) atm->np);
  LOG(2, "Temperature range: %g ... %g K", mini, maxi);
  for (int ig = 0; ig < ctl->ng; ig++) {
    gsl_stats_minmax(&mini, &maxi, atm->q[ig], 1, (size_t) atm->np);
    LOG(2, "Emitter %s range: %g ... %g ppv", ctl->emitter[ig], mini, maxi);
  }
  for (int iw = 0; iw < ctl->nw; iw++) {
    gsl_stats_minmax(&mini, &maxi, atm->k[iw], 1, (size_t) atm->np);
    LOG(2, "Extinction range (window %d): %g ... %g km^-1", iw, mini, maxi);
  }
  if (ctl->ncl > 0 && atm->np == 0) {
    LOG(2, "Cloud layer: z= %g km | dz= %g km | k= %g ... %g km^-1",
        atm->clz, atm->cldz, atm->clk[0], atm->clk[ctl->ncl - 1]);
  } else
    LOG(2, "Cloud layer: none");
  if (ctl->nsf > 0 && atm->np == 0) {
    LOG(2,
        "Surface layer: z_s= %g km | p_s= %g hPa | T_s = %g K | eps= %g ... %g",
        atm->sfz, atm->sfp, atm->sft, atm->sfeps[0],
        atm->sfeps[ctl->nsf - 1]);
  } else
    LOG(2, "Surface layer: none");
}
```

write_atm_rfm()

```
void write_atm_rfm (
            const char * filename,
            const ctl_t * ctl,
            const atm_t * atm )
```

Write atmospheric data in RFM format.

Definition at line 5454 of file jurassic.c.

```
                      {

  FILE *out;

  /* Write info... */
  LOG(1, "Write RFM data: %s", filename);

  /* Create file... */
  if (!(out = fopen(filename, "w")))
    ERRMSG("Cannot create file!");

  /* Write data... */
  fprintf(out, "%d\n", atm->np);
  fprintf(out, "*HGT [km]\n");
  for (int ip = 0; ip < atm->np; ip++)
    fprintf(out, "%g\n", atm->z[ip]);
  fprintf(out, "*PRE [mb]\n");
  for (int ip = 0; ip < atm->np; ip++)
    fprintf(out, "%g\n", atm->p[ip]);
  fprintf(out, "*TEM [K]\n");
  for (int ip = 0; ip < atm->np; ip++)
    fprintf(out, "%g\n", atm->t[ip]);
  for (int ig = 0; ig < ctl->ng; ig++) {
    fprintf(out, "*%s [ppmv]\n", ctl->emitter[ig]);
    for (int ip = 0; ip < atm->np; ip++)
      fprintf(out, "%g\n", atm->q[ig][ip] * 1e6);
  }
```

```
  fprintf(out, "*END\n");

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

write_matrix()

```
void write_matrix (
        const char * dirname,
        const char * filename,
        const ctl_t * ctl,
        const gsl_matrix * matrix,
        const atm_t * atm,
        const obs_t * obs,
        const char * rowspace,
        const char * colspace,
        const char * sort )
```

Write matrix.

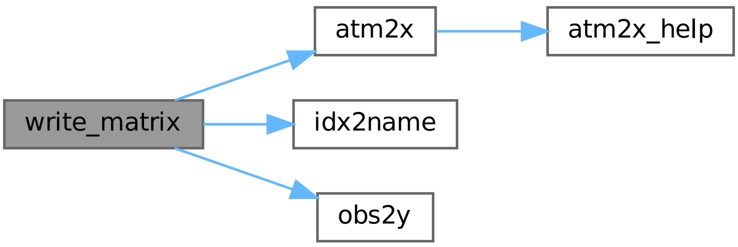
Definition at line 5492 of file jurassic.c.

```
                           {

  FILE *out;

  char file[LEN], quantity[LEN];

  int *cida, *ciqa, *cipa, *cira, *rida, *riqa, *ripa, *rira;

  size_t i, j, nc, nr;

  /* Check output flag... */
  if (!ctl->write_matrix)
    return;

  /* Allocate... */
  ALLOC(cida, int,
        M);
  ALLOC(ciqa, int,
        N);
  ALLOC(cipa, int,
        N);
  ALLOC(cira, int,
        M);
  ALLOC(rida, int,
        M);
  ALLOC(riqa, int,
        N);
  ALLOC(ripa, int,
        N);
  ALLOC(rira, int,
        M);

  /* Set filename... */
  if (dirname != NULL)
    sprintf(file, "%s/%s", dirname, filename);
  else
    sprintf(file, "%s", filename);

  /* Write info... */
  LOG(1, "Write matrix: %s", file);

  /* Create file... */
  if (!(out = fopen(file, "w")))
    ERRMSG("Cannot create file!");

  /* Write header (row space)... */
  if (rowspace[0] == 'y') {

    fprintf(out,
            "# $1 = Row: index (measurement space)\n"
            "# $2 = Row: channel wavenumber [cm^-1]\n"
            "# $3 = Row: time (seconds since 2000-01-01T00:00Z)\n"
            "# $4 = Row: view point altitude [km]\n"
            "# $5 = Row: view point longitude [deg]\n"
```

```
          "# $6 = Row: view point latitude [deg]\n");

    /* Get number of rows... */
    nr = obs2y(ctl, obs, NULL, rida, rira);

  } else {

    fprintf(out,
            "# $1 = Row: index (state space)\n"
            "# $2 = Row: name of quantity\n"
            "# $3 = Row: time (seconds since 2000-01-01T00:00Z)\n"
            "# $4 = Row: altitude [km]\n"
            "# $5 = Row: longitude [deg]\n" "# $6 = Row: latitude [deg]\n");

    /* Get number of rows... */
    nr = atm2x(ctl, atm, NULL, riqa, ripa);
  }

  /* Write header (column space)... */
  if (colspace[0] == 'y') {

    fprintf(out,
            "# $7 = Col: index (measurement space)\n"
            "# $8 = Col: channel wavenumber [cm^-1]\n"
            "# $9 = Col: time (seconds since 2000-01-01T00:00Z)\n"
            "# $10 = Col: view point altitude [km]\n"
            "# $11 = Col: view point longitude [deg]\n"
            "# $12 = Col: view point latitude [deg]\n");

    /* Get number of columns... */
    nc = obs2y(ctl, obs, NULL, cida, cira);

  } else {

    fprintf(out,
            "# $7 = Col: index (state space)\n"
            "# $8 = Col: name of quantity\n"
            "# $9 = Col: time (seconds since 2000-01-01T00:00Z)\n"
            "# $10 = Col: altitude [km]\n"
            "# $11 = Col: longitude [deg]\n" "# $12 = Col: latitude [deg]\n");

    /* Get number of columns... */
    nc = atm2x(ctl, atm, NULL, ciqa, cipa);
  }

  /* Write header entry... */
  fprintf(out, "# $13 = Matrix element\n\n");

  /* Write matrix data... */
  i = j = 0;
  while (i < nr && j < nc) {

    /* Write info about the row... */
    if (rowspace[0] == 'y')
      fprintf(out, "%d %.4f %.2f %g %g %g",
              (int) i, ctl->nu[rida[i]],
              obs->time[rira[i]], obs->vpz[rira[i]],
              obs->vplon[rira[i]], obs->vplat[rira[i]]);
    else {
      idx2name(ctl, riqa[i], quantity);
      fprintf(out, "%d %s %.2f %g %g %g", (int) i, quantity,
              atm->time[ripa[i]], atm->z[ripa[i]],
              atm->lon[ripa[i]], atm->lat[ripa[i]]);
    }

    /* Write info about the column... */
    if (colspace[0] == 'y')
      fprintf(out, " %d %.4f %.2f %g %g %g",
              (int) j, ctl->nu[cida[j]],
              obs->time[cira[j]], obs->vpz[cira[j]],
              obs->vplon[cira[j]], obs->vplat[cira[j]]);
    else {
      idx2name(ctl, ciqa[j], quantity);
      fprintf(out, " %d %s %.2f %g %g %g", (int) j, quantity,
              atm->time[cipa[j]], atm->z[cipa[j]],
              atm->lon[cipa[j]], atm->lat[cipa[j]]);
    }

    /* Write matrix entry... */
    fprintf(out, " %g\n", gsl_matrix_get(matrix, i, j));

    /* Set matrix indices... */
    if (sort[0] == 'r') {
      j++;
      if (j >= nc) {
        j = 0;
        i++;
```

```
        fprintf(out, "\n");
      }
    } else {
      i++;
      if (i >= nr) {
        i = 0;
        j++;
        fprintf(out, "\n");
      }
    }
  }

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

  /* Free... */
  free(cida);
  free(ciqa);
  free(cipa);
  free(cira);
  free(rida);
  free(riqa);
  free(ripa);
  free(rira);
}
```

Here is the call graph for this function:



write_obs()

```
void write_obs (
            const char * dirname,
            const char * filename,
            const ctl_t * ctl,
            const obs_t * obs )
```

Write observation data.

Definition at line 5670 of file jurassic.c.

```
{

  FILE *out;

  char file[LEN];

  int n = 10;

  /* Set filename... */
  if (dirname != NULL)
    sprintf(file, "%s/%s", dirname, filename);
  else
    sprintf(file, "%s", filename);

```

```
/* Write info... */
LOG(1, "Write observation data: %s", file);

/* Create file... */
if (!(out = fopen(file, "w")))
  ERRMSG("Cannot create file!");

/* Write header... */
fprintf(out,
        "# $1 = time (seconds since 2000-01-01T00:00Z)\n"
        "# $2 = observer altitude [km]\n"
        "# $3 = observer longitude [deg]\n"
        "# $4 = observer latitude [deg]\n"
        "# $5 = view point altitude [km]\n"
        "# $6 = view point longitude [deg]\n"
        "# $7 = view point latitude [deg]\n"
        "# $8 = tangent point altitude [km]\n"
        "# $9 = tangent point longitude [deg]\n"
        "# $10 = tangent point latitude [deg]\n");
for (int id = 0; id < ctl->nd; id++)
  if (ctl->write_bbt)
    fprintf(out, "# $%d = brightness temperature (%.4f cm^-1) [K]\n",
            ++n, ctl->nu[id]);
  else
    fprintf(out, "# $%d = radiance (%.4f cm^-1) [W/(m^2 sr cm^-1)]\n",
            ++n, ctl->nu[id]);
for (int id = 0; id < ctl->nd; id++)
  fprintf(out, "# $%d = transmittance (%.4f cm^-1) [-]\n", ++n,
          ctl->nu[id]);

/* Write data... */
for (int ir = 0; ir < obs->nr; ir++) {
  if (ir == 0 || obs->time[ir] != obs->time[ir - 1])
    fprintf(out, "\n");
  fprintf(out, "%.2f %g %g %g %g %g %g %g %g %g", obs->time[ir],
          obs->obsz[ir], obs->obslon[ir], obs->obslat[ir],
          obs->vpz[ir], obs->vplon[ir], obs->vplat[ir],
          obs->tpz[ir], obs->tplon[ir], obs->tplat[ir]);
  for (int id = 0; id < ctl->nd; id++)
    fprintf(out, " %g", obs->rad[id][ir]);
  for (int id = 0; id < ctl->nd; id++)
    fprintf(out, " %g", obs->tau[id][ir]);
  fprintf(out, "\n");
}

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

/* Write info... */
double mini, maxi;
LOG(2, "Number of ray paths: %d", obs->nr);
gsl_stats_minmax(&mini, &maxi, obs->time, 1, (size_t) obs->nr);
LOG(2, "Time range: %.2f ... %.2f s", mini, maxi);
gsl_stats_minmax(&mini, &maxi, obs->obsz, 1, (size_t) obs->nr);
LOG(2, "Observer altitude range: %g ... %g km", mini, maxi);
gsl_stats_minmax(&mini, &maxi, obs->obslon, 1, (size_t) obs->nr);
LOG(2, "Observer longitude range: %g ... %g deg", mini, maxi);
gsl_stats_minmax(&mini, &maxi, obs->obslat, 1, (size_t) obs->nr);
LOG(2, "Observer latitude range: %g ... %g deg", mini, maxi);
gsl_stats_minmax(&mini, &maxi, obs->vpz, 1, (size_t) obs->nr);
LOG(2, "View point altitude range: %g ... %g km", mini, maxi);
gsl_stats_minmax(&mini, &maxi, obs->vplon, 1, (size_t) obs->nr);
LOG(2, "View point longitude range: %g ... %g deg", mini, maxi);
gsl_stats_minmax(&mini, &maxi, obs->vplat, 1, (size_t) obs->nr);
LOG(2, "View point latitude range: %g ... %g deg", mini, maxi);
gsl_stats_minmax(&mini, &maxi, obs->tpz, 1, (size_t) obs->nr);
LOG(2, "Tangent point altitude range: %g ... %g km", mini, maxi);
gsl_stats_minmax(&mini, &maxi, obs->tplon, 1, (size_t) obs->nr);
LOG(2, "Tangent point longitude range: %g ... %g deg", mini, maxi);
gsl_stats_minmax(&mini, &maxi, obs->tplat, 1, (size_t) obs->nr);
LOG(2, "Tangent point latitude range: %g ... %g deg", mini, maxi);
for (int id = 0; id < ctl->nd; id++) {
  gsl_stats_minmax(&mini, &maxi, obs->rad[id], 1, (size_t) obs->nr);
  if (ctl->write_bbt) {
    LOG(2, "Brightness temperature (%.4f cm^-1) range: %g ... %g K",
        ctl->nu[id], mini, maxi);
  } else {
    LOG(2, "Radiance (%.4f cm^-1) range: %g ... %g W/(m^2 sr cm^-1)",
        ctl->nu[id], mini, maxi);
  }
}
for (int id = 0; id < ctl->nd; id++) {
  gsl_stats_minmax(&mini, &maxi, obs->tau[id], 1, (size_t) obs->nr);
  if (ctl->write_bbt) {
    LOG(2, "Transmittance (%.4f cm^-1) range: %g ... %g",
        ctl->nu[id], mini, maxi);
  }
```

```
  }
}
```

write_shape()

```
void write_shape (
            const char * filename,
            const double * x,
            const double * y,
            const int n )
```

Write shape function.

Definition at line 5780 of file jurassic.c.

```
                     {

  FILE *out;

  /* Write info... */
  LOG(1, "Write shape function: %s", filename);

  /* Create file... */
  if (!(out = fopen(filename, "w")))
    ERRMSG("Cannot create file!");

  /* Write header... */
  fprintf(out,
          "# $1 = shape function x-value [-]\n"
          "# $2 = shape function y-value [-]\n\n");

  /* Write data... */
  for (int i = 0; i < n; i++)
    fprintf(out, "%.10g %.10g\n", x[i], y[i]);

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

write_tbl()

```
void write_tbl (
            const ctl_t * ctl,
            const tbl_t * tbl )
```

Write look-up table data.

Definition at line 5810 of file jurassic.c.

```
                          {

  FILE *out;

  char filename[2 * LEN];

  /* Loop over emitters and detectors... */
  for (int ig = 0; ig < ctl->ng; ig++)
    for (int id = 0; id < ctl->nd; id++) {

      /* Set filename... */
      sprintf(filename, "%s_%.4f_%s.%s", ctl->tblbase,
              ctl->nu[id], ctl->emitter[ig],
              ctl->tblfmt == 1 ? "tab" : "bin");

      /* Write info... */
      LOG(1, "Write emissivity table: %s", filename);

      /* Create file... */
      if (!(out = fopen(filename, "w")))
        ERRMSG("Cannot create file!");

      /* Write ASCII data... */
      if (ctl->tblfmt == 1) {
```

```

      /* Write header... */
      fprintf(out,
              "# $1 = pressure [hPa]\n"
              "# $2 = temperature [K]\n"
              "# $3 = column density [molecules/cm^2]\n"
              "# $4 = emissivity [-]\n");

      /* Save table file... */
      for (int ip = 0; ip < tbl->np[id][ig]; ip++)
        for (int it = 0; it < tbl->nt[id][ig][ip]; it++) {
          fprintf(out, "\n");
          for (int iu = 0; iu < tbl->nu[id][ig][ip][it]; iu++)
            fprintf(out, "%g %g %e %e\n",
                    tbl->p[id][ig][ip], tbl->t[id][ig][ip][it],
                    tbl->u[id][ig][ip][it][iu],
                    tbl->eps[id][ig][ip][it][iu]);
        }
    }

    /* Write binary data... */
    else if (ctl->tblfmt == 2) {
      FWRITE(&tbl->np[id][ig], int,
             1,
             out);
      FWRITE(tbl->p[id][ig], double,
               (size_t) tbl->np[id][ig],
             out);
      for (int ip = 0; ip < tbl->np[id][ig]; ip++) {
        FWRITE(&tbl->nt[id][ig][ip], int,
               1,
               out);
        FWRITE(tbl->t[id][ig][ip], double,
                 (size_t) tbl->nt[id][ig][ip],
               out);
        for (int it = 0; it < tbl->nt[id][ig][ip]; it++) {
          FWRITE(&tbl->nu[id][ig][ip][it], int,
                 1,
                 out);
          FWRITE(tbl->u[id][ig][ip][it], float,
                   (size_t) tbl->nu[id][ig][ip][it],
                 out);
          FWRITE(tbl->eps[id][ig][ip][it], float,
                   (size_t) tbl->nu[id][ig][ip][it],
                 out);
        }
      }
    }

    /* Error message... */
    else
      ERRMSG("Unknown look-up table format!");

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

x2atm()

```
void x2atm (
            const ctl_t * ctl,
            const gsl_vector * x,
            atm_t * atm )
```

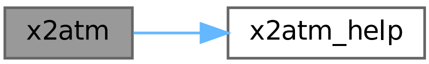
Decompose parameter vector or state vector.

Definition at line 5896 of file jurassic.c.

```
                  {

  size_t n = 0;

  /* Get pressure... */
  for (int ip = 0; ip < atm->np; ip++)
    if (atm->z[ip] >= ctl->retp_zmin && atm->z[ip] <= ctl->retp_zmax)
      x2atm_help(&atm->p[ip], x, &n);

  /* Get temperature... */
  for (int ip = 0; ip < atm->np; ip++)
```

```
    if (atm->z[ip] >= ctl->rett_zmin && atm->z[ip] <= ctl->rett_zmax)
      x2atm_help(&atm->t[ip], x, &n);

  /* Get volume mixing ratio... */
  for (int ig = 0; ig < ctl->ng; ig++)
    for (int ip = 0; ip < atm->np; ip++)
      if (atm->z[ip] >= ctl->retq_zmin[ig]
          && atm->z[ip] <= ctl->retq_zmax[ig])
        x2atm_help(&atm->q[ig][ip], x, &n);

  /* Get extinction... */
  for (int iw = 0; iw < ctl->nw; iw++)
    for (int ip = 0; ip < atm->np; ip++)
      if (atm->z[ip] >= ctl->retk_zmin[iw]
          && atm->z[ip] <= ctl->retk_zmax[iw])
        x2atm_help(&atm->k[iw][ip], x, &n);

  /* Get cloud data... */
  if (ctl->ret_clz)
    x2atm_help(&atm->clz, x, &n);
  if (ctl->ret_cldz)
    x2atm_help(&atm->cldz, x, &n);
  if (ctl->ret_clk)
    for (int icl = 0; icl < ctl->ncl; icl++)
      x2atm_help(&atm->clk[icl], x, &n);

  /* Get surface data... */
  if (ctl->ret_sfz)
    x2atm_help(&atm->sfz, x, &n);
  if (ctl->ret_sfp)
    x2atm_help(&atm->sfp, x, &n);
  if (ctl->ret_sft)
    x2atm_help(&atm->sft, x, &n);
  if (ctl->ret_sfeps)
    for (int isf = 0; isf < ctl->nsf; isf++)
      x2atm_help(&atm->sfeps[isf], x, &n);
}
```

Here is the call graph for this function:



x2atm_help()

```
void x2atm_help (
            double * value,
            const gsl_vector * x,
            size_t * n )
```

Get element from state vector.

Definition at line 5950 of file jurassic.c.

```
             {

  /* Get state vector element... */
  *value = gsl_vector_get(x, *n);
  (*n)++;
}
```

y2obs()

```
void y2obs (
            const ctl_t * ctl,
            const gsl_vector * y,
            obs_t * obs )
```

Decompose measurement vector.

Definition at line 5962 of file jurassic.c.

```
                {

  size_t m = 0;

  /* Decompose measurement vector... */
  for (int ir = 0; ir < obs->nr; ir++)
    for (int id = 0; id < ctl->nd; id++)
      if (isfinite(obs->rad[id][ir])) {
        obs->rad[id][ir] = gsl_vector_get(y, m);
        m++;
      }
}
```

5.6 jurassic.c

Go to the documentation of this file.

```
/*
  This file is part of JURASSIC.

  JURASSIC is free software: you can redistribute it and/or modify
  it under the terms of the GNU General Public License as published by
  the Free Software Foundation, either version 3 of the License, or
  (at your option) any later version.

  JURASSIC is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
  GNU General Public License for more details.

  You should have received a copy of the GNU eneral Public License
  along with JURASSIC. If not, see <http://www.gnu.org/licenses/>.

  Copyright (C) 2003-2025 Forschungszentrum Juelich GmbH
*/

#include "jurassic.h"

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

size_t atm2x(
  const ctl_t *ctl,
  const atm_t *atm,
  gsl_vector *x,
  int *iqa,
  int *ipa) {

  size_t n = 0;

  /* Add pressure... */
  for (int ip = 0; ip < atm->np; ip++)
    if (atm->z[ip] >= ctl->retp_zmin && atm->z[ip] <= ctl->retp_zmax)
      atm2x_help(atm->p[ip], IDXP, ip, x, iqa, ipa, &n);

  /* Add temperature... */
  for (int ip = 0; ip < atm->np; ip++)
    if (atm->z[ip] >= ctl->rett_zmin && atm->z[ip] <= ctl->rett_zmax)
      atm2x_help(atm->t[ip], IDXT, ip, x, iqa, ipa, &n);

  /* Add volume mixing ratios... */
  for (int ig = 0; ig < ctl->ng; ig++)
    for (int ip = 0; ip < atm->np; ip++)
      if (atm->z[ip] >= ctl->retq_zmin[ig]
          && atm->z[ip] <= ctl->retq_zmax[ig])
        atm2x_help(atm->q[ig][ip], IDXQ(ig), ip, x, iqa, ipa, &n);

  /* Add extinction... */
  for (int iw = 0; iw < ctl->nw; iw++)
```

```
    for (int ip = 0; ip < atm->np; ip++)
      if (atm->z[ip] >= ctl->retk_zmin[iw]
          && atm->z[ip] <= ctl->retk_zmax[iw])
        atm2x_help(atm->k[iw][ip], IDXK(iw), ip, x, iqa, ipa, &n);

  /* Add cloud parameters... */
  if (ctl->ret_clz)
    atm2x_help(atm->clz, IDXCLZ, 0, x, iqa, ipa, &n);
  if (ctl->ret_cldz)
    atm2x_help(atm->cldz, IDXCLDZ, 0, x, iqa, ipa, &n);
  if (ctl->ret_clk)
    for (int icl = 0; icl < ctl->ncl; icl++)
      atm2x_help(atm->clk[icl], IDXCLK(icl), 0, x, iqa, ipa, &n);

  /* Add surface parameters... */
  if (ctl->ret_sfz)
    atm2x_help(atm->sfz, IDXSFZ, 0, x, iqa, ipa, &n);
  if (ctl->ret_sfp)
    atm2x_help(atm->sfp, IDXSFP, 0, x, iqa, ipa, &n);
  if (ctl->ret_sft)
    atm2x_help(atm->sft, IDXSFT, 0, x, iqa, ipa, &n);
  if (ctl->ret_sfeps)
    for (int isf = 0; isf < ctl->nsf; isf++)
      atm2x_help(atm->sfeps[isf], IDXSFEPS(isf), 0, x, iqa, ipa, &n);

  return n;
}

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

void atm2x_help(
  const double value,
  const int value_iqa,
  const int value_ip,
  gsl_vector *x,
  int *iqa,
  int *ipa,
  size_t *n) {

  /* Add element to state vector... */
  if (x != NULL)
    gsl_vector_set(x, *n, value);
  if (iqa != NULL)
    iqa[*n] = value_iqa;
  if (ipa != NULL)
    ipa[*n] = value_ip;
  (*n)++;
}

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

void cart2geo(
  const double *x,
  double *z,
  double *lon,
  double *lat) {

  const double radius = NORM(x);

  *lat = RAD2DEG(asin(x[2] / radius));
  *lon = RAD2DEG(atan2(x[1], x[0]));
  *z = radius - RE;
}

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

void climatology(
  const ctl_t *ctl,
  atm_t *atm) {

  static double z[121] = {
    0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19,
    20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37,
    38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55,
    56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73,
    74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91,
    92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107,
    108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 120
  };

  static double pre[121] = {
    1017, 901.083, 796.45, 702.227, 617.614, 541.644, 473.437, 412.288,
    357.603, 308.96, 265.994, 228.348, 195.619, 167.351, 143.039, 122.198,
    104.369, 89.141, 76.1528, 65.0804, 55.641, 47.591, 40.7233, 34.8637,
    29.8633, 25.5956, 21.9534, 18.8445, 16.1909, 13.9258, 11.9913,
    10.34, 8.92988, 7.72454, 6.6924, 5.80701, 5.04654, 4.39238, 3.82902,
    3.34337, 2.92413, 2.56128, 2.2464, 1.97258, 1.73384, 1.52519, 1.34242,
```

```
  1.18197, 1.04086, 0.916546, 0.806832, 0.709875, 0.624101, 0.548176,
  0.480974, 0.421507, 0.368904, 0.322408, 0.281386, 0.245249, 0.213465,
  0.185549, 0.161072, 0.139644, 0.120913, 0.104568, 0.0903249, 0.0779269,
  0.0671493, 0.0577962, 0.0496902, 0.0426736, 0.0366093, 0.0313743,
  0.0268598, 0.0229699, 0.0196206, 0.0167399, 0.0142646, 0.0121397,
  0.0103181, 0.00875775, 0.00742226, 0.00628076, 0.00530519, 0.00447183,
  0.00376124, 0.00315632, 0.00264248, 0.00220738, 0.00184003, 0.00153095,
  0.00127204, 0.00105608, 0.000876652, 0.00072798, 0.00060492,
  0.000503201, 0.000419226, 0.000349896, 0.000292659, 0.000245421,
  0.000206394, 0.000174125, 0.000147441, 0.000125333, 0.000106985,
  9.173e-05, 7.90172e-05, 6.84172e-05, 5.95574e-05, 5.21183e-05,
  4.58348e-05, 4.05127e-05, 3.59987e-05, 3.21583e-05, 2.88718e-05,
  2.60322e-05, 2.35687e-05, 2.14263e-05, 1.95489e-05
};

static double tem[121] = {
  285.14, 279.34, 273.91, 268.3, 263.24, 256.55, 250.2, 242.82, 236.17,
  229.87, 225.04, 221.19, 218.85, 217.19, 216.2, 215.68, 215.42, 215.55,
  215.92, 216.4, 216.93, 217.45, 218, 218.68, 219.39, 220.25, 221.3,
  222.41, 223.88, 225.42, 227.2, 229.52, 231.89, 234.51, 236.85, 239.42,
  241.94, 244.57, 247.36, 250.32, 253.34, 255.82, 258.27, 260.39,
  262.03, 263.45, 264.2, 264.78, 264.67, 264.38, 263.24, 262.03, 260.02,
  258.09, 255.63, 253.28, 250.43, 247.81, 245.26, 242.77, 240.38,
  237.94, 235.79, 233.53, 231.5, 229.53, 227.6, 225.62, 223.77, 222.06,
  220.33, 218.69, 217.18, 215.64, 214.13, 212.52, 210.86, 209.25,
  207.49, 205.81, 204.11, 202.22, 200.32, 198.39, 195.92, 193.46,
  190.94, 188.31, 185.82, 183.57, 181.43, 179.74, 178.64, 178.1, 178.25,
  178.7, 179.41, 180.67, 182.31, 184.18, 186.6, 189.53, 192.66, 196.54,
  201.13, 205.93, 211.73, 217.86, 225, 233.53, 242.57, 252.14, 261.48,
  272.97, 285.26, 299.12, 312.2, 324.17, 338.34, 352.56, 365.28
};

static double c2h2[121] = {
  1.352e-09, 2.83e-10, 1.269e-10, 6.926e-11, 4.346e-11, 2.909e-11,
  2.014e-11, 1.363e-11, 8.71e-12, 5.237e-12, 2.718e-12, 1.375e-12,
  5.786e-13, 2.16e-13, 7.317e-14, 2.551e-14, 1.055e-14, 4.758e-15,
  2.056e-15, 7.703e-16, 2.82e-16, 1.035e-16, 4.382e-17, 1.946e-17,
  9.638e-18, 5.2e-18, 2.811e-18, 1.494e-18, 7.925e-19, 4.213e-19,
  1.998e-19, 8.78e-20, 3.877e-20, 1.728e-20, 7.743e-21, 3.536e-21,
  1.623e-21, 7.508e-22, 3.508e-22, 1.65e-22, 7.837e-23, 3.733e-23,
  1.808e-23, 8.77e-24, 4.285e-24, 2.095e-24, 1.032e-24, 5.082e-25,
  2.506e-25, 1.236e-25, 6.088e-26, 2.996e-26, 1.465e-26, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
};

static double c2h6[121] = {
  2.667e-09, 2.02e-09, 1.658e-09, 1.404e-09, 1.234e-09, 1.109e-09,
  1.012e-09, 9.262e-10, 8.472e-10, 7.71e-10, 6.932e-10, 6.216e-10,
  5.503e-10, 4.87e-10, 4.342e-10, 3.861e-10, 3.347e-10, 2.772e-10,
  2.209e-10, 1.672e-10, 1.197e-10, 8.536e-11, 5.783e-11, 3.846e-11,
  2.495e-11, 1.592e-11, 1.017e-11, 6.327e-12, 3.895e-12, 2.403e-12,
  1.416e-12, 8.101e-13, 4.649e-13, 2.686e-13, 1.557e-13, 9.14e-14,
  5.386e-14, 3.19e-14, 1.903e-14, 1.14e-14, 6.875e-15, 4.154e-15,
  2.538e-15, 1.553e-15, 9.548e-16, 5.872e-16, 3.63e-16, 2.244e-16,
  1.388e-16, 8.587e-17, 5.308e-17, 3.279e-17, 2.017e-17, 1.238e-17,
  7.542e-18, 4.585e-18, 2.776e-18, 1.671e-18, 9.985e-19, 5.937e-19,
  3.518e-19, 2.07e-19, 1.215e-19, 7.06e-20, 4.097e-20, 2.37e-20,
  1.363e-20, 7.802e-21, 4.441e-21, 2.523e-21, 1.424e-21, 8.015e-22,
  4.497e-22, 2.505e-22, 1.391e-22, 7.691e-23, 4.238e-23, 2.331e-23,
  1.274e-23, 6.929e-24, 3.752e-24, 2.02e-24, 1.083e-24, 5.774e-25,
  3.041e-25, 1.593e-25, 8.308e-26, 4.299e-26, 2.195e-26, 1.112e-26,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0
};

static double ccl4[121] = {
  1.075e-10, 1.075e-10, 1.075e-10, 1.075e-10, 1.075e-10, 1.075e-10,
  1.075e-10, 1.075e-10, 1.075e-10, 1.06e-10, 1.024e-10, 9.69e-11,
  8.93e-11, 8.078e-11, 7.213e-11, 6.307e-11, 5.383e-11, 4.49e-11,
  3.609e-11, 2.705e-11, 1.935e-11, 1.385e-11, 8.35e-12, 5.485e-12,
  3.853e-12, 2.22e-12, 5.875e-13, 3.445e-13, 1.015e-13, 6.075e-14,
  4.383e-14, 2.692e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14,
  1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14,
  1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14,
  1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14,
  1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14,
  1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14,
  1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14,
  1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14,
  1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14,
  1e-14, 1e-14, 1e-14
};

static double ch4[121] = {
  1.864e-06, 1.835e-06, 1.819e-06, 1.805e-06, 1.796e-06, 1.788e-06,
```

```
  1.782e-06, 1.776e-06, 1.769e-06, 1.761e-06, 1.749e-06, 1.734e-06,
  1.716e-06, 1.692e-06, 1.654e-06, 1.61e-06, 1.567e-06, 1.502e-06,
  1.433e-06, 1.371e-06, 1.323e-06, 1.277e-06, 1.232e-06, 1.188e-06,
  1.147e-06, 1.108e-06, 1.07e-06, 1.027e-06, 9.854e-07, 9.416e-07,
  8.933e-07, 8.478e-07, 7.988e-07, 7.515e-07, 7.07e-07, 6.64e-07,
  6.239e-07, 5.864e-07, 5.512e-07, 5.184e-07, 4.87e-07, 4.571e-07,
  4.296e-07, 4.04e-07, 3.802e-07, 3.578e-07, 3.383e-07, 3.203e-07,
  3.032e-07, 2.889e-07, 2.76e-07, 2.635e-07, 2.519e-07, 2.409e-07,
  2.302e-07, 2.219e-07, 2.144e-07, 2.071e-07, 1.999e-07, 1.93e-07,
  1.862e-07, 1.795e-07, 1.731e-07, 1.668e-07, 1.607e-07, 1.548e-07,
  1.49e-07, 1.434e-07, 1.38e-07, 1.328e-07, 1.277e-07, 1.227e-07,
  1.18e-07, 1.134e-07, 1.089e-07, 1.046e-07, 1.004e-07, 9.635e-08,
  9.245e-08, 8.867e-08, 8.502e-08, 8.15e-08, 7.809e-08, 7.48e-08,
  7.159e-08, 6.849e-08, 6.55e-08, 6.262e-08, 5.98e-08, 5.708e-08,
  5.448e-08, 5.194e-08, 4.951e-08, 4.72e-08, 4.5e-08, 4.291e-08,
  4.093e-08, 3.905e-08, 3.729e-08, 3.563e-08, 3.408e-08, 3.265e-08,
  3.128e-08, 2.996e-08, 2.87e-08, 2.76e-08, 2.657e-08, 2.558e-08,
  2.467e-08, 2.385e-08, 2.307e-08, 2.234e-08, 2.168e-08, 2.108e-08,
  2.05e-08, 1.998e-08, 1.947e-08, 1.902e-08, 1.86e-08, 1.819e-08,
  1.782e-08
};

static double clo[121] = {
  7.419e-15, 1.061e-14, 1.518e-14, 2.195e-14, 3.175e-14, 4.666e-14,
  6.872e-14, 1.03e-13, 1.553e-13, 2.375e-13, 3.664e-13, 5.684e-13,
  8.915e-13, 1.402e-12, 2.269e-12, 4.125e-12, 7.501e-12, 1.257e-11,
  2.048e-11, 3.338e-11, 5.44e-11, 8.846e-11, 1.008e-10, 1.082e-10,
  1.157e-10, 1.232e-10, 1.312e-10, 1.539e-10, 1.822e-10, 2.118e-10,
  2.387e-10, 2.687e-10, 2.875e-10, 3.031e-10, 3.23e-10, 3.648e-10,
  4.117e-10, 4.477e-10, 4.633e-10, 4.794e-10, 4.95e-10, 5.104e-10,
  5.259e-10, 5.062e-10, 4.742e-10, 4.443e-10, 4.051e-10, 3.659e-10,
  3.305e-10, 2.911e-10, 2.54e-10, 2.215e-10, 1.927e-10, 1.675e-10,
  1.452e-10, 1.259e-10, 1.09e-10, 9.416e-11, 8.119e-11, 6.991e-11,
  6.015e-11, 5.163e-11, 4.43e-11, 3.789e-11, 3.24e-11, 2.769e-11,
  2.361e-11, 2.011e-11, 1.71e-11, 1.453e-11, 1.233e-11, 1.045e-11,
  8.851e-12, 7.48e-12, 6.316e-12, 5.326e-12, 4.487e-12, 3.778e-12,
  3.176e-12, 2.665e-12, 2.234e-12, 1.87e-12, 1.563e-12, 1.304e-12,
  1.085e-12, 9.007e-13, 7.468e-13, 6.179e-13, 5.092e-13, 4.188e-13,
  3.442e-13, 2.816e-13, 2.304e-13, 1.885e-13, 1.542e-13, 1.263e-13,
  1.035e-13, 8.5e-14, 7.004e-14, 5.783e-14, 4.795e-14, 4.007e-14,
  3.345e-14, 2.792e-14, 2.33e-14, 1.978e-14, 1.686e-14, 1.438e-14,
  1.234e-14, 1.07e-14, 9.312e-15, 8.131e-15, 7.164e-15, 6.367e-15,
  5.67e-15, 5.088e-15, 4.565e-15, 4.138e-15, 3.769e-15, 3.432e-15,
  3.148e-15
};

static double clono2[121] = {
  1.011e-13, 1.515e-13, 2.272e-13, 3.446e-13, 5.231e-13, 8.085e-13,
  1.253e-12, 1.979e-12, 3.149e-12, 5.092e-12, 8.312e-12, 1.366e-11,
  2.272e-11, 3.791e-11, 6.209e-11, 9.101e-11, 1.334e-10, 1.951e-10,
  2.853e-10, 3.94e-10, 4.771e-10, 5.771e-10, 6.675e-10, 7.665e-10,
  8.504e-10, 8.924e-10, 9.363e-10, 8.923e-10, 8.411e-10, 7.646e-10,
  6.525e-10, 5.576e-10, 4.398e-10, 3.403e-10, 2.612e-10, 1.915e-10,
  1.407e-10, 1.028e-10, 7.455e-11, 5.42e-11, 3.708e-11, 2.438e-11,
  1.618e-11, 1.075e-11, 7.17e-12, 4.784e-12, 3.205e-12, 2.147e-12,
  1.44e-12, 9.654e-13, 6.469e-13, 4.332e-13, 2.891e-13, 1.926e-13,
  1.274e-13, 8.422e-14, 5.547e-14, 3.636e-14, 2.368e-14, 1.536e-14,
  9.937e-15, 6.39e-15, 4.101e-15, 2.61e-15, 1.659e-15, 1.052e-15,
  6.638e-16, 4.172e-16, 2.61e-16, 1.63e-16, 1.013e-16, 6.275e-17,
  3.879e-17, 2.383e-17, 1.461e-17, 8.918e-18, 5.43e-18, 3.301e-18,
  1.997e-18, 1.203e-18, 7.216e-19, 4.311e-19, 2.564e-19, 1.519e-19,
  8.911e-20, 5.203e-20, 3.026e-20, 1.748e-20, 9.99e-21, 5.673e-21,
  3.215e-21, 1.799e-21, 1.006e-21, 5.628e-22, 3.146e-22, 1.766e-22,
  9.94e-23, 5.614e-23, 3.206e-23, 1.841e-23, 1.071e-23, 6.366e-24,
  3.776e-24, 2.238e-24, 1.326e-24, 8.253e-25, 5.201e-25, 3.279e-25,
  2.108e-25, 1.395e-25, 9.326e-26, 6.299e-26, 4.365e-26, 3.104e-26,
  2.219e-26, 1.621e-26, 1.185e-26, 8.92e-27, 6.804e-27, 5.191e-27,
  4.041e-27
};

static double co[121] = {
  1.907e-07, 1.553e-07, 1.362e-07, 1.216e-07, 1.114e-07, 1.036e-07,
  9.737e-08, 9.152e-08, 8.559e-08, 7.966e-08, 7.277e-08, 6.615e-08,
  5.884e-08, 5.22e-08, 4.699e-08, 4.284e-08, 3.776e-08, 3.274e-08,
  2.845e-08, 2.479e-08, 2.246e-08, 2.054e-08, 1.991e-08, 1.951e-08,
  1.94e-08, 2.009e-08, 2.1e-08, 2.201e-08, 2.322e-08, 2.45e-08,
  2.602e-08, 2.73e-08, 2.867e-08, 2.998e-08, 3.135e-08, 3.255e-08,
  3.352e-08, 3.426e-08, 3.484e-08, 3.53e-08, 3.593e-08, 3.671e-08,
  3.759e-08, 3.945e-08, 4.192e-08, 4.49e-08, 5.03e-08, 5.703e-08,
  6.538e-08, 7.878e-08, 9.644e-08, 1.196e-07, 1.498e-07, 1.904e-07,
  2.422e-07, 3.055e-07, 3.804e-07, 4.747e-07, 5.899e-07, 7.272e-07,
  8.91e-07, 1.071e-06, 1.296e-06, 1.546e-06, 1.823e-06, 2.135e-06,
  2.44e-06, 2.714e-06, 2.967e-06, 3.189e-06, 3.391e-06, 3.58e-06,
  3.773e-06, 4.022e-06, 4.346e-06, 4.749e-06, 5.199e-06, 5.668e-06,
  6.157e-06, 6.688e-06, 7.254e-06, 7.867e-06, 8.539e-06, 9.26e-06,
  1.009e-05, 1.119e-05, 1.228e-05, 1.365e-05, 1.506e-05, 1.641e-05,
  1.784e-05, 1.952e-05, 2.132e-05, 2.323e-05, 2.531e-05, 2.754e-05,
```

```
  3.047e-05, 3.459e-05, 3.922e-05, 4.439e-05, 4.825e-05, 5.077e-05,
  5.34e-05, 5.618e-05, 5.909e-05, 6.207e-05, 6.519e-05, 6.845e-05,
  6.819e-05, 6.726e-05, 6.622e-05, 6.512e-05, 6.671e-05, 6.862e-05,
  7.048e-05, 7.264e-05, 7.3e-05, 7.3e-05, 7.3e-05, 7.3e-05, 7.3e-05
};

static double cof2[121] = {
  7.5e-14, 1.055e-13, 1.485e-13, 2.111e-13, 3.001e-13, 4.333e-13,
  6.269e-13, 9.221e-13, 1.364e-12, 2.046e-12, 3.093e-12, 4.703e-12,
  7.225e-12, 1.113e-11, 1.66e-11, 2.088e-11, 2.626e-11, 3.433e-11,
  4.549e-11, 5.886e-11, 7.21e-11, 8.824e-11, 1.015e-10, 1.155e-10,
  1.288e-10, 1.388e-10, 1.497e-10, 1.554e-10, 1.606e-10, 1.639e-10,
  1.64e-10, 1.64e-10, 1.596e-10, 1.542e-10, 1.482e-10, 1.382e-10,
  1.289e-10, 1.198e-10, 1.109e-10, 1.026e-10, 9.484e-11, 8.75e-11,
  8.086e-11, 7.49e-11, 6.948e-11, 6.446e-11, 5.961e-11, 5.505e-11,
  5.085e-11, 4.586e-11, 4.1e-11, 3.665e-11, 3.235e-11, 2.842e-11,
  2.491e-11, 2.11e-11, 1.769e-11, 1.479e-11, 1.197e-11, 9.631e-12,
  7.74e-12, 6.201e-12, 4.963e-12, 3.956e-12, 3.151e-12, 2.507e-12,
  1.99e-12, 1.576e-12, 1.245e-12, 9.83e-13, 7.742e-13, 6.088e-13,
  4.782e-13, 3.745e-13, 2.929e-13, 2.286e-13, 1.782e-13, 1.388e-13,
  1.079e-13, 8.362e-14, 6.471e-14, 4.996e-14, 3.85e-14, 2.96e-14,
  2.265e-14, 1.729e-14, 1.317e-14, 9.998e-15, 7.549e-15, 5.683e-15,
  4.273e-15, 3.193e-15, 2.385e-15, 1.782e-15, 1.331e-15, 9.957e-16,
  7.461e-16, 5.601e-16, 4.228e-16, 3.201e-16, 2.438e-16, 1.878e-16,
  1.445e-16, 1.111e-16, 8.544e-17, 6.734e-17, 5.341e-17, 4.237e-17,
  3.394e-17, 2.759e-17, 2.254e-17, 1.851e-17, 1.54e-17, 1.297e-17,
  1.096e-17, 9.365e-18, 8e-18, 6.938e-18, 6.056e-18, 5.287e-18,
  4.662e-18
};

static double f11[121] = {
  2.65e-10, 2.65e-10, 2.65e-10, 2.65e-10, 2.65e-10, 2.65e-10, 2.65e-10,
  2.65e-10, 2.65e-10, 2.65e-10, 2.65e-10, 2.65e-10, 2.635e-10, 2.536e-10,
  2.44e-10, 2.348e-10, 2.258e-10, 2.153e-10, 2.046e-10, 1.929e-10,
  1.782e-10, 1.648e-10, 1.463e-10, 1.291e-10, 1.1e-10, 8.874e-11,
  7.165e-11, 5.201e-11, 3.744e-11, 2.577e-11, 1.64e-11, 1.048e-11,
  5.993e-12, 3.345e-12, 1.839e-12, 9.264e-13, 4.688e-13, 2.329e-13,
  1.129e-13, 5.505e-14, 2.825e-14, 1.492e-14, 7.997e-15, 5.384e-15,
  3.988e-15, 2.955e-15, 2.196e-15, 1.632e-15, 1.214e-15, 9.025e-16,
  6.708e-16, 4.984e-16, 3.693e-16, 2.733e-16, 2.013e-16, 1.481e-16,
  1.087e-16, 7.945e-17, 5.782e-17, 4.195e-17, 3.038e-17, 2.19e-17,
  1.577e-17, 1.128e-17, 8.063e-18, 5.753e-18, 4.09e-18, 2.899e-18,
  2.048e-18, 1.444e-18, 1.015e-18, 7.12e-19, 4.985e-19, 3.474e-19,
  2.417e-19, 1.677e-19, 1.161e-19, 8.029e-20, 5.533e-20, 3.799e-20,
  2.602e-20, 1.776e-20, 1.209e-20, 8.202e-21, 5.522e-21, 3.707e-21,
  2.48e-21, 1.652e-21, 1.091e-21, 7.174e-22, 4.709e-22, 3.063e-22,
  1.991e-22, 1.294e-22, 8.412e-23, 5.483e-23, 3.581e-23, 2.345e-23,
  1.548e-23, 1.027e-23, 6.869e-24, 4.673e-24, 3.173e-24, 2.153e-24,
  1.461e-24, 1.028e-24, 7.302e-25, 5.188e-25, 3.739e-25, 2.753e-25,
  2.043e-25, 1.528e-25, 1.164e-25, 9.041e-26, 7.051e-26, 5.587e-26,
  4.428e-26, 3.588e-26, 2.936e-26, 2.402e-26, 1.995e-26
};

static double f12[121] = {
  5.45e-10, 5.45e-10, 5.45e-10, 5.45e-10, 5.45e-10, 5.45e-10, 5.45e-10,
  5.45e-10, 5.45e-10, 5.45e-10, 5.45e-10, 5.45e-10, 5.429e-10, 5.291e-10,
  5.155e-10, 5.022e-10, 4.893e-10, 4.772e-10, 4.655e-10, 4.497e-10,
  4.249e-10, 4.015e-10, 3.632e-10, 3.261e-10, 2.858e-10, 2.408e-10,
  2.03e-10, 1.685e-10, 1.4e-10, 1.163e-10, 9.65e-11, 8.02e-11, 6.705e-11,
  5.624e-11, 4.764e-11, 4.249e-11, 3.792e-11, 3.315e-11, 2.819e-11,
  2.4e-11, 1.999e-11, 1.64e-11, 1.352e-11, 1.14e-11, 9.714e-12,
  8.28e-12, 7.176e-12, 6.251e-12, 5.446e-12, 4.72e-12, 4.081e-12,
  3.528e-12, 3.08e-12, 2.699e-12, 2.359e-12, 2.111e-12, 1.901e-12,
  1.709e-12, 1.534e-12, 1.376e-12, 1.233e-12, 1.103e-12, 9.869e-13,
  8.808e-13, 7.859e-13, 7.008e-13, 6.241e-13, 5.553e-13, 4.935e-13,
  4.383e-13, 3.889e-13, 3.447e-13, 3.054e-13, 2.702e-13, 2.389e-13,
  2.11e-13, 1.862e-13, 1.643e-13, 1.448e-13, 1.274e-13, 1.121e-13,
  9.844e-14, 8.638e-14, 7.572e-14, 6.62e-14, 5.782e-14, 5.045e-14,
  4.394e-14, 3.817e-14, 3.311e-14, 2.87e-14, 2.48e-14, 2.142e-14,
  1.851e-14, 1.599e-14, 1.383e-14, 1.196e-14, 1.036e-14, 9e-15,
  7.828e-15, 6.829e-15, 5.992e-15, 5.254e-15, 4.606e-15, 4.037e-15,
  3.583e-15, 3.19e-15, 2.841e-15, 2.542e-15, 2.291e-15, 2.07e-15,
  1.875e-15, 1.71e-15, 1.57e-15, 1.442e-15, 1.333e-15, 1.232e-15,
  1.147e-15, 1.071e-15, 1.001e-15, 9.396e-16
};

static double f14[121] = {
  9e-11, 9e-11, 9e-11, 9e-11, 9e-11, 9e-11, 9e-11, 9e-11, 9e-11, 9e-11,
  9e-11, 9e-11, 9e-11, 9e-11, 9e-11, 8.91e-11, 8.73e-11, 8.46e-11,
  8.19e-11, 7.92e-11, 7.74e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11,
  7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11,
  7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11,
  7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11,
  7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11,
  7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11,
  7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11,
  7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11,
```

```
  7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11,
  7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11,
  7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11,
  7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11,
  7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11,
  7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11,
  7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11
};

static double f22[121] = {
  1.4e-10, 1.4e-10, 1.4e-10, 1.4e-10, 1.4e-10, 1.4e-10, 1.4e-10,
  1.4e-10, 1.4e-10, 1.4e-10, 1.372e-10, 1.317e-10, 1.235e-10, 1.153e-10,
  1.075e-10, 1.002e-10, 9.332e-11, 8.738e-11, 8.194e-11, 7.7e-11,
  7.165e-11, 6.753e-11, 6.341e-11, 5.971e-11, 5.6e-11, 5.229e-11,
  4.859e-11, 4.488e-11, 4.118e-11, 3.83e-11, 3.568e-11, 3.308e-11,
  3.047e-11, 2.82e-11, 2.594e-11, 2.409e-11, 2.237e-11, 2.065e-11,
  1.894e-11, 1.771e-11, 1.647e-11, 1.532e-11, 1.416e-11, 1.332e-11,
  1.246e-11, 1.161e-11, 1.087e-11, 1.017e-11, 9.471e-12, 8.853e-12,
  8.235e-12, 7.741e-12, 7.247e-12, 6.836e-12, 6.506e-12, 6.176e-12,
  5.913e-12, 5.65e-12, 5.419e-12, 5.221e-12, 5.024e-12, 4.859e-12,
  4.694e-12, 4.546e-12, 4.414e-12, 4.282e-12, 4.15e-12, 4.019e-12,
  3.903e-12, 3.805e-12, 3.706e-12, 3.607e-12, 3.508e-12, 3.41e-12,
  3.31e-12, 3.212e-12, 3.129e-12, 3.047e-12, 2.964e-12, 2.882e-12,
  2.8e-12, 2.734e-12, 2.668e-12, 2.602e-12, 2.537e-12, 2.471e-12,
  2.421e-12, 2.372e-12, 2.322e-12, 2.273e-12, 2.224e-12, 2.182e-12,
  2.141e-12, 2.1e-12, 2.059e-12, 2.018e-12, 1.977e-12, 1.935e-12,
  1.894e-12, 1.853e-12, 1.812e-12, 1.77e-12, 1.73e-12, 1.688e-12,
  1.647e-12, 1.606e-12, 1.565e-12, 1.524e-12, 1.483e-12, 1.441e-12,
  1.4e-12, 1.359e-12, 1.317e-12, 1.276e-12, 1.235e-12, 1.194e-12,
  1.153e-12, 1.112e-12, 1.071e-12, 1.029e-12, 9.883e-13
};

static double h2o[121] = {
  0.01166, 0.008269, 0.005742, 0.003845, 0.00277, 0.001897, 0.001272,
  0.000827, 0.000539, 0.0003469, 0.0001579, 3.134e-05, 1.341e-05,
  6.764e-06, 4.498e-06, 3.703e-06, 3.724e-06, 3.899e-06, 4.002e-06,
  4.122e-06, 4.277e-06, 4.438e-06, 4.558e-06, 4.673e-06, 4.763e-06,
  4.809e-06, 4.856e-06, 4.936e-06, 5.021e-06, 5.114e-06, 5.222e-06,
  5.331e-06, 5.414e-06, 5.488e-06, 5.563e-06, 5.633e-06, 5.704e-06,
  5.767e-06, 5.819e-06, 5.872e-06, 5.914e-06, 5.949e-06, 5.984e-06,
  6.015e-06, 6.044e-06, 6.073e-06, 6.104e-06, 6.136e-06, 6.167e-06,
  6.189e-06, 6.208e-06, 6.226e-06, 6.212e-06, 6.185e-06, 6.158e-06,
  6.114e-06, 6.066e-06, 6.018e-06, 5.877e-06, 5.728e-06, 5.582e-06,
  5.437e-06, 5.296e-06, 5.156e-06, 5.02e-06, 4.886e-06, 4.754e-06,
  4.625e-06, 4.498e-06, 4.374e-06, 4.242e-06, 4.096e-06, 3.955e-06,
  3.817e-06, 3.683e-06, 3.491e-06, 3.204e-06, 2.94e-06, 2.696e-06,
  2.47e-06, 2.252e-06, 2.019e-06, 1.808e-06, 1.618e-06, 1.445e-06,
  1.285e-06, 1.105e-06, 9.489e-07, 8.121e-07, 6.938e-07, 5.924e-07,
  5.04e-07, 4.288e-07, 3.648e-07, 3.103e-07, 2.642e-07, 2.252e-07,
  1.921e-07, 1.643e-07, 1.408e-07, 1.211e-07, 1.048e-07, 9.063e-08,
  7.835e-08, 6.774e-08, 5.936e-08, 5.221e-08, 4.592e-08, 4.061e-08,
  3.62e-08, 3.236e-08, 2.902e-08, 2.62e-08, 2.383e-08, 2.171e-08,
  1.989e-08, 1.823e-08, 1.684e-08, 1.562e-08, 1.449e-08, 1.351e-08
};

static double h2o2[121] = {
  1.779e-10, 7.938e-10, 8.953e-10, 8.032e-10, 6.564e-10, 5.159e-10,
  4.003e-10, 3.026e-10, 2.222e-10, 1.58e-10, 1.044e-10, 6.605e-11,
  3.413e-11, 1.453e-11, 1.062e-11, 1.009e-11, 9.597e-12, 1.175e-11,
  1.572e-11, 2.091e-11, 2.746e-11, 3.603e-11, 4.791e-11, 6.387e-11,
  8.239e-11, 1.007e-10, 1.23e-10, 1.363e-10, 1.489e-10, 1.585e-10,
  1.608e-10, 1.632e-10, 1.576e-10, 1.502e-10, 1.423e-10, 1.302e-10,
  1.192e-10, 1.085e-10, 9.795e-11, 8.854e-11, 8.057e-11, 7.36e-11,
  6.736e-11, 6.362e-11, 6.087e-11, 5.825e-11, 5.623e-11, 5.443e-11,
  5.27e-11, 5.098e-11, 4.931e-11, 4.769e-11, 4.611e-11, 4.458e-11,
  4.308e-11, 4.102e-11, 3.887e-11, 3.682e-11, 3.521e-11, 3.369e-11,
  3.224e-11, 3.082e-11, 2.946e-11, 2.814e-11, 2.687e-11, 2.566e-11,
  2.449e-11, 2.336e-11, 2.227e-11, 2.123e-11, 2.023e-11, 1.927e-11,
  1.835e-11, 1.746e-11, 1.661e-11, 1.58e-11, 1.502e-11, 1.428e-11,
  1.357e-11, 1.289e-11, 1.224e-11, 1.161e-11, 1.102e-11, 1.045e-11,
  9.895e-12, 9.369e-12, 8.866e-12, 8.386e-12, 7.922e-12, 7.479e-12,
  7.06e-12, 6.656e-12, 6.274e-12, 5.914e-12, 5.575e-12, 5.257e-12,
  4.959e-12, 4.679e-12, 4.42e-12, 4.178e-12, 3.954e-12, 3.75e-12,
  3.557e-12, 3.372e-12, 3.198e-12, 3.047e-12, 2.908e-12, 2.775e-12,
  2.653e-12, 2.544e-12, 2.442e-12, 2.346e-12, 2.26e-12, 2.183e-12,
  2.11e-12, 2.044e-12, 1.98e-12, 1.924e-12, 1.871e-12, 1.821e-12,
  1.775e-12
};

static double hcn[121] = {
  5.5e-10, 5.5e-10, 5.5e-10, 5.5e-10, 5.5e-10, 5.5e-10, 5.5e-10,
  5.5e-10, 5.5e-10, 5.5e-10, 5.5e-10, 5.498e-10, 5.495e-10, 5.493e-10,
  5.49e-10, 5.488e-10, 4.717e-10, 3.946e-10, 3.174e-10, 2.4e-10,
  1.626e-10, 1.619e-10, 1.612e-10, 1.602e-10, 1.593e-10, 1.582e-10,
  1.572e-10, 1.56e-10, 1.549e-10, 1.539e-10, 1.53e-10, 1.519e-10,
  1.506e-10, 1.487e-10, 1.467e-10, 1.449e-10, 1.43e-10, 1.413e-10,
  1.397e-10, 1.382e-10, 1.368e-10, 1.354e-10, 1.337e-10, 1.315e-10,
```

```
  1.292e-10, 1.267e-10, 1.241e-10, 1.215e-10, 1.19e-10, 1.165e-10,
  1.141e-10, 1.118e-10, 1.096e-10, 1.072e-10, 1.047e-10, 1.021e-10,
  9.968e-11, 9.739e-11, 9.539e-11, 9.339e-11, 9.135e-11, 8.898e-11,
  8.664e-11, 8.439e-11, 8.249e-11, 8.075e-11, 7.904e-11, 7.735e-11,
  7.565e-11, 7.399e-11, 7.245e-11, 7.109e-11, 6.982e-11, 6.863e-11,
  6.755e-11, 6.657e-11, 6.587e-11, 6.527e-11, 6.476e-11, 6.428e-11,
  6.382e-11, 6.343e-11, 6.307e-11, 6.272e-11, 6.238e-11, 6.205e-11,
  6.17e-11, 6.137e-11, 6.102e-11, 6.072e-11, 6.046e-11, 6.03e-11,
  6.018e-11, 6.01e-11, 6.001e-11, 5.992e-11, 5.984e-11, 5.975e-11,
  5.967e-11, 5.958e-11, 5.95e-11, 5.941e-11, 5.933e-11, 5.925e-11,
  5.916e-11, 5.908e-11, 5.899e-11, 5.891e-11, 5.883e-11, 5.874e-11,
  5.866e-11, 5.858e-11, 5.85e-11, 5.841e-11, 5.833e-11, 5.825e-11,
  5.817e-11, 5.808e-11, 5.8e-11, 5.792e-11, 5.784e-11
};

static double hno3[121] = {
  1.809e-10, 7.234e-10, 5.899e-10, 4.342e-10, 3.277e-10, 2.661e-10,
  2.35e-10, 2.267e-10, 2.389e-10, 2.651e-10, 3.255e-10, 4.099e-10,
  5.42e-10, 6.978e-10, 8.807e-10, 1.112e-09, 1.405e-09, 2.04e-09,
  3.111e-09, 4.5e-09, 5.762e-09, 7.37e-09, 7.852e-09, 8.109e-09,
  8.067e-09, 7.554e-09, 7.076e-09, 6.268e-09, 5.524e-09, 4.749e-09,
  3.909e-09, 3.223e-09, 2.517e-09, 1.942e-09, 1.493e-09, 1.122e-09,
  8.449e-10, 6.361e-10, 4.787e-10, 3.611e-10, 2.804e-10, 2.215e-10,
  1.758e-10, 1.441e-10, 1.197e-10, 9.953e-11, 8.505e-11, 7.334e-11,
  6.325e-11, 5.625e-11, 5.058e-11, 4.548e-11, 4.122e-11, 3.748e-11,
  3.402e-11, 3.088e-11, 2.8e-11, 2.536e-11, 2.293e-11, 2.072e-11,
  1.871e-11, 1.687e-11, 1.52e-11, 1.368e-11, 1.23e-11, 1.105e-11,
  9.922e-12, 8.898e-12, 7.972e-12, 7.139e-12, 6.385e-12, 5.708e-12,
  5.099e-12, 4.549e-12, 4.056e-12, 3.613e-12, 3.216e-12, 2.862e-12,
  2.544e-12, 2.259e-12, 2.004e-12, 1.776e-12, 1.572e-12, 1.391e-12,
  1.227e-12, 1.082e-12, 9.528e-13, 8.379e-13, 7.349e-13, 6.436e-13,
  5.634e-13, 4.917e-13, 4.291e-13, 3.745e-13, 3.267e-13, 2.854e-13,
  2.494e-13, 2.181e-13, 1.913e-13, 1.68e-13, 1.479e-13, 1.31e-13,
  1.159e-13, 1.025e-13, 9.067e-14, 8.113e-14, 7.281e-14, 6.535e-14,
  5.892e-14, 5.348e-14, 4.867e-14, 4.439e-14, 4.073e-14, 3.76e-14,
  3.476e-14, 3.229e-14, 3e-14, 2.807e-14, 2.635e-14, 2.473e-14,
  2.332e-14
};

static double hno4[121] = {
  6.118e-12, 3.594e-12, 2.807e-12, 3.04e-12, 4.458e-12, 7.986e-12,
  1.509e-11, 2.661e-11, 3.738e-11, 4.652e-11, 4.429e-11, 3.992e-11,
  3.347e-11, 3.005e-11, 3.173e-11, 4.055e-11, 5.812e-11, 8.489e-11,
  1.19e-10, 1.482e-10, 1.766e-10, 2.103e-10, 2.35e-10, 2.598e-10,
  2.801e-10, 2.899e-10, 3e-10, 2.817e-10, 2.617e-10, 2.332e-10,
  1.933e-10, 1.605e-10, 1.232e-10, 9.285e-11, 6.941e-11, 4.951e-11,
  3.539e-11, 2.402e-11, 1.522e-11, 9.676e-12, 6.056e-12, 3.745e-12,
  2.34e-12, 1.463e-12, 9.186e-13, 5.769e-13, 3.322e-13, 1.853e-13,
  1.035e-13, 7.173e-14, 5.382e-14, 4.036e-14, 3.401e-14, 2.997e-14,
  2.635e-14, 2.316e-14, 2.034e-14, 1.783e-14, 1.56e-14, 1.363e-14,
  1.19e-14, 1.037e-14, 9.032e-15, 7.846e-15, 6.813e-15, 5.912e-15,
  5.121e-15, 4.431e-15, 3.829e-15, 3.306e-15, 2.851e-15, 2.456e-15,
  2.114e-15, 1.816e-15, 1.559e-15, 1.337e-15, 1.146e-15, 9.811e-16,
  8.389e-16, 7.162e-16, 6.109e-16, 5.203e-16, 4.425e-16, 3.76e-16,
  3.184e-16, 2.692e-16, 2.274e-16, 1.917e-16, 1.61e-16, 1.35e-16,
  1.131e-16, 9.437e-17, 7.874e-17, 6.57e-17, 5.481e-17, 4.579e-17,
  3.828e-17, 3.204e-17, 2.691e-17, 2.264e-17, 1.912e-17, 1.626e-17,
  1.382e-17, 1.174e-17, 9.972e-18, 8.603e-18, 7.45e-18, 6.453e-18,
  5.623e-18, 4.944e-18, 4.361e-18, 3.859e-18, 3.443e-18, 3.096e-18,
  2.788e-18, 2.528e-18, 2.293e-18, 2.099e-18, 1.929e-18, 1.773e-18,
  1.64e-18
};

static double hocl[121] = {
  1.056e-12, 1.194e-12, 1.35e-12, 1.531e-12, 1.737e-12, 1.982e-12,
  2.263e-12, 2.599e-12, 2.991e-12, 3.459e-12, 4.012e-12, 4.662e-12,
  5.438e-12, 6.35e-12, 7.425e-12, 8.686e-12, 1.016e-11, 1.188e-11,
  1.389e-11, 1.659e-11, 2.087e-11, 2.621e-11, 3.265e-11, 4.064e-11,
  4.859e-11, 5.441e-11, 6.09e-11, 6.373e-11, 6.611e-11, 6.94e-11,
  7.44e-11, 7.97e-11, 8.775e-11, 9.722e-11, 1.064e-10, 1.089e-10,
  1.114e-10, 1.106e-10, 1.053e-10, 1.004e-10, 9.006e-11, 7.778e-11,
  6.739e-11, 5.636e-11, 4.655e-11, 3.845e-11, 3.042e-11, 2.368e-11,
  1.845e-11, 1.442e-11, 1.127e-11, 8.814e-12, 6.544e-12, 4.763e-12,
  3.449e-12, 2.612e-12, 1.999e-12, 1.526e-12, 1.16e-12, 8.793e-13,
  6.655e-13, 5.017e-13, 3.778e-13, 2.829e-13, 2.117e-13, 1.582e-13,
  1.178e-13, 8.755e-14, 6.486e-14, 4.799e-14, 3.54e-14, 2.606e-14,
  1.916e-14, 1.403e-14, 1.026e-14, 7.48e-15, 5.446e-15, 3.961e-15,
  2.872e-15, 2.076e-15, 1.498e-15, 1.077e-15, 7.726e-16, 5.528e-16,
  3.929e-16, 2.785e-16, 1.969e-16, 1.386e-16, 9.69e-17, 6.747e-17,
  4.692e-17, 3.236e-17, 2.232e-17, 1.539e-17, 1.061e-17, 7.332e-18,
  5.076e-18, 3.522e-18, 2.461e-18, 1.726e-18, 1.22e-18, 8.75e-19,
  6.264e-19, 4.482e-19, 3.207e-19, 2.368e-19, 1.762e-19, 1.312e-19,
  9.891e-20, 7.595e-20, 5.87e-20, 4.567e-20, 3.612e-20, 2.904e-20,
  2.343e-20, 1.917e-20, 1.568e-20, 1.308e-20, 1.1e-20, 9.25e-21,
  7.881e-21
};
```

```
static double n2o[121] = {
  3.17e-07, 3.17e-07, 3.17e-07, 3.17e-07, 3.17e-07, 3.17e-07, 3.17e-07,
  3.17e-07, 3.17e-07, 3.17e-07, 3.124e-07, 3.077e-07, 3.03e-07,
  2.984e-07, 2.938e-07, 2.892e-07, 2.847e-07, 2.779e-07, 2.705e-07,
  2.631e-07, 2.557e-07, 2.484e-07, 2.345e-07, 2.201e-07, 2.01e-07,
  1.754e-07, 1.532e-07, 1.329e-07, 1.154e-07, 1.003e-07, 8.735e-08,
  7.617e-08, 6.512e-08, 5.547e-08, 4.709e-08, 3.915e-08, 3.259e-08,
  2.738e-08, 2.327e-08, 1.98e-08, 1.711e-08, 1.493e-08, 1.306e-08,
  1.165e-08, 1.049e-08, 9.439e-09, 8.375e-09, 7.391e-09, 6.525e-09,
  5.759e-09, 5.083e-09, 4.485e-09, 3.953e-09, 3.601e-09, 3.27e-09,
  2.975e-09, 2.757e-09, 2.556e-09, 2.37e-09, 2.195e-09, 2.032e-09,
  1.912e-09, 1.79e-09, 1.679e-09, 1.572e-09, 1.482e-09, 1.402e-09,
  1.326e-09, 1.254e-09, 1.187e-09, 1.127e-09, 1.071e-09, 1.02e-09,
  9.673e-10, 9.193e-10, 8.752e-10, 8.379e-10, 8.017e-10, 7.66e-10,
  7.319e-10, 7.004e-10, 6.721e-10, 6.459e-10, 6.199e-10, 5.942e-10,
  5.703e-10, 5.488e-10, 5.283e-10, 5.082e-10, 4.877e-10, 4.696e-10,
  4.52e-10, 4.355e-10, 4.198e-10, 4.039e-10, 3.888e-10, 3.754e-10,
  3.624e-10, 3.499e-10, 3.381e-10, 3.267e-10, 3.163e-10, 3.058e-10,
  2.959e-10, 2.864e-10, 2.77e-10, 2.686e-10, 2.604e-10, 2.534e-10,
  2.462e-10, 2.386e-10, 2.318e-10, 2.247e-10, 2.189e-10, 2.133e-10,
  2.071e-10, 2.014e-10, 1.955e-10, 1.908e-10, 1.86e-10, 1.817e-10
};

static double n2o5[121] = {
  1.231e-11, 3.035e-12, 1.702e-12, 9.877e-13, 8.081e-13, 9.039e-13,
  1.169e-12, 1.474e-12, 1.651e-12, 1.795e-12, 1.998e-12, 2.543e-12,
  4.398e-12, 7.698e-12, 1.28e-11, 2.131e-11, 3.548e-11, 5.894e-11,
  7.645e-11, 1.089e-10, 1.391e-10, 1.886e-10, 2.386e-10, 2.986e-10,
  3.487e-10, 3.994e-10, 4.5e-10, 4.6e-10, 4.591e-10, 4.1e-10, 3.488e-10,
  2.846e-10, 2.287e-10, 1.696e-10, 1.011e-10, 6.428e-11, 4.324e-11,
  2.225e-11, 6.214e-12, 3.608e-12, 8.793e-13, 4.491e-13, 1.04e-13,
  6.1e-14, 3.436e-14, 6.671e-15, 1.171e-15, 5.848e-16, 1.212e-16,
  1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16,
  1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16,
  1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16,
  1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16,
  1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16,
  1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16,
  1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16,
  1e-16, 1e-16
};

static double nh3[121] = {
  1e-10, 1e-10, 1e-10, 1e-10, 1e-10, 1e-10, 1e-10, 1e-10, 1e-10, 1e-10,
  1e-10, 1e-10, 1e-10, 1e-10, 9.444e-11, 8.488e-11, 7.241e-11, 5.785e-11,
  4.178e-11, 3.018e-11, 2.18e-11, 1.574e-11, 1.137e-11, 8.211e-12,
  5.973e-12, 4.327e-12, 3.118e-12, 2.234e-12, 1.573e-12, 1.04e-12,
  6.762e-13, 4.202e-13, 2.406e-13, 1.335e-13, 6.938e-14, 3.105e-14,
  1.609e-14, 1.033e-14, 6.432e-15, 4.031e-15, 2.555e-15, 1.656e-15,
  1.115e-15, 7.904e-16, 5.63e-16, 4.048e-16, 2.876e-16, 2.004e-16,
  1.356e-16, 9.237e-17, 6.235e-17, 4.223e-17, 3.009e-17, 2.328e-17,
  2.002e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17,
  1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17,
  1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17,
  1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17,
  1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17,
  1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17,
  1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17,
  1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17,
  1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17,
  1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17,
  1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17,
  1.914e-17
};

static double no[121] = {
  2.586e-10, 4.143e-11, 1.566e-11, 9.591e-12, 8.088e-12, 8.462e-12,
  1.013e-11, 1.328e-11, 1.855e-11, 2.678e-11, 3.926e-11, 5.464e-11,
  7.012e-11, 8.912e-11, 1.127e-10, 1.347e-10, 1.498e-10, 1.544e-10,
  1.602e-10, 1.824e-10, 2.078e-10, 2.366e-10, 2.691e-10, 5.141e-10,
  8.259e-10, 1.254e-09, 1.849e-09, 2.473e-09, 3.294e-09, 4.16e-09,
  5.095e-09, 6.11e-09, 6.93e-09, 7.888e-09, 8.903e-09, 9.713e-09,
  1.052e-08, 1.115e-08, 1.173e-08, 1.21e-08, 1.228e-08, 1.239e-08,
  1.231e-08, 1.213e-08, 1.192e-08, 1.138e-08, 1.085e-08, 1.008e-08,
  9.224e-09, 8.389e-09, 7.262e-09, 6.278e-09, 5.335e-09, 4.388e-09,
  3.589e-09, 2.761e-09, 2.129e-09, 1.633e-09, 1.243e-09, 9.681e-10,
  8.355e-10, 7.665e-10, 7.442e-10, 8.584e-10, 9.732e-10, 1.063e-09,
  1.163e-09, 1.286e-09, 1.472e-09, 1.707e-09, 2.032e-09, 2.474e-09,
  2.977e-09, 3.506e-09, 4.102e-09, 5.013e-09, 6.493e-09, 8.414e-09,
  1.077e-08, 1.367e-08, 1.777e-08, 2.625e-08, 3.926e-08, 5.545e-08,
  7.195e-08, 9.464e-08, 1.404e-07, 2.183e-07, 3.329e-07, 4.535e-07,
  6.158e-07, 8.187e-07, 1.075e-06, 1.422e-06, 1.979e-06, 2.71e-06,
  3.58e-06, 4.573e-06, 5.951e-06, 7.999e-06, 1.072e-05, 1.372e-05,
  1.697e-05, 2.112e-05, 2.643e-05, 3.288e-05, 3.994e-05, 4.794e-05,
  5.606e-05, 6.383e-05, 7.286e-05, 8.156e-05, 8.883e-05, 9.469e-05,
  9.848e-05, 0.0001023, 0.0001066, 0.0001115, 0.0001145, 0.0001142,
  0.0001133
```

```
};

static double no2[121] = {
  3.036e-09, 2.945e-10, 9.982e-11, 5.069e-11, 3.485e-11, 2.982e-11,
  2.947e-11, 3.164e-11, 3.714e-11, 4.586e-11, 6.164e-11, 8.041e-11,
  9.982e-11, 1.283e-10, 1.73e-10, 2.56e-10, 3.909e-10, 5.959e-10,
  9.081e-10, 1.384e-09, 1.788e-09, 2.189e-09, 2.686e-09, 3.091e-09,
  3.49e-09, 3.796e-09, 4.2e-09, 5.103e-09, 6.005e-09, 6.3e-09, 6.706e-09,
  7.07e-09, 7.434e-09, 7.663e-09, 7.788e-09, 7.8e-09, 7.597e-09,
  7.482e-09, 7.227e-09, 6.403e-09, 5.585e-09, 4.606e-09, 3.703e-09,
  2.984e-09, 2.183e-09, 1.48e-09, 8.441e-10, 5.994e-10, 3.799e-10,
  2.751e-10, 1.927e-10, 1.507e-10, 1.102e-10, 6.971e-11, 5.839e-11,
  3.904e-11, 3.087e-11, 2.176e-11, 1.464e-11, 1.209e-11, 8.497e-12,
  6.477e-12, 4.371e-12, 2.914e-12, 2.424e-12, 1.753e-12, 1.35e-12,
  9.417e-13, 6.622e-13, 5.148e-13, 3.841e-13, 3.446e-13, 3.01e-13,
  2.551e-13, 2.151e-13, 1.829e-13, 1.64e-13, 1.475e-13, 1.352e-13,
  1.155e-13, 9.963e-14, 9.771e-14, 9.577e-14, 9.384e-14, 9.186e-14,
  9e-14, 9e-14, 9e-14, 9e-14, 9e-14, 9e-14, 9e-14, 9e-14, 9e-14, 9e-14,
  9e-14, 9e-14, 9e-14, 9e-14, 9e-14, 9e-14, 9e-14, 9e-14, 9e-14, 9e-14,
  9e-14, 9e-14, 9e-14, 9e-14, 9e-14, 9e-14, 9e-14, 9e-14, 9e-14, 9e-14,
  9e-14, 9e-14, 9e-14, 9e-14, 9e-14, 9e-14
};

static double o3[121] = {
  2.218e-08, 3.394e-08, 3.869e-08, 4.219e-08, 4.501e-08, 4.778e-08,
  5.067e-08, 5.402e-08, 5.872e-08, 6.521e-08, 7.709e-08, 9.461e-08,
  1.269e-07, 1.853e-07, 2.723e-07, 3.964e-07, 5.773e-07, 8.2e-07,
  1.155e-06, 1.59e-06, 2.076e-06, 2.706e-06, 3.249e-06, 3.848e-06,
  4.459e-06, 4.986e-06, 5.573e-06, 5.958e-06, 6.328e-06, 6.661e-06,
  6.9e-06, 7.146e-06, 7.276e-06, 7.374e-06, 7.447e-06, 7.383e-06,
  7.321e-06, 7.161e-06, 6.879e-06, 6.611e-06, 6.216e-06, 5.765e-06,
  5.355e-06, 4.905e-06, 4.471e-06, 4.075e-06, 3.728e-06, 3.413e-06,
  3.125e-06, 2.856e-06, 2.607e-06, 2.379e-06, 2.17e-06, 1.978e-06,
  1.8e-06, 1.646e-06, 1.506e-06, 1.376e-06, 1.233e-06, 1.102e-06,
  9.839e-07, 8.771e-07, 7.814e-07, 6.947e-07, 6.102e-07, 5.228e-07,
  4.509e-07, 3.922e-07, 3.501e-07, 3.183e-07, 2.909e-07, 2.686e-07,
  2.476e-07, 2.284e-07, 2.109e-07, 2.003e-07, 2.013e-07, 2.022e-07,
  2.032e-07, 2.042e-07, 2.097e-07, 2.361e-07, 2.656e-07, 2.989e-07,
  3.37e-07, 3.826e-07, 4.489e-07, 5.26e-07, 6.189e-07, 7.312e-07,
  8.496e-07, 8.444e-07, 8.392e-07, 8.339e-07, 8.286e-07, 8.234e-07,
  8.181e-07, 8.129e-07, 8.077e-07, 8.026e-07, 6.918e-07, 5.176e-07,
  3.865e-07, 2.885e-07, 2.156e-07, 1.619e-07, 1.219e-07, 9.161e-08,
  6.972e-08, 5.399e-08, 3.498e-08, 2.111e-08, 1.322e-08, 8.482e-09,
  5.527e-09, 3.423e-09, 2.071e-09, 1.314e-09, 8.529e-10, 5.503e-10,
  3.665e-10
};

static double ocs[121] = {
  6e-10, 6e-10, 6e-10, 6e-10, 6e-10, 6e-10, 6e-10, 6e-10, 6e-10, 5.997e-10,
  5.989e-10, 5.881e-10, 5.765e-10, 5.433e-10, 5.074e-10, 4.567e-10,
  4.067e-10, 3.601e-10, 3.093e-10, 2.619e-10, 2.232e-10, 1.805e-10,
  1.46e-10, 1.187e-10, 8.03e-11, 5.435e-11, 3.686e-11, 2.217e-11,
  1.341e-11, 8.756e-12, 4.511e-12, 2.37e-12, 1.264e-12, 8.28e-13,
  5.263e-13, 3.209e-13, 1.717e-13, 9.068e-14, 4.709e-14, 2.389e-14,
  1.236e-14, 1.127e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14,
  1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14,
  1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14,
  1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14,
  1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14,
  1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14,
  1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14,
  1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14,
  1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14,
  1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14,
  1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14,
  1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14,
  1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14,
  1.091e-14, 1.091e-14, 1.091e-14
};

static double sf6[121] = {
  4.103e-12, 4.103e-12, 4.103e-12, 4.103e-12, 4.103e-12, 4.103e-12,
  4.103e-12, 4.103e-12, 4.103e-12, 4.087e-12, 4.064e-12, 4.023e-12,
  3.988e-12, 3.941e-12, 3.884e-12, 3.755e-12, 3.622e-12, 3.484e-12,
  3.32e-12, 3.144e-12, 2.978e-12, 2.811e-12, 2.653e-12, 2.489e-12,
  2.332e-12, 2.199e-12, 2.089e-12, 2.013e-12, 1.953e-12, 1.898e-12,
  1.859e-12, 1.826e-12, 1.798e-12, 1.776e-12, 1.757e-12, 1.742e-12,
  1.728e-12, 1.717e-12, 1.707e-12, 1.698e-12, 1.691e-12, 1.685e-12,
  1.679e-12, 1.675e-12, 1.671e-12, 1.668e-12, 1.665e-12, 1.663e-12,
  1.661e-12, 1.659e-12, 1.658e-12, 1.657e-12, 1.656e-12, 1.655e-12,
  1.654e-12, 1.653e-12, 1.653e-12, 1.652e-12, 1.652e-12, 1.652e-12,
  1.651e-12, 1.651e-12, 1.651e-12, 1.651e-12, 1.651e-12, 1.651e-12,
  1.651e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12,
  1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12,
  1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12,
  1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12,
  1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12,
```

```
    1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12,
    1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12,
    1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12
  };

  static double so2[121] = {
    1e-10, 1e-10, 1e-10, 1e-10, 1e-10, 1e-10, 1e-10, 1e-10, 1e-10, 1e-10,
    1e-10, 1e-10, 9.867e-11, 9.537e-11, 9e-11, 8.404e-11, 7.799e-11,
    7.205e-11, 6.616e-11, 6.036e-11, 5.475e-11, 5.007e-11, 4.638e-11,
    4.346e-11, 4.055e-11, 3.763e-11, 3.471e-11, 3.186e-11, 2.905e-11,
    2.631e-11, 2.358e-11, 2.415e-11, 2.949e-11, 3.952e-11, 5.155e-11,
    6.76e-11, 8.741e-11, 1.099e-10, 1.278e-10, 1.414e-10, 1.512e-10,
    1.607e-10, 1.699e-10, 1.774e-10, 1.832e-10, 1.871e-10, 1.907e-10,
    1.943e-10, 1.974e-10, 1.993e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10,
    2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10,
    2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10,
    2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10,
    2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10,
    2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10,
    2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10,
    2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10
  };

  static int ig_co2 = -999;

  double *q[NG] = { NULL };

  /* Find emitter index of CO2... */
  if (ig_co2 == -999)
    ig_co2 = find_emitter(ctl, "CO2");

  /* Identify variable... */
  for (int ig = 0; ig < ctl->ng; ig++) {
    q[ig] = NULL;
    if (strcasecmp(ctl->emitter[ig], "C2H2") == 0)
      q[ig] = c2h2;
    if (strcasecmp(ctl->emitter[ig], "C2H6") == 0)
      q[ig] = c2h6;
    if (strcasecmp(ctl->emitter[ig], "CCl4") == 0)
      q[ig] = ccl4;
    if (strcasecmp(ctl->emitter[ig], "CH4") == 0)
      q[ig] = ch4;
    if (strcasecmp(ctl->emitter[ig], "ClO") == 0)
      q[ig] = clo;
    if (strcasecmp(ctl->emitter[ig], "ClONO2") == 0)
      q[ig] = clono2;
    if (strcasecmp(ctl->emitter[ig], "CO") == 0)
      q[ig] = co;
    if (strcasecmp(ctl->emitter[ig], "COF2") == 0)
      q[ig] = cof2;
    if (strcasecmp(ctl->emitter[ig], "F11") == 0)
      q[ig] = f11;
    if (strcasecmp(ctl->emitter[ig], "F12") == 0)
      q[ig] = f12;
    if (strcasecmp(ctl->emitter[ig], "F14") == 0)
      q[ig] = f14;
    if (strcasecmp(ctl->emitter[ig], "F22") == 0)
      q[ig] = f22;
    if (strcasecmp(ctl->emitter[ig], "H2O") == 0)
      q[ig] = h2o;
    if (strcasecmp(ctl->emitter[ig], "H2O2") == 0)
      q[ig] = h2o2;
    if (strcasecmp(ctl->emitter[ig], "HCN") == 0)
      q[ig] = hcn;
    if (strcasecmp(ctl->emitter[ig], "HNO3") == 0)
      q[ig] = hno3;
    if (strcasecmp(ctl->emitter[ig], "HNO4") == 0)
      q[ig] = hno4;
    if (strcasecmp(ctl->emitter[ig], "HOCl") == 0)
      q[ig] = hocl;
    if (strcasecmp(ctl->emitter[ig], "N2O") == 0)
      q[ig] = n2o;
    if (strcasecmp(ctl->emitter[ig], "N2O5") == 0)
      q[ig] = n2o5;
    if (strcasecmp(ctl->emitter[ig], "NH3") == 0)
      q[ig] = nh3;
    if (strcasecmp(ctl->emitter[ig], "NO") == 0)
      q[ig] = no;
    if (strcasecmp(ctl->emitter[ig], "NO2") == 0)
      q[ig] = no2;
    if (strcasecmp(ctl->emitter[ig], "O3") == 0)
      q[ig] = o3;
    if (strcasecmp(ctl->emitter[ig], "OCS") == 0)
      q[ig] = ocs;
    if (strcasecmp(ctl->emitter[ig], "SF6") == 0)
      q[ig] = sf6;
    if (strcasecmp(ctl->emitter[ig], "SO2") == 0)
```

```
      q[ig] = so2;
  }

  /* Loop over atmospheric data points... */
  for (int ip = 0; ip < atm->np; ip++) {

    /* Get altitude index... */
    const int iz = locate_reg(z, 121, atm->z[ip]);

    /* Interpolate pressure... */
    atm->p[ip] = LOGY(z[iz], pre[iz], z[iz + 1], pre[iz + 1], atm->z[ip]);

    /* Interpolate temperature... */
    atm->t[ip] = LIN(z[iz], tem[iz], z[iz + 1], tem[iz + 1], atm->z[ip]);

    /* Interpolate trace gases... */
    for (int ig = 0; ig < ctl->ng; ig++)
      if (q[ig] != NULL)
        atm->q[ig][ip] =
          LIN(z[iz], q[ig][iz], z[iz + 1], q[ig][iz + 1], atm->z[ip]);
      else
        atm->q[ig][ip] = 0;

    /* Set CO2... */
    if (ig_co2 >= 0)
      atm->q[ig_co2][ip] =
        371.789948e-6 + 2.026214e-6 * (atm->time[ip] - 63158400.) / 31557600.;

    /* Set extinction to zero... */
    for (int iw = 0; iw < ctl->nw; iw++)
      atm->k[iw][ip] = 0;

    /* Set cloud layer... */
    atm->clz = atm->cldz = 0;
    for (int icl = 0; icl < ctl->ncl; icl++)
      atm->clk[icl] = 0;

    /* Set surface layer... */
    atm->sfz = atm->sfp = atm->sft = 0;
    for (int isf = 0; isf < ctl->nsf; isf++)
      atm->sfeps[isf] = 1;
  }
}

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

double ctmco2(
  const double nu,
  const double p,
  const double t,
  const double u) {

  static double co2296[2001] = { 9.3388e-5, 9.7711e-5, 1.0224e-4, 1.0697e-4,
    1.1193e-4, 1.1712e-4, 1.2255e-4, 1.2824e-4, 1.3419e-4, 1.4043e-4,
    1.4695e-4, 1.5378e-4, 1.6094e-4, 1.6842e-4, 1.7626e-4, 1.8447e-4,
    1.9307e-4, 2.0207e-4, 2.1149e-4, 2.2136e-4, 2.3169e-4, 2.4251e-4,
    2.5384e-4, 2.657e-4, 2.7813e-4, 2.9114e-4, 3.0477e-4, 3.1904e-4,
    3.3399e-4, 3.4965e-4, 3.6604e-4, 3.8322e-4, 4.0121e-4, 4.2006e-4,
    4.398e-4, 4.6047e-4, 4.8214e-4, 5.0483e-4, 5.286e-4, 5.535e-4,
    5.7959e-4, 6.0693e-4, 6.3557e-4, 6.6558e-4, 6.9702e-4, 7.2996e-4,
    7.6449e-4, 8.0066e-4, 8.3856e-4, 8.7829e-4, 9.1991e-4, 9.6354e-4,
    .0010093, .0010572, .0011074, .00116, .0012152, .001273,
    .0013336, .0013972, .0014638, .0015336, .0016068, .0016835,
    .001764, .0018483, .0019367, .0020295, .0021267, .0022286,
    .0023355, .0024476, .0025652, .0026885, .0028178, .0029534,
    .0030956, .0032448, .0034012, .0035654, .0037375, .0039181,
    .0041076, .0043063, .0045148, .0047336, .0049632, .005204,
    .0054567, .0057219, .0060002, .0062923, .0065988, .0069204,
    .007258, .0076123, .0079842, .0083746, .0087844, .0092146,
    .0096663, .01014, .010638, .011161, .01171, .012286, .012891,
    .013527, .014194, .014895, .015631, .016404, .017217, .01807,
    .018966, .019908, .020897, .021936, .023028, .024176, .025382,
    .026649, .027981, .02938, .030851, .032397, .034023, .035732,
    .037528, .039416, .041402, .04349, .045685, .047994, .050422,
    .052975, .055661, .058486, .061458, .064584, .067873, .071334,
    .074975, .078807, .082839, .087082, .091549, .096249, .1012,
    .10641, .11189, .11767, .12375, .13015, .13689, .14399, .15147,
    .15935, .16765, .17639, .18561, .19531, .20554, .21632, .22769,
    .23967, .25229, .2656, .27964, .29443, .31004, .3265, .34386,
    .36218, .3815, .40188, .42339, .44609, .47004, .49533, .52202,
    .5502, .57995, .61137, .64455, .6796, .71663, .75574, .79707,
    .84075, .88691, .9357, .98728, 1.0418, 1.0995, 1.1605, 1.225,
    1.2932, 1.3654, 1.4418, 1.5227, 1.6083, 1.6989, 1.7948, 1.8964,
    2.004, 2.118, 2.2388, 2.3668, 2.5025, 2.6463, 2.7988, 2.9606,
    3.1321, 3.314, 3.5071, 3.712, 3.9296, 4.1605, 4.4058, 4.6663,
    4.9431, 5.2374, 5.5501, 5.8818, 6.2353, 6.6114, 7.0115, 7.4372,
    7.8905, 8.3731, 8.8871, 9.4349, 10.019, 10.641, 11.305, 12.013,
```

```
12.769, 13.576, 14.437, 15.358, 16.342, 17.39, 18.513, 19.716,
21.003, 22.379, 23.854, 25.436, 27.126, 28.942, 30.89, 32.973,
35.219, 37.634, 40.224, 43.021, 46.037, 49.29, 52.803, 56.447,
60.418, 64.792, 69.526, 74.637, 80.182, 86.193, 92.713, 99.786,
107.47, 115.84, 124.94, 134.86, 145.69, 157.49, 170.3, 184.39,
199.83, 216.4, 234.55, 254.72, 276.82, 299.85, 326.16, 354.99,
386.51, 416.68, 449.89, 490.12, 534.35, 578.25, 632.26, 692.61,
756.43, 834.75, 924.11, 1016.9, 996.96, 1102.7, 1219.2, 1351.9,
1494.3, 1654.1, 1826.5, 2027.9, 2249., 2453.8, 2714.4, 2999.4,
3209.5, 3509., 3840.4, 3907.5, 4190.7, 4533.5, 4648.3, 5059.1,
5561.6, 6191.4, 6820.8, 7905.9, 9362.2, 2431.3, 2211.3, 2046.8,
2023.8, 1985.9, 1905.9, 1491.1, 1369.8, 1262.2, 1200.7, 887.74,
820.25, 885.23, 887.21, 816.73, 1126.9, 1216.2, 1272.4, 1579.5,
1634.2, 1656.3, 1657.9, 1789.5, 1670.8, 1509.5, 8474.6, 7489.2,
6793.6, 6117., 5574.1, 5141.2, 5084.6, 4745.1, 4413.2, 4102.8,
4024.7, 3715., 3398.6, 3100.8, 2900.4, 2629.2, 2374., 2144.7,
1955.8, 1760.8, 1591.2, 1435.2, 1296.2, 1174., 1065.1, 967.76,
999.48, 897.45, 809.23, 732.77, 670.26, 611.93, 560.11, 518.77,
476.84, 438.8, 408.48, 380.21, 349.24, 322.71, 296.65, 272.85,
251.96, 232.04, 213.88, 197.69, 182.41, 168.41, 155.79, 144.05,
133.31, 123.48, 114.5, 106.21, 98.591, 91.612, 85.156, 79.204,
73.719, 68.666, 63.975, 59.637, 56.35, 52.545, 49.042, 45.788,
42.78, 39.992, 37.441, 35.037, 32.8, 30.744, 28.801, 26.986,
25.297, 23.731, 22.258, 20.883, 19.603, 18.403, 17.295, 16.249,
15.271, 14.356, 13.501, 12.701, 11.954, 11.254, 10.6, 9.9864,
9.4118, 8.8745, 8.3714, 7.8997, 7.4578, 7.0446, 6.6573, 6.2949,
5.9577, 5.6395, 5.3419, 5.063, 4.8037, 4.5608, 4.3452, 4.1364,
3.9413, 3.7394, 3.562, 3.3932, 3.2325, 3.0789, 2.9318, 2.7898,
2.6537, 2.5225, 2.3958, 2.2305, 2.1215, 2.0245, 1.9427, 1.8795,
1.8336, 1.7604, 1.7016, 1.6419, 1.5282, 1.4611, 1.3443, 1.27,
1.1675, 1.0824, 1.0534, .99833, .95854, .92981, .90887, .89346,
.88113, .87068, .86102, .85096, .88262, .86151, .83565, .80518,
.77045, .73736, .74744, .74954, .75773, .82267, .83493, .89402,
.89725, .93426, .95564, .94045, .94174, .93404, .92035, .90456,
.88621, .86673, .78117, .7515, .72056, .68822, .65658, .62764,
.55984, .55598, .57407, .60963, .63763, .66198, .61132, .60972,
.52496, .50649, .41872, .3964, .32422, .27276, .24048, .23772,
.2286, .22711, .23999, .32038, .34371, .36621, .38561, .39953,
.40636, .44913, .42716, .3919, .35477, .33935, .3351, .39746,
.40993, .49398, .49956, .56157, .54742, .57295, .57386, .55417,
.50745, .471, .43446, .39102, .34993, .31269, .27888, .24912,
.22291, .19994, .17972, .16197, .14633, .13252, .12029, .10942,
.099745, .091118, .083404, .076494, .070292, .064716, .059697,
.055173, .051093, .047411, .044089, .041092, .038392, .035965,
.033789, .031846, .030122, .028607, .02729, .026169, .025209,
.024405, .023766, .023288, .022925, .022716, .022681, .022685,
.022768, .023133, .023325, .023486, .024004, .024126, .024083,
.023785, .024023, .023029, .021649, .021108, .019454, .017809,
.017292, .016635, .017037, .018068, .018977, .018756, .017847,
.016557, .016142, .014459, .012869, .012381, .010875, .0098701,
.009285, .0091698, .0091701, .0096145, .010553, .01106, .012613,
.014362, .015017, .016507, .017741, .01768, .017784, .0171,
.016357, .016172, .017257, .018978, .020935, .021741, .023567,
.025183, .025589, .026732, .027648, .028278, .028215, .02856,
.029015, .029062, .028851, .028497, .027825, .027801, .026523,
.02487, .022967, .022168, .020194, .018605, .017903, .018439,
.019697, .020311, .020855, .020057, .018608, .016738, .015963,
.013844, .011801, .011134, .0097573, .0086007, .0086226,
.0083721, .0090978, .0097616, .0098426, .011317, .012853, .01447,
.014657, .015771, .016351, .016079, .014829, .013431, .013185,
.013207, .01448, .016176, .017971, .018265, .019526, .020455,
.019797, .019802, .0194, .018176, .017505, .016197, .015339,
.014401, .013213, .012203, .011186, .010236, .0093288, .0084854,
.0076837, .0069375, .0062614, .0056628, .0051153, .0046015,
.0041501, .003752, .0033996, .0030865, .0028077, .0025586,
.0023355, .0021353, .0019553, .0017931, .0016466, .0015141,
.0013941, .0012852, .0011862, .0010962, .0010142, 9.3935e-4,
8.71e-4, 8.0851e-4, 7.5132e-4, 6.9894e-4, 6.5093e-4, 6.0689e-4,
5.6647e-4, 5.2935e-4, 4.9525e-4, 4.6391e-4, 4.3509e-4, 4.086e-4,
3.8424e-4, 3.6185e-4, 3.4126e-4, 3.2235e-4, 3.0498e-4, 2.8904e-4,
2.7444e-4, 2.6106e-4, 2.4883e-4, 2.3766e-4, 2.275e-4, 2.1827e-4,
2.0992e-4, 2.0239e-4, 1.9563e-4, 1.896e-4, 1.8427e-4, 1.796e-4,
1.7555e-4, 1.7209e-4, 1.692e-4, 1.6687e-4, 1.6505e-4, 1.6375e-4,
1.6294e-4, 1.6261e-4, 1.6274e-4, 1.6334e-4, 1.6438e-4, 1.6587e-4,
1.678e-4, 1.7017e-4, 1.7297e-4, 1.762e-4, 1.7988e-4, 1.8399e-4,
1.8855e-4, 1.9355e-4, 1.9902e-4, 2.0494e-4, 2.1134e-4, 2.1823e-4,
2.2561e-4, 2.335e-4, 2.4192e-4, 2.5088e-4, 2.604e-4, 2.705e-4,
2.8119e-4, 2.9251e-4, 3.0447e-4, 3.171e-4, 3.3042e-4, 3.4447e-4,
3.5927e-4, 3.7486e-4, 3.9127e-4, 4.0854e-4, 4.267e-4, 4.4579e-4,
4.6586e-4, 4.8696e-4, 5.0912e-4, 5.324e-4, 5.5685e-4, 5.8253e-4,
6.0949e-4, 6.378e-4, 6.6753e-4, 6.9873e-4, 7.3149e-4, 7.6588e-4,
8.0198e-4, 8.3987e-4, 8.7964e-4, 9.2139e-4, 9.6522e-4, .0010112,
.0010595, .0011102, .0011634, .0012193, .001278, .0013396,
.0014043, .0014722, .0015436, .0016185, .0016972, .0017799,
.0018668, .001958, .0020539, .0021547, .0022606, .0023719,
.002489, .002612, .0027414, .0028775, .0030206, .0031712,
.0033295, .0034962, .0036716, .0038563, .0040506, .0042553,
```

```
.0044709, .004698, .0049373, .0051894, .0054552, .0057354,
.006031, .0063427, .0066717, .0070188, .0073854, .0077726,
.0081816, .0086138, .0090709, .0095543, .010066, .010607,
.011181, .011789, .012433, .013116, .013842, .014613, .015432,
.016304, .017233, .018224, .019281, .020394, .021574, .022836,
.024181, .025594, .027088, .028707, .030401, .032245, .034219,
.036262, .038539, .040987, .043578, .04641, .04949, .052726,
.056326, .0602, .064093, .068521, .073278, .077734, .083064,
.088731, .093885, .1003, .1072, .11365, .12187, .13078, .13989,
.15095, .16299, .17634, .19116, .20628, .22419, .24386, .26587,
.28811, .31399, .34321, .36606, .39675, .42742, .44243, .47197,
.49993, .49027, .51147, .52803, .48931, .49729, .5026, .43854,
.441, .44766, .43414, .46151, .50029, .55247, .43855, .32115,
.32607, .3431, .36119, .38029, .41179, .43996, .47144, .51853,
.55362, .59122, .66338, .69877, .74001, .82923, .86907, .90361,
1.0025, 1.031, 1.0559, 1.104, 1.1178, 1.1341, 1.1547, 1.351,
1.4772, 1.4812, 1.4907, 1.512, 1.5442, 1.5853, 1.6358, 1.6963,
1.7674, 1.8474, 1.9353, 2.0335, 2.143, 2.2592, 2.3853, 2.5217,
2.6686, 2.8273, 2.9998, 3.183, 3.3868, 3.6109, 3.8564, 4.1159,
4.4079, 4.7278, 5.0497, 5.3695, 5.758, 6.0834, 6.4976, 6.9312,
7.38, 7.5746, 7.9833, 8.3791, 8.3956, 8.7501, 9.1067, 9.072,
9.4649, 9.9112, 10.402, 10.829, 11.605, 12.54, 12.713, 10.443,
10.825, 11.375, 11.955, 12.623, 13.326, 14.101, 15.041, 15.547,
16.461, 17.439, 18.716, 19.84, 21.036, 22.642, 23.901, 25.244,
27.03, 28.411, 29.871, 31.403, 33.147, 34.744, 36.456, 39.239,
43.605, 45.162, 47.004, 49.093, 51.391, 53.946, 56.673, 59.629,
63.167, 66.576, 70.254, 74.222, 78.477, 83.034, 87.914, 93.18,
98.77, 104.74, 111.15, 117.95, 125.23, 133.01, 141.33, 150.21,
159.71, 169.89, 180.93, 192.54, 204.99, 218.34, 232.65, 248.,
264.47, 282.14, 301.13, 321.53, 343.48, 367.08, 392.5, 419.88,
449.4, 481.26, 515.64, 552.79, 592.99, 636.48, 683.61, 734.65,
789.99, 850.02, 915.14, 985.81, 1062.5, 1147.1, 1237.8, 1336.4,
1443.2, 1558.9, 1684.2, 1819.2, 1965.2, 2122.6, 2291.7, 2470.8,
2665.7, 2874.9, 3099.4, 3337.9, 3541., 3813.3, 4111.9, 4439.3,
4798.9, 5196., 5639.2, 6087.5, 6657.7, 7306.7, 8040.7, 8845.5,
9702.2, 10670., 11739., 12842., 14141., 15498., 17068., 18729.,
20557., 22559., 25248., 27664., 30207., 32915., 35611., 38081.,
40715., 43191., 41651., 42750., 43785., 44353., 44366., 44189.,
43618., 42862., 41878., 35133., 35215., 36383., 39420., 44055.,
44155., 45850., 46853., 39197., 38274., 29942., 28553., 21792.,
21228., 17106., 14955., 18181., 19557., 21427., 23728., 26301.,
28584., 30775., 32536., 33867., 40089., 39204., 37329., 34452.,
31373., 33921., 34800., 36043., 44415., 45162., 52181., 50895.,
54140., 50840., 50468., 48302., 44915., 40910., 36754., 32755.,
29093., 25860., 22962., 20448., 18247., 16326., 14645., 13165.,
11861., 10708., 9686.9, 8779.7, 7971.9, 7250.8, 6605.7, 6027.2,
5507.3, 5039.1, 4616.6, 4234.8, 3889., 3575.4, 3290.5, 3031.3,
2795.2, 2579.9, 2383.1, 2203.3, 2038.6, 1887.6, 1749.1, 1621.9,
1505., 1397.4, 1298.3, 1207., 1122.8, 1045., 973.1, 906.64,
845.16, 788.22, 735.48, 686.57, 641.21, 599.1, 559.99, 523.64,
489.85, 458.42, 429.16, 401.92, 376.54, 352.88, 330.82, 310.24,
291.03, 273.09, 256.34, 240.69, 226.05, 212.37, 199.57, 187.59,
176.37, 165.87, 156.03, 146.82, 138.17, 130.07, 122.47, 115.34,
108.65, 102.37, 96.473, 90.934, 85.73, 80.84, 76.243, 71.922,
67.858, 64.034, 60.438, 57.052, 53.866, 50.866, 48.04, 45.379,
42.872, 40.51, 38.285, 36.188, 34.211, 32.347, 30.588, 28.929,
27.362, 25.884, 24.489, 23.171, 21.929, 20.755, 19.646, 18.599,
17.61, 16.677, 15.795, 14.961, 14.174, 13.43, 12.725, 12.06,
11.431, 10.834, 10.27, 9.7361, 9.2302, 8.7518, 8.2997, 7.8724,
7.4674, 7.0848, 6.7226, 6.3794, 6.054, 5.745, 5.4525, 5.1752,
4.9121, 4.6625, 4.4259, 4.2015, 3.9888, 3.7872, 3.5961, 3.4149,
3.2431, 3.0802, 2.9257, 2.7792, 2.6402, 2.5084, 2.3834, 2.2648,
2.1522, 2.0455, 1.9441, 1.848, 1.7567, 1.6701, 1.5878, 1.5097,
1.4356, 1.3651, 1.2981, 1.2345, 1.174, 1.1167, 1.062, 1.0101,
.96087, .91414, .86986, .82781, .78777, .74971, .71339, .67882,
.64604, .61473, .58507, .55676, .52987, .5044, .48014, .45715,
.43527, .41453, .3948, .37609, .35831, .34142, .32524, .30995,
.29536, .28142, .26807, .25527, .24311, .23166, .22077, .21053,
.20081, .19143, .18261, .17407, .16603, .15833, .15089, .14385,
.13707, .13065, .12449, .11865, .11306, .10774, .10266, .097818,
.093203, .088815, .084641, .080671, .076892, .073296, .069873,
.066613, .06351, .060555, .05774, .055058, .052504, .050071,
.047752, .045543, .043438, .041432, .039521, .037699, .035962,
.034307, .032729, .031225, .029791, .028423, .02712, .025877,
.024692, .023563, .022485, .021458, .020478, .019543, .018652,
.017802, .016992, .016219, .015481, .014778, .014107, .013467,
.012856, .012274, .011718, .011188, .010682, .0102, .0097393,
.0093001, .008881, .0084812, .0080997, .0077358, .0073885,
.0070571, .0067409, .0064393, .0061514, .0058768, .0056147,
.0053647, .0051262, .0048987, .0046816, .0044745, .0042769,
.0040884, .0039088, .0037373, .0035739, .003418, .0032693,
.0031277, .0029926, .0028639, .0027413, .0026245, .0025133,
.0024074, .0023066, .0022108, .0021196, .002033, .0019507,
.0018726, .0017985, .0017282, .0016617, .0015988, .0015394,
.0014834, .0014306, .0013811, .0013346, .0012911, .0012506,
.0012131, .0011784, .0011465, .0011175, .0010912, .0010678,
.0010472, .0010295, .0010147, .001003, 9.9428e-4, 9.8883e-4,
```

```
  9.8673e-4, 9.8821e-4, 9.9343e-4, .0010027, .0010164, .0010348,
  .0010586, .0010882, .0011245, .0011685, .0012145, .0012666,
  .0013095, .0013688, .0014048, .0014663, .0015309, .0015499,
  .0016144, .0016312, .001705, .0017892, .0018499, .0019715,
  .0021102, .0022442, .0024284, .0025893, .0027703, .0029445,
  .0031193, .003346, .0034552, .0036906, .0037584, .0040084,
  .0041934, .0044587, .0047093, .0049759, .0053421, .0055134,
  .0059048, .0058663, .0061036, .0063259, .0059657, .0060653,
  .0060972, .0055539, .0055653, .0055772, .005331, .0054953,
  .0055919, .0058684, .006183, .0066675, .0069808, .0075142,
  .0078536, .0084282, .0089454, .0094625, .0093703, .0095857,
  .0099283, .010063, .010521, .0097778, .0098175, .010379, .010447,
  .0105, .010617, .010706, .01078, .011177, .011212, .011304,
  .011446, .011603, .011816, .012165, .012545, .013069, .013539,
  .01411, .014776, .016103, .017016, .017994, .018978, .01998,
  .021799, .022745, .023681, .024627, .025562, .026992, .027958,
  .029013, .030154, .031402, .03228, .033651, .035272, .037088,
  .039021, .041213, .043597, .045977, .04877, .051809, .054943,
  .058064, .061528, .06537, .069309, .071928, .075752, .079589,
  .083352, .084096, .087497, .090817, .091198, .094966, .099045,
  .10429, .10867, .11518, .12269, .13126, .14087, .15161, .16388,
  .16423, .1759, .18721, .19994, .21275, .22513, .23041, .24231,
  .25299, .25396, .26396, .27696, .27929, .2908, .30595, .31433,
  .3282, .3429, .35944, .37467, .39277, .41245, .43326, .45649,
  .48152, .51897, .54686, .57877, .61263, .64962, .68983, .73945,
  .78619, .83537, .89622, .95002, 1.0067, 1.0742, 1.1355, 1.2007,
  1.2738, 1.347, 1.4254, 1.5094, 1.6009, 1.6976, 1.8019, 1.9148,
  2.0357, 2.166, 2.3066, 2.4579, 2.6208, 2.7966, 2.986, 3.188,
  3.4081, 3.6456, 3.9, 4.1747, 4.4712, 4.7931, 5.1359, 5.5097,
  5.9117, 6.3435, 6.8003, 7.3001, 7.8385, 8.3945, 9.011, 9.6869,
  10.392, 11.18, 12.036, 12.938, 13.944, 14.881, 16.029, 17.255,
  18.574, 19.945, 21.38, 22.9, 24.477, 26.128, 27.87, 29.037,
  30.988, 33.145, 35.506, 37.76, 40.885, 44.487, 48.505, 52.911,
  57.56, 61.964, 67.217, 72.26, 78.343, 85.08, 91.867, 99.435,
  107.68, 116.97, 127.12, 138.32, 150.26, 163.04, 174.81, 189.26,
  205.61, 224.68, 240.98, 261.88, 285.1, 307.58, 334.35, 363.53,
  394.68, 427.85, 458.85, 489.25, 472.87, 486.93, 496.27, 501.52,
  501.57, 497.14, 488.09, 476.32, 393.76, 388.51, 393.42, 414.45,
  455.12, 514.62, 520.38, 547.42, 562.6, 487.47, 480.83, 391.06,
  376.92, 303.7, 295.91, 256.03, 236.73, 280.38, 310.71, 335.53,
  367.88, 401.94, 435.52, 469.13, 497.94, 588.82, 597.94, 597.2,
  588.28, 571.2, 555.75, 603.56, 638.15, 680.75, 801.72, 848.01,
  962.15, 990.06, 1068.1, 1076.2, 1115.3, 1134.2, 1136.6, 1119.1,
  1108.9, 1090.6, 1068.7, 1041.9, 1005.4, 967.98, 927.08, 780.1,
  751.41, 733.12, 742.65, 785.56, 855.16, 852.45, 878.1, 784.59,
  777.81, 765.13, 622.93, 498.09, 474.89, 386.9, 378.48, 336.17,
  322.04, 329.57, 350.5, 383.38, 420.02, 462.39, 499.71, 531.98,
  654.99, 653.43, 639.99, 605.16, 554.16, 504.42, 540.64, 552.33,
  679.46, 699.51, 713.91, 832.17, 919.91, 884.96, 907.57, 846.56,
  818.56, 768.93, 706.71, 642.17, 575.95, 515.38, 459.07, 409.02,
  364.61, 325.46, 291.1, 260.89, 234.39, 211.01, 190.38, 172.11,
  155.91, 141.49, 128.63, 117.13, 106.84, 97.584, 89.262, 81.756,
  74.975, 68.842, 63.28, 58.232, 53.641, 49.46, 45.649, 42.168,
  38.991, 36.078, 33.409, 30.96, 28.71, 26.642, 24.737, 22.985,
  21.37, 19.882, 18.512, 17.242, 16.073, 14.987, 13.984, 13.05,
  12.186, 11.384, 10.637, 9.9436, 9.2988, 8.6991, 8.141, 7.6215,
  7.1378, 6.6872, 6.2671, 5.8754, 5.51, 5.1691, 4.851, 4.5539,
  4.2764, 4.0169, 3.7742, 3.5472, 3.3348, 3.1359, 2.9495, 2.7749,
  2.6113, 2.4578, 2.3139, 2.1789, 2.0523, 1.9334, 1.8219, 1.7171,
  1.6188, 1.5263, 1.4395, 1.3579, 1.2812, 1.209, 1.1411, 1.0773,
  1.0171, .96048, .90713, .85684, .80959, .76495, .72282, .68309,
  .64563, .61035, .57707, .54573, .51622, .48834, .46199, .43709,
  .41359, .39129, .37034, .35064, .33198, .31442, .29784, .28218,
  .26732, .25337, .24017, .22774, .21601, .20479, .19426
};

static double co2260[2001] = { 5.7971e-5, 6.0733e-5, 6.3628e-5, 6.6662e-5,
  6.9843e-5, 7.3176e-5, 7.6671e-5, 8.0334e-5, 8.4175e-5, 8.8201e-5,
  9.2421e-5, 9.6846e-5, 1.0149e-4, 1.0635e-4, 1.1145e-4, 1.1679e-4,
  1.224e-4, 1.2828e-4, 1.3444e-4, 1.409e-4, 1.4768e-4, 1.5479e-4,
  1.6224e-4, 1.7006e-4, 1.7826e-4, 1.8685e-4, 1.9587e-4, 2.0532e-4,
  2.1524e-4, 2.2565e-4, 2.3656e-4, 2.48e-4, 2.6001e-4, 2.7261e-4,
  2.8582e-4, 2.9968e-4, 3.1422e-4, 3.2948e-4, 3.4548e-4, 3.6228e-4,
  3.799e-4, 3.9838e-4, 4.1778e-4, 4.3814e-4, 4.595e-4, 4.8191e-4,
  5.0543e-4, 5.3012e-4, 5.5603e-4, 5.8321e-4, 6.1175e-4, 6.417e-4,
  6.7314e-4, 7.0614e-4, 7.4078e-4, 7.7714e-4, 8.1531e-4, 8.5538e-4,
  8.9745e-4, 9.4162e-4, 9.8798e-4, .0010367, .0010878, .0011415,
  .0011978, .001257, .0013191, .0013844, .001453, .0015249,
  .0016006, .00168, .0017634, .001851, .001943, .0020397, .0021412,
  .0022479, .00236, .0024778, .0026015, .0027316, .0028682,
  .0030117, .0031626, .0033211, .0034877, .0036628, .0038469,
  .0040403, .0042436, .0044574, .004682, .0049182, .0051665,
  .0054276, .0057021, .0059907, .0062942, .0066133, .0069489,
  .0073018, .0076729, .0080632, .0084738, .0089056, .0093599,
  .0098377, .01034, .010869, .011426, .012011, .012627, .013276,
  .013958, .014676, .015431, .016226, .017063, .017944, .018872,
  .019848, .020876, .021958, .023098, .024298, .025561, .026892,
```

```
.028293, .029769, .031323, .032961, .034686, .036503, .038418,
.040435, .042561, .044801, .047161, .049649, .052271, .055035,
.057948, .061019, .064256, .06767, .07127, .075066, .079069,
.083291, .087744, .092441, .097396, .10262, .10814, .11396,
.1201, .12658, .13342, .14064, .14826, .1563, .1648, .17376,
.18323, .19324, .2038, .21496, .22674, .23919, .25234, .26624,
.28093, .29646, .31287, .33021, .34855, .36794, .38844, .41012,
.43305, .45731, .48297, .51011, .53884, .56924, .60141, .63547,
.67152, .70969, .75012, .79292, .83826, .8863, .93718, .99111,
1.0482, 1.1088, 1.173, 1.2411, 1.3133, 1.3898, 1.471, 1.5571,
1.6485, 1.7455, 1.8485, 1.9577, 2.0737, 2.197, 2.3278, 2.4668,
2.6145, 2.7715, 2.9383, 3.1156, 3.3042, 3.5047, 3.7181, 3.9451,
4.1866, 4.4437, 4.7174, 5.0089, 5.3192, 5.65, 6.0025, 6.3782,
6.7787, 7.206, 7.6617, 8.1479, 8.6669, 9.221, 9.8128, 10.445,
11.12, 11.843, 12.615, 13.441, 14.325, 15.271, 16.283, 17.367,
18.529, 19.776, 21.111, 22.544, 24.082, 25.731, 27.504, 29.409,
31.452, 33.654, 36.024, 38.573, 41.323, 44.29, 47.492, 50.951,
54.608, 58.588, 62.929, 67.629, 72.712, 78.226, 84.207, 90.699,
97.749, 105.42, 113.77, 122.86, 132.78, 143.61, 155.44, 168.33,
182.48, 198.01, 214.87, 233.39, 253.86, 276.34, 300.3, 327.28,
356.89, 389.48, 422.29, 458.99, 501.39, 548.13, 595.62, 652.74,
716.54, 784.57, 866.78, 960.59, 1062.8, 1072.5, 1189.5, 1319.4,
1467.6, 1630.2, 1813.7, 2016.9, 2253., 2515.3, 2773.5, 3092.8,
3444.4, 3720.4, 4104.3, 4527.5, 4645.9, 5021.7, 5462.2, 5597.,
6110.6, 6732.5, 7513.8, 8270.6, 9640.6, 11487., 2796.1, 2680.1,
2441.6, 2404.2, 2334.8, 2215.2, 1642.5, 1477.9, 1328.1, 1223.5,
843.34, 766.96, 831.65, 834.84, 774.85, 1156.3, 1275.6, 1366.1,
1795.6, 1885., 1936.5, 1953.4, 2154.4, 2002.7, 1789.8, 10381.,
9040., 8216.5, 7384.7, 6721.9, 6187.7, 6143.8, 5703.9, 5276.6,
4873.1, 4736., 4325.3, 3927., 3554.1, 3286.1, 2950.1, 2642.4,
2368.7, 2138.9, 1914., 1719.6, 1543.9, 1388.6, 1252.1, 1132.2,
1024.1, 1025.4, 920.58, 829.59, 750.54, 685.01, 624.25, 570.14,
525.81, 481.85, 441.95, 408.71, 377.23, 345.86, 318.51, 292.26,
268.34, 247.04, 227.14, 209.02, 192.69, 177.59, 163.78, 151.26,
139.73, 129.19, 119.53, 110.7, 102.57, 95.109, 88.264, 81.948,
76.13, 70.768, 65.827, 61.251, 57.022, 53.495, 49.824, 46.443,
43.307, 40.405, 37.716, 35.241, 32.923, 30.77, 28.78, 26.915,
25.177, 23.56, 22.059, 20.654, 19.345, 18.126, 16.988, 15.93,
14.939, 14.014, 13.149, 12.343, 11.589, 10.884, 10.225, 9.6093,
9.0327, 8.4934, 7.9889, 7.5166, 7.0744, 6.6604, 6.2727, 5.9098,
5.5701, 5.2529, 4.955, 4.676, 4.4148, 4.171, 3.9426, 3.7332,
3.5347, 3.3493, 3.1677, 3.0025, 2.8466, 2.6994, 2.5601, 2.4277,
2.3016, 2.1814, 2.0664, 1.9564, 1.8279, 1.7311, 1.6427, 1.5645,
1.4982, 1.443, 1.374, 1.3146, 1.2562, 1.17, 1.1105, 1.0272,
.96863, .89718, .83654, .80226, .75908, .72431, .69573, .67174,
.65126, .63315, .61693, .60182, .58715, .59554, .57649, .55526,
.53177, .50622, .48176, .4813, .47642, .47492, .50273, .50293,
.52687, .52239, .53419, .53814, .52626, .52211, .51492, .50622,
.49746, .48841, .4792, .43534, .41999, .40349, .38586, .36799,
.35108, .31089, .30803, .3171, .33599, .35041, .36149, .32924,
.32462, .27309, .25961, .20922, .19504, .15683, .13098, .11588,
.11478, .11204, .11363, .12135, .16423, .17785, .19094, .20236,
.21084, .2154, .24108, .22848, .20871, .18797, .17963, .17834,
.21552, .22284, .26945, .27052, .30108, .28977, .29772, .29224,
.27658, .24956, .22777, .20654, .18392, .16338, .1452, .12916,
.1152, .10304, .092437, .083163, .075031, .067878, .061564,
.055976, .051018, .046609, .042679, .03917, .036032, .033223,
.030706, .02845, .026428, .024617, .022998, .021554, .02027,
.019136, .018141, .017278, .016541, .015926, .015432, .015058,
.014807, .014666, .014635, .014728, .014947, .01527, .015728,
.016345, .017026, .017798, .018839, .019752, .020636, .021886,
.022695, .02327, .023478, .024292, .023544, .022222, .021932,
.020052, .018143, .017722, .017031, .017782, .01938, .020734,
.020476, .019255, .017477, .016878, .014617, .012489, .011765,
.0099077, .0086446, .0079446, .0078644, .0079763, .008671,
.01001, .0108, .012933, .015349, .016341, .018484, .020254,
.020254, .020478, .019591, .018595, .018385, .019913, .022254,
.024847, .025809, .028053, .029924, .030212, .031367, .03222,
.032739, .032537, .03286, .033344, .033507, .033499, .033339,
.032809, .033041, .031723, .029837, .027511, .026603, .024032,
.021914, .020948, .021701, .023425, .024259, .024987, .023818,
.021768, .019223, .018144, .015282, .012604, .01163, .0097907,
.008336, .0082473, .0079582, .0088077, .009779, .010129, .012145,
.014378, .016761, .01726, .018997, .019998, .019809, .01819,
.016358, .016099, .01617, .017939, .020223, .022521, .02277,
.024279, .025247, .024222, .023989, .023224, .021493, .020362,
.018596, .017309, .015975, .014466, .013171, .011921, .01078,
.0097229, .0087612, .0078729, .0070682, .0063494, .0057156,
.0051459, .0046273, .0041712, .0037686, .0034119, .003095,
.0028126, .0025603, .0023342, .0021314, .0019489, .0017845,
.001636, .0015017, .00138, .0012697, .0011694, .0010782,
9.9507e-4, 9.1931e-4, 8.5013e-4, 7.869e-4, 7.2907e-4, 6.7611e-4,
6.2758e-4, 5.8308e-4, 5.4223e-4, 5.0473e-4, 4.7027e-4, 4.3859e-4,
4.0946e-4, 3.8265e-4, 3.5798e-4, 3.3526e-4, 3.1436e-4, 2.9511e-4,
2.7739e-4, 2.6109e-4, 2.4609e-4, 2.3229e-4, 2.1961e-4, 2.0797e-4,
1.9729e-4, 1.875e-4, 1.7855e-4, 1.7038e-4, 1.6294e-4, 1.5619e-4,
1.5007e-4, 1.4456e-4, 1.3961e-4, 1.3521e-4, 1.3131e-4, 1.2789e-4,
```

```
    1.2494e-4, 1.2242e-4, 1.2032e-4, 1.1863e-4, 1.1733e-4, 1.1641e-4,
    1.1585e-4, 1.1565e-4, 1.158e-4, 1.1629e-4, 1.1712e-4, 1.1827e-4,
    1.1976e-4, 1.2158e-4, 1.2373e-4, 1.262e-4, 1.2901e-4, 1.3214e-4,
    1.3562e-4, 1.3944e-4, 1.4361e-4, 1.4814e-4, 1.5303e-4, 1.5829e-4,
    1.6394e-4, 1.6999e-4, 1.7644e-4, 1.8332e-4, 1.9063e-4, 1.984e-4,
    2.0663e-4, 2.1536e-4, 2.246e-4, 2.3436e-4, 2.4468e-4, 2.5558e-4,
    2.6708e-4, 2.7921e-4, 2.92e-4, 3.0548e-4, 3.1968e-4, 3.3464e-4,
    3.5039e-4, 3.6698e-4, 3.8443e-4, 4.0281e-4, 4.2214e-4, 4.4248e-4,
    4.6389e-4, 4.864e-4, 5.1009e-4, 5.3501e-4, 5.6123e-4, 5.888e-4,
    6.1781e-4, 6.4833e-4, 6.8043e-4, 7.142e-4, 7.4973e-4, 7.8711e-4,
    8.2644e-4, 8.6783e-4, 9.1137e-4, 9.5721e-4, .0010054, .0010562,
    .0011096, .0011659, .0012251, .0012875, .0013532, .0014224,
    .0014953, .001572, .0016529, .0017381, .0018279, .0019226,
    .0020224, .0021277, .0022386, .0023557, .0024792, .0026095,
    .002747, .0028921, .0030453, .0032071, .003378, .0035586,
    .0037494, .003951, .0041642, .0043897, .0046282, .0048805,
    .0051476, .0054304, .00573, .0060473, .0063837, .0067404,
    .0071188, .0075203, .0079466, .0083994, .0088806, .0093922,
    .0099366, .010516, .011134, .011792, .012494, .013244, .014046,
    .014898, .015808, .016781, .017822, .018929, .020108, .02138,
    .022729, .02419, .02576, .027412, .029233, .031198, .033301,
    .035594, .038092, .040767, .04372, .046918, .050246, .053974,
    .058009, .061976, .066586, .071537, .076209, .081856, .087998,
    .093821, .10113, .10913, .11731, .12724, .13821, .15025, .1639,
    .17807, .19472, .21356, .23496, .25758, .28387, .31389, .34104,
    .37469, .40989, .43309, .46845, .5042, .5023, .52981, .55275,
    .51075, .51976, .52457, .44779, .44721, .4503, .4243, .45244,
    .49491, .55399, .39021, .24802, .2501, .2618, .27475, .28879,
    .31317, .33643, .36257, .4018, .43275, .46525, .53333, .56599,
    .60557, .70142, .74194, .77736, .88567, .91182, .93294, .98407,
    .98772, .99176, .9995, 1.2405, 1.3602, 1.338, 1.3255, 1.3267,
    1.3404, 1.3634, 1.3967, 1.4407, 1.4961, 1.5603, 1.6328, 1.7153,
    1.8094, 1.9091, 2.018, 2.1367, 2.264, 2.4035, 2.5562, 2.7179,
    2.9017, 3.1052, 3.3304, 3.5731, 3.8488, 4.1553, 4.4769, 4.7818,
    5.1711, 5.5204, 5.9516, 6.4097, 6.8899, 7.1118, 7.5469, 7.9735,
    7.9511, 8.3014, 8.6418, 8.4757, 8.8256, 9.2294, 9.6923, 10.033,
    10.842, 11.851, 11.78, 8.8435, 9.1381, 9.5956, 10.076, 10.629,
    11.22, 11.883, 12.69, 13.163, 13.974, 14.846, 16.027, 17.053,
    18.148, 19.715, 20.907, 22.163, 23.956, 25.235, 26.566, 27.94,
    29.576, 30.956, 32.432, 35.337, 39.911, 41.128, 42.625, 44.386,
    46.369, 48.619, 51.031, 53.674, 56.825, 59.921, 63.286, 66.929,
    70.859, 75.081, 79.618, 84.513, 89.739, 95.335, 101.35, 107.76,
    114.63, 121.98, 129.87, 138.3, 147.34, 157.04, 167.56, 178.67,
    190.61, 203.43, 217.19, 231.99, 247.88, 264.98, 283.37, 303.17,
    324.49, 347.47, 372.25, 398.98, 427.85, 459.06, 492.8, 529.31,
    568.89, 611.79, 658.35, 708.91, 763.87, 823.65, 888.72, 959.58,
    1036.8, 1121.8, 1213.9, 1314.3, 1423.8, 1543., 1672.8, 1813.4,
    1966.1, 2131.4, 2309.5, 2499.3, 2705., 2925.7, 3161.6, 3411.3,
    3611.5, 3889.2, 4191.1, 4519.3, 4877.9, 5272.9, 5712.9, 6142.7,
    6719.6, 7385., 8145., 8977.7, 9831.9, 10827., 11934., 13063.,
    14434., 15878., 17591., 19435., 21510., 23835., 26835., 29740.,
    32878., 36305., 39830., 43273., 46931., 50499., 49586., 51598.,
    53429., 54619., 55081., 55102., 54485., 53487., 52042., 42689.,
    42607., 44020., 47994., 54169., 53916., 55808., 56642., 46049.,
    44243., 32929., 30658., 21963., 20835., 15962., 13679., 17652.,
    19680., 22388., 25625., 29184., 32520., 35720., 38414., 40523.,
    49228., 48173., 45678., 41768., 37600., 41313., 42654., 44465.,
    55736., 56630., 65409., 63308., 66572., 61845., 60379., 56777.,
    51920., 46601., 41367., 36529., 32219., 28470., 25192., 22362.,
    19907., 17772., 15907., 14273., 12835., 11567., 10445., 9450.2,
    8565.1, 7776., 7070.8, 6439.2, 5872.3, 5362.4, 4903., 4488.3,
    4113.4, 3773.8, 3465.8, 3186.1, 2931.7, 2700.1, 2488.8, 2296.,
    2119.8, 1958.6, 1810.9, 1675.6, 1551.4, 1437.3, 1332.4, 1236.,
    1147.2, 1065.3, 989.86, 920.22, 855.91, 796.48, 741.53, 690.69,
    643.62, 600.02, 559.6, 522.13, 487.35, 455.06, 425.08, 397.21,
    371.3, 347.2, 324.78, 303.9, 284.46, 266.34, 249.45, 233.7,
    219.01, 205.3, 192.5, 180.55, 169.38, 158.95, 149.2, 140.07,
    131.54, 123.56, 116.09, 109.09, 102.54, 96.405, 90.655, 85.266,
    80.213, 75.475, 71.031, 66.861, 62.948, 59.275, 55.827, 52.587,
    49.544, 46.686, 43.998, 41.473, 39.099, 36.867, 34.768, 32.795,
    30.939, 29.192, 27.546, 25.998, 24.539, 23.164, 21.869, 20.65,
    19.501, 18.419, 17.399, 16.438, 15.532, 14.678, 13.874, 13.115,
    12.4, 11.726, 11.088, 10.488, 9.921, 9.3846, 8.8784, 8.3996,
    7.9469, 7.5197, 7.1174, 6.738, 6.379, 6.0409, 5.7213, 5.419,
    5.1327, 4.8611, 4.6046, 4.3617, 4.1316, 3.9138, 3.7077, 3.5125,
    3.3281, 3.1536, 2.9885, 2.8323, 2.6846, 2.5447, 2.4124, 2.2871,
    2.1686, 2.0564, 1.9501, 1.8495, 1.7543, 1.6641, 1.5787, 1.4978,
    1.4212, 1.3486, 1.2799, 1.2147, 1.1529, 1.0943, 1.0388, .98602,
    .93596, .8886, .84352, .80078, .76029, .722, .68585, .65161,
    .61901, .58808, .55854, .53044, .5039, .47853, .45459, .43173,
    .41008, .38965, .37021, .35186, .33444, .31797, .30234, .28758,
    .2736, .26036, .24764, .2357, .22431, .21342, .20295, .19288,
    .18334, .17444, .166, .15815, .15072, .14348, .13674, .13015,
    .12399, .11807, .11231, .10689, .10164, .096696, .091955,
    .087476, .083183, .079113, .075229, .071536, .068026, .064698,
    .06154, .058544, .055699, .052997, .050431, .047993, .045676,
    .043475, .041382, .039392, .037501, .035702, .033991, .032364,
```

```
    .030817, .029345, .027945, .026613, .025345, .024139, .022991,
    .021899, .02086, .019871, .018929, .018033, .01718, .016368,
    .015595, .014859, .014158, .013491, .012856, .012251, .011675,
    .011126, .010604, .010107, .0096331, .009182, .0087523, .0083431,
    .0079533, .0075821, .0072284, .0068915, .0065706, .0062649,
    .0059737, .0056963, .005432, .0051802, .0049404, .0047118,
    .0044941, .0042867, .0040891, .0039009, .0037216, .0035507,
    .003388, .0032329, .0030852, .0029445, .0028105, .0026829,
    .0025613, .0024455, .0023353, .0022303, .0021304, .0020353,
    .0019448, .0018587, .0017767, .0016988, .0016247, .0015543,
    .0014874, .0014238, .0013635, .0013062, .0012519, .0012005,
    .0011517, .0011057, .0010621, .001021, 9.8233e-4, 9.4589e-4,
    9.1167e-4, 8.7961e-4, 8.4964e-4, 8.2173e-4, 7.9582e-4, 7.7189e-4,
    7.499e-4, 7.2983e-4, 7.1167e-4, 6.9542e-4, 6.8108e-4, 6.6866e-4,
    6.5819e-4, 6.4971e-4, 6.4328e-4, 6.3895e-4, 6.3681e-4, 6.3697e-4,
    6.3956e-4, 6.4472e-4, 6.5266e-4, 6.6359e-4, 6.778e-4, 6.9563e-4,
    7.1749e-4, 7.4392e-4, 7.7556e-4, 8.1028e-4, 8.4994e-4, 8.8709e-4,
    9.3413e-4, 9.6953e-4, .0010202, .0010738, .0010976, .0011507,
    .0011686, .0012264, .001291, .0013346, .0014246, .0015293,
    .0016359, .0017824, .0019255, .0020854, .002247, .0024148,
    .0026199, .0027523, .0029704, .0030702, .0033047, .0035013,
    .0037576, .0040275, .0043089, .0046927, .0049307, .0053486,
    .0053809, .0056699, .0059325, .0055488, .005634, .0056392,
    .004946, .0048855, .0048208, .0044386, .0045498, .0046377,
    .0048939, .0052396, .0057324, .0060859, .0066906, .0071148,
    .0077224, .0082687, .008769, .0084471, .008572, .0087729,
    .008775, .0090742, .0080704, .0080288, .0085747, .0086087,
    .0086408, .0088752, .0089381, .0089757, .0093532, .0092824,
    .0092566, .0092645, .0092735, .009342, .0095806, .0097991,
    .010213, .010611, .011129, .011756, .013237, .01412, .015034,
    .015936, .01682, .018597, .019315, .019995, .020658, .021289,
    .022363, .022996, .023716, .024512, .025434, .026067, .027118,
    .028396, .029865, .031442, .033253, .03525, .037296, .039701,
    .042356, .045154, .048059, .051294, .054893, .058636, .061407,
    .065172, .068974, .072676, .073379, .076547, .079556, .079134,
    .082308, .085739, .090192, .09359, .099599, .10669, .11496,
    .1244, .13512, .14752, .14494, .15647, .1668, .17863, .19029,
    .20124, .20254, .21179, .21982, .21625, .22364, .23405, .23382,
    .2434, .25708, .26406, .27621, .28909, .30395, .31717, .33271,
    .3496, .36765, .38774, .40949, .446, .46985, .49846, .5287, .562,
    .59841, .64598, .68834, .7327, .78978, .8373, .88708, .94744,
    1.0006, 1.0574, 1.1215, 1.1856, 1.2546, 1.3292, 1.4107, 1.4974,
    1.5913, 1.6931, 1.8028, 1.9212, 2.0492, 2.1874, 2.3365, 2.4978,
    2.6718, 2.8588, 3.062, 3.2818, 3.5188, 3.7752, 4.0527, 4.3542,
    4.6782, 5.0312, 5.4123, 5.8246, 6.2639, 6.7435, 7.2636, 7.8064,
    8.4091, 9.0696, 9.7677, 10.548, 11.4, 12.309, 13.324, 14.284,
    15.445, 16.687, 18.019, 19.403, 20.847, 22.366, 23.925, 25.537,
    27.213, 28.069, 29.864, 31.829, 33.988, 35.856, 38.829, 42.321,
    46.319, 50.606, 55.126, 59.126, 64.162, 68.708, 74.615, 81.176,
    87.739, 95.494, 103.83, 113.38, 123.99, 135.8, 148.7, 162.58,
    176.32, 192.6, 211.47, 232.7, 252.64, 277.41, 305.38, 333.44,
    366.42, 402.66, 442.14, 484.53, 526.42, 568.15, 558.78, 582.6,
    600.98, 613.94, 619.44, 618.24, 609.84, 595.96, 484.86, 475.59,
    478.49, 501.56, 552.19, 628.44, 630.39, 658.92, 671.96, 562.7,
    545.88, 423.43, 400.14, 306.59, 294.13, 246.8, 226.51, 278.21,
    314.39, 347.22, 389.13, 433.16, 477.48, 521.67, 560.54, 683.6,
    696.37, 695.91, 683.1, 658.24, 634.89, 698.85, 742.87, 796.66,
    954.49, 1009.5, 1150.5, 1179.1, 1267.9, 1272.4, 1312.7, 1330.4,
    1331.6, 1315.8, 1308.3, 1293.3, 1274.6, 1249.5, 1213.2, 1172.1,
    1124.4, 930.33, 893.36, 871.27, 883.54, 940.76, 1036., 1025.6,
    1053.1, 914.51, 894.15, 865.03, 670.63, 508.41, 475.15, 370.85,
    361.06, 319.38, 312.75, 331.87, 367.13, 415., 467.94, 525.49,
    578.41, 624.66, 794.82, 796.97, 780.29, 736.49, 670.18, 603.75,
    659.67, 679.8, 857.12, 884.05, 900.65, 1046.1, 1141.9, 1083.,
    1089.2, 1e3, 947.08, 872.31, 787.91, 704.75, 624.93, 553.68,
    489.91, 434.21, 385.64, 343.3, 306.42, 274.18, 245.94, 221.11,
    199.23, 179.88, 162.73, 147.48, 133.88, 121.73, 110.86, 101.1,
    92.323, 84.417, 77.281, 70.831, 64.991, 59.694, 54.884, 50.509,
    46.526, 42.893, 39.58, 36.549, 33.776, 31.236, 28.907, 26.77,
    24.805, 23., 21.339, 19.81, 18.404, 17.105, 15.909, 14.801,
    13.778, 12.83, 11.954, 11.142, 10.389, 9.691, 9.0434, 8.4423,
    7.8842, 7.3657, 6.8838, 6.4357, 6.0189, 5.6308, 5.2696, 4.9332,
    4.6198, 4.3277, 4.0553, 3.8012, 3.5639, 3.3424, 3.1355, 2.9422,
    2.7614, 2.5924, 2.4343, 2.2864, 2.148, 2.0184, 1.8971, 1.7835,
    1.677, 1.5773, 1.4838, 1.3961, 1.3139, 1.2369, 1.1645, 1.0966,
    1.0329, .97309, .91686, .86406, .81439, .76767, .72381, .68252,
    .64359, .60695, .57247, .54008, .50957, .48092, .45401, .42862,
    .40465, .38202, .36072, .34052, .3216, .30386, .28711, .27135,
    .25651, .24252, .2293, .21689, .20517, .19416, .18381, .17396,
    .16469
};

static double co2230[2001] = { 2.743e-5, 2.8815e-5, 3.027e-5, 3.1798e-5,
  3.3405e-5, 3.5094e-5, 3.6869e-5, 3.8734e-5, 4.0694e-5, 4.2754e-5,
  4.492e-5, 4.7196e-5, 4.9588e-5, 5.2103e-5, 5.4747e-5, 5.7525e-5,
  6.0446e-5, 6.3516e-5, 6.6744e-5, 7.0137e-5, 7.3704e-5, 7.7455e-5,
  8.1397e-5, 8.5543e-5, 8.9901e-5, 9.4484e-5, 9.9302e-5, 1.0437e-4,
```

```
    1.097e-4, 1.153e-4, 1.2119e-4, 1.2738e-4, 1.3389e-4, 1.4074e-4,
    1.4795e-4, 1.5552e-4, 1.6349e-4, 1.7187e-4, 1.8068e-4, 1.8995e-4,
    1.997e-4, 2.0996e-4, 2.2075e-4, 2.321e-4, 2.4403e-4, 2.5659e-4,
    2.698e-4, 2.837e-4, 2.9832e-4, 3.137e-4, 3.2988e-4, 3.4691e-4,
    3.6483e-4, 3.8368e-4, 4.0351e-4, 4.2439e-4, 4.4635e-4, 4.6947e-4,
    4.9379e-4, 5.1939e-4, 5.4633e-4, 5.7468e-4, 6.0452e-4, 6.3593e-4,
    6.69e-4, 7.038e-4, 7.4043e-4, 7.79e-4, 8.1959e-4, 8.6233e-4,
    9.0732e-4, 9.5469e-4, .0010046, .0010571, .0011124, .0011706,
    .0012319, .0012964, .0013644, .001436, .0015114, .0015908,
    .0016745, .0017625, .0018553, .0019531, .002056, .0021645,
    .0022788, .0023992, .002526, .0026596, .0028004, .0029488,
    .0031052, .0032699, .0034436, .0036265, .0038194, .0040227,
    .0042369, .0044628, .0047008, .0049518, .0052164, .0054953,
    .0057894, .0060995, .0064265, .0067713, .007135, .0075184,
    .0079228, .0083494, .0087993, .0092738, .0097745, .010303,
    .01086, .011448, .012068, .012722, .013413, .014142, .014911,
    .015723, .01658, .017484, .018439, .019447, .020511, .021635,
    .022821, .024074, .025397, .026794, .02827, .029829, .031475,
    .033215, .035052, .036994, .039045, .041213, .043504, .045926,
    .048485, .05119, .05405, .057074, .060271, .063651, .067225,
    .071006, .075004, .079233, .083708, .088441, .093449, .098749,
    .10436, .11029, .11657, .12322, .13026, .13772, .14561, .15397,
    .16282, .1722, .18214, .19266, .20381, .21563, .22816, .24143,
    .2555, .27043, .28625, .30303, .32082, .3397, .35972, .38097,
    .40352, .42746, .45286, .47983, .50847, .53888, .57119, .6055,
    .64196, .6807, .72187, .76564, .81217, .86165, .91427, .97025,
    1.0298, 1.0932, 1.1606, 1.2324, 1.3088, 1.3902, 1.477, 1.5693,
    1.6678, 1.7727, 1.8845, 2.0038, 2.131, 2.2666, 2.4114, 2.5659,
    2.7309, 2.907, 3.0951, 3.2961, 3.5109, 3.7405, 3.986, 4.2485,
    4.5293, 4.8299, 5.1516, 5.4961, 5.8651, 6.2605, 6.6842, 7.1385,
    7.6256, 8.1481, 8.7089, 9.3109, 9.9573, 10.652, 11.398, 12.2,
    13.063, 13.992, 14.99, 16.064, 17.222, 18.469, 19.813, 21.263,
    22.828, 24.516, 26.34, 28.31, 30.437, 32.738, 35.226, 37.914,
    40.824, 43.974, 47.377, 51.061, 55.011, 59.299, 63.961, 69.013,
    74.492, 80.444, 86.919, 93.836, 101.23, 109.25, 117.98, 127.47,
    137.81, 149.07, 161.35, 174.75, 189.42, 205.49, 223.02, 242.26,
    263.45, 286.75, 311.94, 340.01, 370.86, 404.92, 440.44, 480.27,
    525.17, 574.71, 626.22, 686.8, 754.38, 827.07, 913.38, 1011.7,
    1121.5, 1161.6, 1289.5, 1432.2, 1595.4, 1777., 1983.3, 2216.1,
    2485.7, 2788.3, 3101.5, 3481., 3902.1, 4257.1, 4740., 5272.8,
    5457.9, 5946.2, 6505.3, 6668.4, 7302.4, 8061.6, 9015.8, 9908.3,
    11613., 13956., 3249.6, 3243., 2901.5, 2841.3, 2729.6, 2558.2,
    1797.8, 1583.2, 1386., 1233.5, 787.74, 701.46, 761.66, 767.21,
    722.83, 1180.6, 1332.1, 1461.6, 2032.9, 2166., 2255.9, 2294.7,
    2587.2, 2396.5, 2122.4, 12553., 10784., 9832.5, 8827.3, 8029.1,
    7377.9, 7347.1, 6783.8, 6239.1, 5721.1, 5503., 4975.1, 4477.8,
    4021.3, 3676.8, 3275.3, 2914.9, 2597.4, 2328.2, 2075.4, 1857.6,
    1663.6, 1493.3, 1343.8, 1213.3, 1095.6, 1066.5, 958.91, 865.15,
    783.31, 714.35, 650.77, 593.98, 546.2, 499.9, 457.87, 421.75,
    387.61, 355.25, 326.62, 299.7, 275.21, 253.17, 232.83, 214.31,
    197.5, 182.08, 167.98, 155.12, 143.32, 132.5, 122.58, 113.48,
    105.11, 97.415, 90.182, 83.463, 77.281, 71.587, 66.341, 61.493,
    57.014, 53.062, 49.21, 45.663, 42.38, 39.348, 36.547, 33.967,
    31.573, 29.357, 27.314, 25.415, 23.658, 22.03, 20.524, 19.125,
    17.829, 16.627, 15.511, 14.476, 13.514, 12.618, 11.786, 11.013,
    10.294, 9.6246, 9.0018, 8.4218, 7.8816, 7.3783, 6.9092, 6.4719,
    6.0641, 5.6838, 5.3289, 4.998, 4.6893, 4.4014, 4.1325, 3.8813,
    3.6469, 3.4283, 3.2241, 3.035, 2.8576, 2.6922, 2.5348, 2.3896,
    2.2535, 2.1258, 2.0059, 1.8929, 1.7862, 1.6854, 1.5898, 1.4992,
    1.4017, 1.3218, 1.2479, 1.1809, 1.1215, 1.0693, 1.0116, .96016,
    .9105, .84859, .80105, .74381, .69982, .65127, .60899, .57843,
    .54592, .51792, .49336, .47155, .45201, .43426, .41807, .40303,
    .38876, .3863, .37098, .35492, .33801, .32032, .30341, .29874,
    .29193, .28689, .29584, .29155, .29826, .29195, .29287, .2904,
    .28199, .27709, .27162, .26622, .26133, .25676, .25235, .23137,
    .22365, .21519, .20597, .19636, .18699, .16485, .16262, .16643,
    .17542, .18198, .18631, .16759, .16338, .13505, .1267, .10053,
    .092554, .074093, .062159, .055523, .054849, .05401, .05528,
    .058982, .07952, .08647, .093244, .099285, .10393, .10661,
    .12072, .11417, .10396, .093265, .089137, .088909, .10902,
    .11277, .13625, .13565, .14907, .14167, .1428, .13744, .12768,
    .11382, .10244, .091686, .08109, .071739, .063616, .056579,
    .050504, .045251, .040689, .036715, .033237, .030181, .027488,
    .025107, .022998, .021125, .01946, .017979, .016661, .015489,
    .014448, .013526, .012712, .011998, .011375, .010839, .010384,
    .010007, .0097053, .0094783, .0093257, .0092489, .0092504,
    .0093346, .0095077, .0097676, .01012, .01058, .011157, .011844,
    .012672, .013665, .014766, .015999, .017509, .018972, .020444,
    .022311, .023742, .0249, .025599, .026981, .026462, .025143,
    .025066, .022814, .020458, .020026, .019142, .020189, .022371,
    .024163, .023728, .02199, .019506, .018591, .015576, .012784,
    .011744, .0094777, .0079148, .0070652, .006986, .0071758,
    .008086, .0098025, .01087, .013609, .016764, .018137, .021061,
    .023498, .023576, .023965, .022828, .021519, .021283, .023364,
    .026457, .029782, .030856, .033486, .035515, .035543, .036558,
    .037198, .037472, .037045, .037284, .03777, .038085, .038366,
    .038526, .038282, .038915, .037697, .035667, .032941, .031959,
```

```
      .028692, .025918, .024596, .025592, .027873, .028935, .02984,
      .028148, .025305, .021912, .020454, .016732, .013357, .01205,
      .009731, .0079881, .0077704, .0074387, .0083895, .0096776,
      .010326, .01293, .015955, .019247, .020145, .02267, .024231,
      .024184, .022131, .019784, .01955, .01971, .022119, .025116,
      .027978, .028107, .029808, .030701, .029164, .028551, .027286,
      .024946, .023259, .020982, .019221, .017471, .015643, .014074,
      .01261, .011301, .010116, .0090582, .0081036, .0072542, .0065034,
      .0058436, .0052571, .0047321, .0042697, .0038607, .0034977,
      .0031747, .0028864, .0026284, .002397, .002189, .0020017,
      .0018326, .0016798, .0015414, .0014159, .0013019, .0011983,
      .0011039, .0010177, 9.391e-4, 8.6717e-4, 8.0131e-4, 7.4093e-4,
      6.8553e-4, 6.3464e-4, 5.8787e-4, 5.4487e-4, 5.0533e-4, 4.69e-4,
      4.3556e-4, 4.0474e-4, 3.7629e-4, 3.5e-4, 3.2569e-4, 3.032e-4,
      2.8239e-4, 2.6314e-4, 2.4535e-4, 2.2891e-4, 2.1374e-4, 1.9975e-4,
      1.8685e-4, 1.7498e-4, 1.6406e-4, 1.5401e-4, 1.4479e-4, 1.3633e-4,
      1.2858e-4, 1.2148e-4, 1.1499e-4, 1.0907e-4, 1.0369e-4, 9.8791e-5,
      9.4359e-5, 9.0359e-5, 8.6766e-5, 8.3555e-5, 8.0703e-5, 7.8192e-5,
      7.6003e-5, 7.4119e-5, 7.2528e-5, 7.1216e-5, 7.0171e-5, 6.9385e-5,
      6.8848e-5, 6.8554e-5, 6.8496e-5, 6.8669e-5, 6.9069e-5, 6.9694e-5,
      7.054e-5, 7.1608e-5, 7.2896e-5, 7.4406e-5, 7.6139e-5, 7.8097e-5,
      8.0283e-5, 8.2702e-5, 8.5357e-5, 8.8255e-5, 9.1402e-5, 9.4806e-5,
      9.8473e-5, 1.0241e-4, 1.0664e-4, 1.1115e-4, 1.1598e-4, 1.2112e-4,
      1.2659e-4, 1.3241e-4, 1.3859e-4, 1.4515e-4, 1.521e-4, 1.5947e-4,
      1.6728e-4, 1.7555e-4, 1.8429e-4, 1.9355e-4, 2.0334e-4, 2.1369e-4,
      2.2463e-4, 2.3619e-4, 2.4841e-4, 2.6132e-4, 2.7497e-4, 2.8938e-4,
      3.0462e-4, 3.2071e-4, 3.3771e-4, 3.5567e-4, 3.7465e-4, 3.947e-4,
      4.1588e-4, 4.3828e-4, 4.6194e-4, 4.8695e-4, 5.1338e-4, 5.4133e-4,
      5.7087e-4, 6.0211e-4, 6.3515e-4, 6.701e-4, 7.0706e-4, 7.4617e-4,
      7.8756e-4, 8.3136e-4, 8.7772e-4, 9.2681e-4, 9.788e-4, .0010339,
      .0010922, .001154, .0012195, .0012889, .0013626, .0014407,
      .0015235, .0016114, .0017048, .0018038, .001909, .0020207,
      .0021395, .0022657, .0023998, .0025426, .0026944, .002856,
      .0030281, .0032114, .0034068, .003615, .0038371, .004074,
      .004327, .0045971, .0048857, .0051942, .0055239, .0058766,
      .0062538, .0066573, .0070891, .007551, .0080455, .0085747,
      .0091412, .0097481, .010397, .011092, .011837, .012638, .013495,
      .014415, .01541, .016475, .017621, .018857, .020175, .02162,
      .023185, .024876, .02672, .028732, .030916, .033319, .035939,
      .038736, .041847, .04524, .048715, .052678, .056977, .061203,
      .066184, .07164, .076952, .083477, .090674, .098049, .10697,
      .1169, .1277, .14011, .15323, .1684, .18601, .20626, .22831,
      .25417, .28407, .31405, .34957, .38823, .41923, .46026, .50409,
      .51227, .54805, .57976, .53818, .55056, .557, .46741, .46403,
      .4636, .42265, .45166, .49852, .56663, .34306, .17779, .17697,
      .18346, .19129, .20014, .21778, .23604, .25649, .28676, .31238,
      .33856, .39998, .4288, .46568, .56654, .60786, .64473, .76466,
      .7897, .80778, .86443, .85736, .84798, .84157, 1.1385, 1.2446,
      1.1923, 1.1552, 1.1338, 1.1266, 1.1292, 1.1431, 1.1683, 1.2059,
      1.2521, 1.3069, 1.3712, 1.4471, 1.5275, 1.6165, 1.7145, 1.8189,
      1.9359, 2.065, 2.2007, 2.3591, 2.5362, 2.7346, 2.9515, 3.2021,
      3.4851, 3.7935, 4.0694, 4.4463, 4.807, 5.2443, 5.7178, 6.2231,
      6.4796, 6.9461, 7.4099, 7.3652, 7.7182, 8.048, 7.7373, 8.0363,
      8.3855, 8.8044, 9.0257, 9.8574, 10.948, 10.563, 6.8979, 7.0744,
      7.4121, 7.7663, 8.1768, 8.6243, 9.1437, 9.7847, 10.182, 10.849,
      11.572, 12.602, 13.482, 14.431, 15.907, 16.983, 18.11, 19.884,
      21.02, 22.18, 23.355, 24.848, 25.954, 27.13, 30.186, 34.893,
      35.682, 36.755, 38.111, 39.703, 41.58, 43.606, 45.868, 48.573,
      51.298, 54.291, 57.559, 61.116, 64.964, 69.124, 73.628, 78.471,
      83.683, 89.307, 95.341, 101.84, 108.83, 116.36, 124.46, 133.18,
      142.57, 152.79, 163.69, 175.43, 188.11, 201.79, 216.55, 232.51,
      249.74, 268.38, 288.54, 310.35, 333.97, 359.55, 387.26, 417.3,
      449.88, 485.2, 523.54, 565.14, 610.28, 659.31, 712.56, 770.43,
      833.36, 901.82, 976.36, 1057.6, 1146.8, 1243.8, 1350., 1466.3,
      1593.6, 1732.7, 1884.1, 2049.1, 2228.2, 2421.9, 2629.4, 2853.7,
      3094.4, 3351.1, 3622.3, 3829.8, 4123.1, 4438.3, 4777.2, 5144.1,
      5545.4, 5990.5, 6404.5, 6996.8, 7687.6, 8482.9, 9349.4, 10203.,
      11223., 12358., 13493., 14916., 16416., 18236., 20222., 22501.,
      25102., 28358., 31707., 35404., 39538., 43911., 48391., 53193.,
      58028., 58082., 61276., 64193., 66294., 67480., 67921., 67423.,
      66254., 64341., 51737., 51420., 53072., 58145., 66195., 65358.,
      67377., 67869., 53509., 50553., 35737., 32425., 21704., 19974.,
      14457., 12142., 16798., 19489., 23049., 27270., 31910., 36457.,
      40877., 44748., 47876., 59793., 58626., 55454., 50337., 44893.,
      50228., 52216., 54747., 69541., 70455., 81014., 77694., 80533.,
      73953., 70927., 65539., 59002., 52281., 45953., 40292., 35360.,
      31124., 27478., 24346., 21647., 19308., 17271., 15491., 13927.,
      12550., 11331., 10250., 9288.8, 8431.4, 7664.9, 6978.3, 6361.8,
      5807.4, 5307.7, 4856.8, 4449., 4079.8, 3744.9, 3440.8, 3164.2,
      2912.3, 2682.7, 2473., 2281.4, 2106., 1945.3, 1797.9, 1662.5,
      1538.1, 1423.6, 1318.1, 1221., 1131.5, 1049., 972.99, 902.87,
      838.01, 777.95, 722.2, 670.44, 622.35, 577.68, 536.21, 497.76,
      462.12, 429.13, 398.61, 370.39, 344.29, 320.16, 297.85, 277.2,
      258.08, 240.38, 223.97, 208.77, 194.66, 181.58, 169.43, 158.15,
      147.67, 137.92, 128.86, 120.44, 112.6, 105.3, 98.499, 92.166,
      86.264, 80.763, 75.632, 70.846, 66.381, 62.213, 58.321, 54.685,
      51.288, 48.114, 45.145, 42.368, 39.772, 37.341, 35.065, 32.937,
```

```
30.943, 29.077, 27.33, 25.693, 24.158, 22.717, 21.367, 20.099,
18.909, 17.792, 16.744, 15.761, 14.838, 13.971, 13.157, 12.393,
11.676, 11.003, 10.369, 9.775, 9.2165, 8.6902, 8.1963, 7.7314,
7.2923, 6.8794, 6.4898, 6.122, 5.7764, 5.4525, 5.1484, 4.8611,
4.5918, 4.3379, 4.0982, 3.8716, 3.6567, 3.4545, 3.2634, 3.0828,
2.9122, 2.7512, 2.5993, 2.4561, 2.3211, 2.1938, 2.0737, 1.9603,
1.8534, 1.7525, 1.6572, 1.5673, 1.4824, 1.4022, 1.3265, 1.2551,
1.1876, 1.1239, 1.0637, 1.0069, .9532, .90248, .85454, .80921,
.76631, .72569, .6872, .65072, .61635, .5836, .55261, .52336,
.49581, .46998, .44559, .42236, .40036, .37929, .35924, .34043,
.32238, .30547, .28931, .27405, .25975, .24616, .23341, .22133,
.20997, .19924, .18917, .17967, .17075, .16211, .15411, .14646,
.13912, .13201, .12509, .11857, .11261, .10698, .10186, .097039,
.092236, .087844, .083443, .07938, .075452, .071564, .067931,
.064389, .061078, .057901, .054921, .052061, .049364, .046789,
.04435, .042044, .039866, .037808, .035863, .034023, .032282,
.030634, .029073, .027595, .026194, .024866, .023608, .022415,
.021283, .02021, .019193, .018228, .017312, .016443, .015619,
.014837, .014094, .01339, .012721, .012086, .011483, .010911,
.010368, .009852, .0093623, .0088972, .0084556, .0080362,
.0076379, .0072596, .0069003, .006559, .0062349, .0059269,
.0056344, .0053565, .0050925, .0048417, .0046034, .004377,
.0041618, .0039575, .0037633, .0035788, .0034034, .0032368,
.0030785, .002928, .0027851, .0026492, .0025201, .0023975,
.0022809, .0021701, .0020649, .0019649, .0018699, .0017796,
.0016938, .0016122, .0015348, .0014612, .0013913, .001325,
.0012619, .0012021, .0011452, .0010913, .0010401, 9.9149e-4,
9.454e-4, 9.0169e-4, 8.6024e-4, 8.2097e-4, 7.8377e-4, 7.4854e-4,
7.1522e-4, 6.8371e-4, 6.5393e-4, 6.2582e-4, 5.9932e-4, 5.7435e-4,
5.5087e-4, 5.2882e-4, 5.0814e-4, 4.8881e-4, 4.7076e-4, 4.5398e-4,
4.3843e-4, 4.2407e-4, 4.109e-4, 3.9888e-4, 3.88e-4, 3.7826e-4,
3.6963e-4, 3.6213e-4, 3.5575e-4, 3.505e-4, 3.464e-4, 3.4346e-4,
3.4173e-4, 3.4125e-4, 3.4206e-4, 3.4424e-4, 3.4787e-4, 3.5303e-4,
3.5986e-4, 3.6847e-4, 3.7903e-4, 3.9174e-4, 4.0681e-4, 4.2455e-4,
4.4527e-4, 4.6942e-4, 4.9637e-4, 5.2698e-4, 5.5808e-4, 5.9514e-4,
6.2757e-4, 6.689e-4, 7.1298e-4, 7.3955e-4, 7.8403e-4, 8.0449e-4,
8.5131e-4, 9.0256e-4, 9.3692e-4, .0010051, .0010846, .0011678,
.001282, .0014016, .0015355, .0016764, .0018272, .0020055,
.0021455, .0023421, .0024615, .0026786, .0028787, .0031259,
.0034046, .0036985, .0040917, .0043902, .0048349, .0049531,
.0052989, .0056148, .0052452, .0053357, .005333, .0045069,
.0043851, .004253, .003738, .0038084, .0039013, .0041505,
.0045372, .0050569, .0054507, .0061267, .0066122, .0072449,
.0078012, .0082651, .0076538, .0076573, .0076806, .0075227,
.0076269, .0063758, .006254, .0067749, .0067909, .0068231,
.0072143, .0072762, .0072954, .007679, .0075107, .0073658,
.0072441, .0071074, .0070378, .007176, .0072472, .0075844,
.0079291, .008412, .0090165, .010688, .011535, .012375, .013166,
.013895, .015567, .016011, .016392, .016737, .017043, .017731,
.018031, .018419, .018877, .019474, .019868, .020604, .021538,
.022653, .023869, .025288, .026879, .028547, .030524, .03274,
.035132, .03769, .040567, .043793, .047188, .049962, .053542,
.057205, .060776, .061489, .064419, .067124, .065945, .068487,
.071209, .074783, .077039, .082444, .08902, .09692, .10617,
.11687, .12952, .12362, .13498, .14412, .15492, .16519, .1744,
.17096, .17714, .18208, .17363, .17813, .18564, .18295, .19045,
.20252, .20815, .21844, .22929, .24229, .25321, .26588, .2797,
.29465, .31136, .32961, .36529, .38486, .41027, .43694, .4667,
.49943, .54542, .58348, .62303, .67633, .71755, .76054, .81371,
.85934, .90841, .96438, 1.0207, 1.0821, 1.1491, 1.2226, 1.3018,
1.388, 1.4818, 1.5835, 1.6939, 1.8137, 1.9435, 2.0843, 2.237,
2.4026, 2.5818, 2.7767, 2.9885, 3.2182, 3.4679, 3.7391, 4.0349,
4.3554, 4.7053, 5.0849, 5.4986, 5.9436, 6.4294, 6.9598, 7.5203,
8.143, 8.8253, 9.5568, 10.371, 11.267, 12.233, 13.31, 14.357,
15.598, 16.93, 18.358, 19.849, 21.408, 23.04, 24.706, 26.409,
28.153, 28.795, 30.549, 32.43, 34.49, 36.027, 38.955, 42.465,
46.565, 50.875, 55.378, 59.002, 63.882, 67.949, 73.693, 80.095,
86.403, 94.264, 102.65, 112.37, 123.3, 135.54, 149.14, 163.83,
179.17, 196.89, 217.91, 240.94, 264.13, 292.39, 324.83, 358.21,
397.16, 440.5, 488.6, 541.04, 595.3, 650.43, 652.03, 688.74,
719.47, 743.54, 757.68, 762.35, 756.43, 741.42, 595.43, 580.97,
580.83, 605.68, 667.88, 764.49, 759.93, 789.12, 798.17, 645.66,
615.65, 455.05, 421.09, 306.45, 289.14, 235.7, 215.52, 274.57,
316.53, 357.73, 409.89, 465.06, 521.84, 579.02, 630.64, 794.46,
813., 813.56, 796.25, 761.57, 727.97, 812.14, 866.75, 932.5,
1132.8, 1194.8, 1362.2, 1387.2, 1482.3, 1479.7, 1517.9, 1533.1,
1534.2, 1523.3, 1522.5, 1515.5, 1505.2, 1486.5, 1454., 1412.,
1358.8, 1107.8, 1060.9, 1033.5, 1048.2, 1122.4, 1248.9, 1227.1,
1255.4, 1058.9, 1020.7, 970.59, 715.24, 512.56, 468.47, 349.3,
338.26, 299.22, 301.26, 332.38, 382.08, 445.49, 515.87, 590.85,
662.3, 726.05, 955.59, 964.11, 945.17, 891.48, 807.11, 720.9,
803.36, 834.46, 1073.9, 1107.1, 1123.6, 1296., 1393.7, 1303.1,
1284.3, 1161.8, 1078.8, 976.13, 868.72, 767.4, 674.72, 593.73,
523.12, 462.24, 409.75, 364.34, 325., 290.73, 260.76, 234.46,
211.28, 190.78, 172.61, 156.44, 142.01, 129.12, 117.57, 107.2,
97.877, 89.47, 81.882, 75.021, 68.807, 63.171, 58.052, 53.396,
49.155, 45.288, 41.759, 38.531, 35.576, 32.868, 30.384, 28.102,
```

```
    26.003, 24.071, 22.293, 20.655, 19.147, 17.756, 16.476, 15.292,
    14.198, 13.183, 12.241, 11.367, 10.554, 9.7989, 9.0978, 8.4475,
    7.845, 7.2868, 6.7704, 6.2927, 5.8508, 5.4421, 5.064, 4.714,
    4.3902, 4.0902, 3.8121, 3.5543, 3.315, 3.093, 2.8869, 2.6953,
    2.5172, 2.3517, 2.1977, 2.0544, 1.9211, 1.7969, 1.6812, 1.5735,
    1.4731, 1.3794, 1.2921, 1.2107, 1.1346, 1.0637, .99744, .93554,
    .87771, .82368, .77313, .72587, .6816, .64014, .60134, .565,
    .53086, .49883, .46881, .44074, .4144, .38979, .36679, .34513,
    .32474, .30552, .28751, .27045, .25458, .23976, .22584, .21278,
    .20051, .18899, .17815, .16801, .15846, .14954, .14117, .13328,
    .12584
  };

  /* Get CO2 continuum absorption... */
  const double xw = nu / 2 + 1;
  if (xw >= 1 && xw < 2001) {
    const int iw = (int) xw;
    const double dw = xw - iw;
    const double ew = 1 - dw;
    const double cw296 = ew * co2296[iw - 1] + dw * co2296[iw];
    const double cw260 = ew * co2260[iw - 1] + dw * co2260[iw];
    const double cw230 = ew * co2230[iw - 1] + dw * co2230[iw];
    const double dt230 = t - 230;
    const double dt260 = t - 260;
    const double dt296 = t - 296;
    const double ctw =
      dt260 * 5.050505e-4 * dt296 * cw230 -
      dt230 * 9.259259e-4 * dt296 * cw260 +
      dt230 * 4.208754e-4 * dt260 * cw296;
    return u / NA / 1000 * p / P0 * ctw;
  } else
    return 0;
}

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

double ctmh2o(
  const double nu,
  const double p,
  const double t,
  const double q,
  const double u) {

  static double h2o296[2001] = { .17, .1695, .172, .168, .1687, .1624, .1606,
    .1508, .1447, .1344, .1214, .1133, .1009, .09217, .08297, .06989,
    .06513, .05469, .05056, .04417, .03779, .03484, .02994, .0272,
    .02325, .02063, .01818, .01592, .01405, .01251, .0108, .009647,
    .008424, .007519, .006555, .00588, .005136, .004511, .003989,
    .003509, .003114, .00274, .002446, .002144, .001895, .001676,
    .001486, .001312, .001164, .001031, 9.129e-4, 8.106e-4, 7.213e-4,
    6.4e-4, 5.687e-4, 5.063e-4, 4.511e-4, 4.029e-4, 3.596e-4,
    3.22e-4, 2.889e-4, 2.597e-4, 2.337e-4, 2.108e-4, 1.907e-4,
    1.728e-4, 1.57e-4, 1.43e-4, 1.305e-4, 1.195e-4, 1.097e-4,
    1.009e-4, 9.307e-5, 8.604e-5, 7.971e-5, 7.407e-5, 6.896e-5,
    6.433e-5, 6.013e-5, 5.631e-5, 5.283e-5, 4.963e-5, 4.669e-5,
    4.398e-5, 4.148e-5, 3.917e-5, 3.702e-5, 3.502e-5, 3.316e-5,
    3.142e-5, 2.978e-5, 2.825e-5, 2.681e-5, 2.546e-5, 2.419e-5,
    2.299e-5, 2.186e-5, 2.079e-5, 1.979e-5, 1.884e-5, 1.795e-5,
    1.711e-5, 1.633e-5, 1.559e-5, 1.49e-5, 1.426e-5, 1.367e-5,
    1.312e-5, 1.263e-5, 1.218e-5, 1.178e-5, 1.143e-5, 1.112e-5,
    1.088e-5, 1.07e-5, 1.057e-5, 1.05e-5, 1.051e-5, 1.059e-5,
    1.076e-5, 1.1e-5, 1.133e-5, 1.18e-5, 1.237e-5, 1.308e-5,
    1.393e-5, 1.483e-5, 1.614e-5, 1.758e-5, 1.93e-5, 2.123e-5,
    2.346e-5, 2.647e-5, 2.93e-5, 3.279e-5, 3.745e-5, 4.152e-5,
    4.813e-5, 5.477e-5, 6.203e-5, 7.331e-5, 8.056e-5, 9.882e-5,
    1.05e-4, 1.21e-4, 1.341e-4, 1.572e-4, 1.698e-4, 1.968e-4,
    2.175e-4, 2.431e-4, 2.735e-4, 2.867e-4, 3.19e-4, 3.371e-4,
    3.554e-4, 3.726e-4, 3.837e-4, 3.878e-4, 3.864e-4, 3.858e-4,
    3.841e-4, 3.852e-4, 3.815e-4, 3.762e-4, 3.618e-4, 3.579e-4,
    3.45e-4, 3.202e-4, 3.018e-4, 2.785e-4, 2.602e-4, 2.416e-4,
    2.097e-4, 1.939e-4, 1.689e-4, 1.498e-4, 1.308e-4, 1.17e-4,
    1.011e-4, 9.237e-5, 7.909e-5, 7.006e-5, 6.112e-5, 5.401e-5,
    4.914e-5, 4.266e-5, 3.963e-5, 3.316e-5, 3.037e-5, 2.598e-5,
    2.294e-5, 2.066e-5, 1.813e-5, 1.583e-5, 1.423e-5, 1.247e-5,
    1.116e-5, 9.76e-6, 8.596e-6, 7.72e-6, 6.825e-6, 6.108e-6,
    5.366e-6, 4.733e-6, 4.229e-6, 3.731e-6, 3.346e-6, 2.972e-6,
    2.628e-6, 2.356e-6, 2.102e-6, 1.878e-6, 1.678e-6, 1.507e-6,
    1.348e-6, 1.21e-6, 1.089e-6, 9.806e-7, 8.857e-7, 8.004e-7,
    7.261e-7, 6.599e-7, 6.005e-7, 5.479e-7, 5.011e-7, 4.595e-7,
    4.219e-7, 3.885e-7, 3.583e-7, 3.314e-7, 3.071e-7, 2.852e-7,
    2.654e-7, 2.474e-7, 2.311e-7, 2.162e-7, 2.026e-7, 1.902e-7,
    1.788e-7, 1.683e-7, 1.587e-7, 1.497e-7, 1.415e-7, 1.338e-7,
    1.266e-7, 1.2e-7, 1.138e-7, 1.08e-7, 1.027e-7, 9.764e-8,
    9.296e-8, 8.862e-8, 8.458e-8, 8.087e-8, 7.744e-8, 7.429e-8,
    7.145e-8, 6.893e-8, 6.664e-8, 6.468e-8, 6.322e-8, 6.162e-8,
    6.07e-8, 5.992e-8, 5.913e-8, 5.841e-8, 5.796e-8, 5.757e-8,
    5.746e-8, 5.731e-8, 5.679e-8, 5.577e-8, 5.671e-8, 5.656e-8,
```

```
    5.594e-8, 5.593e-8, 5.602e-8, 5.62e-8, 5.693e-8, 5.725e-8,
    5.858e-8, 6.037e-8, 6.249e-8, 6.535e-8, 6.899e-8, 7.356e-8,
    7.918e-8, 8.618e-8, 9.385e-8, 1.039e-7, 1.158e-7, 1.29e-7,
    1.437e-7, 1.65e-7, 1.871e-7, 2.121e-7, 2.427e-7, 2.773e-7,
    3.247e-7, 3.677e-7, 4.037e-7, 4.776e-7, 5.101e-7, 6.214e-7,
    6.936e-7, 7.581e-7, 8.486e-7, 9.355e-7, 9.942e-7, 1.063e-6,
    1.123e-6, 1.191e-6, 1.215e-6, 1.247e-6, 1.26e-6, 1.271e-6,
    1.284e-6, 1.317e-6, 1.323e-6, 1.349e-6, 1.353e-6, 1.362e-6,
    1.344e-6, 1.329e-6, 1.336e-6, 1.327e-6, 1.325e-6, 1.359e-6,
    1.374e-6, 1.415e-6, 1.462e-6, 1.526e-6, 1.619e-6, 1.735e-6,
    1.863e-6, 2.034e-6, 2.265e-6, 2.482e-6, 2.756e-6, 3.103e-6,
    3.466e-6, 3.832e-6, 4.378e-6, 4.913e-6, 5.651e-6, 6.311e-6,
    7.169e-6, 8.057e-6, 9.253e-6, 1.047e-5, 1.212e-5, 1.36e-5,
    1.569e-5, 1.776e-5, 2.02e-5, 2.281e-5, 2.683e-5, 2.994e-5,
    3.488e-5, 3.896e-5, 4.499e-5, 5.175e-5, 6.035e-5, 6.34e-5,
    7.281e-5, 7.923e-5, 8.348e-5, 9.631e-5, 1.044e-4, 1.102e-4,
    1.176e-4, 1.244e-4, 1.283e-4, 1.326e-4, 1.4e-4, 1.395e-4,
    1.387e-4, 1.363e-4, 1.314e-4, 1.241e-4, 1.228e-4, 1.148e-4,
    1.086e-4, 1.018e-4, 8.89e-5, 8.316e-5, 7.292e-5, 6.452e-5,
    5.625e-5, 5.045e-5, 4.38e-5, 3.762e-5, 3.29e-5, 2.836e-5,
    2.485e-5, 2.168e-5, 1.895e-5, 1.659e-5, 1.453e-5, 1.282e-5,
    1.132e-5, 1.001e-5, 8.836e-6, 7.804e-6, 6.922e-6, 6.116e-6,
    5.429e-6, 4.824e-6, 4.278e-6, 3.788e-6, 3.371e-6, 2.985e-6,
    2.649e-6, 2.357e-6, 2.09e-6, 1.858e-6, 1.647e-6, 1.462e-6,
    1.299e-6, 1.155e-6, 1.028e-6, 9.142e-7, 8.132e-7, 7.246e-7,
    6.451e-7, 5.764e-7, 5.151e-7, 4.603e-7, 4.121e-7, 3.694e-7,
    3.318e-7, 2.985e-7, 2.69e-7, 2.428e-7, 2.197e-7, 1.992e-7,
    1.81e-7, 1.649e-7, 1.506e-7, 1.378e-7, 1.265e-7, 1.163e-7,
    1.073e-7, 9.918e-8, 9.191e-8, 8.538e-8, 7.949e-8, 7.419e-8,
    6.94e-8, 6.508e-8, 6.114e-8, 5.761e-8, 5.437e-8, 5.146e-8,
    4.89e-8, 4.636e-8, 4.406e-8, 4.201e-8, 4.015e-8, 3.84e-8,
    3.661e-8, 3.51e-8, 3.377e-8, 3.242e-8, 3.13e-8, 3.015e-8,
    2.918e-8, 2.83e-8, 2.758e-8, 2.707e-8, 2.656e-8, 2.619e-8,
    2.609e-8, 2.615e-8, 2.63e-8, 2.675e-8, 2.745e-8, 2.842e-8,
    2.966e-8, 3.125e-8, 3.318e-8, 3.565e-8, 3.85e-8, 4.191e-8,
    4.59e-8, 5.059e-8, 5.607e-8, 6.239e-8, 6.958e-8, 7.796e-8,
    8.773e-8, 9.88e-8, 1.114e-7, 1.258e-7, 1.422e-7, 1.61e-7,
    1.822e-7, 2.06e-7, 2.337e-7, 2.645e-7, 2.996e-7, 3.393e-7,
    3.843e-7, 4.363e-7, 4.935e-7, 5.607e-7, 6.363e-7, 7.242e-7,
    8.23e-7, 9.411e-7, 1.071e-6, 1.232e-6, 1.402e-6, 1.6e-6, 1.82e-6,
    2.128e-6, 2.386e-6, 2.781e-6, 3.242e-6, 3.653e-6, 4.323e-6,
    4.747e-6, 5.321e-6, 5.919e-6, 6.681e-6, 7.101e-6, 7.983e-6,
    8.342e-6, 8.741e-6, 9.431e-6, 9.952e-6, 1.026e-5, 1.055e-5,
    1.095e-5, 1.095e-5, 1.087e-5, 1.056e-5, 1.026e-5, 9.715e-6,
    9.252e-6, 8.452e-6, 7.958e-6, 7.268e-6, 6.295e-6, 6.003e-6, 5e-6,
    4.591e-6, 3.983e-6, 3.479e-6, 3.058e-6, 2.667e-6, 2.293e-6,
    1.995e-6, 1.747e-6, 1.517e-6, 1.335e-6, 1.165e-6, 1.028e-6,
    9.007e-7, 7.956e-7, 7.015e-7, 6.192e-7, 5.491e-7, 4.859e-7,
    4.297e-7, 3.799e-7, 3.38e-7, 3.002e-7, 2.659e-7, 2.366e-7,
    2.103e-7, 1.861e-7, 1.655e-7, 1.469e-7, 1.309e-7, 1.162e-7,
    1.032e-7, 9.198e-8, 8.181e-8, 7.294e-8, 6.516e-8, 5.787e-8,
    5.163e-8, 4.612e-8, 4.119e-8, 3.695e-8, 3.308e-8, 2.976e-8,
    2.67e-8, 2.407e-8, 2.171e-8, 1.965e-8, 1.78e-8, 1.617e-8,
    1.47e-8, 1.341e-8, 1.227e-8, 1.125e-8, 1.033e-8, 9.524e-9,
    8.797e-9, 8.162e-9, 7.565e-9, 7.04e-9, 6.56e-9, 6.129e-9,
    5.733e-9, 5.376e-9, 5.043e-9, 4.75e-9, 4.466e-9, 4.211e-9,
    3.977e-9, 3.759e-9, 3.558e-9, 3.373e-9, 3.201e-9, 3.043e-9,
    2.895e-9, 2.76e-9, 2.635e-9, 2.518e-9, 2.411e-9, 2.314e-9,
    2.23e-9, 2.151e-9, 2.087e-9, 2.035e-9, 1.988e-9, 1.946e-9,
    1.927e-9, 1.916e-9, 1.916e-9, 1.933e-9, 1.966e-9, 2.018e-9,
    2.09e-9, 2.182e-9, 2.299e-9, 2.442e-9, 2.623e-9, 2.832e-9,
    3.079e-9, 3.368e-9, 3.714e-9, 4.104e-9, 4.567e-9, 5.091e-9,
    5.701e-9, 6.398e-9, 7.194e-9, 8.127e-9, 9.141e-9, 1.035e-8,
    1.177e-8, 1.338e-8, 1.508e-8, 1.711e-8, 1.955e-8, 2.216e-8,
    2.534e-8, 2.871e-8, 3.291e-8, 3.711e-8, 4.285e-8, 4.868e-8,
    5.509e-8, 6.276e-8, 7.262e-8, 8.252e-8, 9.4e-8, 1.064e-7,
    1.247e-7, 1.411e-7, 1.626e-7, 1.827e-7, 2.044e-7, 2.284e-7,
    2.452e-7, 2.854e-7, 3.026e-7, 3.278e-7, 3.474e-7, 3.693e-7,
    3.93e-7, 4.104e-7, 4.22e-7, 4.439e-7, 4.545e-7, 4.778e-7,
    4.812e-7, 5.018e-7, 4.899e-7, 5.075e-7, 5.073e-7, 5.171e-7,
    5.131e-7, 5.25e-7, 5.617e-7, 5.846e-7, 6.239e-7, 6.696e-7,
    7.398e-7, 8.073e-7, 9.15e-7, 1.009e-6, 1.116e-6, 1.264e-6,
    1.439e-6, 1.644e-6, 1.856e-6, 2.147e-6, 2.317e-6, 2.713e-6,
    2.882e-6, 2.99e-6, 3.489e-6, 3.581e-6, 4.033e-6, 4.26e-6,
    4.543e-6, 4.84e-6, 4.826e-6, 5.013e-6, 5.252e-6, 5.277e-6,
    5.306e-6, 5.236e-6, 5.123e-6, 5.171e-6, 4.843e-6, 4.615e-6,
    4.385e-6, 3.97e-6, 3.693e-6, 3.231e-6, 2.915e-6, 2.495e-6,
    2.144e-6, 1.91e-6, 1.639e-6, 1.417e-6, 1.226e-6, 1.065e-6,
    9.29e-7, 8.142e-7, 7.161e-7, 6.318e-7, 5.581e-7, 4.943e-7,
    4.376e-7, 3.884e-7, 3.449e-7, 3.06e-7, 2.712e-7, 2.412e-7,
    2.139e-7, 1.903e-7, 1.689e-7, 1.499e-7, 1.331e-7, 1.183e-7,
    1.05e-7, 9.362e-8, 8.306e-8, 7.403e-8, 6.578e-8, 5.853e-8,
    5.216e-8, 4.632e-8, 4.127e-8, 3.678e-8, 3.279e-8, 2.923e-8,
    2.612e-8, 2.339e-8, 2.094e-8, 1.877e-8, 1.686e-8, 1.516e-8,
    1.366e-8, 1.234e-8, 1.114e-8, 1.012e-8, 9.182e-9, 8.362e-9,
    7.634e-9, 6.981e-9, 6.406e-9, 5.888e-9, 5.428e-9, 5.021e-9,
    4.65e-9, 4.326e-9, 4.033e-9, 3.77e-9, 3.536e-9, 3.327e-9,
```

```
3.141e-9, 2.974e-9, 2.825e-9, 2.697e-9, 2.584e-9, 2.488e-9,
2.406e-9, 2.34e-9, 2.292e-9, 2.259e-9, 2.244e-9, 2.243e-9,
2.272e-9, 2.31e-9, 2.378e-9, 2.454e-9, 2.618e-9, 2.672e-9,
2.831e-9, 3.05e-9, 3.225e-9, 3.425e-9, 3.677e-9, 3.968e-9,
4.221e-9, 4.639e-9, 4.96e-9, 5.359e-9, 5.649e-9, 6.23e-9,
6.716e-9, 7.218e-9, 7.746e-9, 7.988e-9, 8.627e-9, 8.999e-9,
9.442e-9, 9.82e-9, 1.015e-8, 1.06e-8, 1.079e-8, 1.109e-8,
1.137e-8, 1.186e-8, 1.18e-8, 1.187e-8, 1.194e-8, 1.192e-8,
1.224e-8, 1.245e-8, 1.246e-8, 1.318e-8, 1.377e-8, 1.471e-8,
1.582e-8, 1.713e-8, 1.853e-8, 2.063e-8, 2.27e-8, 2.567e-8,
2.891e-8, 3.264e-8, 3.744e-8, 4.286e-8, 4.915e-8, 5.623e-8,
6.336e-8, 7.293e-8, 8.309e-8, 9.319e-8, 1.091e-7, 1.243e-7,
1.348e-7, 1.449e-7, 1.62e-7, 1.846e-7, 1.937e-7, 2.04e-7,
2.179e-7, 2.298e-7, 2.433e-7, 2.439e-7, 2.464e-7, 2.611e-7,
2.617e-7, 2.582e-7, 2.453e-7, 2.401e-7, 2.349e-7, 2.203e-7,
2.066e-7, 1.939e-7, 1.78e-7, 1.558e-7, 1.391e-7, 1.203e-7,
1.048e-7, 9.464e-8, 8.306e-8, 7.239e-8, 6.317e-8, 5.52e-8,
4.847e-8, 4.282e-8, 3.796e-8, 3.377e-8, 2.996e-8, 2.678e-8,
2.4e-8, 2.134e-8, 1.904e-8, 1.705e-8, 1.523e-8, 1.35e-8,
1.204e-8, 1.07e-8, 9.408e-9, 8.476e-9, 7.47e-9, 6.679e-9,
5.929e-9, 5.267e-9, 4.711e-9, 4.172e-9, 3.761e-9, 3.288e-9,
2.929e-9, 2.609e-9, 2.315e-9, 2.042e-9, 1.844e-9, 1.64e-9,
1.47e-9, 1.31e-9, 1.176e-9, 1.049e-9, 9.377e-10, 8.462e-10,
7.616e-10, 6.854e-10, 6.191e-10, 5.596e-10, 5.078e-10, 4.611e-10,
4.197e-10, 3.83e-10, 3.505e-10, 3.215e-10, 2.956e-10, 2.726e-10,
2.521e-10, 2.338e-10, 2.173e-10, 2.026e-10, 1.895e-10, 1.777e-10,
1.672e-10, 1.579e-10, 1.496e-10, 1.423e-10, 1.358e-10, 1.302e-10,
1.254e-10, 1.216e-10, 1.187e-10, 1.163e-10, 1.147e-10, 1.145e-10,
1.15e-10, 1.17e-10, 1.192e-10, 1.25e-10, 1.298e-10, 1.345e-10,
1.405e-10, 1.538e-10, 1.648e-10, 1.721e-10, 1.872e-10, 1.968e-10,
2.089e-10, 2.172e-10, 2.317e-10, 2.389e-10, 2.503e-10, 2.585e-10,
2.686e-10, 2.8e-10, 2.895e-10, 3.019e-10, 3.037e-10, 3.076e-10,
3.146e-10, 3.198e-10, 3.332e-10, 3.397e-10, 3.54e-10, 3.667e-10,
3.895e-10, 4.071e-10, 4.565e-10, 4.983e-10, 5.439e-10, 5.968e-10,
6.676e-10, 7.456e-10, 8.405e-10, 9.478e-10, 1.064e-9, 1.218e-9,
1.386e-9, 1.581e-9, 1.787e-9, 2.032e-9, 2.347e-9, 2.677e-9,
3.008e-9, 3.544e-9, 4.056e-9, 4.687e-9, 5.331e-9, 6.227e-9,
6.854e-9, 8.139e-9, 8.945e-9, 9.865e-9, 1.125e-8, 1.178e-8,
1.364e-8, 1.436e-8, 1.54e-8, 1.672e-8, 1.793e-8, 1.906e-8,
2.036e-8, 2.144e-8, 2.292e-8, 2.371e-8, 2.493e-8, 2.606e-8,
2.706e-8, 2.866e-8, 3.036e-8, 3.136e-8, 3.405e-8, 3.665e-8,
3.837e-8, 4.229e-8, 4.748e-8, 5.32e-8, 5.763e-8, 6.677e-8,
7.216e-8, 7.716e-8, 8.958e-8, 9.419e-8, 1.036e-7, 1.108e-7,
1.189e-7, 1.246e-7, 1.348e-7, 1.31e-7, 1.361e-7, 1.364e-7,
1.363e-7, 1.343e-7, 1.293e-7, 1.254e-7, 1.235e-7, 1.158e-7,
1.107e-7, 9.961e-8, 9.011e-8, 7.91e-8, 6.916e-8, 6.338e-8,
5.564e-8, 4.827e-8, 4.198e-8, 3.695e-8, 3.276e-8, 2.929e-8,
2.633e-8, 2.391e-8, 2.192e-8, 2.021e-8, 1.89e-8, 1.772e-8,
1.667e-8, 1.603e-8, 1.547e-8, 1.537e-8, 1.492e-8, 1.515e-8,
1.479e-8, 1.45e-8, 1.513e-8, 1.495e-8, 1.529e-8, 1.565e-8,
1.564e-8, 1.553e-8, 1.569e-8, 1.584e-8, 1.57e-8, 1.538e-8,
1.513e-8, 1.472e-8, 1.425e-8, 1.349e-8, 1.328e-8, 1.249e-8,
1.17e-8, 1.077e-8, 9.514e-9, 8.614e-9, 7.46e-9, 6.621e-9,
5.775e-9, 5.006e-9, 4.308e-9, 3.747e-9, 3.24e-9, 2.84e-9,
2.481e-9, 2.184e-9, 1.923e-9, 1.71e-9, 1.504e-9, 1.334e-9,
1.187e-9, 1.053e-9, 9.367e-10, 8.306e-10, 7.419e-10, 6.63e-10,
5.918e-10, 5.277e-10, 4.717e-10, 4.222e-10, 3.783e-10, 3.39e-10,
3.036e-10, 2.729e-10, 2.455e-10, 2.211e-10, 1.995e-10, 1.804e-10,
1.635e-10, 1.485e-10, 1.355e-10, 1.24e-10, 1.139e-10, 1.051e-10,
9.757e-11, 9.114e-11, 8.577e-11, 8.139e-11, 7.792e-11, 7.52e-11,
7.39e-11, 7.311e-11, 7.277e-11, 7.482e-11, 7.698e-11, 8.162e-11,
8.517e-11, 8.968e-11, 9.905e-11, 1.075e-10, 1.187e-10, 1.291e-10,
1.426e-10, 1.573e-10, 1.734e-10, 1.905e-10, 2.097e-10, 2.28e-10,
2.473e-10, 2.718e-10, 2.922e-10, 3.128e-10, 3.361e-10, 3.641e-10,
3.91e-10, 4.196e-10, 4.501e-10, 4.932e-10, 5.258e-10, 5.755e-10,
6.253e-10, 6.664e-10, 7.344e-10, 7.985e-10, 8.877e-10, 1.005e-9,
1.118e-9, 1.251e-9, 1.428e-9, 1.61e-9, 1.888e-9, 2.077e-9,
2.331e-9, 2.751e-9, 3.061e-9, 3.522e-9, 3.805e-9, 4.181e-9,
4.575e-9, 5.167e-9, 5.634e-9, 6.007e-9, 6.501e-9, 6.829e-9,
7.211e-9, 7.262e-9, 7.696e-9, 7.832e-9, 7.799e-9, 7.651e-9,
7.304e-9, 7.15e-9, 6.977e-9, 6.603e-9, 6.209e-9, 5.69e-9,
5.432e-9, 4.764e-9, 4.189e-9, 3.64e-9, 3.203e-9, 2.848e-9,
2.51e-9, 2.194e-9, 1.946e-9, 1.75e-9, 1.567e-9, 1.426e-9,
1.302e-9, 1.197e-9, 1.109e-9, 1.035e-9, 9.719e-10, 9.207e-10,
8.957e-10, 8.578e-10, 8.262e-10, 8.117e-10, 7.987e-10, 7.875e-10,
7.741e-10, 7.762e-10, 7.537e-10, 7.424e-10, 7.474e-10, 7.294e-10,
7.216e-10, 7.233e-10, 7.075e-10, 6.892e-10, 6.618e-10, 6.314e-10,
6.208e-10, 5.689e-10, 5.55e-10, 4.984e-10, 4.6e-10, 4.078e-10,
3.879e-10, 3.459e-10, 2.982e-10, 2.626e-10, 2.329e-10, 1.988e-10,
1.735e-10, 1.487e-10, 1.297e-10, 1.133e-10, 9.943e-11, 8.736e-11,
7.726e-11, 6.836e-11, 6.053e-11, 5.384e-11, 4.789e-11, 4.267e-11,
3.804e-11, 3.398e-11, 3.034e-11, 2.71e-11, 2.425e-11, 2.173e-11,
1.95e-11, 1.752e-11, 1.574e-11, 1.418e-11, 1.278e-11, 1.154e-11,
1.044e-11, 9.463e-12, 8.602e-12, 7.841e-12, 7.171e-12, 6.584e-12,
6.073e-12, 5.631e-12, 5.254e-12, 4.937e-12, 4.679e-12, 4.476e-12,
4.328e-12, 4.233e-12, 4.194e-12, 4.211e-12, 4.286e-12, 4.424e-12,
4.628e-12, 4.906e-12, 5.262e-12, 5.708e-12, 6.254e-12, 6.914e-12,
```

```
7.714e-12, 8.677e-12, 9.747e-12, 1.101e-11, 1.256e-11, 1.409e-11,
1.597e-11, 1.807e-11, 2.034e-11, 2.316e-11, 2.622e-11, 2.962e-11,
3.369e-11, 3.819e-11, 4.329e-11, 4.932e-11, 5.589e-11, 6.364e-11,
7.284e-11, 8.236e-11, 9.447e-11, 1.078e-10, 1.229e-10, 1.417e-10,
1.614e-10, 1.843e-10, 2.107e-10, 2.406e-10, 2.728e-10, 3.195e-10,
3.595e-10, 4.153e-10, 4.736e-10, 5.41e-10, 6.088e-10, 6.769e-10,
7.691e-10, 8.545e-10, 9.621e-10, 1.047e-9, 1.161e-9, 1.296e-9,
1.424e-9, 1.576e-9, 1.739e-9, 1.893e-9, 2.08e-9, 2.336e-9,
2.604e-9, 2.76e-9, 3.001e-9, 3.365e-9, 3.55e-9, 3.895e-9,
4.183e-9, 4.614e-9, 4.846e-9, 5.068e-9, 5.427e-9, 5.541e-9,
5.864e-9, 5.997e-9, 5.997e-9, 6.061e-9, 5.944e-9, 5.855e-9,
5.661e-9, 5.523e-9, 5.374e-9, 4.94e-9, 4.688e-9, 4.17e-9,
3.913e-9, 3.423e-9, 2.997e-9, 2.598e-9, 2.253e-9, 1.946e-9,
1.71e-9, 1.507e-9, 1.336e-9, 1.19e-9, 1.068e-9, 9.623e-10,
8.772e-10, 8.007e-10, 7.42e-10, 6.884e-10, 6.483e-10, 6.162e-10,
5.922e-10, 5.688e-10, 5.654e-10, 5.637e-10, 5.701e-10, 5.781e-10,
5.874e-10, 6.268e-10, 6.357e-10, 6.525e-10, 7.137e-10, 7.441e-10,
8.024e-10, 8.485e-10, 9.143e-10, 9.536e-10, 9.717e-10, 1.018e-9,
1.042e-9, 1.054e-9, 1.092e-9, 1.079e-9, 1.064e-9, 1.043e-9,
1.02e-9, 9.687e-10, 9.273e-10, 9.208e-10, 9.068e-10, 7.687e-10,
7.385e-10, 6.595e-10, 5.87e-10, 5.144e-10, 4.417e-10, 3.804e-10,
3.301e-10, 2.866e-10, 2.509e-10, 2.202e-10, 1.947e-10, 1.719e-10,
1.525e-10, 1.361e-10, 1.21e-10, 1.084e-10, 9.8e-11, 8.801e-11,
7.954e-11, 7.124e-11, 6.335e-11, 5.76e-11, 5.132e-11, 4.601e-11,
4.096e-11, 3.657e-11, 3.25e-11, 2.909e-11, 2.587e-11, 2.297e-11,
2.05e-11, 1.828e-11, 1.632e-11, 1.462e-11, 1.314e-11, 1.185e-11,
1.073e-11, 9.76e-12, 8.922e-12, 8.206e-12, 7.602e-12, 7.1e-12,
6.694e-12, 6.378e-12, 6.149e-12, 6.004e-12, 5.941e-12, 5.962e-12,
6.069e-12, 6.265e-12, 6.551e-12, 6.935e-12, 7.457e-12, 8.074e-12,
8.811e-12, 9.852e-12, 1.086e-11, 1.207e-11, 1.361e-11, 1.553e-11,
1.737e-11, 1.93e-11, 2.175e-11, 2.41e-11, 2.706e-11, 3.023e-11,
3.313e-11, 3.657e-11, 4.118e-11, 4.569e-11, 5.025e-11, 5.66e-11,
6.231e-11, 6.881e-11, 7.996e-11, 8.526e-11, 9.694e-11, 1.106e-10,
1.222e-10, 1.355e-10, 1.525e-10, 1.775e-10, 1.924e-10, 2.181e-10,
2.379e-10, 2.662e-10, 2.907e-10, 3.154e-10, 3.366e-10, 3.579e-10,
3.858e-10, 4.046e-10, 4.196e-10, 4.166e-10, 4.457e-10, 4.466e-10,
4.404e-10, 4.337e-10, 4.15e-10, 4.083e-10, 3.91e-10, 3.723e-10,
3.514e-10, 3.303e-10, 2.847e-10, 2.546e-10, 2.23e-10, 1.994e-10,
1.733e-10, 1.488e-10, 1.297e-10, 1.144e-10, 1.004e-10, 8.741e-11,
7.928e-11, 7.034e-11, 6.323e-11, 5.754e-11, 5.25e-11, 4.85e-11,
4.502e-11, 4.286e-11, 4.028e-11, 3.899e-11, 3.824e-11, 3.761e-11,
3.804e-11, 3.839e-11, 3.845e-11, 4.244e-11, 4.382e-11, 4.582e-11,
4.847e-11, 5.209e-11, 5.384e-11, 5.887e-11, 6.371e-11, 6.737e-11,
7.168e-11, 7.415e-11, 7.827e-11, 8.037e-11, 8.12e-11, 8.071e-11,
8.008e-11, 7.851e-11, 7.544e-11, 7.377e-11, 7.173e-11, 6.801e-11,
6.267e-11, 5.727e-11, 5.288e-11, 4.853e-11, 4.082e-11, 3.645e-11,
3.136e-11, 2.672e-11, 2.304e-11, 1.986e-11, 1.725e-11, 1.503e-11,
1.315e-11, 1.153e-11, 1.014e-11, 8.942e-12, 7.901e-12, 6.993e-12,
6.199e-12, 5.502e-12, 4.89e-12, 4.351e-12, 3.878e-12, 3.461e-12,
3.094e-12, 2.771e-12, 2.488e-12, 2.241e-12, 2.025e-12, 1.838e-12,
1.677e-12, 1.541e-12, 1.427e-12, 1.335e-12, 1.262e-12, 1.209e-12,
1.176e-12, 1.161e-12, 1.165e-12, 1.189e-12, 1.234e-12, 1.3e-12,
1.389e-12, 1.503e-12, 1.644e-12, 1.814e-12, 2.017e-12, 2.255e-12,
2.534e-12, 2.858e-12, 3.231e-12, 3.661e-12, 4.153e-12, 4.717e-12,
5.36e-12, 6.094e-12, 6.93e-12, 7.882e-12, 8.966e-12, 1.02e-11,
1.162e-11, 1.324e-11, 1.51e-11, 1.72e-11, 1.965e-11, 2.237e-11,
2.56e-11, 2.927e-11, 3.371e-11, 3.842e-11, 4.429e-11, 5.139e-11,
5.798e-11, 6.697e-11, 7.626e-11, 8.647e-11, 1.022e-10, 1.136e-10,
1.3e-10, 1.481e-10, 1.672e-10, 1.871e-10, 2.126e-10, 2.357e-10,
2.583e-10, 2.997e-10, 3.289e-10, 3.702e-10, 4.012e-10, 4.319e-10,
4.527e-10, 5.001e-10, 5.448e-10, 5.611e-10, 5.76e-10, 5.965e-10,
6.079e-10, 6.207e-10, 6.276e-10, 6.222e-10, 6.137e-10, 6e-10,
5.814e-10, 5.393e-10, 5.35e-10, 4.947e-10, 4.629e-10, 4.117e-10,
3.712e-10, 3.372e-10, 2.923e-10, 2.55e-10, 2.232e-10, 1.929e-10,
1.679e-10, 1.46e-10, 1.289e-10, 1.13e-10, 9.953e-11, 8.763e-11,
7.76e-11, 6.9e-11, 6.16e-11, 5.525e-11, 4.958e-11, 4.489e-11,
4.072e-11, 3.728e-11, 3.438e-11, 3.205e-11, 3.006e-11, 2.848e-11,
2.766e-11, 2.688e-11, 2.664e-11, 2.67e-11, 2.696e-11, 2.786e-11,
2.861e-11, 3.009e-11, 3.178e-11, 3.389e-11, 3.587e-11, 3.819e-11,
4.054e-11, 4.417e-11, 4.703e-11, 5.137e-11, 5.46e-11, 6.055e-11,
6.333e-11, 6.773e-11, 7.219e-11, 7.717e-11, 8.131e-11, 8.491e-11,
8.574e-11, 9.01e-11, 9.017e-11, 8.999e-11, 8.959e-11, 8.838e-11,
8.579e-11, 8.162e-11, 8.098e-11, 7.472e-11, 7.108e-11, 6.559e-11,
5.994e-11, 5.172e-11, 4.424e-11, 3.951e-11, 3.34e-11, 2.902e-11,
2.541e-11, 2.215e-11, 1.945e-11, 1.716e-11, 1.503e-11, 1.339e-11,
1.185e-11, 1.05e-11, 9.336e-12, 8.307e-12, 7.312e-12, 6.55e-12,
5.836e-12, 5.178e-12, 4.6e-12, 4.086e-12, 3.639e-12, 3.247e-12,
2.904e-12, 2.604e-12, 2.341e-12, 2.112e-12, 1.914e-12, 1.744e-12,
1.598e-12, 1.476e-12, 1.374e-12, 1.293e-12, 1.23e-12, 1.185e-12,
1.158e-12, 1.147e-12, 1.154e-12, 1.177e-12, 1.219e-12, 1.28e-12,
1.36e-12, 1.463e-12, 1.591e-12, 1.75e-12, 1.94e-12, 2.156e-12,
2.43e-12, 2.748e-12, 3.052e-12, 3.533e-12, 3.967e-12, 4.471e-12,
5.041e-12, 5.86e-12, 6.664e-12, 7.522e-12, 8.342e-12, 9.412e-12,
1.072e-11, 1.213e-11, 1.343e-11, 1.496e-11, 1.664e-11, 1.822e-11,
2.029e-11, 2.233e-11, 2.457e-11, 2.709e-11, 2.928e-11, 3.115e-11,
3.356e-11, 3.592e-11, 3.818e-11, 3.936e-11, 4.061e-11, 4.149e-11,
4.299e-11, 4.223e-11, 4.251e-11, 4.287e-11, 4.177e-11, 4.094e-11,
```

```
  3.942e-11, 3.772e-11, 3.614e-11, 3.394e-11, 3.222e-11, 2.791e-11,
  2.665e-11, 2.309e-11, 2.032e-11, 1.74e-11, 1.535e-11, 1.323e-11,
  1.151e-11, 9.803e-12, 8.65e-12, 7.54e-12, 6.619e-12, 5.832e-12,
  5.113e-12, 4.503e-12, 3.975e-12, 3.52e-12, 3.112e-12, 2.797e-12,
  2.5e-12, 2.24e-12, 2.013e-12, 1.819e-12, 1.653e-12, 1.513e-12,
  1.395e-12, 1.299e-12, 1.225e-12, 1.168e-12, 1.124e-12, 1.148e-12,
  1.107e-12, 1.128e-12, 1.169e-12, 1.233e-12, 1.307e-12, 1.359e-12,
  1.543e-12, 1.686e-12, 1.794e-12, 2.028e-12, 2.21e-12, 2.441e-12,
  2.653e-12, 2.828e-12, 3.093e-12, 3.28e-12, 3.551e-12, 3.677e-12,
  3.803e-12, 3.844e-12, 4.068e-12, 4.093e-12, 4.002e-12, 3.904e-12,
  3.624e-12, 3.633e-12, 3.622e-12, 3.443e-12, 3.184e-12, 2.934e-12,
  2.476e-12, 2.212e-12, 1.867e-12, 1.594e-12, 1.37e-12, 1.192e-12,
  1.045e-12, 9.211e-13, 8.17e-13, 7.29e-13, 6.55e-13, 5.929e-13,
  5.415e-13, 4.995e-13, 4.661e-13, 4.406e-13, 4.225e-13, 4.116e-13,
  4.075e-13, 4.102e-13, 4.198e-13, 4.365e-13, 4.606e-13, 4.925e-13,
  5.326e-13, 5.818e-13, 6.407e-13, 7.104e-13, 7.92e-13, 8.868e-13,
  9.964e-13, 1.123e-12, 1.268e-12, 1.434e-12, 1.626e-12, 1.848e-12,
  2.107e-12, 2.422e-12, 2.772e-12, 3.145e-12, 3.704e-12, 4.27e-12,
  4.721e-12, 5.361e-12, 6.083e-12, 7.095e-12, 7.968e-12, 9.228e-12,
  1.048e-11, 1.187e-11, 1.336e-11, 1.577e-11, 1.772e-11, 2.017e-11,
  2.25e-11, 2.63e-11, 2.911e-11, 3.356e-11, 3.82e-11, 4.173e-11,
  4.811e-11, 5.254e-11, 5.839e-11, 6.187e-11, 6.805e-11, 7.118e-11,
  7.369e-11, 7.664e-11, 7.794e-11, 7.947e-11, 8.036e-11, 7.954e-11,
  7.849e-11, 7.518e-11, 7.462e-11, 6.926e-11, 6.531e-11, 6.197e-11,
  5.421e-11, 4.777e-11, 4.111e-11, 3.679e-11, 3.166e-11, 2.786e-11,
  2.436e-11, 2.144e-11, 1.859e-11, 1.628e-11, 1.414e-11, 1.237e-11,
  1.093e-11, 9.558e-12
};

static double h2o260[2001] = { .2752, .2732, .2749, .2676, .2667, .2545,
  .2497, .2327, .2218, .2036, .1825, .1694, .1497, .1353, .121,
  .1014, .09405, .07848, .07195, .06246, .05306, .04853, .04138,
  .03735, .03171, .02785, .02431, .02111, .01845, .0164, .01405,
  .01255, .01098, .009797, .008646, .007779, .006898, .006099,
  .005453, .004909, .004413, .003959, .003581, .003199, .002871,
  .002583, .00233, .002086, .001874, .001684, .001512, .001361,
  .001225, .0011, 9.89e-4, 8.916e-4, 8.039e-4, 7.256e-4, 6.545e-4,
  5.918e-4, 5.359e-4, 4.867e-4, 4.426e-4, 4.033e-4, 3.682e-4,
  3.366e-4, 3.085e-4, 2.833e-4, 2.605e-4, 2.403e-4, 2.221e-4,
  2.055e-4, 1.908e-4, 1.774e-4, 1.653e-4, 1.544e-4, 1.443e-4,
  1.351e-4, 1.267e-4, 1.19e-4, 1.119e-4, 1.053e-4, 9.922e-5,
  9.355e-5, 8.831e-5, 8.339e-5, 7.878e-5, 7.449e-5, 7.043e-5,
  6.664e-5, 6.307e-5, 5.969e-5, 5.654e-5, 5.357e-5, 5.075e-5,
  4.81e-5, 4.56e-5, 4.322e-5, 4.102e-5, 3.892e-5, 3.696e-5,
  3.511e-5, 3.339e-5, 3.177e-5, 3.026e-5, 2.886e-5, 2.756e-5,
  2.636e-5, 2.527e-5, 2.427e-5, 2.337e-5, 2.257e-5, 2.185e-5,
  2.127e-5, 2.08e-5, 2.041e-5, 2.013e-5, 2e-5, 1.997e-5, 2.009e-5,
  2.031e-5, 2.068e-5, 2.124e-5, 2.189e-5, 2.267e-5, 2.364e-5,
  2.463e-5, 2.618e-5, 2.774e-5, 2.937e-5, 3.144e-5, 3.359e-5,
  3.695e-5, 4.002e-5, 4.374e-5, 4.947e-5, 5.431e-5, 6.281e-5,
  7.169e-5, 8.157e-5, 9.728e-5, 1.079e-4, 1.337e-4, 1.442e-4,
  1.683e-4, 1.879e-4, 2.223e-4, 2.425e-4, 2.838e-4, 3.143e-4,
  3.527e-4, 4.012e-4, 4.237e-4, 4.747e-4, 5.057e-4, 5.409e-4,
  5.734e-4, 5.944e-4, 6.077e-4, 6.175e-4, 6.238e-4, 6.226e-4,
  6.248e-4, 6.192e-4, 6.098e-4, 5.818e-4, 5.709e-4, 5.465e-4,
  5.043e-4, 4.699e-4, 4.294e-4, 3.984e-4, 3.672e-4, 3.152e-4,
  2.883e-4, 2.503e-4, 2.211e-4, 1.92e-4, 1.714e-4, 1.485e-4,
  1.358e-4, 1.156e-4, 1.021e-4, 8.887e-5, 7.842e-5, 7.12e-5,
  6.186e-5, 5.73e-5, 4.792e-5, 4.364e-5, 3.72e-5, 3.28e-5,
  2.946e-5, 2.591e-5, 2.261e-5, 2.048e-5, 1.813e-5, 1.63e-5,
  1.447e-5, 1.282e-5, 1.167e-5, 1.041e-5, 9.449e-6, 8.51e-6,
  7.596e-6, 6.961e-6, 6.272e-6, 5.728e-6, 5.198e-6, 4.667e-6,
  4.288e-6, 3.897e-6, 3.551e-6, 3.235e-6, 2.952e-6, 2.688e-6,
  2.449e-6, 2.241e-6, 2.05e-6, 1.879e-6, 1.722e-6, 1.582e-6,
  1.456e-6, 1.339e-6, 1.236e-6, 1.144e-6, 1.06e-6, 9.83e-7,
  9.149e-7, 8.535e-7, 7.973e-7, 7.466e-7, 6.999e-7, 6.574e-7,
  6.18e-7, 5.821e-7, 5.487e-7, 5.18e-7, 4.896e-7, 4.631e-7,
  4.386e-7, 4.16e-7, 3.945e-7, 3.748e-7, 3.562e-7, 3.385e-7,
  3.222e-7, 3.068e-7, 2.922e-7, 2.788e-7, 2.659e-7, 2.539e-7,
  2.425e-7, 2.318e-7, 2.219e-7, 2.127e-7, 2.039e-7, 1.958e-7,
  1.885e-7, 1.818e-7, 1.758e-7, 1.711e-7, 1.662e-7, 1.63e-7,
  1.605e-7, 1.58e-7, 1.559e-7, 1.545e-7, 1.532e-7, 1.522e-7,
  1.51e-7, 1.495e-7, 1.465e-7, 1.483e-7, 1.469e-7, 1.448e-7,
  1.444e-7, 1.436e-7, 1.426e-7, 1.431e-7, 1.425e-7, 1.445e-7,
  1.477e-7, 1.515e-7, 1.567e-7, 1.634e-7, 1.712e-7, 1.802e-7,
  1.914e-7, 2.024e-7, 2.159e-7, 2.295e-7, 2.461e-7, 2.621e-7,
  2.868e-7, 3.102e-7, 3.394e-7, 3.784e-7, 4.223e-7, 4.864e-7,
  5.501e-7, 6.039e-7, 7.193e-7, 7.728e-7, 9.514e-7, 1.073e-6,
  1.18e-6, 1.333e-6, 1.472e-6, 1.566e-6, 1.677e-6, 1.784e-6,
  1.904e-6, 1.953e-6, 2.02e-6, 2.074e-6, 2.128e-6, 2.162e-6,
  2.219e-6, 2.221e-6, 2.249e-6, 2.239e-6, 2.235e-6, 2.185e-6,
  2.141e-6, 2.124e-6, 2.09e-6, 2.068e-6, 2.1e-6, 2.104e-6,
  2.142e-6, 2.181e-6, 2.257e-6, 2.362e-6, 2.5e-6, 2.664e-6,
  2.884e-6, 3.189e-6, 3.48e-6, 3.847e-6, 4.313e-6, 4.79e-6,
  5.25e-6, 5.989e-6, 6.692e-6, 7.668e-6, 8.52e-6, 9.606e-6,
  1.073e-5, 1.225e-5, 1.377e-5, 1.582e-5, 1.761e-5, 2.029e-5,
  2.284e-5, 2.602e-5, 2.94e-5, 3.483e-5, 3.928e-5, 4.618e-5,
```

```
5.24e-5, 6.132e-5, 7.183e-5, 8.521e-5, 9.111e-5, 1.07e-4,
1.184e-4, 1.264e-4, 1.475e-4, 1.612e-4, 1.704e-4, 1.818e-4,
1.924e-4, 1.994e-4, 2.061e-4, 2.18e-4, 2.187e-4, 2.2e-4,
2.196e-4, 2.131e-4, 2.015e-4, 1.988e-4, 1.847e-4, 1.729e-4,
1.597e-4, 1.373e-4, 1.262e-4, 1.087e-4, 9.439e-5, 8.061e-5,
7.093e-5, 6.049e-5, 5.12e-5, 4.435e-5, 3.817e-5, 3.34e-5,
2.927e-5, 2.573e-5, 2.291e-5, 2.04e-5, 1.827e-5, 1.636e-5,
1.463e-5, 1.309e-5, 1.17e-5, 1.047e-5, 9.315e-6, 8.328e-6,
7.458e-6, 6.665e-6, 5.94e-6, 5.316e-6, 4.752e-6, 4.252e-6,
3.825e-6, 3.421e-6, 3.064e-6, 2.746e-6, 2.465e-6, 2.216e-6,
1.99e-6, 1.79e-6, 1.609e-6, 1.449e-6, 1.306e-6, 1.177e-6,
1.063e-6, 9.607e-7, 8.672e-7, 7.855e-7, 7.118e-7, 6.46e-7,
5.871e-7, 5.34e-7, 4.868e-7, 4.447e-7, 4.068e-7, 3.729e-7,
3.423e-7, 3.151e-7, 2.905e-7, 2.686e-7, 2.484e-7, 2.306e-7,
2.142e-7, 1.995e-7, 1.86e-7, 1.738e-7, 1.626e-7, 1.522e-7,
1.427e-7, 1.338e-7, 1.258e-7, 1.183e-7, 1.116e-7, 1.056e-7,
9.972e-8, 9.46e-8, 9.007e-8, 8.592e-8, 8.195e-8, 7.816e-8,
7.483e-8, 7.193e-8, 6.892e-8, 6.642e-8, 6.386e-8, 6.154e-8,
5.949e-8, 5.764e-8, 5.622e-8, 5.479e-8, 5.364e-8, 5.301e-8,
5.267e-8, 5.263e-8, 5.313e-8, 5.41e-8, 5.55e-8, 5.745e-8,
6.003e-8, 6.311e-8, 6.713e-8, 7.173e-8, 7.724e-8, 8.368e-8,
9.121e-8, 9.986e-8, 1.097e-7, 1.209e-7, 1.338e-7, 1.486e-7,
1.651e-7, 1.837e-7, 2.048e-7, 2.289e-7, 2.557e-7, 2.857e-7,
3.195e-7, 3.587e-7, 4.015e-7, 4.497e-7, 5.049e-7, 5.665e-7,
6.366e-7, 7.121e-7, 7.996e-7, 8.946e-7, 1.002e-6, 1.117e-6,
1.262e-6, 1.416e-6, 1.611e-6, 1.807e-6, 2.056e-6, 2.351e-6,
2.769e-6, 3.138e-6, 3.699e-6, 4.386e-6, 5.041e-6, 6.074e-6,
6.812e-6, 7.79e-6, 8.855e-6, 1.014e-5, 1.095e-5, 1.245e-5,
1.316e-5, 1.39e-5, 1.504e-5, 1.583e-5, 1.617e-5, 1.652e-5,
1.713e-5, 1.724e-5, 1.715e-5, 1.668e-5, 1.629e-5, 1.552e-5,
1.478e-5, 1.34e-5, 1.245e-5, 1.121e-5, 9.575e-6, 8.956e-6,
7.345e-6, 6.597e-6, 5.612e-6, 4.818e-6, 4.165e-6, 3.579e-6,
3.041e-6, 2.623e-6, 2.29e-6, 1.984e-6, 1.748e-6, 1.534e-6,
1.369e-6, 1.219e-6, 1.092e-6, 9.8e-7, 8.762e-7, 7.896e-7,
7.104e-7, 6.364e-7, 5.691e-7, 5.107e-7, 4.575e-7, 4.09e-7,
3.667e-7, 3.287e-7, 2.931e-7, 2.633e-7, 2.356e-7, 2.111e-7,
1.895e-7, 1.697e-7, 1.525e-7, 1.369e-7, 1.233e-7, 1.114e-7,
9.988e-8, 9.004e-8, 8.149e-8, 7.352e-8, 6.662e-8, 6.03e-8,
5.479e-8, 4.974e-8, 4.532e-8, 4.129e-8, 3.781e-8, 3.462e-8,
3.176e-8, 2.919e-8, 2.687e-8, 2.481e-8, 2.292e-8, 2.119e-8,
1.967e-8, 1.828e-8, 1.706e-8, 1.589e-8, 1.487e-8, 1.393e-8,
1.307e-8, 1.228e-8, 1.156e-8, 1.089e-8, 1.028e-8, 9.696e-9,
9.159e-9, 8.658e-9, 8.187e-9, 7.746e-9, 7.34e-9, 6.953e-9,
6.594e-9, 6.259e-9, 5.948e-9, 5.66e-9, 5.386e-9, 5.135e-9,
4.903e-9, 4.703e-9, 4.515e-9, 4.362e-9, 4.233e-9, 4.117e-9,
4.017e-9, 3.962e-9, 3.924e-9, 3.905e-9, 3.922e-9, 3.967e-9,
4.046e-9, 4.165e-9, 4.32e-9, 4.522e-9, 4.769e-9, 5.083e-9,
5.443e-9, 5.872e-9, 6.366e-9, 6.949e-9, 7.601e-9, 8.371e-9,
9.22e-9, 1.02e-8, 1.129e-8, 1.251e-8, 1.393e-8, 1.542e-8,
1.72e-8, 1.926e-8, 2.152e-8, 2.392e-8, 2.678e-8, 3.028e-8,
3.39e-8, 3.836e-8, 4.309e-8, 4.9e-8, 5.481e-8, 6.252e-8,
7.039e-8, 7.883e-8, 8.849e-8, 1.012e-7, 1.142e-7, 1.3e-7,
1.475e-7, 1.732e-7, 1.978e-7, 2.304e-7, 2.631e-7, 2.988e-7,
3.392e-7, 3.69e-7, 4.355e-7, 4.672e-7, 5.11e-7, 5.461e-7,
5.828e-7, 6.233e-7, 6.509e-7, 6.672e-7, 6.969e-7, 7.104e-7,
7.439e-7, 7.463e-7, 7.708e-7, 7.466e-7, 7.668e-7, 7.549e-7,
7.586e-7, 7.384e-7, 7.439e-7, 7.785e-7, 7.915e-7, 8.31e-7,
8.745e-7, 9.558e-7, 1.038e-6, 1.173e-6, 1.304e-6, 1.452e-6,
1.671e-6, 1.931e-6, 2.239e-6, 2.578e-6, 3.032e-6, 3.334e-6,
3.98e-6, 4.3e-6, 4.518e-6, 5.321e-6, 5.508e-6, 6.211e-6, 6.59e-6,
7.046e-6, 7.555e-6, 7.558e-6, 7.875e-6, 8.319e-6, 8.433e-6,
8.59e-6, 8.503e-6, 8.304e-6, 8.336e-6, 7.739e-6, 7.301e-6,
6.827e-6, 6.078e-6, 5.551e-6, 4.762e-6, 4.224e-6, 3.538e-6,
2.984e-6, 2.619e-6, 2.227e-6, 1.923e-6, 1.669e-6, 1.462e-6,
1.294e-6, 1.155e-6, 1.033e-6, 9.231e-7, 8.238e-7, 7.36e-7,
6.564e-7, 5.869e-7, 5.236e-7, 4.673e-7, 4.174e-7, 3.736e-7,
3.33e-7, 2.976e-7, 2.657e-7, 2.367e-7, 2.106e-7, 1.877e-7,
1.671e-7, 1.494e-7, 1.332e-7, 1.192e-7, 1.065e-7, 9.558e-8,
8.586e-8, 7.717e-8, 6.958e-8, 6.278e-8, 5.666e-8, 5.121e-8,
4.647e-8, 4.213e-8, 3.815e-8, 3.459e-8, 3.146e-8, 2.862e-8,
2.604e-8, 2.375e-8, 2.162e-8, 1.981e-8, 1.817e-8, 1.67e-8,
1.537e-8, 1.417e-8, 1.31e-8, 1.215e-8, 1.128e-8, 1.05e-8,
9.793e-9, 9.158e-9, 8.586e-9, 8.068e-9, 7.595e-9, 7.166e-9,
6.778e-9, 6.427e-9, 6.108e-9, 5.826e-9, 5.571e-9, 5.347e-9,
5.144e-9, 4.968e-9, 4.822e-9, 4.692e-9, 4.589e-9, 4.506e-9,
4.467e-9, 4.44e-9, 4.466e-9, 4.515e-9, 4.718e-9, 4.729e-9,
4.937e-9, 5.249e-9, 5.466e-9, 5.713e-9, 6.03e-9, 6.436e-9,
6.741e-9, 7.33e-9, 7.787e-9, 8.414e-9, 8.908e-9, 9.868e-9,
1.069e-8, 1.158e-8, 1.253e-8, 1.3e-8, 1.409e-8, 1.47e-8,
1.548e-8, 1.612e-8, 1.666e-8, 1.736e-8, 1.763e-8, 1.812e-8,
1.852e-8, 1.923e-8, 1.897e-8, 1.893e-8, 1.888e-8, 1.868e-8,
1.895e-8, 1.899e-8, 1.876e-8, 1.96e-8, 2.02e-8, 2.121e-8,
2.239e-8, 2.379e-8, 2.526e-8, 2.766e-8, 2.994e-8, 3.332e-8,
3.703e-8, 4.158e-8, 4.774e-8, 5.499e-8, 6.355e-8, 7.349e-8,
8.414e-8, 9.846e-8, 1.143e-7, 1.307e-7, 1.562e-7, 1.817e-7,
2.011e-7, 2.192e-7, 2.485e-7, 2.867e-7, 3.035e-7, 3.223e-7,
3.443e-7, 3.617e-7, 3.793e-7, 3.793e-7, 3.839e-7, 4.081e-7,
```

```
4.117e-7, 4.085e-7, 3.92e-7, 3.851e-7, 3.754e-7, 3.49e-7,
3.229e-7, 2.978e-7, 2.691e-7, 2.312e-7, 2.029e-7, 1.721e-7,
1.472e-7, 1.308e-7, 1.132e-7, 9.736e-8, 8.458e-8, 7.402e-8,
6.534e-8, 5.811e-8, 5.235e-8, 4.762e-8, 4.293e-8, 3.896e-8,
3.526e-8, 3.165e-8, 2.833e-8, 2.551e-8, 2.288e-8, 2.036e-8,
1.82e-8, 1.626e-8, 1.438e-8, 1.299e-8, 1.149e-8, 1.03e-8,
9.148e-9, 8.122e-9, 7.264e-9, 6.425e-9, 5.777e-9, 5.06e-9,
4.502e-9, 4.013e-9, 3.567e-9, 3.145e-9, 2.864e-9, 2.553e-9,
2.311e-9, 2.087e-9, 1.886e-9, 1.716e-9, 1.556e-9, 1.432e-9,
1.311e-9, 1.202e-9, 1.104e-9, 1.013e-9, 9.293e-10, 8.493e-10,
7.79e-10, 7.185e-10, 6.642e-10, 6.141e-10, 5.684e-10, 5.346e-10,
5.032e-10, 4.725e-10, 4.439e-10, 4.176e-10, 3.93e-10, 3.714e-10,
3.515e-10, 3.332e-10, 3.167e-10, 3.02e-10, 2.887e-10, 2.769e-10,
2.665e-10, 2.578e-10, 2.503e-10, 2.436e-10, 2.377e-10, 2.342e-10,
2.305e-10, 2.296e-10, 2.278e-10, 2.321e-10, 2.355e-10, 2.402e-10,
2.478e-10, 2.67e-10, 2.848e-10, 2.982e-10, 3.263e-10, 3.438e-10,
3.649e-10, 3.829e-10, 4.115e-10, 4.264e-10, 4.473e-10, 4.63e-10,
4.808e-10, 4.995e-10, 5.142e-10, 5.313e-10, 5.318e-10, 5.358e-10,
5.452e-10, 5.507e-10, 5.698e-10, 5.782e-10, 5.983e-10, 6.164e-10,
6.532e-10, 6.811e-10, 7.624e-10, 8.302e-10, 9.067e-10, 9.937e-10,
1.104e-9, 1.221e-9, 1.361e-9, 1.516e-9, 1.675e-9, 1.883e-9,
2.101e-9, 2.349e-9, 2.614e-9, 2.92e-9, 3.305e-9, 3.724e-9,
4.142e-9, 4.887e-9, 5.614e-9, 6.506e-9, 7.463e-9, 8.817e-9,
9.849e-9, 1.187e-8, 1.321e-8, 1.474e-8, 1.698e-8, 1.794e-8,
2.09e-8, 2.211e-8, 2.362e-8, 2.556e-8, 2.729e-8, 2.88e-8,
3.046e-8, 3.167e-8, 3.367e-8, 3.457e-8, 3.59e-8, 3.711e-8,
3.826e-8, 4.001e-8, 4.211e-8, 4.315e-8, 4.661e-8, 5.01e-8,
5.249e-8, 5.84e-8, 6.628e-8, 7.512e-8, 8.253e-8, 9.722e-8,
1.067e-7, 1.153e-7, 1.347e-7, 1.428e-7, 1.577e-7, 1.694e-7,
1.833e-7, 1.938e-7, 2.108e-7, 2.059e-7, 2.157e-7, 2.185e-7,
2.208e-7, 2.182e-7, 2.093e-7, 2.014e-7, 1.962e-7, 1.819e-7,
1.713e-7, 1.51e-7, 1.34e-7, 1.154e-7, 9.89e-8, 8.88e-8, 7.673e-8,
6.599e-8, 5.73e-8, 5.081e-8, 4.567e-8, 4.147e-8, 3.773e-8,
3.46e-8, 3.194e-8, 2.953e-8, 2.759e-8, 2.594e-8, 2.442e-8,
2.355e-8, 2.283e-8, 2.279e-8, 2.231e-8, 2.279e-8, 2.239e-8,
2.21e-8, 2.309e-8, 2.293e-8, 2.352e-8, 2.415e-8, 2.43e-8,
2.426e-8, 2.465e-8, 2.5e-8, 2.496e-8, 2.465e-8, 2.445e-8,
2.383e-8, 2.299e-8, 2.165e-8, 2.113e-8, 1.968e-8, 1.819e-8,
1.644e-8, 1.427e-8, 1.27e-8, 1.082e-8, 9.428e-9, 8.091e-9,
6.958e-9, 5.988e-9, 5.246e-9, 4.601e-9, 4.098e-9, 3.664e-9,
3.287e-9, 2.942e-9, 2.656e-9, 2.364e-9, 2.118e-9, 1.903e-9,
1.703e-9, 1.525e-9, 1.365e-9, 1.229e-9, 1.107e-9, 9.96e-10,
8.945e-10, 8.08e-10, 7.308e-10, 6.616e-10, 5.994e-10, 5.422e-10,
4.929e-10, 4.478e-10, 4.07e-10, 3.707e-10, 3.379e-10, 3.087e-10,
2.823e-10, 2.592e-10, 2.385e-10, 2.201e-10, 2.038e-10, 1.897e-10,
1.774e-10, 1.667e-10, 1.577e-10, 1.502e-10, 1.437e-10, 1.394e-10,
1.358e-10, 1.324e-10, 1.329e-10, 1.324e-10, 1.36e-10, 1.39e-10,
1.424e-10, 1.544e-10, 1.651e-10, 1.817e-10, 1.984e-10, 2.195e-10,
2.438e-10, 2.7e-10, 2.991e-10, 3.322e-10, 3.632e-10, 3.957e-10,
4.36e-10, 4.701e-10, 5.03e-10, 5.381e-10, 5.793e-10, 6.19e-10,
6.596e-10, 7.004e-10, 7.561e-10, 7.934e-10, 8.552e-10, 9.142e-10,
9.57e-10, 1.027e-9, 1.097e-9, 1.193e-9, 1.334e-9, 1.47e-9,
1.636e-9, 1.871e-9, 2.122e-9, 2.519e-9, 2.806e-9, 3.203e-9,
3.846e-9, 4.362e-9, 5.114e-9, 5.643e-9, 6.305e-9, 6.981e-9,
7.983e-9, 8.783e-9, 9.419e-9, 1.017e-8, 1.063e-8, 1.121e-8,
1.13e-8, 1.201e-8, 1.225e-8, 1.232e-8, 1.223e-8, 1.177e-8,
1.151e-8, 1.116e-8, 1.047e-8, 9.698e-9, 8.734e-9, 8.202e-9,
7.041e-9, 6.074e-9, 5.172e-9, 4.468e-9, 3.913e-9, 3.414e-9,
2.975e-9, 2.65e-9, 2.406e-9, 2.173e-9, 2.009e-9, 1.861e-9,
1.727e-9, 1.612e-9, 1.514e-9, 1.43e-9, 1.362e-9, 1.333e-9,
1.288e-9, 1.249e-9, 1.238e-9, 1.228e-9, 1.217e-9, 1.202e-9,
1.209e-9, 1.177e-9, 1.157e-9, 1.165e-9, 1.142e-9, 1.131e-9,
1.138e-9, 1.117e-9, 1.1e-9, 1.069e-9, 1.023e-9, 1.005e-9,
9.159e-10, 8.863e-10, 7.865e-10, 7.153e-10, 6.247e-10, 5.846e-10,
5.133e-10, 4.36e-10, 3.789e-10, 3.335e-10, 2.833e-10, 2.483e-10,
2.155e-10, 1.918e-10, 1.709e-10, 1.529e-10, 1.374e-10, 1.235e-10,
1.108e-10, 9.933e-11, 8.932e-11, 8.022e-11, 7.224e-11, 6.52e-11,
5.896e-11, 5.328e-11, 4.813e-11, 4.365e-11, 3.961e-11, 3.594e-11,
3.266e-11, 2.967e-11, 2.701e-11, 2.464e-11, 2.248e-11, 2.054e-11,
1.878e-11, 1.721e-11, 1.579e-11, 1.453e-11, 1.341e-11, 1.241e-11,
1.154e-11, 1.078e-11, 1.014e-11, 9.601e-12, 9.167e-12, 8.838e-12,
8.614e-12, 8.493e-12, 8.481e-12, 8.581e-12, 8.795e-12, 9.131e-12,
9.601e-12, 1.021e-11, 1.097e-11, 1.191e-11, 1.303e-11, 1.439e-11,
1.601e-11, 1.778e-11, 1.984e-11, 2.234e-11, 2.474e-11, 2.766e-11,
3.085e-11, 3.415e-11, 3.821e-11, 4.261e-11, 4.748e-11, 5.323e-11,
5.935e-11, 6.619e-11, 7.418e-11, 8.294e-11, 9.26e-11, 1.039e-10,
1.156e-10, 1.297e-10, 1.46e-10, 1.641e-10, 1.858e-10, 2.1e-10,
2.383e-10, 2.724e-10, 3.116e-10, 3.538e-10, 4.173e-10, 4.727e-10,
5.503e-10, 6.337e-10, 7.32e-10, 8.298e-10, 9.328e-10, 1.059e-9,
1.176e-9, 1.328e-9, 1.445e-9, 1.593e-9, 1.77e-9, 1.954e-9,
2.175e-9, 2.405e-9, 2.622e-9, 2.906e-9, 3.294e-9, 3.713e-9,
3.98e-9, 4.384e-9, 4.987e-9, 5.311e-9, 5.874e-9, 6.337e-9,
7.027e-9, 7.39e-9, 7.769e-9, 8.374e-9, 8.605e-9, 9.165e-9,
9.415e-9, 9.511e-9, 9.704e-9, 9.588e-9, 9.45e-9, 9.086e-9,
8.798e-9, 8.469e-9, 7.697e-9, 7.168e-9, 6.255e-9, 5.772e-9,
4.97e-9, 4.271e-9, 3.653e-9, 3.154e-9, 2.742e-9, 2.435e-9,
2.166e-9, 1.936e-9, 1.731e-9, 1.556e-9, 1.399e-9, 1.272e-9,
```

```
1.157e-9, 1.066e-9, 9.844e-10, 9.258e-10, 8.787e-10, 8.421e-10,
8.083e-10, 8.046e-10, 8.067e-10, 8.181e-10, 8.325e-10, 8.517e-10,
9.151e-10, 9.351e-10, 9.677e-10, 1.071e-9, 1.126e-9, 1.219e-9,
1.297e-9, 1.408e-9, 1.476e-9, 1.517e-9, 1.6e-9, 1.649e-9,
1.678e-9, 1.746e-9, 1.742e-9, 1.728e-9, 1.699e-9, 1.655e-9,
1.561e-9, 1.48e-9, 1.451e-9, 1.411e-9, 1.171e-9, 1.106e-9,
9.714e-10, 8.523e-10, 7.346e-10, 6.241e-10, 5.371e-10, 4.704e-10,
4.144e-10, 3.683e-10, 3.292e-10, 2.942e-10, 2.62e-10, 2.341e-10,
2.104e-10, 1.884e-10, 1.7e-10, 1.546e-10, 1.394e-10, 1.265e-10,
1.14e-10, 1.019e-10, 9.279e-11, 8.283e-11, 7.458e-11, 6.668e-11,
5.976e-11, 5.33e-11, 4.794e-11, 4.289e-11, 3.841e-11, 3.467e-11,
3.13e-11, 2.832e-11, 2.582e-11, 2.356e-11, 2.152e-11, 1.97e-11,
1.808e-11, 1.664e-11, 1.539e-11, 1.434e-11, 1.344e-11, 1.269e-11,
1.209e-11, 1.162e-11, 1.129e-11, 1.108e-11, 1.099e-11, 1.103e-11,
1.119e-11, 1.148e-11, 1.193e-11, 1.252e-11, 1.329e-11, 1.421e-11,
1.555e-11, 1.685e-11, 1.839e-11, 2.054e-11, 2.317e-11, 2.571e-11,
2.839e-11, 3.171e-11, 3.49e-11, 3.886e-11, 4.287e-11, 4.645e-11,
5.047e-11, 5.592e-11, 6.109e-11, 6.628e-11, 7.381e-11, 8.088e-11,
8.966e-11, 1.045e-10, 1.12e-10, 1.287e-10, 1.486e-10, 1.662e-10,
1.866e-10, 2.133e-10, 2.524e-10, 2.776e-10, 3.204e-10, 3.559e-10,
4.028e-10, 4.448e-10, 4.882e-10, 5.244e-10, 5.605e-10, 6.018e-10,
6.328e-10, 6.579e-10, 6.541e-10, 7.024e-10, 7.074e-10, 7.068e-10,
7.009e-10, 6.698e-10, 6.545e-10, 6.209e-10, 5.834e-10, 5.412e-10,
5.001e-10, 4.231e-10, 3.727e-10, 3.211e-10, 2.833e-10, 2.447e-10,
2.097e-10, 1.843e-10, 1.639e-10, 1.449e-10, 1.27e-10, 1.161e-10,
1.033e-10, 9.282e-11, 8.407e-11, 7.639e-11, 7.023e-11, 6.474e-11,
6.142e-11, 5.76e-11, 5.568e-11, 5.472e-11, 5.39e-11, 5.455e-11,
5.54e-11, 5.587e-11, 6.23e-11, 6.49e-11, 6.868e-11, 7.382e-11,
8.022e-11, 8.372e-11, 9.243e-11, 1.004e-10, 1.062e-10, 1.13e-10,
1.176e-10, 1.244e-10, 1.279e-10, 1.298e-10, 1.302e-10, 1.312e-10,
1.295e-10, 1.244e-10, 1.211e-10, 1.167e-10, 1.098e-10, 9.927e-11,
8.854e-11, 8.011e-11, 7.182e-11, 5.923e-11, 5.212e-11, 4.453e-11,
3.832e-11, 3.371e-11, 2.987e-11, 2.651e-11, 2.354e-11, 2.093e-11,
1.863e-11, 1.662e-11, 1.486e-11, 1.331e-11, 1.193e-11, 1.071e-11,
9.628e-12, 8.66e-12, 7.801e-12, 7.031e-12, 6.347e-12, 5.733e-12,
5.182e-12, 4.695e-12, 4.26e-12, 3.874e-12, 3.533e-12, 3.235e-12,
2.979e-12, 2.76e-12, 2.579e-12, 2.432e-12, 2.321e-12, 2.246e-12,
2.205e-12, 2.196e-12, 2.223e-12, 2.288e-12, 2.387e-12, 2.525e-12,
2.704e-12, 2.925e-12, 3.191e-12, 3.508e-12, 3.876e-12, 4.303e-12,
4.793e-12, 5.347e-12, 5.978e-12, 6.682e-12, 7.467e-12, 8.34e-12,
9.293e-12, 1.035e-11, 1.152e-11, 1.285e-11, 1.428e-11, 1.586e-11,
1.764e-11, 1.972e-11, 2.214e-11, 2.478e-11, 2.776e-11, 3.151e-11,
3.591e-11, 4.103e-11, 4.66e-11, 5.395e-11, 6.306e-11, 7.172e-11,
8.358e-11, 9.67e-11, 1.11e-10, 1.325e-10, 1.494e-10, 1.736e-10,
2.007e-10, 2.296e-10, 2.608e-10, 3.004e-10, 3.361e-10, 3.727e-10,
4.373e-10, 4.838e-10, 5.483e-10, 6.006e-10, 6.535e-10, 6.899e-10,
7.687e-10, 8.444e-10, 8.798e-10, 9.135e-10, 9.532e-10, 9.757e-10,
9.968e-10, 1.006e-9, 9.949e-10, 9.789e-10, 9.564e-10, 9.215e-10,
8.51e-10, 8.394e-10, 7.707e-10, 7.152e-10, 6.274e-10, 5.598e-10,
5.028e-10, 4.3e-10, 3.71e-10, 3.245e-10, 2.809e-10, 2.461e-10,
2.154e-10, 1.91e-10, 1.685e-10, 1.487e-10, 1.313e-10, 1.163e-10,
1.031e-10, 9.172e-11, 8.221e-11, 7.382e-11, 6.693e-11, 6.079e-11,
5.581e-11, 5.167e-11, 4.811e-11, 4.506e-11, 4.255e-11, 4.083e-11,
3.949e-11, 3.881e-11, 3.861e-11, 3.858e-11, 3.951e-11, 4.045e-11,
4.24e-11, 4.487e-11, 4.806e-11, 5.133e-11, 5.518e-11, 5.919e-11,
6.533e-11, 7.031e-11, 7.762e-11, 8.305e-11, 9.252e-11, 9.727e-11,
1.045e-10, 1.117e-10, 1.2e-10, 1.275e-10, 1.341e-10, 1.362e-10,
1.438e-10, 1.45e-10, 1.455e-10, 1.455e-10, 1.434e-10, 1.381e-10,
1.301e-10, 1.276e-10, 1.163e-10, 1.089e-10, 9.911e-11, 8.943e-11,
7.618e-11, 6.424e-11, 5.717e-11, 4.866e-11, 4.257e-11, 3.773e-11,
3.331e-11, 2.958e-11, 2.629e-11, 2.316e-11, 2.073e-11, 1.841e-11,
1.635e-11, 1.464e-11, 1.31e-11, 1.16e-11, 1.047e-11, 9.408e-12,
8.414e-12, 7.521e-12, 6.705e-12, 5.993e-12, 5.371e-12, 4.815e-12,
4.338e-12, 3.921e-12, 3.567e-12, 3.265e-12, 3.01e-12, 2.795e-12,
2.613e-12, 2.464e-12, 2.346e-12, 2.256e-12, 2.195e-12, 2.165e-12,
2.166e-12, 2.198e-12, 2.262e-12, 2.364e-12, 2.502e-12, 2.682e-12,
2.908e-12, 3.187e-12, 3.533e-12, 3.946e-12, 4.418e-12, 5.013e-12,
5.708e-12, 6.379e-12, 7.43e-12, 8.39e-12, 9.51e-12, 1.078e-11,
1.259e-11, 1.438e-11, 1.63e-11, 1.814e-11, 2.055e-11, 2.348e-11,
2.664e-11, 2.956e-11, 3.3e-11, 3.677e-11, 4.032e-11, 4.494e-11,
4.951e-11, 5.452e-11, 6.014e-11, 6.5e-11, 6.915e-11, 7.45e-11,
7.971e-11, 8.468e-11, 8.726e-11, 8.995e-11, 9.182e-11, 9.509e-11,
9.333e-11, 9.386e-11, 9.457e-11, 9.21e-11, 9.019e-11, 8.68e-11,
8.298e-11, 7.947e-11, 7.46e-11, 7.082e-11, 6.132e-11, 5.855e-11,
5.073e-11, 4.464e-11, 3.825e-11, 3.375e-11, 2.911e-11, 2.535e-11,
2.16e-11, 1.907e-11, 1.665e-11, 1.463e-11, 1.291e-11, 1.133e-11,
9.997e-12, 8.836e-12, 7.839e-12, 6.943e-12, 6.254e-12, 5.6e-12,
5.029e-12, 4.529e-12, 4.102e-12, 3.737e-12, 3.428e-12, 3.169e-12,
2.959e-12, 2.798e-12, 2.675e-12, 2.582e-12, 2.644e-12, 2.557e-12,
2.614e-12, 2.717e-12, 2.874e-12, 3.056e-12, 3.187e-12, 3.631e-12,
3.979e-12, 4.248e-12, 4.817e-12, 5.266e-12, 5.836e-12, 6.365e-12,
6.807e-12, 7.47e-12, 7.951e-12, 8.636e-12, 8.972e-12, 9.314e-12,
9.445e-12, 1.003e-11, 1.013e-11, 9.937e-12, 9.729e-12, 9.064e-12,
9.119e-12, 9.124e-12, 8.704e-12, 8.078e-12, 7.47e-12, 6.329e-12,
5.674e-12, 4.808e-12, 4.119e-12, 3.554e-12, 3.103e-12, 2.731e-12,
2.415e-12, 2.15e-12, 1.926e-12, 1.737e-12, 1.578e-12, 1.447e-12,
1.34e-12, 1.255e-12, 1.191e-12, 1.146e-12, 1.121e-12, 1.114e-12,
```

```
  1.126e-12, 1.156e-12, 1.207e-12, 1.278e-12, 1.372e-12, 1.49e-12,
  1.633e-12, 1.805e-12, 2.01e-12, 2.249e-12, 2.528e-12, 2.852e-12,
  3.228e-12, 3.658e-12, 4.153e-12, 4.728e-12, 5.394e-12, 6.176e-12,
  7.126e-12, 8.188e-12, 9.328e-12, 1.103e-11, 1.276e-11, 1.417e-11,
  1.615e-11, 1.84e-11, 2.155e-11, 2.429e-11, 2.826e-11, 3.222e-11,
  3.664e-11, 4.14e-11, 4.906e-11, 5.536e-11, 6.327e-11, 7.088e-11,
  8.316e-11, 9.242e-11, 1.07e-10, 1.223e-10, 1.341e-10, 1.553e-10,
  1.703e-10, 1.9e-10, 2.022e-10, 2.233e-10, 2.345e-10, 2.438e-10,
  2.546e-10, 2.599e-10, 2.661e-10, 2.703e-10, 2.686e-10, 2.662e-10,
  2.56e-10, 2.552e-10, 2.378e-10, 2.252e-10, 2.146e-10, 1.885e-10,
  1.668e-10, 1.441e-10, 1.295e-10, 1.119e-10, 9.893e-11, 8.687e-11,
  7.678e-11, 6.685e-11, 5.879e-11, 5.127e-11, 4.505e-11, 3.997e-11,
  3.511e-11
};

static double h2ofrn[2001] = { .01095, .01126, .01205, .01322, .0143,
  .01506, .01548, .01534, .01486, .01373, .01262, .01134, .01001,
  .008702, .007475, .006481, .00548, .0046, .003833, .00311,
  .002543, .002049, .00168, .001374, .001046, 8.193e-4, 6.267e-4,
  4.968e-4, 3.924e-4, 2.983e-4, 2.477e-4, 1.997e-4, 1.596e-4,
  1.331e-4, 1.061e-4, 8.942e-5, 7.168e-5, 5.887e-5, 4.848e-5,
  3.817e-5, 3.17e-5, 2.579e-5, 2.162e-5, 1.768e-5, 1.49e-5,
  1.231e-5, 1.013e-5, 8.555e-6, 7.328e-6, 6.148e-6, 5.207e-6,
  4.387e-6, 3.741e-6, 3.22e-6, 2.753e-6, 2.346e-6, 1.985e-6,
  1.716e-6, 1.475e-6, 1.286e-6, 1.122e-6, 9.661e-7, 8.284e-7,
  7.057e-7, 6.119e-7, 5.29e-7, 4.571e-7, 3.948e-7, 3.432e-7,
  2.983e-7, 2.589e-7, 2.265e-7, 1.976e-7, 1.704e-7, 1.456e-7,
  1.26e-7, 1.101e-7, 9.648e-8, 8.415e-8, 7.34e-8, 6.441e-8,
  5.643e-8, 4.94e-8, 4.276e-8, 3.703e-8, 3.227e-8, 2.825e-8,
  2.478e-8, 2.174e-8, 1.898e-8, 1.664e-8, 1.458e-8, 1.278e-8,
  1.126e-8, 9.891e-9, 8.709e-9, 7.652e-9, 6.759e-9, 5.975e-9,
  5.31e-9, 4.728e-9, 4.214e-9, 3.792e-9, 3.463e-9, 3.226e-9,
  2.992e-9, 2.813e-9, 2.749e-9, 2.809e-9, 2.913e-9, 3.037e-9,
  3.413e-9, 3.738e-9, 4.189e-9, 4.808e-9, 5.978e-9, 7.088e-9,
  8.071e-9, 9.61e-9, 1.21e-8, 1.5e-8, 1.764e-8, 2.221e-8, 2.898e-8,
  3.948e-8, 5.068e-8, 6.227e-8, 7.898e-8, 1.033e-7, 1.437e-7,
  1.889e-7, 2.589e-7, 3.59e-7, 4.971e-7, 7.156e-7, 9.983e-7,
  1.381e-6, 1.929e-6, 2.591e-6, 3.453e-6, 4.57e-6, 5.93e-6,
  7.552e-6, 9.556e-6, 1.183e-5, 1.425e-5, 1.681e-5, 1.978e-5,
  2.335e-5, 2.668e-5, 3.022e-5, 3.371e-5, 3.715e-5, 3.967e-5,
  4.06e-5, 4.01e-5, 3.809e-5, 3.491e-5, 3.155e-5, 2.848e-5,
  2.678e-5, 2.66e-5, 2.811e-5, 3.071e-5, 3.294e-5, 3.459e-5,
  3.569e-5, 3.56e-5, 3.434e-5, 3.186e-5, 2.916e-5, 2.622e-5,
  2.275e-5, 1.918e-5, 1.62e-5, 1.373e-5, 1.182e-5, 1.006e-5,
  8.556e-6, 7.26e-6, 6.107e-6, 5.034e-6, 4.211e-6, 3.426e-6,
  2.865e-6, 2.446e-6, 1.998e-6, 1.628e-6, 1.242e-6, 1.005e-6,
  7.853e-7, 6.21e-7, 5.071e-7, 4.156e-7, 3.548e-7, 2.825e-7,
  2.261e-7, 1.916e-7, 1.51e-7, 1.279e-7, 1.059e-7, 9.14e-8,
  7.707e-8, 6.17e-8, 5.311e-8, 4.263e-8, 3.518e-8, 2.961e-8,
  2.457e-8, 2.119e-8, 1.712e-8, 1.439e-8, 1.201e-8, 1.003e-8,
  8.564e-9, 7.199e-9, 6.184e-9, 5.206e-9, 4.376e-9, 3.708e-9,
  3.157e-9, 2.725e-9, 2.361e-9, 2.074e-9, 1.797e-9, 1.562e-9,
  1.364e-9, 1.196e-9, 1.042e-9, 8.862e-10, 7.648e-10, 6.544e-10,
  5.609e-10, 4.791e-10, 4.108e-10, 3.531e-10, 3.038e-10, 2.618e-10,
  2.268e-10, 1.969e-10, 1.715e-10, 1.496e-10, 1.308e-10, 1.147e-10,
  1.008e-10, 8.894e-11, 7.885e-11, 7.031e-11, 6.355e-11, 5.854e-11,
  5.534e-11, 5.466e-11, 5.725e-11, 6.447e-11, 7.943e-11, 1.038e-10,
  1.437e-10, 2.04e-10, 2.901e-10, 4.051e-10, 5.556e-10, 7.314e-10,
  9.291e-10, 1.134e-9, 1.321e-9, 1.482e-9, 1.596e-9, 1.669e-9,
  1.715e-9, 1.762e-9, 1.817e-9, 1.828e-9, 1.848e-9, 1.873e-9,
  1.902e-9, 1.894e-9, 1.864e-9, 1.841e-9, 1.797e-9, 1.704e-9,
  1.559e-9, 1.382e-9, 1.187e-9, 1.001e-9, 8.468e-10, 7.265e-10,
  6.521e-10, 6.381e-10, 6.66e-10, 7.637e-10, 9.705e-10, 1.368e-9,
  1.856e-9, 2.656e-9, 3.954e-9, 5.96e-9, 8.72e-9, 1.247e-8,
  1.781e-8, 2.491e-8, 3.311e-8, 4.272e-8, 5.205e-8, 6.268e-8,
  7.337e-8, 8.277e-8, 9.185e-8, 1.004e-7, 1.091e-7, 1.159e-7,
  1.188e-7, 1.175e-7, 1.124e-7, 1.033e-7, 9.381e-8, 8.501e-8,
  7.956e-8, 7.894e-8, 8.331e-8, 9.102e-8, 9.836e-8, 1.035e-7,
  1.064e-7, 1.06e-7, 1.032e-7, 9.808e-8, 9.139e-8, 8.442e-8,
  7.641e-8, 6.881e-8, 6.161e-8, 5.404e-8, 4.804e-8, 4.446e-8,
  4.328e-8, 4.259e-8, 4.421e-8, 4.673e-8, 4.985e-8, 5.335e-8,
  5.796e-8, 6.542e-8, 7.714e-8, 8.827e-8, 1.04e-7, 1.238e-7,
  1.499e-7, 1.829e-7, 2.222e-7, 2.689e-7, 3.303e-7, 3.981e-7,
  4.84e-7, 5.91e-7, 7.363e-7, 9.087e-7, 1.139e-6, 1.455e-6,
  1.866e-6, 2.44e-6, 3.115e-6, 3.941e-6, 4.891e-6, 5.992e-6,
  7.111e-6, 8.296e-6, 9.21e-6, 9.987e-6, 1.044e-5, 1.073e-5,
  1.092e-5, 1.106e-5, 1.138e-5, 1.171e-5, 1.186e-5, 1.186e-5,
  1.179e-5, 1.166e-5, 1.151e-5, 1.16e-5, 1.197e-5, 1.241e-5,
  1.268e-5, 1.26e-5, 1.184e-5, 1.063e-5, 9.204e-6, 7.584e-6,
  6.053e-6, 4.482e-6, 3.252e-6, 2.337e-6, 1.662e-6, 1.18e-6,
  8.15e-7, 5.95e-7, 4.354e-7, 3.302e-7, 2.494e-7, 1.93e-7,
  1.545e-7, 1.25e-7, 1.039e-7, 8.602e-8, 7.127e-8, 5.897e-8,
  4.838e-8, 4.018e-8, 3.28e-8, 2.72e-8, 2.307e-8, 1.972e-8,
  1.654e-8, 1.421e-8, 1.174e-8, 1.004e-8, 8.739e-9, 7.358e-9,
  6.242e-9, 5.303e-9, 4.567e-9, 3.94e-9, 3.375e-9, 2.864e-9,
  2.422e-9, 2.057e-9, 1.75e-9, 1.505e-9, 1.294e-9, 1.101e-9,
  9.401e-10, 8.018e-10, 6.903e-10, 5.965e-10, 5.087e-10, 4.364e-10,
```

```
3.759e-10, 3.247e-10, 2.809e-10, 2.438e-10, 2.123e-10, 1.853e-10,
1.622e-10, 1.426e-10, 1.26e-10, 1.125e-10, 1.022e-10, 9.582e-11,
9.388e-11, 9.801e-11, 1.08e-10, 1.276e-10, 1.551e-10, 1.903e-10,
2.291e-10, 2.724e-10, 3.117e-10, 3.4e-10, 3.562e-10, 3.625e-10,
3.619e-10, 3.429e-10, 3.221e-10, 2.943e-10, 2.645e-10, 2.338e-10,
2.062e-10, 1.901e-10, 1.814e-10, 1.827e-10, 1.906e-10, 1.984e-10,
2.04e-10, 2.068e-10, 2.075e-10, 2.018e-10, 1.959e-10, 1.897e-10,
1.852e-10, 1.791e-10, 1.696e-10, 1.634e-10, 1.598e-10, 1.561e-10,
1.518e-10, 1.443e-10, 1.377e-10, 1.346e-10, 1.342e-10, 1.375e-10,
1.525e-10, 1.767e-10, 2.108e-10, 2.524e-10, 2.981e-10, 3.477e-10,
4.262e-10, 5.326e-10, 6.646e-10, 8.321e-10, 1.069e-9, 1.386e-9,
1.743e-9, 2.216e-9, 2.808e-9, 3.585e-9, 4.552e-9, 5.907e-9,
7.611e-9, 9.774e-9, 1.255e-8, 1.666e-8, 2.279e-8, 3.221e-8,
4.531e-8, 6.4e-8, 9.187e-8, 1.295e-7, 1.825e-7, 2.431e-7,
3.181e-7, 4.009e-7, 4.941e-7, 5.88e-7, 6.623e-7, 7.155e-7,
7.451e-7, 7.594e-7, 7.541e-7, 7.467e-7, 7.527e-7, 7.935e-7,
8.461e-7, 8.954e-7, 9.364e-7, 9.843e-7, 1.024e-6, 1.05e-6,
1.059e-6, 1.074e-6, 1.072e-6, 1.043e-6, 9.789e-7, 8.803e-7,
7.662e-7, 6.378e-7, 5.133e-7, 3.958e-7, 2.914e-7, 2.144e-7,
1.57e-7, 1.14e-7, 8.47e-8, 6.2e-8, 4.657e-8, 3.559e-8, 2.813e-8,
2.222e-8, 1.769e-8, 1.391e-8, 1.125e-8, 9.186e-9, 7.704e-9,
6.447e-9, 5.381e-9, 4.442e-9, 3.669e-9, 3.057e-9, 2.564e-9,
2.153e-9, 1.784e-9, 1.499e-9, 1.281e-9, 1.082e-9, 9.304e-10,
8.169e-10, 6.856e-10, 5.866e-10, 5.043e-10, 4.336e-10, 3.731e-10,
3.175e-10, 2.745e-10, 2.374e-10, 2.007e-10, 1.737e-10, 1.508e-10,
1.302e-10, 1.13e-10, 9.672e-11, 8.375e-11, 7.265e-11, 6.244e-11,
5.343e-11, 4.654e-11, 3.975e-11, 3.488e-11, 3.097e-11, 2.834e-11,
2.649e-11, 2.519e-11, 2.462e-11, 2.443e-11, 2.44e-11, 2.398e-11,
2.306e-11, 2.183e-11, 2.021e-11, 1.821e-11, 1.599e-11, 1.403e-11,
1.196e-11, 1.023e-11, 8.728e-12, 7.606e-12, 6.941e-12, 6.545e-12,
6.484e-12, 6.6e-12, 6.718e-12, 6.785e-12, 6.746e-12, 6.724e-12,
6.764e-12, 6.995e-12, 7.144e-12, 7.32e-12, 7.33e-12, 7.208e-12,
6.789e-12, 6.09e-12, 5.337e-12, 4.62e-12, 4.037e-12, 3.574e-12,
3.311e-12, 3.346e-12, 3.566e-12, 3.836e-12, 4.076e-12, 4.351e-12,
4.691e-12, 5.114e-12, 5.427e-12, 6.167e-12, 7.436e-12, 8.842e-12,
1.038e-11, 1.249e-11, 1.54e-11, 1.915e-11, 2.48e-11, 3.256e-11,
4.339e-11, 5.611e-11, 7.519e-11, 1.037e-10, 1.409e-10, 1.883e-10,
2.503e-10, 3.38e-10, 4.468e-10, 5.801e-10, 7.335e-10, 8.98e-10,
1.11e-9, 1.363e-9, 1.677e-9, 2.104e-9, 2.681e-9, 3.531e-9,
4.621e-9, 6.106e-9, 8.154e-9, 1.046e-8, 1.312e-8, 1.607e-8,
1.948e-8, 2.266e-8, 2.495e-8, 2.655e-8, 2.739e-8, 2.739e-8,
2.662e-8, 2.589e-8, 2.59e-8, 2.664e-8, 2.833e-8, 3.023e-8,
3.305e-8, 3.558e-8, 3.793e-8, 3.961e-8, 4.056e-8, 4.102e-8,
4.025e-8, 3.917e-8, 3.706e-8, 3.493e-8, 3.249e-8, 3.096e-8,
3.011e-8, 3.111e-8, 3.395e-8, 3.958e-8, 4.875e-8, 6.066e-8,
7.915e-8, 1.011e-7, 1.3e-7, 1.622e-7, 2.003e-7, 2.448e-7,
2.863e-7, 3.317e-7, 3.655e-7, 3.96e-7, 4.098e-7, 4.168e-7,
4.198e-7, 4.207e-7, 4.289e-7, 4.384e-7, 4.471e-7, 4.524e-7,
4.574e-7, 4.633e-7, 4.785e-7, 5.028e-7, 5.371e-7, 5.727e-7,
5.955e-7, 5.998e-7, 5.669e-7, 5.082e-7, 4.397e-7, 3.596e-7,
2.814e-7, 2.074e-7, 1.486e-7, 1.057e-7, 7.25e-8, 4.946e-8,
3.43e-8, 2.447e-8, 1.793e-8, 1.375e-8, 1.096e-8, 9.091e-9,
7.709e-9, 6.631e-9, 5.714e-9, 4.886e-9, 4.205e-9, 3.575e-9,
3.07e-9, 2.631e-9, 2.284e-9, 2.002e-9, 1.745e-9, 1.509e-9,
1.284e-9, 1.084e-9, 9.163e-10, 7.663e-10, 6.346e-10, 5.283e-10,
4.354e-10, 3.59e-10, 2.982e-10, 2.455e-10, 2.033e-10, 1.696e-10,
1.432e-10, 1.211e-10, 1.02e-10, 8.702e-11, 7.38e-11, 6.293e-11,
5.343e-11, 4.532e-11, 3.907e-11, 3.365e-11, 2.945e-11, 2.558e-11,
2.192e-11, 1.895e-11, 1.636e-11, 1.42e-11, 1.228e-11, 1.063e-11,
9.348e-12, 8.2e-12, 7.231e-12, 6.43e-12, 5.702e-12, 5.052e-12,
4.469e-12, 4e-12, 3.679e-12, 3.387e-12, 3.197e-12, 3.158e-12,
3.327e-12, 3.675e-12, 4.292e-12, 5.437e-12, 7.197e-12, 1.008e-11,
1.437e-11, 2.035e-11, 2.905e-11, 4.062e-11, 5.528e-11, 7.177e-11,
9.064e-11, 1.109e-10, 1.297e-10, 1.473e-10, 1.652e-10, 1.851e-10,
2.079e-10, 2.313e-10, 2.619e-10, 2.958e-10, 3.352e-10, 3.796e-10,
4.295e-10, 4.923e-10, 5.49e-10, 5.998e-10, 6.388e-10, 6.645e-10,
6.712e-10, 6.549e-10, 6.38e-10, 6.255e-10, 6.253e-10, 6.459e-10,
6.977e-10, 7.59e-10, 8.242e-10, 8.92e-10, 9.403e-10, 9.701e-10,
9.483e-10, 9.135e-10, 8.617e-10, 7.921e-10, 7.168e-10, 6.382e-10,
5.677e-10, 5.045e-10, 4.572e-10, 4.312e-10, 4.145e-10, 4.192e-10,
4.541e-10, 5.368e-10, 6.771e-10, 8.962e-10, 1.21e-9, 1.659e-9,
2.33e-9, 3.249e-9, 4.495e-9, 5.923e-9, 7.642e-9, 9.607e-9,
1.178e-8, 1.399e-8, 1.584e-8, 1.73e-8, 1.816e-8, 1.87e-8,
1.868e-8, 1.87e-8, 1.884e-8, 1.99e-8, 2.15e-8, 2.258e-8,
2.364e-8, 2.473e-8, 2.602e-8, 2.689e-8, 2.731e-8, 2.816e-8,
2.859e-8, 2.839e-8, 2.703e-8, 2.451e-8, 2.149e-8, 1.787e-8,
1.449e-8, 1.111e-8, 8.282e-9, 6.121e-9, 4.494e-9, 3.367e-9,
2.487e-9, 1.885e-9, 1.503e-9, 1.249e-9, 1.074e-9, 9.427e-10,
8.439e-10, 7.563e-10, 6.772e-10, 6.002e-10, 5.254e-10, 4.588e-10,
3.977e-10, 3.449e-10, 3.003e-10, 2.624e-10, 2.335e-10, 2.04e-10,
1.771e-10, 1.534e-10, 1.296e-10, 1.097e-10, 9.173e-11, 7.73e-11,
6.547e-11, 5.191e-11, 4.198e-11, 3.361e-11, 2.732e-11, 2.244e-11,
1.791e-11, 1.509e-11, 1.243e-11, 1.035e-11, 8.969e-12, 7.394e-12,
6.323e-12, 5.282e-12, 4.543e-12, 3.752e-12, 3.14e-12, 2.6e-12,
2.194e-12, 1.825e-12, 1.511e-12, 1.245e-12, 1.024e-12, 8.539e-13,
7.227e-13, 6.102e-13, 5.189e-13, 4.43e-13, 3.774e-13, 3.236e-13,
2.8e-13, 2.444e-13, 2.156e-13, 1.932e-13, 1.775e-13, 1.695e-13,
```

```
1.672e-13, 1.704e-13, 1.825e-13, 2.087e-13, 2.614e-13, 3.377e-13,
4.817e-13, 6.989e-13, 1.062e-12, 1.562e-12, 2.288e-12, 3.295e-12,
4.55e-12, 5.965e-12, 7.546e-12, 9.395e-12, 1.103e-11, 1.228e-11,
1.318e-11, 1.38e-11, 1.421e-11, 1.39e-11, 1.358e-11, 1.336e-11,
1.342e-11, 1.356e-11, 1.424e-11, 1.552e-11, 1.73e-11, 1.951e-11,
2.128e-11, 2.249e-11, 2.277e-11, 2.226e-11, 2.111e-11, 1.922e-11,
1.775e-11, 1.661e-11, 1.547e-11, 1.446e-11, 1.323e-11, 1.21e-11,
1.054e-11, 9.283e-12, 8.671e-12, 8.67e-12, 9.429e-12, 1.062e-11,
1.255e-11, 1.506e-11, 1.818e-11, 2.26e-11, 2.831e-11, 3.723e-11,
5.092e-11, 6.968e-11, 9.826e-11, 1.349e-10, 1.87e-10, 2.58e-10,
3.43e-10, 4.424e-10, 5.521e-10, 6.812e-10, 8.064e-10, 9.109e-10,
9.839e-10, 1.028e-9, 1.044e-9, 1.029e-9, 1.005e-9, 1.002e-9,
1.038e-9, 1.122e-9, 1.233e-9, 1.372e-9, 1.524e-9, 1.665e-9,
1.804e-9, 1.908e-9, 2.015e-9, 2.117e-9, 2.219e-9, 2.336e-9,
2.531e-9, 2.805e-9, 3.189e-9, 3.617e-9, 4.208e-9, 4.911e-9,
5.619e-9, 6.469e-9, 7.188e-9, 7.957e-9, 8.503e-9, 9.028e-9,
9.571e-9, 9.99e-9, 1.055e-8, 1.102e-8, 1.132e-8, 1.141e-8,
1.145e-8, 1.145e-8, 1.176e-8, 1.224e-8, 1.304e-8, 1.388e-8,
1.445e-8, 1.453e-8, 1.368e-8, 1.22e-8, 1.042e-8, 8.404e-9,
6.403e-9, 4.643e-9, 3.325e-9, 2.335e-9, 1.638e-9, 1.19e-9,
9.161e-10, 7.412e-10, 6.226e-10, 5.516e-10, 5.068e-10, 4.831e-10,
4.856e-10, 5.162e-10, 5.785e-10, 6.539e-10, 7.485e-10, 8.565e-10,
9.534e-10, 1.052e-9, 1.115e-9, 1.173e-9, 1.203e-9, 1.224e-9,
1.243e-9, 1.248e-9, 1.261e-9, 1.265e-9, 1.25e-9, 1.217e-9,
1.176e-9, 1.145e-9, 1.153e-9, 1.199e-9, 1.278e-9, 1.366e-9,
1.426e-9, 1.444e-9, 1.365e-9, 1.224e-9, 1.051e-9, 8.539e-10,
6.564e-10, 4.751e-10, 3.404e-10, 2.377e-10, 1.631e-10, 1.114e-10,
7.87e-11, 5.793e-11, 4.284e-11, 3.3e-11, 2.62e-11, 2.152e-11,
1.777e-11, 1.496e-11, 1.242e-11, 1.037e-11, 8.725e-12, 7.004e-12,
5.718e-12, 4.769e-12, 3.952e-12, 3.336e-12, 2.712e-12, 2.213e-12,
1.803e-12, 1.492e-12, 1.236e-12, 1.006e-12, 8.384e-13, 7.063e-13,
5.879e-13, 4.93e-13, 4.171e-13, 3.569e-13, 3.083e-13, 2.688e-13,
2.333e-13, 2.035e-13, 1.82e-13, 1.682e-13, 1.635e-13, 1.674e-13,
1.769e-13, 2.022e-13, 2.485e-13, 3.127e-13, 4.25e-13, 5.928e-13,
8.514e-13, 1.236e-12, 1.701e-12, 2.392e-12, 3.231e-12, 4.35e-12,
5.559e-12, 6.915e-12, 8.519e-12, 1.013e-11, 1.146e-11, 1.24e-11,
1.305e-11, 1.333e-11, 1.318e-11, 1.263e-11, 1.238e-11, 1.244e-11,
1.305e-11, 1.432e-11, 1.623e-11, 1.846e-11, 2.09e-11, 2.328e-11,
2.526e-11, 2.637e-11, 2.702e-11, 2.794e-11, 2.889e-11, 2.989e-11,
3.231e-11, 3.68e-11, 4.375e-11, 5.504e-11, 7.159e-11, 9.502e-11,
1.279e-10, 1.645e-10, 2.098e-10, 2.618e-10, 3.189e-10, 3.79e-10,
4.303e-10, 4.753e-10, 5.027e-10, 5.221e-10, 5.293e-10, 5.346e-10,
5.467e-10, 5.796e-10, 6.2e-10, 6.454e-10, 6.705e-10, 6.925e-10,
7.233e-10, 7.35e-10, 7.538e-10, 7.861e-10, 8.077e-10, 8.132e-10,
7.749e-10, 7.036e-10, 6.143e-10, 5.093e-10, 4.089e-10, 3.092e-10,
2.299e-10, 1.705e-10, 1.277e-10, 9.723e-11, 7.533e-11, 6.126e-11,
5.154e-11, 4.428e-11, 3.913e-11, 3.521e-11, 3.297e-11, 3.275e-11,
3.46e-11, 3.798e-11, 4.251e-11, 4.745e-11, 5.232e-11, 5.606e-11,
5.82e-11, 5.88e-11, 5.79e-11, 5.661e-11, 5.491e-11, 5.366e-11,
5.341e-11, 5.353e-11, 5.336e-11, 5.293e-11, 5.248e-11, 5.235e-11,
5.208e-11, 5.322e-11, 5.521e-11, 5.725e-11, 5.827e-11, 5.685e-11,
5.245e-11, 4.612e-11, 3.884e-11, 3.129e-11, 2.404e-11, 1.732e-11,
1.223e-11, 8.574e-12, 5.888e-12, 3.986e-12, 2.732e-12, 1.948e-12,
1.414e-12, 1.061e-12, 8.298e-13, 6.612e-13, 5.413e-13, 4.472e-13,
3.772e-13, 3.181e-13, 2.645e-13, 2.171e-13, 1.778e-13, 1.464e-13,
1.183e-13, 9.637e-14, 7.991e-14, 6.668e-14, 5.57e-14, 4.663e-14,
3.848e-14, 3.233e-14, 2.706e-14, 2.284e-14, 1.944e-14, 1.664e-14,
1.43e-14, 1.233e-14, 1.066e-14, 9.234e-15, 8.023e-15, 6.993e-15,
6.119e-15, 5.384e-15, 4.774e-15, 4.283e-15, 3.916e-15, 3.695e-15,
3.682e-15, 4.004e-15, 4.912e-15, 6.853e-15, 1.056e-14, 1.712e-14,
2.804e-14, 4.516e-14, 7.113e-14, 1.084e-13, 1.426e-13, 1.734e-13,
1.978e-13, 2.194e-13, 2.388e-13, 2.489e-13, 2.626e-13, 2.865e-13,
3.105e-13, 3.387e-13, 3.652e-13, 3.984e-13, 4.398e-13, 4.906e-13,
5.55e-13, 6.517e-13, 7.813e-13, 9.272e-13, 1.164e-12, 1.434e-12,
1.849e-12, 2.524e-12, 3.328e-12, 4.523e-12, 6.108e-12, 8.207e-12,
1.122e-11, 1.477e-11, 1.9e-11, 2.412e-11, 2.984e-11, 3.68e-11,
4.353e-11, 4.963e-11, 5.478e-11, 5.903e-11, 6.233e-11, 6.483e-11,
6.904e-11, 7.569e-11, 8.719e-11, 1.048e-10, 1.278e-10, 1.557e-10,
1.869e-10, 2.218e-10, 2.61e-10, 2.975e-10, 3.371e-10, 3.746e-10,
4.065e-10, 4.336e-10, 4.503e-10, 4.701e-10, 4.8e-10, 4.917e-10,
5.038e-10, 5.128e-10, 5.143e-10, 5.071e-10, 5.019e-10, 5.025e-10,
5.183e-10, 5.496e-10, 5.877e-10, 6.235e-10, 6.42e-10, 6.234e-10,
5.698e-10, 4.916e-10, 4.022e-10, 3.126e-10, 2.282e-10, 1.639e-10,
1.142e-10, 7.919e-11, 5.69e-11, 4.313e-11, 3.413e-11, 2.807e-11,
2.41e-11, 2.166e-11, 2.024e-11, 1.946e-11, 1.929e-11, 1.963e-11,
2.035e-11, 2.162e-11, 2.305e-11, 2.493e-11, 2.748e-11, 3.048e-11,
3.413e-11, 3.754e-11, 4.155e-11, 4.635e-11, 5.11e-11, 5.734e-11,
6.338e-11, 6.99e-11, 7.611e-11, 8.125e-11, 8.654e-11, 8.951e-11,
9.182e-11, 9.31e-11, 9.273e-11, 9.094e-11, 8.849e-11, 8.662e-11,
8.67e-11, 8.972e-11, 9.566e-11, 1.025e-10, 1.083e-10, 1.111e-10,
1.074e-10, 9.771e-11, 8.468e-11, 6.958e-11, 5.47e-11, 4.04e-11,
2.94e-11, 2.075e-11, 1.442e-11, 1.01e-11, 7.281e-12, 5.409e-12,
4.138e-12, 3.304e-12, 2.784e-12, 2.473e-12, 2.273e-12, 2.186e-12,
2.118e-12, 2.066e-12, 1.958e-12, 1.818e-12, 1.675e-12, 1.509e-12,
1.349e-12, 1.171e-12, 9.838e-13, 8.213e-13, 6.765e-13, 5.378e-13,
4.161e-13, 3.119e-13, 2.279e-13, 1.637e-13, 1.152e-13, 8.112e-14,
5.919e-14, 4.47e-14, 3.492e-14, 2.811e-14, 2.319e-14, 1.948e-14,
```

```
1.66e-14, 1.432e-14, 1.251e-14, 1.109e-14, 1.006e-14, 9.45e-15,
9.384e-15, 1.012e-14, 1.216e-14, 1.636e-14, 2.305e-14, 3.488e-14,
5.572e-14, 8.479e-14, 1.265e-13, 1.905e-13, 2.73e-13, 3.809e-13,
4.955e-13, 6.303e-13, 7.861e-13, 9.427e-13, 1.097e-12, 1.212e-12,
1.328e-12, 1.415e-12, 1.463e-12, 1.495e-12, 1.571e-12, 1.731e-12,
1.981e-12, 2.387e-12, 2.93e-12, 3.642e-12, 4.584e-12, 5.822e-12,
7.278e-12, 9.193e-12, 1.135e-11, 1.382e-11, 1.662e-11, 1.958e-11,
2.286e-11, 2.559e-11, 2.805e-11, 2.988e-11, 3.106e-11, 3.182e-11,
3.2e-11, 3.258e-11, 3.362e-11, 3.558e-11, 3.688e-11, 3.8e-11,
3.929e-11, 4.062e-11, 4.186e-11, 4.293e-11, 4.48e-11, 4.643e-11,
4.704e-11, 4.571e-11, 4.206e-11, 3.715e-11, 3.131e-11, 2.541e-11,
1.978e-11, 1.508e-11, 1.146e-11, 8.7e-12, 6.603e-12, 5.162e-12,
4.157e-12, 3.408e-12, 2.829e-12, 2.405e-12, 2.071e-12, 1.826e-12,
1.648e-12, 1.542e-12, 1.489e-12, 1.485e-12, 1.493e-12, 1.545e-12,
1.637e-12, 1.814e-12, 2.061e-12, 2.312e-12, 2.651e-12, 3.03e-12,
3.46e-12, 3.901e-12, 4.306e-12, 4.721e-12, 5.008e-12, 5.281e-12,
5.541e-12, 5.791e-12, 6.115e-12, 6.442e-12, 6.68e-12, 6.791e-12,
6.831e-12, 6.839e-12, 6.946e-12, 7.128e-12, 7.537e-12, 8.036e-12,
8.392e-12, 8.526e-12, 8.11e-12, 7.325e-12, 6.329e-12, 5.183e-12,
4.081e-12, 2.985e-12, 2.141e-12, 1.492e-12, 1.015e-12, 6.684e-13,
4.414e-13, 2.987e-13, 2.038e-13, 1.391e-13, 9.86e-14, 7.24e-14,
5.493e-14, 4.288e-14, 3.427e-14, 2.787e-14, 2.296e-14, 1.909e-14,
1.598e-14, 1.344e-14, 1.135e-14, 9.616e-15, 8.169e-15, 6.957e-15,
5.938e-15, 5.08e-15, 4.353e-15, 3.738e-15, 3.217e-15, 2.773e-15,
2.397e-15, 2.077e-15, 1.805e-15, 1.575e-15, 1.382e-15, 1.221e-15,
1.09e-15, 9.855e-16, 9.068e-16, 8.537e-16, 8.27e-16, 8.29e-16,
8.634e-16, 9.359e-16, 1.055e-15, 1.233e-15, 1.486e-15, 1.839e-15,
2.326e-15, 2.998e-15, 3.934e-15, 5.256e-15, 7.164e-15, 9.984e-15,
1.427e-14, 2.099e-14, 3.196e-14, 5.121e-14, 7.908e-14, 1.131e-13,
1.602e-13, 2.239e-13, 3.075e-13, 4.134e-13, 5.749e-13, 7.886e-13,
1.071e-12, 1.464e-12, 2.032e-12, 2.8e-12, 3.732e-12, 4.996e-12,
6.483e-12, 8.143e-12, 1.006e-11, 1.238e-11, 1.484e-11, 1.744e-11,
2.02e-11, 2.274e-11, 2.562e-11, 2.848e-11, 3.191e-11, 3.617e-11,
4.081e-11, 4.577e-11, 4.937e-11, 5.204e-11, 5.401e-11, 5.462e-11,
5.507e-11, 5.51e-11, 5.605e-11, 5.686e-11, 5.739e-11, 5.766e-11,
5.74e-11, 5.754e-11, 5.761e-11, 5.777e-11, 5.712e-11, 5.51e-11,
5.088e-11, 4.438e-11, 3.728e-11, 2.994e-11, 2.305e-11, 1.715e-11,
1.256e-11, 9.208e-12, 6.745e-12, 5.014e-12, 3.785e-12, 2.9e-12,
2.239e-12, 1.757e-12, 1.414e-12, 1.142e-12, 9.482e-13, 8.01e-13,
6.961e-13, 6.253e-13, 5.735e-13, 5.433e-13, 5.352e-13, 5.493e-13,
5.706e-13, 6.068e-13, 6.531e-13, 7.109e-13, 7.767e-13, 8.59e-13,
9.792e-13, 1.142e-12, 1.371e-12, 1.65e-12, 1.957e-12, 2.302e-12,
2.705e-12, 3.145e-12, 3.608e-12, 4.071e-12, 4.602e-12, 5.133e-12,
5.572e-12, 5.987e-12, 6.248e-12, 6.533e-12, 6.757e-12, 6.935e-12,
7.224e-12, 7.422e-12, 7.538e-12, 7.547e-12, 7.495e-12, 7.543e-12,
7.725e-12, 8.139e-12, 8.627e-12, 9.146e-12, 9.443e-12, 9.318e-12,
8.649e-12, 7.512e-12, 6.261e-12, 4.915e-12, 3.647e-12, 2.597e-12,
1.785e-12, 1.242e-12, 8.66e-13, 6.207e-13, 4.61e-13, 3.444e-13,
2.634e-13, 2.1e-13, 1.725e-13, 1.455e-13, 1.237e-13, 1.085e-13,
9.513e-14, 7.978e-14, 6.603e-14, 5.288e-14, 4.084e-14, 2.952e-14,
2.157e-14, 1.593e-14, 1.199e-14, 9.267e-15, 7.365e-15, 6.004e-15,
4.995e-15, 4.218e-15, 3.601e-15, 3.101e-15, 2.692e-15, 2.36e-15,
2.094e-15, 1.891e-15, 1.755e-15, 1.699e-15, 1.755e-15, 1.987e-15,
2.506e-15, 3.506e-15, 5.289e-15, 8.311e-15, 1.325e-14, 2.129e-14,
3.237e-14, 4.595e-14, 6.441e-14, 8.433e-14, 1.074e-13, 1.383e-13,
1.762e-13, 2.281e-13, 2.831e-13, 3.523e-13, 4.38e-13, 5.304e-13,
6.29e-13, 7.142e-13, 8.032e-13, 8.934e-13, 9.888e-13, 1.109e-12,
1.261e-12, 1.462e-12, 1.74e-12, 2.099e-12, 2.535e-12, 3.008e-12,
3.462e-12, 3.856e-12, 4.098e-12, 4.239e-12, 4.234e-12, 4.132e-12,
3.986e-12, 3.866e-12, 3.829e-12, 3.742e-12, 3.705e-12, 3.694e-12,
3.765e-12, 3.849e-12, 3.929e-12, 4.056e-12, 4.092e-12, 4.047e-12,
3.792e-12, 3.407e-12, 2.953e-12, 2.429e-12, 1.931e-12, 1.46e-12,
1.099e-12, 8.199e-13, 6.077e-13, 4.449e-13, 3.359e-13, 2.524e-13,
1.881e-13, 1.391e-13, 1.02e-13, 7.544e-14, 5.555e-14, 4.22e-14,
3.321e-14, 2.686e-14, 2.212e-14, 1.78e-14, 1.369e-14, 1.094e-14,
9.13e-15, 8.101e-15, 7.828e-15, 8.393e-15, 1.012e-14, 1.259e-14,
1.538e-14, 1.961e-14, 2.619e-14, 3.679e-14, 5.049e-14, 6.917e-14,
8.88e-14, 1.115e-13, 1.373e-13, 1.619e-13, 1.878e-13, 2.111e-13,
2.33e-13, 2.503e-13, 2.613e-13, 2.743e-13, 2.826e-13, 2.976e-13,
3.162e-13, 3.36e-13, 3.491e-13, 3.541e-13, 3.595e-13, 3.608e-13,
3.709e-13, 3.869e-13, 4.12e-13, 4.366e-13, 4.504e-13, 4.379e-13,
3.955e-13, 3.385e-13, 2.741e-13, 2.089e-13, 1.427e-13, 9.294e-14,
5.775e-14, 3.565e-14, 2.21e-14, 1.398e-14, 9.194e-15, 6.363e-15,
4.644e-15, 3.55e-15, 2.808e-15, 2.274e-15, 1.871e-15, 1.557e-15,
1.308e-15, 1.108e-15, 9.488e-16, 8.222e-16, 7.238e-16, 6.506e-16,
6.008e-16, 5.742e-16, 5.724e-16, 5.991e-16, 6.625e-16, 7.775e-16,
9.734e-16, 1.306e-15, 1.88e-15, 2.879e-15, 4.616e-15, 7.579e-15,
1.248e-14, 2.03e-14, 3.244e-14, 5.171e-14, 7.394e-14, 9.676e-14,
1.199e-13, 1.467e-13, 1.737e-13, 2.02e-13, 2.425e-13, 3.016e-13,
3.7e-13, 4.617e-13, 5.949e-13, 7.473e-13, 9.378e-13, 1.191e-12,
1.481e-12, 1.813e-12, 2.232e-12, 2.722e-12, 3.254e-12, 3.845e-12,
4.458e-12, 5.048e-12, 5.511e-12, 5.898e-12, 6.204e-12, 6.293e-12,
6.386e-12, 6.467e-12, 6.507e-12, 6.466e-12, 6.443e-12, 6.598e-12,
6.873e-12, 7.3e-12, 7.816e-12, 8.368e-12, 8.643e-12, 8.466e-12,
7.871e-12, 6.853e-12, 5.714e-12, 4.482e-12, 3.392e-12, 2.613e-12,
2.008e-12, 1.562e-12, 1.228e-12, 9.888e-13, 7.646e-13, 5.769e-13,
4.368e-13, 3.324e-13, 2.508e-13, 1.916e-13
```

```
  };

  static double xfcrev[15] =
    { 1.003, 1.009, 1.015, 1.023, 1.029, 1.033, 1.037,
    1.039, 1.04, 1.046, 1.036, 1.027, 1.01, 1.002, 1.
  };

  double sfac;

  /* Get H2O continuum absorption... */
  const double xw = nu / 10 + 1;
  if (xw >= 1 && xw < 2001) {
    const int iw = (int) xw;
    const double dw = xw - iw;
    const double ew = 1 - dw;
    const double cw296 = ew * h2o296[iw - 1] + dw * h2o296[iw];
    const double cw260 = ew * h2o260[iw - 1] + dw * h2o260[iw];
    const double cwfrn = ew * h2ofrn[iw - 1] + dw * h2ofrn[iw];
    if (nu <= 820 || nu >= 960) {
      sfac = 1;
    } else {
      const double xx = (nu - 820) / 10;
      const int ix = (int) xx;
      const double dx = xx - ix;
      sfac = (1 - dx) * xfcrev[ix] + dx * xfcrev[ix + 1];
    }
    const double ctwslf =
      sfac * cw296 * pow(cw260 / cw296, (296 - t) / (296 - 260));
    const double vf2 = POW2(nu - 370);
    const double vf6 = POW3(vf2);
    const double fscal = 36100 / (vf2 + vf6 * 1e-8 + 36100) * -.25 + 1;
    const double ctwfrn = cwfrn * fscal;
    const double a1 = nu * u * tanh(.7193876 / t * nu);
    const double a2 = 296 / t;
    const double a3 = p / P0 * (q * ctwslf + (1 - q) * ctwfrn) * 1e-20;
    return a1 * a2 * a3;
  } else
    return 0;
}

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

double ctmn2(
  const double nu,
  const double p,
  const double t) {

  static double ba[98] = { 0., 4.45e-8, 5.22e-8, 6.46e-8, 7.75e-8, 9.03e-8,
    1.06e-7, 1.21e-7, 1.37e-7, 1.57e-7, 1.75e-7, 2.01e-7, 2.3e-7,
    2.59e-7, 2.95e-7, 3.26e-7, 3.66e-7, 4.05e-7, 4.47e-7, 4.92e-7,
    5.34e-7, 5.84e-7, 6.24e-7, 6.67e-7, 7.14e-7, 7.26e-7, 7.54e-7,
    7.84e-7, 8.09e-7, 8.42e-7, 8.62e-7, 8.87e-7, 9.11e-7, 9.36e-7,
    9.76e-7, 1.03e-6, 1.11e-6, 1.23e-6, 1.39e-6, 1.61e-6, 1.76e-6,
    1.94e-6, 1.97e-6, 1.87e-6, 1.75e-6, 1.56e-6, 1.42e-6, 1.35e-6,
    1.32e-6, 1.29e-6, 1.29e-6, 1.29e-6, 1.3e-6, 1.32e-6, 1.33e-6,
    1.34e-6, 1.35e-6, 1.33e-6, 1.31e-6, 1.29e-6, 1.24e-6, 1.2e-6,
    1.16e-6, 1.1e-6, 1.04e-6, 9.96e-7, 9.38e-7, 8.63e-7, 7.98e-7,
    7.26e-7, 6.55e-7, 5.94e-7, 5.35e-7, 4.74e-7, 4.24e-7, 3.77e-7,
    3.33e-7, 2.96e-7, 2.63e-7, 2.34e-7, 2.08e-7, 1.85e-7, 1.67e-7,
    1.47e-7, 1.32e-7, 1.2e-7, 1.09e-7, 9.85e-8, 9.08e-8, 8.18e-8,
    7.56e-8, 6.85e-8, 6.14e-8, 5.83e-8, 5.77e-8, 5e-8, 4.32e-8, 0.
  };

  static double betaa[98] = { 802., 802., 761., 722., 679., 646., 609., 562.,
    511., 472., 436., 406., 377., 355., 338., 319., 299., 278., 255.,
    233., 208., 184., 149., 107., 66., 25., -13., -49., -82., -104.,
    -119., -130., -139., -144., -146., -146., -147., -148., -150.,
    -153., -160., -169., -181., -189., -195., -200., -205., -209.,
    -211., -210., -210., -209., -205., -199., -190., -180., -168.,
    -157., -143., -126., -108., -89., -63., -32., 1., 35., 65., 95.,
    121., 141., 152., 161., 164., 164., 161., 155., 148., 143., 137.,
    133., 131., 133., 139., 150., 165., 187., 213., 248., 284., 321.,
    372., 449., 514., 569., 609., 642., 673., 673.
  };

  static double nua[98] = { 2120., 2125., 2130., 2135., 2140., 2145., 2150.,
    2155., 2160., 2165., 2170., 2175., 2180., 2185., 2190., 2195.,
    2200., 2205., 2210., 2215., 2220., 2225., 2230., 2235., 2240.,
    2245., 2250., 2255., 2260., 2265., 2270., 2275., 2280., 2285.,
    2290., 2295., 2300., 2305., 2310., 2315., 2320., 2325., 2330.,
    2335., 2340., 2345., 2350., 2355., 2360., 2365., 2370., 2375.,
    2380., 2385., 2390., 2395., 2400., 2405., 2410., 2415., 2420.,
    2425., 2430., 2435., 2440., 2445., 2450., 2455., 2460., 2465.,
    2470., 2475., 2480., 2485., 2490., 2495., 2500., 2505., 2510.,
    2515., 2520., 2525., 2530., 2535., 2540., 2545., 2550., 2555.,
    2560., 2565., 2570., 2575., 2580., 2585., 2590., 2595., 2600., 2605.
  };
```

```
  const double q_n2 = 0.79, t0 = 273.0, tr = 296.0;

  /* Check wavenumber range... */
  if (nu < nua[0] || nu > nua[97])
    return 0;

  /* Interpolate B and beta... */
  const int idx = locate_reg(nua, 98, nu);
  const double b = LIN(nua[idx], ba[idx], nua[idx + 1], ba[idx + 1], nu);
  const double beta =
    LIN(nua[idx], betaa[idx], nua[idx + 1], betaa[idx + 1], nu);

  /* Compute absorption coefficient... */
  return 0.1 * POW2(p / P0 * t0 / t) * exp(beta * (1 / tr - 1 / t))
    * q_n2 * b * (q_n2 + (1 - q_n2) * (1.294 - 0.4545 * t / tr));
}

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

double ctmo2(
  const double nu,
  const double p,
  const double t) {

  static double ba[90] = { 0., .061, .074, .084, .096, .12, .162, .208, .246,
    .285, .314, .38, .444, .5, .571, .673, .768, .853, .966, 1.097,
    1.214, 1.333, 1.466, 1.591, 1.693, 1.796, 1.922, 2.037, 2.154,
    2.264, 2.375, 2.508, 2.671, 2.847, 3.066, 3.417, 3.828, 4.204,
    4.453, 4.599, 4.528, 4.284, 3.955, 3.678, 3.477, 3.346, 3.29,
    3.251, 3.231, 3.226, 3.212, 3.192, 3.108, 3.033, 2.911, 2.798,
    2.646, 2.508, 2.322, 2.13, 1.928, 1.757, 1.588, 1.417, 1.253,
    1.109, .99, .888, .791, .678, .587, .524, .464, .403, .357, .32,
    .29, .267, .242, .215, .182, .16, .146, .128, .103, .087, .081,
    .071, .064, 0.
  };

  static double betaa[90] = { 467., 467., 400., 315., 379., 368., 475., 521.,
    531., 512., 442., 444., 430., 381., 335., 324., 296., 248., 215.,
    193., 158., 127., 101., 71., 31., -6., -26., -47., -63., -79.,
    -88., -88., -87., -90., -98., -99., -109., -134., -160., -167.,
    -164., -158., -153., -151., -156., -166., -168., -173., -170.,
    -161., -145., -126., -108., -84., -59., -29., 4., 41., 73., 97.,
    123., 159., 198., 220., 242., 256., 281., 311., 334., 319., 313.,
    321., 323., 310., 315., 320., 335., 361., 378., 373., 338., 319.,
    346., 322., 291., 290., 350., 371., 504., 504.
  };

  static double nua[90] = { 1360., 1365., 1370., 1375., 1380., 1385., 1390.,
    1395., 1400., 1405., 1410., 1415., 1420., 1425., 1430., 1435.,
    1440., 1445., 1450., 1455., 1460., 1465., 1470., 1475., 1480.,
    1485., 1490., 1495., 1500., 1505., 1510., 1515., 1520., 1525.,
    1530., 1535., 1540., 1545., 1550., 1555., 1560., 1565., 1570.,
    1575., 1580., 1585., 1590., 1595., 1600., 1605., 1610., 1615.,
    1620., 1625., 1630., 1635., 1640., 1645., 1650., 1655., 1660.,
    1665., 1670., 1675., 1680., 1685., 1690., 1695., 1700., 1705.,
    1710., 1715., 1720., 1725., 1730., 1735., 1740., 1745., 1750.,
    1755., 1760., 1765., 1770., 1775., 1780., 1785., 1790., 1795.,
    1800., 1805.
  };

  const double q_o2 = 0.21, t0 = 273, tr = 296;

  /* Check wavenumber range... */
  if (nu < nua[0] || nu > nua[89])
    return 0;

  /* Interpolate B and beta... */
  const int idx = locate_reg(nua, 90, nu);
  const double b = LIN(nua[idx], ba[idx], nua[idx + 1], ba[idx + 1], nu);
  const double beta =
    LIN(nua[idx], betaa[idx], nua[idx + 1], betaa[idx + 1], nu);

  /* Compute absorption coefficient... */
  return 0.1 * POW2(p / P0 * t0 / t) * exp(beta * (1 / tr - 1 / t)) * q_o2 *
    b;
}

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

void copy_atm(
  const ctl_t *ctl,
  atm_t *atm_dest,
  const atm_t *atm_src,
  const int init) {

  /* Data size... */
```

```
  const size_t s = (size_t) atm_src->np * sizeof(double);

  /* Copy data... */
  atm_dest->np = atm_src->np;
  memcpy(atm_dest->time, atm_src->time, s);
  memcpy(atm_dest->z, atm_src->z, s);
  memcpy(atm_dest->lon, atm_src->lon, s);
  memcpy(atm_dest->lat, atm_src->lat, s);
  memcpy(atm_dest->p, atm_src->p, s);
  memcpy(atm_dest->t, atm_src->t, s);
  for (int ig = 0; ig < ctl->ng; ig++)
    memcpy(atm_dest->q[ig], atm_src->q[ig], s);
  for (int iw = 0; iw < ctl->nw; iw++)
    memcpy(atm_dest->k[iw], atm_src->k[iw], s);
  atm_dest->clz = atm_src->clz;
  atm_dest->cldz = atm_src->cldz;
  for (int icl = 0; icl < ctl->ncl; icl++)
    atm_dest->clk[icl] = atm_src->clk[icl];
  atm_dest->sfz = atm_src->sfz;
  atm_dest->sfp = atm_src->sfp;
  atm_dest->sft = atm_src->sft;
  for (int isf = 0; isf < ctl->nsf; isf++)
    atm_dest->sfeps[isf] = atm_src->sfeps[isf];

  /* Initialize... */
  if (init)
    for (int ip = 0; ip < atm_dest->np; ip++) {
      atm_dest->p[ip] = 0;
      atm_dest->t[ip] = 0;
      for (int ig = 0; ig < ctl->ng; ig++)
        atm_dest->q[ig][ip] = 0;
      for (int iw = 0; iw < ctl->nw; iw++)
        atm_dest->k[iw][ip] = 0;
      atm_dest->clz = 0;
      atm_dest->cldz = 0;
      for (int icl = 0; icl < ctl->ncl; icl++)
        atm_dest->clk[icl] = 0;
      atm_dest->sfz = 0;
      atm_dest->sfp = 0;
      atm_dest->sft = 0;
      for (int isf = 0; isf < ctl->nsf; isf++)
        atm_dest->sfeps[isf] = 1;
    }
}

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

void copy_obs(
  const ctl_t *ctl,
  obs_t *obs_dest,
  const obs_t *obs_src,
  const int init) {

  /* Data size... */
  const size_t s = (size_t) obs_src->nr * sizeof(double);

  /* Copy data... */
  obs_dest->nr = obs_src->nr;
  memcpy(obs_dest->time, obs_src->time, s);
  memcpy(obs_dest->obsz, obs_src->obsz, s);
  memcpy(obs_dest->obslon, obs_src->obslon, s);
  memcpy(obs_dest->obslat, obs_src->obslat, s);
  memcpy(obs_dest->vpz, obs_src->vpz, s);
  memcpy(obs_dest->vplon, obs_src->vplon, s);
  memcpy(obs_dest->vplat, obs_src->vplat, s);
  memcpy(obs_dest->tpz, obs_src->tpz, s);
  memcpy(obs_dest->tplon, obs_src->tplon, s);
  memcpy(obs_dest->tplat, obs_src->tplat, s);
  for (int id = 0; id < ctl->nd; id++)
    memcpy(obs_dest->rad[id], obs_src->rad[id], s);
  for (int id = 0; id < ctl->nd; id++)
    memcpy(obs_dest->tau[id], obs_src->tau[id], s);

  /* Initialize... */
  if (init)
    for (int id = 0; id < ctl->nd; id++)
      for (int ir = 0; ir < obs_dest->nr; ir++)
        if (isfinite(obs_dest->rad[id][ir])) {
          obs_dest->rad[id][ir] = 0;
          obs_dest->tau[id][ir] = 0;
        }
}

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

int find_emitter(
  const ctl_t *ctl,
```

```
  const char *emitter) {

  for (int ig = 0; ig < ctl->ng; ig++)
    if (strcasecmp(ctl->emitter[ig], emitter) == 0)
      return ig;

  return -1;
}

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

void formod(
  const ctl_t *ctl,
  atm_t *atm,
  obs_t *obs) {

  int *mask;

  /* Allocate... */
  ALLOC(mask, int,
        ND * NR);

  /* Save observation mask... */
  for (int id = 0; id < ctl->nd; id++)
    for (int ir = 0; ir < obs->nr; ir++)
      mask[id * NR + ir] = !isfinite(obs->rad[id][ir]);

  /* Hydrostatic equilibrium... */
  hydrostatic(ctl, atm);

  /* CGA or EGA forward model... */
  if (ctl->formod == 0 || ctl->formod == 1)
    for (int ir = 0; ir < obs->nr; ir++)
      formod_pencil(ctl, atm, obs, ir);

  /* Call RFM... */
  else if (ctl->formod == 2)
    formod_rfm(ctl, atm, obs);

  /* Apply field-of-view convolution... */
  formod_fov(ctl, obs);

  /* Convert radiance to brightness temperature... */
  if (ctl->write_bbt)
    for (int id = 0; id < ctl->nd; id++)
      for (int ir = 0; ir < obs->nr; ir++)
        obs->rad[id][ir] = BRIGHT(obs->rad[id][ir], ctl->nu[id]);

  /* Apply observation mask... */
  for (int id = 0; id < ctl->nd; id++)
    for (int ir = 0; ir < obs->nr; ir++)
      if (mask[id * NR + ir])
        obs->rad[id][ir] = NAN;

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

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

void formod_continua(
  const ctl_t *ctl,
  const los_t *los,
  const int ip,
  double *beta) {

  static int ig_co2 = -999, ig_h2o = -999;

  /* Extinction... */
  for (int id = 0; id < ctl->nd; id++)
    beta[id] = los->k[ip][id];

  /* CO2 continuum... */
  if (ctl->ctm_co2) {
    if (ig_co2 == -999)
      ig_co2 = find_emitter(ctl, "CO2");
    if (ig_co2 >= 0)
      for (int id = 0; id < ctl->nd; id++)
        beta[id] += ctmco2(ctl->nu[id], los->p[ip], los->t[ip],
                           los->u[ip][ig_co2]) / los->ds[ip];
  }

  /* H2O continuum... */
  if (ctl->ctm_h2o) {
    if (ig_h2o == -999)
      ig_h2o = find_emitter(ctl, "H2O");
    if (ig_h2o >= 0)
```

```
      for (int id = 0; id < ctl->nd; id++)
        beta[id] += ctmh2o(ctl->nu[id], los->p[ip], los->t[ip],
                           los->q[ip][ig_h2o], los->u[ip][ig_h2o])
          / los->ds[ip];
  }

  /* N2 continuum... */
  if (ctl->ctm_n2)
    for (int id = 0; id < ctl->nd; id++)
      beta[id] += ctmn2(ctl->nu[id], los->p[ip], los->t[ip]);

  /* O2 continuum... */
  if (ctl->ctm_o2)
    for (int id = 0; id < ctl->nd; id++)
      beta[id] += ctmo2(ctl->nu[id], los->p[ip], los->t[ip]);
}

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

void formod_fov(
  const ctl_t *ctl,
  obs_t *obs) {

  static double dz[NSHAPE], w[NSHAPE];

  static int init = 0, n;

  obs_t *obs2;

  double rad[ND][NR], tau[ND][NR], z[NR];

  /* Do not take into account FOV... */
  if (ctl->fov[0] == '-')
    return;

  /* Initialize FOV data... */
  if (!init) {
    init = 1;
    read_shape(ctl->fov, dz, w, &n);
  }

  /* Allocate... */
  ALLOC(obs2, obs_t, 1);

  /* Copy observation data... */
  copy_obs(ctl, obs2, obs, 0);

  /* Loop over ray paths... */
  for (int ir = 0; ir < obs->nr; ir++) {

    /* Get radiance and transmittance profiles... */
    int nz = 0;
    for (int ir2 = MAX(ir - NFOV, 0);
         ir2 < MIN(ir + 1 + NFOV, obs->nr); ir2++)
      if (obs->time[ir2] == obs->time[ir]) {
        z[nz] = obs2->vpz[ir2];
        for (int id = 0; id < ctl->nd; id++) {
          rad[id][nz] = obs2->rad[id][ir2];
          tau[id][nz] = obs2->tau[id][ir2];
        }
        nz++;
      }
    if (nz < 2)
      ERRMSG("Cannot apply FOV convolution!");

    /* Convolute profiles with FOV... */
    double wsum = 0;
    for (int id = 0; id < ctl->nd; id++) {
      obs->rad[id][ir] = 0;
      obs->tau[id][ir] = 0;
    }
    for (int i = 0; i < n; i++) {
      const double zfov = obs->vpz[ir] + dz[i];
      const int idx = locate_irr(z, nz, zfov);
      for (int id = 0; id < ctl->nd; id++) {
        obs->rad[id][ir] += w[i]
          * LIN(z[idx], rad[id][idx], z[idx + 1], rad[id][idx + 1], zfov);
        obs->tau[id][ir] += w[i]
          * LIN(z[idx], tau[id][idx], z[idx + 1], tau[id][idx + 1], zfov);
      }
      wsum += w[i];
    }
    for (int id = 0; id < ctl->nd; id++) {
      obs->rad[id][ir] /= wsum;
      obs->tau[id][ir] /= wsum;
    }
  }
```

```
  /* Free... */
  free(obs2);
}

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

void formod_pencil(
  const ctl_t *ctl,
  const atm_t *atm,
  obs_t *obs,
  const int ir) {

  static tbl_t *tbl;

  static int init = 0;

  los_t *los;

  double beta_ctm[ND], rad[ND], tau[ND], tau_refl[ND],
    tau_path[ND][NG], tau_gas[ND], x0[3], x1[3];

  /* Initialize look-up tables... */
  if (!init) {
    init = 1;
    ALLOC(tbl, tbl_t, 1);
    read_tbl(ctl, tbl);
    init_srcfunc(ctl, tbl);
  }

  /* Allocate... */
  ALLOC(los, los_t, 1);

  /* Initialize... */
  for (int id = 0; id < ctl->nd; id++) {
    rad[id] = 0;
    tau[id] = 1;
    for (int ig = 0; ig < ctl->ng; ig++)
      tau_path[id][ig] = 1;
  }

  /* Raytracing... */
  raytrace(ctl, atm, obs, los, ir);

  /* Loop over LOS points... */
  for (int ip = 0; ip < los->np; ip++) {

    /* Get trace gas transmittance... */
    if (ctl->formod == 0)
      intpol_tbl_cga(ctl, tbl, los, ip, tau_path, tau_gas);
    else
      intpol_tbl_ega(ctl, tbl, los, ip, tau_path, tau_gas);

    /* Get continuum absorption... */
    formod_continua(ctl, los, ip, beta_ctm);

    /* Compute Planck function... */
    formod_srcfunc(ctl, tbl, los->t[ip], los->src[ip]);

    /* Loop over channels... */
    for (int id = 0; id < ctl->nd; id++)
      if (tau_gas[id] > 0) {

        /* Get segment emissivity... */
        los->eps[ip][id] = 1 - tau_gas[id] * exp(-beta_ctm[id] * los->ds[ip]);

        /* Compute radiance... */
        rad[id] += los->src[ip][id] * los->eps[ip][id] * tau[id];

        /* Compute path transmittance... */
        tau[id] *= (1 - los->eps[ip][id]);
      }
  }

  /* Check whether LOS hit the ground... */
  if (ctl->sftype >= 1 && los->sft > 0) {

    /* Add surface emissions... */
    double src_sf[ND];
    formod_srcfunc(ctl, tbl, los->sft, src_sf);
    for (int id = 0; id < ctl->nd; id++)
      rad[id] += los->sfeps[id] * src_sf[id] * tau[id];

    /* Check reflectivity... */
    int refl = 0;
    if (ctl->sftype >= 2)
      for (int id = 0; id < ctl->nd; id++)
```

```
        if (los->sfeps[id] < 1) {
          refl = 1;
          break;
        }

    /* Calculate reflection... */
    if (refl) {

      /* Initialize... */
      for (int id = 0; id < ctl->nd; id++)
        tau_refl[id] = 1;

      /* Add down-welling radiance... */
      for (int ip = los->np - 1; ip >= 0; ip--)
        for (int id = 0; id < ctl->nd; id++) {
          rad[id] += los->src[ip][id] * los->eps[ip][id] * tau_refl[id]
            * tau[id] * (1 - los->sfeps[id]);
          tau_refl[id] *= (1 - los->eps[ip][id]);
        }

      /* Add solar term... */
      if (ctl->sftype >= 3) {

        /* Get solar zenith angle... */
        double sza2;
        if (ctl->sfsza < 0)
          sza2 =
            sza(obs->time[ir], los->lon[los->np - 1], los->lat[los->np - 1]);
        else
          sza2 = ctl->sfsza;

        /* Check solar zenith angle... */
        if (sza2 < 89.999) {

          /* Get angle of incidence... */
          geo2cart(los->z[los->np - 1], los->lon[los->np - 1],
                   los->lat[los->np - 1], x0);
          geo2cart(los->z[0], los->lon[0], los->lat[0], x1);
          for (int i = 0; i < 3; i++)
            x1[i] -= x0[i];
          const double cosa = DOTP(x0, x1) / NORM(x0) / NORM(x1);

          /* Get ratio of SZA and incident radiation... */
          const double rcos = cosa / cos(DEG2RAD(sza2));

          /* Add solar radiation... */
          for (int id = 0; id < ctl->nd; id++)
            rad[id] += 6.764e-5 / (2. * M_PI) * PLANCK(TSUN, ctl->nu[id])
              * tau_refl[id] * (1 - los->sfeps[id]) * tau[id] * rcos;
        }
      }
    }
  }

  /* Copy results... */
  for (int id = 0; id < ctl->nd; id++) {
    obs->rad[id][ir] = rad[id];
    obs->tau[id][ir] = tau[id];
  }

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

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

void formod_rfm(
  const ctl_t *ctl,
  const atm_t *atm,
  obs_t *obs) {

  los_t *los;

  FILE *out;

  char cmd[2 * LEN], filename[2 * LEN],
    rfmflg[LEN] = { "RAD TRA MIX LIN SFC" };

  double f[NSHAPE], nu[NSHAPE], nu0, nu1, obsz = -999, tsurf,
    xd[3], xo[3], xv[3], z[NR], zmin, zmax;

  int n, nadir = 0;

  /* Allocate... */
  ALLOC(los, los_t, 1);

  /* Check observer positions... */
```

```
  for (int ir = 1; ir < obs->nr; ir++)
    if (obs->obsz[ir] != obs->obsz[0]
        || obs->obslon[ir] != obs->obslon[0]
        || obs->obslat[ir] != obs->obslat[0])
      ERRMSG("RFM interface requires identical observer positions!");

  /* Check extinction data... */
  for (int iw = 0; iw < ctl->nw; iw++)
    for (int ip = 0; ip < atm->np; ip++)
      if (atm->k[iw][ip] != 0)
        ERRMSG("RFM interface cannot handle extinction data!");

  /* Get altitude range of atmospheric data... */
  gsl_stats_minmax(&zmin, &zmax, atm->z, 1, (size_t) atm->np);

  /* Observer within atmosphere? */
  if (obs->obsz[0] >= zmin && obs->obsz[0] <= zmax) {
    obsz = obs->obsz[0];
    strcat(rfmflg, " OBS");
  }

  /* Determine tangent altitude or air mass factor... */
  for (int ir = 0; ir < obs->nr; ir++) {

    /* Raytracing... */
    raytrace(ctl, atm, obs, los, ir);

    /* Nadir? */
    if (obs->tpz[ir] <= zmin) {
      geo2cart(obs->obsz[ir], obs->obslon[ir], obs->obslat[ir], xo);
      geo2cart(obs->vpz[ir], obs->vplon[ir], obs->vplat[ir], xv);
      for (int i = 0; i < 3; i++)
        xd[i] = xo[i] - xv[i];
      z[ir] = NORM(xo) * NORM(xd) / DOTP(xo, xd);
      nadir++;
    } else
      z[ir] = obs->tpz[ir];
  }
  if (nadir > 0 && nadir < obs->nr)
    ERRMSG("Limb and nadir not simultaneously possible!");

  /* Nadir? */
  if (nadir)
    strcat(rfmflg, " NAD");

  /* Get surface temperature... */
  tsurf = atm->t[gsl_stats_min_index(atm->z, 1, (size_t) atm->np)];

  /* Refraction? */
  if (!nadir && !ctl->refrac)
    strcat(rfmflg, " GEO");

  /* Continua? */
  if (ctl->ctm_co2 || ctl->ctm_h2o || ctl->ctm_n2 || ctl->ctm_o2)
    strcat(rfmflg, " CTM");

  /* Write atmospheric data file... */
  write_atm_rfm("rfm.atm", ctl, atm);

  /* Loop over channels... */
  for (int id = 0; id < ctl->nd; id++) {

    /* Read filter function... */
    sprintf(filename, "%s_%.4f.filt", ctl->tblbase, ctl->nu[id]);
    read_shape(filename, nu, f, &n);

    /* Set spectral range... */
    nu0 = nu[0];
    nu1 = nu[n - 1];

    /* Create RFM driver file... */
    if (!(out = fopen("rfm.drv", "w")))
      ERRMSG("Cannot create file!");
    fprintf(out, "*HDR\nRFM call by JURASSIC.\n");
    fprintf(out, "*FLG\n%s\n", rfmflg);
    fprintf(out, "*SPC\n%.4f %.4f 0.0005\n", nu0, nu1);
    fprintf(out, "*GAS\n");
    for (int ig = 0; ig < ctl->ng; ig++)
      fprintf(out, "%s\n", ctl->emitter[ig]);
    fprintf(out, "*ATM\nrfm.atm\n");
    fprintf(out, "*TAN\n");
    for (int ir = 0; ir < obs->nr; ir++)
      fprintf(out, "%g\n", z[ir]);
    fprintf(out, "*SFC\n%g 1.0\n", tsurf);
    if (obsz >= 0)
      fprintf(out, "*OBS\n%g\n", obsz);
    fprintf(out, "*HIT\n%s\n", ctl->rfmhit);
```

```
    fprintf(out, "*XSC\n");
    for (int ig = 0; ig < ctl->ng; ig++)
      if (ctl->rfmxsc[ig][0] != '-')
        fprintf(out, "%s\n", ctl->rfmxsc[ig]);
    fprintf(out, "*END\n");
    fclose(out);

    /* Remove temporary files... */
    if (system("rm -f rfm.runlog rad_*.asc tra_*.asc"))
      ERRMSG("Cannot remove temporary files!");

    /* Call RFM... */
    sprintf(cmd, "echo | %s", ctl->rfmbin);
    if (system(cmd))
      ERRMSG("Error while calling RFM!");

    /* Read data... */
    for (int ir = 0; ir < obs->nr; ir++) {
      obs->rad[id][ir] = read_obs_rfm("rad", z[ir], nu, f, n) * 1e-5;
      obs->tau[id][ir] = read_obs_rfm("tra", z[ir], nu, f, n);
    }
  }

  /* Remove temporary files... */
  if (system("rm -f rfm.drv rfm.atm rfm.runlog rad_*.asc tra_*.asc"))
    ERRMSG("Error while removing temporary files!");

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

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

void formod_srcfunc(
  const ctl_t *ctl,
  const tbl_t *tbl,
  const double t,
  double *src) {

  /* Determine index in temperature array... */
  const int it = locate_reg(tbl->st, TBLNS, t);

  /* Interpolate Planck function value... */
  for (int id = 0; id < ctl->nd; id++)
    src[id] = LIN(tbl->st[it], tbl->sr[it][id],
                  tbl->st[it + 1], tbl->sr[it + 1][id], t);
}

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

void geo2cart(
  const double z,
  const double lon,
  const double lat,
  double *x) {

  const double radius = z + RE;

  const double latrad = lat / 180. * M_PI;
  const double lonrad = lon / 180. * M_PI;

  const double coslat = cos(latrad);

  x[0] = radius * coslat * cos(lonrad);
  x[1] = radius * coslat * sin(lonrad);
  x[2] = radius * sin(latrad);
}

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

void hydrostatic(
  const ctl_t *ctl,
  atm_t *atm) {

  const double mmair = 28.96456e-3, mmh2o = 18.0153e-3;

  const int ipts = 20;

  static int ig_h2o = -999;

  double dzmin = 1e99, e = 0;

  int ipref = 0;

  /* Check reference height... */
  if (ctl->hydz < 0)
    return;
```

```
  /* Determine emitter index of H2O... */
  if (ig_h2o == -999)
    ig_h2o = find_emitter(ctl, "H2O");

  /* Find air parcel next to reference height... */
  for (int ip = 0; ip < atm->np; ip++)
    if (fabs(atm->z[ip] - ctl->hydz) < dzmin) {
      dzmin = fabs(atm->z[ip] - ctl->hydz);
      ipref = ip;
    }

  /* Upper part of profile... */
  for (int ip = ipref + 1; ip < atm->np; ip++) {
    double mean = 0;
    for (int i = 0; i < ipts; i++) {
      if (ig_h2o >= 0)
        e = LIN(0.0, atm->q[ig_h2o][ip - 1],
                ipts - 1.0, atm->q[ig_h2o][ip], (double) i);
      mean += (e * mmh2o + (1 - e) * mmair)
        * G0 / RI
        / LIN(0.0, atm->t[ip - 1], ipts - 1.0, atm->t[ip], (double) i) / ipts;
    }

    /* Compute p(z,T)... */
    atm->p[ip] =
      exp(log(atm->p[ip - 1]) - mean * 1000 * (atm->z[ip] - atm->z[ip - 1]));
  }

  /* Lower part of profile... */
  for (int ip = ipref - 1; ip >= 0; ip--) {
    double mean = 0;
    for (int i = 0; i < ipts; i++) {
      if (ig_h2o >= 0)
        e = LIN(0.0, atm->q[ig_h2o][ip + 1],
                ipts - 1.0, atm->q[ig_h2o][ip], (double) i);
      mean += (e * mmh2o + (1 - e) * mmair)
        * G0 / RI
        / LIN(0.0, atm->t[ip + 1], ipts - 1.0, atm->t[ip], (double) i) / ipts;
    }

    /* Compute p(z,T)... */
    atm->p[ip] =
      exp(log(atm->p[ip + 1]) - mean * 1000 * (atm->z[ip] - atm->z[ip + 1]));
  }
}

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

void idx2name(
  const ctl_t *ctl,
  const int idx,
  char *quantity) {

  if (idx == IDXP)
    sprintf(quantity, "PRESSURE");

  if (idx == IDXT)
    sprintf(quantity, "TEMPERATURE");

  for (int ig = 0; ig < ctl->ng; ig++)
    if (idx == IDXQ(ig))
      sprintf(quantity, "%s", ctl->emitter[ig]);

  for (int iw = 0; iw < ctl->nw; iw++)
    if (idx == IDXK(iw))
      sprintf(quantity, "EXTINCT_WINDOW_%d", iw);

  if (idx == IDXCLZ)
    sprintf(quantity, "CLOUD_HEIGHT");

  if (idx == IDXCLDZ)
    sprintf(quantity, "CLOUD_DEPTH");

  for (int icl = 0; icl < ctl->ncl; icl++)
    if (idx == IDXCLK(icl))
      sprintf(quantity, "CLOUD_EXTINCT_%.4f", ctl->clnu[icl]);

  if (idx == IDXSFZ)
    sprintf(quantity, "SURFACE_HEIGHT");

  if (idx == IDXSFP)
    sprintf(quantity, "SURFACE_PRESSURE");

  if (idx == IDXSFT)
    sprintf(quantity, "SURFACE_TEMPERATURE");

```

```
  for (int isf = 0; isf < ctl->nsf; isf++)
    if (idx == IDXSFEPS(isf))
      sprintf(quantity, "SURFACE_EMISSIVITY_%.4f", ctl->sfnu[isf]);
}

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

void init_srcfunc(
  const ctl_t *ctl,
  tbl_t *tbl) {

  char filename[2 * LEN];

  double f[NSHAPE], nu[NSHAPE];

  int n;

  /* Write info... */
  LOG(1, "Initialize source function table...");
  LOG(2, "Number of data points: %d", TBLNS);

  /* Loop over channels... */
  for (int id = 0; id < ctl->nd; id++) {

    /* Read filter function... */
    sprintf(filename, "%s_%.4f.filt", ctl->tblbase, ctl->nu[id]);
    read_shape(filename, nu, f, &n);

    /* Get minimum grid spacing... */
    double dnu = 1.0;
    for (int i = 1; i < n; i++)
      dnu = MIN(dnu, nu[i] - nu[i - 1]);

    /* Compute source function table... */
#pragma omp parallel for default(none) shared(ctl,tbl,id,nu,f,n,dnu)
    for (int it = 0; it < TBLNS; it++) {

      /* Set temperature... */
      tbl->st[it] = LIN(0.0, TMIN, TBLNS - 1.0, TMAX, (double) it);

      /* Integrate Planck function... */
      double fsum = tbl->sr[it][id] = 0;
      for (double fnu = nu[0]; fnu <= nu[n - 1]; fnu += dnu) {
        const int i = locate_irr(nu, n, fnu);
        const double ff = LIN(nu[i], f[i], nu[i + 1], f[i + 1], fnu);
        fsum += ff;
        tbl->sr[it][id] += ff * PLANCK(tbl->st[it], fnu);
      }
      tbl->sr[it][id] /= fsum;
    }

    /* Write info... */
    LOG(2,
        "channel= %.4f cm^-1 | T= %g ... %g K | B= %g ... %g W/(m^2 sr cm^-1)",
        ctl->nu[id], tbl->st[0], tbl->st[TBLNS - 1], tbl->sr[0][id],
        tbl->sr[TBLNS - 1][id]);
  }
}

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

void intpol_atm(
  const ctl_t *ctl,
  const atm_t *atm,
  const double z,
  double *p,
  double *t,
  double *q,
  double *k) {

  /* Get array index... */
  const int ip = locate_irr(atm->z, atm->np, z);

  /* Interpolate... */
  *p = LOGY(atm->z[ip], atm->p[ip], atm->z[ip + 1], atm->p[ip + 1], z);
  *t = LIN(atm->z[ip], atm->t[ip], atm->z[ip + 1], atm->t[ip + 1], z);
  for (int ig = 0; ig < ctl->ng; ig++)
    q[ig] =
      LIN(atm->z[ip], atm->q[ig][ip], atm->z[ip + 1], atm->q[ig][ip + 1], z);
  for (int iw = 0; iw < ctl->nw; iw++)
    k[iw] =
      LIN(atm->z[ip], atm->k[iw][ip], atm->z[ip + 1], atm->k[iw][ip + 1], z);
}

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

void intpol_tbl_cga(
```

```
  const ctl_t *ctl,
  const tbl_t *tbl,
  const los_t *los,
  const int ip,
  double tau_path[ND][NG],
  double tau_seg[ND]) {

  double eps;

  /* Loop over channels... */
  for (int id = 0; id < ctl->nd; id++) {

    /* Initialize... */
    tau_seg[id] = 1;

    /* Loop over emitters.... */
    for (int ig = 0; ig < ctl->ng; ig++) {

      /* Check size of table (pressure)... */
      if (tbl->np[id][ig] < 30)
        eps = 0;

      /* Check transmittance... */
      else if (tau_path[id][ig] < 1e-9)
        eps = 1;

      /* Interpolate... */
      else {

        /* Determine pressure and temperature indices... */
        const int ipr =
          locate_irr(tbl->p[id][ig], tbl->np[id][ig], los->cgp[ip][ig]);
        const int it0 = locate_reg(tbl->t[id][ig][ipr], tbl->nt[id][ig][ipr],
                                   los->cgt[ip][ig]);
        const int it1 =
          locate_reg(tbl->t[id][ig][ipr + 1], tbl->nt[id][ig][ipr + 1],
                     los->cgt[ip][ig]);

        /* Check size of table (temperature and column density)... */
        if (tbl->nt[id][ig][ipr] < 2 || tbl->nt[id][ig][ipr + 1] < 2
            || tbl->nu[id][ig][ipr][it0] < 2
            || tbl->nu[id][ig][ipr][it0 + 1] < 2
            || tbl->nu[id][ig][ipr + 1][it1] < 2
            || tbl->nu[id][ig][ipr + 1][it1 + 1] < 2)
          eps = 0;

        else {

          /* Get emissivities of extended path... */
          double eps00
            = intpol_tbl_eps(tbl, ig, id, ipr, it0, los->cgu[ip][ig]);
          double eps01 =
            intpol_tbl_eps(tbl, ig, id, ipr, it0 + 1, los->cgu[ip][ig]);
          double eps10 =
            intpol_tbl_eps(tbl, ig, id, ipr + 1, it1, los->cgu[ip][ig]);
          double eps11 =
            intpol_tbl_eps(tbl, ig, id, ipr + 1, it1 + 1, los->cgu[ip][ig]);

          /* Interpolate with respect to temperature... */
          eps00 = LIN(tbl->t[id][ig][ipr][it0], eps00,
                      tbl->t[id][ig][ipr][it0 + 1], eps01, los->cgt[ip][ig]);
          eps11 = LIN(tbl->t[id][ig][ipr + 1][it1], eps10,
                      tbl->t[id][ig][ipr + 1][it1 + 1],
                      eps11, los->cgt[ip][ig]);

          /* Interpolate with respect to pressure... */
          eps00 = LOGX(tbl->p[id][ig][ipr], eps00,
                       tbl->p[id][ig][ipr + 1], eps11, los->cgp[ip][ig]);

          /* Check emssivity range... */
          eps00 = MAX(MIN(eps00, 1), 0);

          /* Determine segment emissivity... */
          eps = 1 - (1 - eps00) / tau_path[id][ig];
        }
      }

      /* Get transmittance of extended path... */
      tau_path[id][ig] *= (1 - eps);

      /* Get segment transmittance... */
      tau_seg[id] *= (1 - eps);
    }
  }
}

/*****************************************************************************/
```

```
void intpol_tbl_ega(
  const ctl_t *ctl,
  const tbl_t *tbl,
  const los_t *los,
  const int ip,
  double tau_path[ND][NG],
  double tau_seg[ND]) {

  double eps, u;

  /* Loop over channels... */
  for (int id = 0; id < ctl->nd; id++) {

    /* Initialize... */
    tau_seg[id] = 1;

    /* Loop over emitters.... */
    for (int ig = 0; ig < ctl->ng; ig++) {

      /* Check size of table (pressure)... */
      if (tbl->np[id][ig] < 30)
        eps = 0;

      /* Check transmittance... */
      else if (tau_path[id][ig] < 1e-9)
        eps = 1;

      /* Interpolate... */
      else {

        /* Determine pressure and temperature indices... */
        const int ipr =
          locate_irr(tbl->p[id][ig], tbl->np[id][ig], los->p[ip]);
        const int it0 =
          locate_reg(tbl->t[id][ig][ipr], tbl->nt[id][ig][ipr], los->t[ip]);
        const int it1 =
          locate_reg(tbl->t[id][ig][ipr + 1], tbl->nt[id][ig][ipr + 1],
                     los->t[ip]);

        /* Check size of table (temperature and column density)... */
        if (tbl->nt[id][ig][ipr] < 2 || tbl->nt[id][ig][ipr + 1] < 2
            || tbl->nu[id][ig][ipr][it0] < 2
            || tbl->nu[id][ig][ipr][it0 + 1] < 2
            || tbl->nu[id][ig][ipr + 1][it1] < 2
            || tbl->nu[id][ig][ipr + 1][it1 + 1] < 2)
          eps = 0;

        else {

          /* Get emissivities of extended path... */
          u = intpol_tbl_u(tbl, ig, id, ipr, it0, 1 - tau_path[id][ig]);
          double eps00
            = intpol_tbl_eps(tbl, ig, id, ipr, it0, u + los->u[ip][ig]);

          u = intpol_tbl_u(tbl, ig, id, ipr, it0 + 1, 1 - tau_path[id][ig]);
          double eps01 =
            intpol_tbl_eps(tbl, ig, id, ipr, it0 + 1, u + los->u[ip][ig]);

          u = intpol_tbl_u(tbl, ig, id, ipr + 1, it1, 1 - tau_path[id][ig]);
          double eps10 =
            intpol_tbl_eps(tbl, ig, id, ipr + 1, it1, u + los->u[ip][ig]);

          u =
            intpol_tbl_u(tbl, ig, id, ipr + 1, it1 + 1, 1 - tau_path[id][ig]);
          double eps11 =
            intpol_tbl_eps(tbl, ig, id, ipr + 1, it1 + 1, u + los->u[ip][ig]);

          /* Interpolate with respect to temperature... */
          eps00 = LIN(tbl->t[id][ig][ipr][it0], eps00,
                      tbl->t[id][ig][ipr][it0 + 1], eps01, los->t[ip]);
          eps11 = LIN(tbl->t[id][ig][ipr + 1][it1], eps10,
                      tbl->t[id][ig][ipr + 1][it1 + 1], eps11, los->t[ip]);

          /* Interpolate with respect to pressure... */
          eps00 = LIN(tbl->p[id][ig][ipr], eps00,
                      tbl->p[id][ig][ipr + 1], eps11, los->p[ip]);

          /* Check emssivity range... */
          eps00 = MAX(MIN(eps00, 1), 0);

          /* Determine segment emissivity... */
          eps = 1 - (1 - eps00) / tau_path[id][ig];
        }
      }

      /* Get transmittance of extended path... */
```

```
      tau_path[id][ig] *= (1 - eps);

      /* Get segment transmittance... */
      tau_seg[id] *= (1 - eps);
    }
  }
}

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

double intpol_tbl_eps(
  const tbl_t *tbl,
  const int ig,
  const int id,
  const int ip,
  const int it,
  const double u) {

  /* Lower boundary... */
  if (u < tbl->u[id][ig][ip][it][0])
    return LIN(0, 0, tbl->u[id][ig][ip][it][0], tbl->eps[id][ig][ip][it][0],
               u);

  /* Upper boundary... */
  else if (u > tbl->u[id][ig][ip][it][tbl->nu[id][ig][ip][it] - 1]) {
    const double a =
      log(1 - tbl->eps[id][ig][ip][it][tbl->nu[id][ig][ip][it] - 1])
      / tbl->u[id][ig][ip][it][tbl->nu[id][ig][ip][it] - 1];
    return 1 - exp(a * u);
  }

  /* Interpolation... */
  else {

    /* Get index... */
    const int idx =
      locate_tbl(tbl->u[id][ig][ip][it], tbl->nu[id][ig][ip][it], u);

    /* Interpolate... */
    return
      LIN(tbl->u[id][ig][ip][it][idx], tbl->eps[id][ig][ip][it][idx],
          tbl->u[id][ig][ip][it][idx + 1], tbl->eps[id][ig][ip][it][idx + 1],
          u);
  }
}

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

double intpol_tbl_u(
  const tbl_t *tbl,
  const int ig,
  const int id,
  const int ip,
  const int it,
  const double eps) {

  /* Lower boundary... */
  if (eps < tbl->eps[id][ig][ip][it][0])
    return LIN(0, 0, tbl->eps[id][ig][ip][it][0], tbl->u[id][ig][ip][it][0],
               eps);

  /* Upper boundary... */
  else if (eps > tbl->eps[id][ig][ip][it][tbl->nu[id][ig][ip][it] - 1]) {
    const double a =
      log(1 - tbl->eps[id][ig][ip][it][tbl->nu[id][ig][ip][it] - 1])
      / tbl->u[id][ig][ip][it][tbl->nu[id][ig][ip][it] - 1];
    return log(1 - eps) / a;
  }

  /* Interpolation... */
  else {

    /* Get index... */
    const int idx =
      locate_tbl(tbl->eps[id][ig][ip][it], tbl->nu[id][ig][ip][it], eps);

    /* Interpolate... */
    return
      LIN(tbl->eps[id][ig][ip][it][idx], tbl->u[id][ig][ip][it][idx],
          tbl->eps[id][ig][ip][it][idx + 1], tbl->u[id][ig][ip][it][idx + 1],
          eps);
  }
}

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

void jsec2time(
```

```
  const double jsec,
  int *year,
  int *mon,
  int *day,
  int *hour,
  int *min,
  int *sec,
  double *remain) {

  struct tm t0, *t1;

  t0.tm_year = 100;
  t0.tm_mon = 0;
  t0.tm_mday = 1;
  t0.tm_hour = 0;
  t0.tm_min = 0;
  t0.tm_sec = 0;

  time_t jsec0 = (time_t) jsec + timegm(&t0);
  t1 = gmtime(&jsec0);

  *year = t1->tm_year + 1900;
  *mon = t1->tm_mon + 1;
  *day = t1->tm_mday;
  *hour = t1->tm_hour;
  *min = t1->tm_min;
  *sec = t1->tm_sec;
  *remain = jsec - floor(jsec);
}

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

void kernel(
  ctl_t *ctl,
  atm_t *atm,
  obs_t *obs,
  gsl_matrix *k) {

  atm_t *atm1;
  obs_t *obs1;

  int *iqa;

  /* Get sizes... */
  const size_t m = k->size1;
  const size_t n = k->size2;

  /* Allocate... */
  gsl_vector *x0 = gsl_vector_alloc(n);
  gsl_vector *yy0 = gsl_vector_alloc(m);
  ALLOC(iqa, int,
        N);

  /* Compute radiance for undisturbed atmospheric data... */
  formod(ctl, atm, obs);

  /* Compose vectors... */
  atm2x(ctl, atm, x0, iqa, NULL);
  obs2y(ctl, obs, yy0, NULL, NULL);

  /* Initialize kernel matrix... */
  gsl_matrix_set_zero(k);

  /* Loop over state vector elements... */
#pragma omp parallel for default(none) shared(ctl,atm,obs,k,x0,yy0,n,m,iqa) private(atm1, obs1)
  for (size_t j = 0; j < n; j++) {

    /* Allocate... */
    gsl_vector *x1 = gsl_vector_alloc(n);
    gsl_vector *yy1 = gsl_vector_alloc(m);
    ALLOC(atm1, atm_t, 1);
    ALLOC(obs1, obs_t, 1);

    /* Set perturbation size... */
    double h;
    if (iqa[j] == IDXP)
      h = MAX(fabs(0.01 * gsl_vector_get(x0, j)), 1e-7);
    else if (iqa[j] == IDXT)
      h = 1.0;
    else if (iqa[j] >= IDXQ(0) && iqa[j] < IDXQ(ctl->ng))
      h = MAX(fabs(0.01 * gsl_vector_get(x0, j)), 1e-15);
    else if (iqa[j] >= IDXK(0) && iqa[j] < IDXK(ctl->nw))
      h = 1e-4;
    else if (iqa[j] == IDXCLZ || iqa[j] == IDXCLDZ)
      h = 1.0;
    else if (iqa[j] >= IDXCLK(0) && iqa[j] < IDXCLK(ctl->ncl))
      h = 1e-4;
```

```
    else if (iqa[j] == IDXSFZ)
      h = 0.1;
    else if (iqa[j] == IDXSFP)
      h = 10.0;
    else if (iqa[j] == IDXSFT)
      h = 1.0;
    else if (iqa[j] >= IDXSFEPS(0) && iqa[j] < IDXSFEPS(ctl->nsf))
      h = 1e-2;
    else
      ERRMSG("Cannot set perturbation size!");

    /* Disturb state vector element... */
    gsl_vector_memcpy(x1, x0);
    gsl_vector_set(x1, j, gsl_vector_get(x1, j) + h);
    copy_atm(ctl, atm1, atm, 0);
    copy_obs(ctl, obs1, obs, 0);
    x2atm(ctl, x1, atm1);

    /* Compute radiance for disturbed atmospheric data... */
    formod(ctl, atm1, obs1);

    /* Compose measurement vector for disturbed radiance data... */
    obs2y(ctl, obs1, yy1, NULL, NULL);

    /* Compute derivatives... */
    for (size_t i = 0; i < m; i++)
      gsl_matrix_set(k, i, j,
                     (gsl_vector_get(yy1, i) - gsl_vector_get(yy0, i)) / h);

    /* Free... */
    gsl_vector_free(x1);
    gsl_vector_free(yy1);
    free(atm1);
    free(obs1);
  }

  /* Free... */
  gsl_vector_free(x0);
  gsl_vector_free(yy0);
  free(iqa);
}

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

int locate_irr(
  const double *xx,
  const int n,
  const double x) {

  int ilo = 0;
  int ihi = n - 1;
  int i = (ihi + ilo) >> 1;

  if (xx[i] < xx[i + 1])
    while (ihi > ilo + 1) {
      i = (ihi + ilo) >> 1;
      if (xx[i] > x)
        ihi = i;
      else
        ilo = i;
  } else
    while (ihi > ilo + 1) {
      i = (ihi + ilo) >> 1;
      if (xx[i] <= x)
        ihi = i;
      else
        ilo = i;
    }

  return ilo;
}

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

int locate_reg(
  const double *xx,
  const int n,
  const double x) {

  /* Calculate index... */
  const int i = (int) ((x - xx[0]) / (xx[1] - xx[0]));

  /* Check range... */
  if (i < 0)
    return 0;
  else if (i > n - 2)
    return n - 2;
```

```
  else
    return i;
}

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

int locate_tbl(
  const float *xx,
  const int n,
  const double x) {

  int ilo = 0;
  int ihi = n - 1;
  int i = (ihi + ilo) >> 1;

  while (ihi > ilo + 1) {
    i = (ihi + ilo) >> 1;
    if (xx[i] > x)
      ihi = i;
    else
      ilo = i;
  }

  return ilo;
}

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

size_t obs2y(
  const ctl_t *ctl,
  const obs_t *obs,
  gsl_vector *y,
  int *ida,
  int *ira) {

  size_t m = 0;

  /* Determine measurement vector... */
  for (int ir = 0; ir < obs->nr; ir++)
    for (int id = 0; id < ctl->nd; id++)
      if (isfinite(obs->rad[id][ir])) {
        if (y != NULL)
          gsl_vector_set(y, m, obs->rad[id][ir]);
        if (ida != NULL)
          ida[m] = id;
        if (ira != NULL)
          ira[m] = ir;
        m++;
      }

  return m;
}

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

void raytrace(
  const ctl_t *ctl,
  const atm_t *atm,
  obs_t *obs,
  los_t *los,
  const int ir) {

  const double h = 0.02, zrefrac = 60;

  double ex0[3], ex1[3], k[NW], lat, lon, n, ng[3], norm, p, q[NG], t,
    x[3], xh[3], xobs[3], xvp[3], z = 1e99, zmax, zmin;

  int stop = 0;

  /* Initialize... */
  los->np = 0;
  los->sft = -999;
  obs->tpz[ir] = obs->vpz[ir];
  obs->tplon[ir] = obs->vplon[ir];
  obs->tplat[ir] = obs->vplat[ir];

  /* Get altitude range of atmospheric data... */
  gsl_stats_minmax(&zmin, &zmax, atm->z, 1, (size_t) atm->np);
  if (ctl->nsf > 0) {
    zmin = MAX(atm->sfz, zmin);
    if (atm->sfp > 0) {
      const int ip = locate_irr(atm->p, atm->np, atm->sfp);
      const double zip =
        LIN(log(atm->p[ip]), atm->z[ip], log(atm->p[ip + 1]), atm->z[ip + 1],
            log(atm->sfp));
      zmin = MAX(zip, zmin);
    }
```

```
	}

	/* Check observer altitude... */
	if (obs->obsz[ir] < zmin)
	  ERRMSG("Observer below surface!");

	/* Check view point altitude... */
	if (obs->vpz[ir] > zmax)
	  return;

	/* Determine Cartesian coordinates for observer and view point... */
	geo2cart(obs->obsz[ir], obs->obslon[ir], obs->obslat[ir], xobs);
	geo2cart(obs->vpz[ir], obs->vplon[ir], obs->vplat[ir], xvp);

	/* Determine initial tangent vector... */
	for (int i = 0; i < 3; i++)
	  ex0[i] = xvp[i] - xobs[i];
	norm = NORM(ex0);
	for (int i = 0; i < 3; i++)
	  ex0[i] /= norm;

	/* Observer within atmosphere... */
	for (int i = 0; i < 3; i++)
	  x[i] = xobs[i];

	/* Observer above atmosphere (search entry point)... */
	if (obs->obsz[ir] > zmax) {
	  double dmax = norm, dmin = 0;
	  while (fabs(dmin - dmax) > 0.001) {
	    const double d = (dmax + dmin) / 2;
	    for (int i = 0; i < 3; i++)
	      x[i] = xobs[i] + d * ex0[i];
	    cart2geo(x, &z, &lon, &lat);
	    if (z <= zmax && z > zmax - 0.001)
	      break;
	    if (z < zmax - 0.0005)
	      dmax = d;
	    else
	      dmin = d;
	  }
	}

	/* Ray-tracing... */
	while (1) {

	  /* Set step length... */
	  double ds = ctl->rayds;
	  if (ctl->raydz > 0) {
	    norm = NORM(x);
	    for (int i = 0; i < 3; i++)
	      xh[i] = x[i] / norm;
	    const double cosa = fabs(DOTP(ex0, xh));
	    if (cosa != 0)
	      ds = MIN(ctl->rayds, ctl->raydz / cosa);
	  }

	  /* Determine geolocation... */
	  cart2geo(x, &z, &lon, &lat);

	  /* Check if LOS hits the ground or has left atmosphere... */
	  if (z < zmin || z > zmax) {
	    stop = (z < zmin ? 2 : 1);
	    const double frac =
	      ((z <
	        zmin ? zmin : zmax) - los->z[los->np - 1]) / (z - los->z[los->np -
	                                                                1]);
	    geo2cart(los->z[los->np - 1], los->lon[los->np - 1],
	             los->lat[los->np - 1], xh);
	    for (int i = 0; i < 3; i++)
	      x[i] = xh[i] + frac * (x[i] - xh[i]);
	    cart2geo(x, &z, &lon, &lat);
	    los->ds[los->np - 1] = ds * frac;
	    ds = 0;
	  }

	  /* Interpolate atmospheric data... */
	  intpol_atm(ctl, atm, z, &p, &t, q, k);

	  /* Save data... */
	  los->lon[los->np] = lon;
	  los->lat[los->np] = lat;
	  los->z[los->np] = z;
	  los->p[los->np] = p;
	  los->t[los->np] = t;
	  for (int ig = 0; ig < ctl->ng; ig++)
	    los->q[los->np][ig] = q[ig];
	  for (int id = 0; id < ctl->nd; id++)
```

```
      los->k[los->np][id] = k[ctl->window[id]];
    los->ds[los->np] = ds;

    /* Add cloud extinction... */
    if (ctl->ncl > 0 && atm->cldz > 0) {
      const double aux = exp(-0.5 * POW2((z - atm->clz) / atm->cldz));
      for (int id = 0; id < ctl->nd; id++) {
        const int icl = locate_irr(ctl->clnu, ctl->ncl, ctl->nu[id]);
        los->k[los->np][id]
          += aux * LIN(ctl->clnu[icl], atm->clk[icl],
                      ctl->clnu[icl + 1], atm->clk[icl + 1], ctl->nu[id]);
      }
    }

    /* Increment and check number of LOS points... */
    if ((++los->np) > NLOS)
      ERRMSG("Too many LOS points!");

    /* Check stop flag... */
    if (stop) {

      /* Set surface temperature... */
      if (ctl->nsf > 0 && atm->sft > 0)
        t = atm->sft;
      los->sft = (stop == 2 ? t : -999);

      /* Set surface emissivity... */
      for (int id = 0; id < ctl->nd; id++) {
        los->sfeps[id] = 1.0;
        if (ctl->nsf > 0) {
          const int isf = locate_irr(ctl->sfnu, ctl->nsf, ctl->nu[id]);
          los->sfeps[id] = LIN(ctl->sfnu[isf], atm->sfeps[isf],
                              ctl->sfnu[isf + 1], atm->sfeps[isf + 1],
                              ctl->nu[id]);
        }
      }

      /* Leave raytracer... */
      break;
    }

    /* Determine refractivity... */
    if (ctl->refrac && z <= zrefrac)
      n = 1 + REFRAC(p, t);
    else
      n = 1;

    /* Construct new tangent vector (first term)... */
    for (int i = 0; i < 3; i++)
      ex1[i] = ex0[i] * n;

    /* Compute gradient of refractivity... */
    if (ctl->refrac && z <= zrefrac) {
      for (int i = 0; i < 3; i++)
        xh[i] = x[i] + 0.5 * ds * ex0[i];
      cart2geo(xh, &z, &lon, &lat);
      intpol_atm(ctl, atm, z, &p, &t, q, k);
      n = REFRAC(p, t);
      for (int i = 0; i < 3; i++) {
        xh[i] += h;
        cart2geo(xh, &z, &lon, &lat);
        intpol_atm(ctl, atm, z, &p, &t, q, k);
        ng[i] = (REFRAC(p, t) - n) / h;
        xh[i] -= h;
      }
    } else
      for (int i = 0; i < 3; i++)
        ng[i] = 0;

    /* Construct new tangent vector (second term)... */
    for (int i = 0; i < 3; i++)
      ex1[i] += ds * ng[i];

    /* Normalize new tangent vector... */
    norm = NORM(ex1);
    for (int i = 0; i < 3; i++)
      ex1[i] /= norm;

    /* Determine next point of LOS... */
    for (int i = 0; i < 3; i++)
      x[i] += 0.5 * ds * (ex0[i] + ex1[i]);

    /* Copy tangent vector... */
    for (int i = 0; i < 3; i++)
      ex0[i] = ex1[i];
  }

```

```
  /* Get tangent point (to be done before changing segment lengths!)... */
  tangent_point(los, &obs->tpz[ir], &obs->tplon[ir], &obs->tplat[ir]);

  /* Change segment lengths according to trapezoid rule... */
  for (int ip = los->np - 1; ip >= 1; ip--)
    los->ds[ip] = 0.5 * (los->ds[ip - 1] + los->ds[ip]);
  los->ds[0] *= 0.5;

  /* Compute column density... */
  for (int ip = 0; ip < los->np; ip++)
    for (int ig = 0; ig < ctl->ng; ig++)
      los->u[ip][ig] = 10 * los->q[ip][ig] * los->p[ip]
        / (KB * los->t[ip]) * los->ds[ip];

  /* Compute Curtis-Godson means... */
  for (int ig = 0; ig < ctl->ng; ig++) {
    los->cgu[0][ig] = los->u[0][ig];
    los->cgp[0][ig] = los->u[0][ig] * los->p[0];
    los->cgt[0][ig] = los->u[0][ig] * los->t[0];
  }
  for (int ip = 1; ip < los->np; ip++)
    for (int ig = 0; ig < ctl->ng; ig++) {
      los->cgu[ip][ig] = los->cgu[ip - 1][ig] + los->u[ip][ig];
      los->cgp[ip][ig] = los->cgp[ip - 1][ig] + los->u[ip][ig] * los->p[ip];
      los->cgt[ip][ig] = los->cgt[ip - 1][ig] + los->u[ip][ig] * los->t[ip];
    }
  for (int ip = 0; ip < los->np; ip++)
    for (int ig = 0; ig < ctl->ng; ig++) {
      los->cgp[ip][ig] /= los->cgu[ip][ig];
      los->cgt[ip][ig] /= los->cgu[ip][ig];
    }
}

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

void read_atm(
  const char *dirname,
  const char *filename,
  const ctl_t *ctl,
  atm_t *atm) {

  FILE *in;

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

  /* Init... */
  atm->np = 0;

  /* Set filename... */
  if (dirname != NULL)
    sprintf(file, "%s/%s", dirname, filename);
  else
    sprintf(file, "%s", filename);

  /* Write info... */
  LOG(1, "Read atmospheric data: %s", file);

  /* Open file... */
  if (!(in = fopen(file, "r")))
    ERRMSG("Cannot open file!");

  /* Read line... */
  while (fgets(line, LEN, in)) {

    /* Read data... */
    TOK(line, tok, "%lg", atm->time[atm->np]);
    TOK(NULL, tok, "%lg", atm->z[atm->np]);
    TOK(NULL, tok, "%lg", atm->lon[atm->np]);
    TOK(NULL, tok, "%lg", atm->lat[atm->np]);
    TOK(NULL, tok, "%lg", atm->p[atm->np]);
    TOK(NULL, tok, "%lg", atm->t[atm->np]);
    for (int ig = 0; ig < ctl->ng; ig++)
      TOK(NULL, tok, "%lg", atm->q[ig][atm->np]);
    for (int iw = 0; iw < ctl->nw; iw++)
      TOK(NULL, tok, "%lg", atm->k[iw][atm->np]);
    if (ctl->ncl > 0 && atm->np == 0) {
      TOK(NULL, tok, "%lg", atm->clz);
      TOK(NULL, tok, "%lg", atm->cldz);
      for (int icl = 0; icl < ctl->ncl; icl++)
        TOK(NULL, tok, "%lg", atm->clk[icl]);
    }
    if (ctl->nsf > 0 && atm->np == 0) {
      TOK(NULL, tok, "%lg", atm->sfz);
      TOK(NULL, tok, "%lg", atm->sfp);
      TOK(NULL, tok, "%lg", atm->sft);
      for (int isf = 0; isf < ctl->nsf; isf++)
        TOK(NULL, tok, "%lg", atm->sfeps[isf]);
```

```
    }

    /* Increment data point counter... */
    if ((++atm->np) > NP)
      ERRMSG("Too many data points!");
  }

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

  /* Check number of points... */
  if (atm->np < 1)
    ERRMSG("Could not read any data!");

  /* Write info... */
  double mini, maxi;
  LOG(2, "Number of data points: %d", atm->np);
  gsl_stats_minmax(&mini, &maxi, atm->time, 1, (size_t) atm->np);
  LOG(2, "Time range: %.2f ... %.2f s", mini, maxi);
  gsl_stats_minmax(&mini, &maxi, atm->z, 1, (size_t) atm->np);
  LOG(2, "Altitude range: %g ... %g km", mini, maxi);
  gsl_stats_minmax(&mini, &maxi, atm->lon, 1, (size_t) atm->np);
  LOG(2, "Longitude range: %g ... %g deg", mini, maxi);
  gsl_stats_minmax(&mini, &maxi, atm->lat, 1, (size_t) atm->np);
  LOG(2, "Latitude range: %g ... %g deg", mini, maxi);
  gsl_stats_minmax(&mini, &maxi, atm->p, 1, (size_t) atm->np);
  LOG(2, "Pressure range: %g ... %g hPa", maxi, mini);
  gsl_stats_minmax(&mini, &maxi, atm->t, 1, (size_t) atm->np);
  LOG(2, "Temperature range: %g ... %g K", mini, maxi);
  for (int ig = 0; ig < ctl->ng; ig++) {
    gsl_stats_minmax(&mini, &maxi, atm->q[ig], 1, (size_t) atm->np);
    LOG(2, "Emitter %s range: %g ... %g ppv", ctl->emitter[ig], mini, maxi);
  }
  for (int iw = 0; iw < ctl->nw; iw++) {
    gsl_stats_minmax(&mini, &maxi, atm->k[iw], 1, (size_t) atm->np);
    LOG(2, "Extinction range (window %d): %g ... %g km^-1", iw, mini, maxi);
  }
  if (ctl->ncl > 0 && atm->np == 0) {
    LOG(2, "Cloud layer: z= %g km | dz= %g km | k= %g ... %g km^-1",
        atm->clz, atm->cldz, atm->clk[0], atm->clk[ctl->ncl - 1]);
  } else
    LOG(2, "Cloud layer: none");
  if (ctl->nsf > 0 && atm->np == 0) {
    LOG(2,
        "Surface layer: z_s= %g km | p_s= %g hPa | T_s = %g K | eps= %g ... %g",
        atm->sfz, atm->sfp, atm->sft, atm->sfeps[0],
        atm->sfeps[ctl->nsf - 1]);
  } else
    LOG(2, "Surface layer: none");
}

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

void read_ctl(
  int argc,
  char *argv[],
  ctl_t *ctl) {

  /* Write info... */
  LOG(1, "\nJuelich Rapid Spectral Simulation Code (JURASSIC)\n"
      "(executable: %s | version: %s | compiled: %s, %s)\n",
      argv[0], VERSION, __DATE__, __TIME__);

  /* Emitters... */
  ctl->ng = (int) scan_ctl(argc, argv, "NG", -1, "0", NULL);
  if (ctl->ng < 0 || ctl->ng > NG)
    ERRMSG("Set 0 <= NG <= MAX!");
  for (int ig = 0; ig < ctl->ng; ig++)
    scan_ctl(argc, argv, "EMITTER", ig, "", ctl->emitter[ig]);

  /* Radiance channels... */
  ctl->nd = (int) scan_ctl(argc, argv, "ND", -1, "0", NULL);
  if (ctl->nd < 0 || ctl->nd > ND)
    ERRMSG("Set 0 <= ND <= MAX!");
  for (int id = 0; id < ctl->nd; id++)
    ctl->nu[id] = scan_ctl(argc, argv, "NU", id, "", NULL);

  /* Spectral windows... */
  ctl->nw = (int) scan_ctl(argc, argv, "NW", -1, "1", NULL);
  if (ctl->nw < 0 || ctl->nw > NW)
    ERRMSG("Set 0 <= NW <= MAX!");
  for (int id = 0; id < ctl->nd; id++)
    ctl->window[id] = (int) scan_ctl(argc, argv, "WINDOW", id, "0", NULL);

  /* Cloud data... */
  ctl->ncl = (int) scan_ctl(argc, argv, "NCL", -1, "0", NULL);
  if (ctl->ncl < 0 || ctl->ncl > NCL)
```

```
    ERRMSG("Set 0 <= NCL <= MAX!");
  if (ctl->ncl == 1)
    ERRMSG("Set NCL > 1!");
  for (int icl = 0; icl < ctl->ncl; icl++)
    ctl->clnu[icl] = scan_ctl(argc, argv, "CLNU", icl, "", NULL);

  /* Surface data... */
  ctl->nsf = (int) scan_ctl(argc, argv, "NSF", -1, "0", NULL);
  if (ctl->nsf < 0 || ctl->nsf > NSF)
    ERRMSG("Set 0 <= NSF <= MAX!");
  if (ctl->nsf == 1)
    ERRMSG("Set NSF > 1!");
  for (int isf = 0; isf < ctl->nsf; isf++)
    ctl->sfnu[isf] = scan_ctl(argc, argv, "SFNU", isf, "", NULL);
  ctl->sftype = (int) scan_ctl(argc, argv, "SFTYPE", -1, "2", NULL);
  if (ctl->sftype < 0 || ctl->sftype > 3)
    ERRMSG("Set 0 <= SFTYPE <= 3!");
  ctl->sfsza = scan_ctl(argc, argv, "SFSZA", -1, "-999", NULL);

  /* Emissivity look-up tables... */
  scan_ctl(argc, argv, "TBLBASE", -1, "-", ctl->tblbase);
  ctl->tblfmt = (int) scan_ctl(argc, argv, "TBLFMT", -1, "1", NULL);

  /* Hydrostatic equilibrium... */
  ctl->hydz = scan_ctl(argc, argv, "HYDZ", -1, "-999", NULL);

  /* Continua... */
  ctl->ctm_co2 = (int) scan_ctl(argc, argv, "CTM_CO2", -1, "1", NULL);
  ctl->ctm_h2o = (int) scan_ctl(argc, argv, "CTM_H2O", -1, "1", NULL);
  ctl->ctm_n2 = (int) scan_ctl(argc, argv, "CTM_N2", -1, "1", NULL);
  ctl->ctm_o2 = (int) scan_ctl(argc, argv, "CTM_O2", -1, "1", NULL);

  /* Ray-tracing... */
  ctl->refrac = (int) scan_ctl(argc, argv, "REFRAC", -1, "1", NULL);
  ctl->rayds = scan_ctl(argc, argv, "RAYDS", -1, "10", NULL);
  ctl->raydz = scan_ctl(argc, argv, "RAYDZ", -1, "0.1", NULL);

  /* Field of view... */
  scan_ctl(argc, argv, "FOV", -1, "-", ctl->fov);

  /* Retrieval interface... */
  ctl->retp_zmin = scan_ctl(argc, argv, "RETP_ZMIN", -1, "-999", NULL);
  ctl->retp_zmax = scan_ctl(argc, argv, "RETP_ZMAX", -1, "-999", NULL);
  ctl->rett_zmin = scan_ctl(argc, argv, "RETT_ZMIN", -1, "-999", NULL);
  ctl->rett_zmax = scan_ctl(argc, argv, "RETT_ZMAX", -1, "-999", NULL);
  for (int ig = 0; ig < ctl->ng; ig++) {
    ctl->retq_zmin[ig] = scan_ctl(argc, argv, "RETQ_ZMIN", ig, "-999", NULL);
    ctl->retq_zmax[ig] = scan_ctl(argc, argv, "RETQ_ZMAX", ig, "-999", NULL);
  }
  for (int iw = 0; iw < ctl->nw; iw++) {
    ctl->retk_zmin[iw] = scan_ctl(argc, argv, "RETK_ZMIN", iw, "-999", NULL);
    ctl->retk_zmax[iw] = scan_ctl(argc, argv, "RETK_ZMAX", iw, "-999", NULL);
  }
  ctl->ret_clz = (int) scan_ctl(argc, argv, "RET_CLZ", -1, "0", NULL);
  ctl->ret_cldz = (int) scan_ctl(argc, argv, "RET_CLDZ", -1, "0", NULL);
  ctl->ret_clk = (int) scan_ctl(argc, argv, "RET_CLK", -1, "0", NULL);
  ctl->ret_sfz = (int) scan_ctl(argc, argv, "RET_SFZ", -1, "0", NULL);
  ctl->ret_sfp = (int) scan_ctl(argc, argv, "RET_SFP", -1, "0", NULL);
  ctl->ret_sft = (int) scan_ctl(argc, argv, "RET_SFT", -1, "0", NULL);
  ctl->ret_sfeps = (int) scan_ctl(argc, argv, "RET_SFEPS", -1, "0", NULL);

  /* Output flags... */
  ctl->write_bbt = (int) scan_ctl(argc, argv, "WRITE_BBT", -1, "0", NULL);
  ctl->write_matrix =
    (int) scan_ctl(argc, argv, "WRITE_MATRIX", -1, "0", NULL);

  /* External forward models... */
  ctl->formod = (int) scan_ctl(argc, argv, "FORMOD", -1, "1", NULL);
  scan_ctl(argc, argv, "RFMBIN", -1, "-", ctl->rfmbin);
  scan_ctl(argc, argv, "RFMHIT", -1, "-", ctl->rfmhit);
  for (int ig = 0; ig < ctl->ng; ig++)
    scan_ctl(argc, argv, "RFMXSC", ig, "-", ctl->rfmxsc[ig]);
}

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

void read_matrix(
  const char *dirname,
  const char *filename,
  gsl_matrix *matrix) {

  FILE *in;

  char dum[LEN], file[LEN], line[LEN];

  double value;

```

```
  int i, j;

  /* Set filename... */
  if (dirname != NULL)
    sprintf(file, "%s/%s", dirname, filename);
  else
    sprintf(file, "%s", filename);

  /* Write info... */
  LOG(1, "Read matrix: %s", file);

  /* Open file... */
  if (!(in = fopen(file, "r")))
    ERRMSG("Cannot open file!");

  /* Read data... */
  gsl_matrix_set_zero(matrix);
  while (fgets(line, LEN, in))
    if (sscanf(line, "%d %s %s %s %s %s %d %s %s %s %s %s %lg",
               &i, dum, dum, dum, dum, dum,
               &j, dum, dum, dum, dum, dum, &value) == 13)
      gsl_matrix_set(matrix, (size_t) i, (size_t) j, value);

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

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

void read_obs(
  const char *dirname,
  const char *filename,
  const ctl_t *ctl,
  obs_t *obs) {

  FILE *in;

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

  /* Init... */
  obs->nr = 0;

  /* Set filename... */
  if (dirname != NULL)
    sprintf(file, "%s/%s", dirname, filename);
  else
    sprintf(file, "%s", filename);

  /* Write info... */
  LOG(1, "Read observation data: %s", file);

  /* Open file... */
  if (!(in = fopen(file, "r")))
    ERRMSG("Cannot open file!");

  /* Read line... */
  while (fgets(line, LEN, in)) {

    /* Read data... */
    TOK(line, tok, "%lg", obs->time[obs->nr]);
    TOK(NULL, tok, "%lg", obs->obsz[obs->nr]);
    TOK(NULL, tok, "%lg", obs->obslon[obs->nr]);
    TOK(NULL, tok, "%lg", obs->obslat[obs->nr]);
    TOK(NULL, tok, "%lg", obs->vpz[obs->nr]);
    TOK(NULL, tok, "%lg", obs->vplon[obs->nr]);
    TOK(NULL, tok, "%lg", obs->vplat[obs->nr]);
    TOK(NULL, tok, "%lg", obs->tpz[obs->nr]);
    TOK(NULL, tok, "%lg", obs->tplon[obs->nr]);
    TOK(NULL, tok, "%lg", obs->tplat[obs->nr]);
    for (int id = 0; id < ctl->nd; id++)
      TOK(NULL, tok, "%lg", obs->rad[id][obs->nr]);
    for (int id = 0; id < ctl->nd; id++)
      TOK(NULL, tok, "%lg", obs->tau[id][obs->nr]);

    /* Increment counter... */
    if ((++obs->nr) > NR)
      ERRMSG("Too many rays!");
  }

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

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

  /* Write info... */
```

```
  double mini, maxi;
  LOG(2, "Number of ray paths: %d", obs->nr);
  gsl_stats_minmax(&mini, &maxi, obs->time, 1, (size_t) obs->nr);
  LOG(2, "Time range: %.2f ... %.2f s", mini, maxi);
  gsl_stats_minmax(&mini, &maxi, obs->obsz, 1, (size_t) obs->nr);
  LOG(2, "Observer altitude range: %g ... %g km", mini, maxi);
  gsl_stats_minmax(&mini, &maxi, obs->obslon, 1, (size_t) obs->nr);
  LOG(2, "Observer longitude range: %g ... %g deg", mini, maxi);
  gsl_stats_minmax(&mini, &maxi, obs->obslat, 1, (size_t) obs->nr);
  LOG(2, "Observer latitude range: %g ... %g deg", mini, maxi);
  gsl_stats_minmax(&mini, &maxi, obs->vpz, 1, (size_t) obs->nr);
  LOG(2, "View point altitude range: %g ... %g km", mini, maxi);
  gsl_stats_minmax(&mini, &maxi, obs->vplon, 1, (size_t) obs->nr);
  LOG(2, "View point longitude range: %g ... %g deg", mini, maxi);
  gsl_stats_minmax(&mini, &maxi, obs->vplat, 1, (size_t) obs->nr);
  LOG(2, "View point latitude range: %g ... %g deg", mini, maxi);
  gsl_stats_minmax(&mini, &maxi, obs->tpz, 1, (size_t) obs->nr);
  LOG(2, "Tangent point altitude range: %g ... %g km", mini, maxi);
  gsl_stats_minmax(&mini, &maxi, obs->tplon, 1, (size_t) obs->nr);
  LOG(2, "Tangent point longitude range: %g ... %g deg", mini, maxi);
  gsl_stats_minmax(&mini, &maxi, obs->tplat, 1, (size_t) obs->nr);
  LOG(2, "Tangent point latitude range: %g ... %g deg", mini, maxi);
  for (int id = 0; id < ctl->nd; id++) {
    gsl_stats_minmax(&mini, &maxi, obs->rad[id], 1, (size_t) obs->nr);
    if (ctl->write_bbt) {
      LOG(2, "Brightness temperature (%.4f cm^-1) range: %g ... %g K",
          ctl->nu[id], mini, maxi);
    } else {
      LOG(2, "Radiance (%.4f cm^-1) range: %g ... %g W/(m^2 sr cm^-1)",
          ctl->nu[id], mini, maxi);
    }
  }
  for (int id = 0; id < ctl->nd; id++) {
    gsl_stats_minmax(&mini, &maxi, obs->tau[id], 1, (size_t) obs->nr);
    if (ctl->write_bbt) {
      LOG(2, "Transmittance (%.4f cm^-1) range: %g ... %g",
          ctl->nu[id], mini, maxi);
    }
  }
}

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

double read_obs_rfm(
  const char *basename,
  const double z,
  double *nu,
  double *f,
  int n) {

  FILE *in;

  char filename[LEN];

  double filt, fsum = 0, nu2[NSHAPE], *nurfm, *rad, radsum = 0;

  int npts;

  /* Allocate... */
  ALLOC(nurfm, double,
        RFMNPTS);
  ALLOC(rad, double,
        RFMNPTS);

  /* Search RFM spectrum... */
  sprintf(filename, "%s_%05d.asc", basename, (int) (z * 1000));
  if (!(in = fopen(filename, "r"))) {
    sprintf(filename, "%s_%05d.asc", basename, (int) (z * 1000) + 1);
    if (!(in = fopen(filename, "r")))
      ERRMSG("Cannot find RFM data file!");
  }
  fclose(in);

  /* Read RFM spectrum... */
  read_rfm_spec(filename, nurfm, rad, &npts);

  /* Set wavenumbers... */
  nu2[0] = nu[0];
  nu2[n - 1] = nu[n - 1];
  for (int i = 1; i < n - 1; i++)
    nu2[i] = LIN(0.0, nu2[0], n - 1.0, nu2[n - 1], i);

  /* Convolute... */
  for (int ipts = 0; ipts < npts; ipts++)
    if (nurfm[ipts] >= nu2[0] && nurfm[ipts] <= nu2[n - 1]) {
      const int idx = locate_irr(nu2, n, nurfm[ipts]);
      filt = LIN(nu2[idx], f[idx], nu2[idx + 1], f[idx + 1], nurfm[ipts]);
```

```
      fsum += filt;
      radsum += filt * rad[ipts];
    }

  /* Free... */
  free(nurfm);
  free(rad);

  /* Return radiance... */
  return radsum / fsum;
}

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

void read_rfm_spec(
  const char *filename,
  double *nu,
  double *rad,
  int *npts) {

  FILE *in;

  char line[RFMLINE], *tok;

  double dnu, nu0, nu1;

  int ipts = 0;

  /* Write info... */
  LOG(1, "Read RFM data: %s", filename);

  /* Open file... */
  if (!(in = fopen(filename, "r")))
    ERRMSG("Cannot open file!");

  /* Read header...... */
  for (int i = 0; i < 4; i++)
    if (fgets(line, RFMLINE, in) == NULL)
      ERRMSG("Error while reading file header!");
  sscanf(line, "%d %lg %lg %lg", npts, &nu0, &dnu, &nu1);
  if (*npts > RFMNPTS)
    ERRMSG("Too many spectral grid points!");

  /* Read radiance data... */
  while (fgets(line, RFMLINE, in) && ipts < *npts) {
    if ((tok = strtok(line, " \t\n")) != NULL)
      if (sscanf(tok, "%lg", &rad[ipts]) == 1)
        ipts++;
    while ((tok = strtok(NULL, " \t\n")) != NULL)
      if (sscanf(tok, "%lg", &rad[ipts]) == 1)
        ipts++;
  }
  if (ipts != *npts)
    ERRMSG("Error while reading RFM data!");

  /* Compute wavenumbers... */
  for (ipts = 0; ipts < *npts; ipts++)
    nu[ipts] = LIN(0.0, nu0, (double) (*npts - 1), nu1, (double) ipts);

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

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

void read_shape(
  const char *filename,
  double *x,
  double *y,
  int *n) {

  FILE *in;

  char line[LEN];

  /* Write info... */
  LOG(1, "Read shape function: %s", filename);

  /* Open file... */
  if (!(in = fopen(filename, "r")))
    ERRMSG("Cannot open file!");

  /* Read data... */
  *n = 0;
  while (fgets(line, LEN, in))
    if (sscanf(line, "%lg %lg", &x[*n], &y[*n]) == 2)
      if ((++(*n)) > NSHAPE)
```

```
      ERRMSG("Too many data points!");

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

  /* Check number of data points... */
  if (*n < 2)
    ERRMSG("Could not read any data!");

  /* Write info... */
  double mini, maxi;
  LOG(2, "Number of data points: %d", *n);
  gsl_stats_minmax(&mini, &maxi, x, 1, (size_t) *n);
  LOG(2, "Range of x values: %.4f ... %.4f", mini, maxi);
  gsl_stats_minmax(&mini, &maxi, y, 1, (size_t) *n);
  LOG(2, "Range of y values: %g ... %g", mini, maxi);
}

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

void read_tbl(
  const ctl_t *ctl,
  tbl_t *tbl) {

  FILE *in;

  char filename[2 * LEN], line[LEN];

  double eps, press, temp, u;

  /* Loop over trace gases and channels... */
  for (int id = 0; id < ctl->nd; id++)
    for (int ig = 0; ig < ctl->ng; ig++) {

      /* Initialize... */
      tbl->np[id][ig] = -1;
      double eps_old = -999;
      double press_old = -999;
      double temp_old = -999;
      double u_old = -999;
      int nrange = 0;

      /* Set filename... */
      sprintf(filename, "%s_%.4f_%s.%s", ctl->tblbase,
              ctl->nu[id], ctl->emitter[ig],
              ctl->tblfmt == 1 ? "tab" : "bin");

      /* Write info... */
      LOG(1, "Read emissivity table: %s", filename);

      /* Try to open file... */
      if (!(in = fopen(filename, "r"))) {
        WARN("Missing emissivity table: %s", filename);
        continue;
      }

      /* Read ASCII tables... */
      if (ctl->tblfmt == 1) {

        /* Read data... */
        while (fgets(line, LEN, in)) {

          /* Parse line... */
          if (sscanf(line, "%lg %lg %lg %lg", &press, &temp, &u, &eps) != 4)
            continue;

          /* Check ranges... */
          if (u < UMIN || u > UMAX || eps < EPSMIN || eps > EPSMAX) {
            nrange++;
            continue;
          }

          /* Determine pressure index... */
          if (press != press_old) {
            press_old = press;
            if ((++tbl->np[id][ig]) >= TBLNP)
              ERRMSG("Too many pressure levels!");
            tbl->nt[id][ig][tbl->np[id][ig]] = -1;
          }

          /* Determine temperature index... */
          if (temp != temp_old) {
            temp_old = temp;
            if ((++tbl->nt[id][ig][tbl->np[id][ig]]) >= TBLNT)
              ERRMSG("Too many temperatures!");
            tbl->nu[id][ig][tbl->np[id][ig]]
              [tbl->nt[id][ig][tbl->np[id][ig]]] = -1;
```

```
        }

        /* Determine column density index... */
        if ((eps > eps_old && u > u_old) || tbl->nu[id][ig][tbl->np[id][ig]]
            [tbl->nt[id][ig][tbl->np[id][ig]]] < 0) {
          eps_old = eps;
          u_old = u;
          if ((++tbl->nu[id][ig][tbl->np[id][ig]]
               [tbl->nt[id][ig][tbl->np[id][ig]]]) >= TBLNU)
            ERRMSG("Too many column densities!");
        }

        /* Store data... */
        tbl->p[id][ig][tbl->np[id][ig]] = press;
        tbl->t[id][ig][tbl->np[id][ig]][tbl->nt[id][ig][tbl->np[id][ig]]]
          = temp;
        tbl->u[id][ig][tbl->np[id][ig]][tbl->nt[id][ig][tbl->np[id][ig]]]
          [tbl->nu[id][ig][tbl->np[id][ig]]
           [tbl->nt[id][ig][tbl->np[id][ig]]]] = (float) u;
        tbl->eps[id][ig][tbl->np[id][ig]][tbl->nt[id][ig][tbl->np[id][ig]]]
          [tbl->nu[id][ig][tbl->np[id][ig]]
           [tbl->nt[id][ig][tbl->np[id][ig]]]] = (float) eps;
      }

      /* Increment counters... */
      tbl->np[id][ig]++;
      for (int ip = 0; ip < tbl->np[id][ig]; ip++) {
        tbl->nt[id][ig][ip]++;
        for (int it = 0; it < tbl->nt[id][ig][ip]; it++)
          tbl->nu[id][ig][ip][it]++;
      }
    }

    /* Read binary data... */
    else if (ctl->tblfmt == 2) {

      /* Read data... */
      FREAD(&tbl->np[id][ig], int,
            1,
            in);
      if (tbl->np[id][ig] > TBLNP)
        ERRMSG("Too many pressure levels!");
      FREAD(tbl->p[id][ig], double,
              (size_t) tbl->np[id][ig],
            in);
      for (int ip = 0; ip < tbl->np[id][ig]; ip++) {
        FREAD(&tbl->nt[id][ig][ip], int,
              1,
              in);
        if (tbl->nt[id][ig][ip] > TBLNT)
          ERRMSG("Too many temperatures!");
        FREAD(tbl->t[id][ig][ip], double,
                (size_t) tbl->nt[id][ig][ip],
              in);
        for (int it = 0; it < tbl->nt[id][ig][ip]; it++) {
          FREAD(&tbl->nu[id][ig][ip][it], int,
                1,
                in);
          if (tbl->nu[id][ig][ip][it] > TBLNU)
            ERRMSG("Too many column densities!");
          FREAD(tbl->u[id][ig][ip][it], float,
                  (size_t) tbl->nu[id][ig][ip][it],
                in);
          FREAD(tbl->eps[id][ig][ip][it], float,
                  (size_t) tbl->nu[id][ig][ip][it],
                in);
        }
      }
    }

    /* Error message... */
    else
      ERRMSG("Unknown look-up table format!");

    /* Check ranges... */
    if (nrange > 0)
      WARN("Column density or emissivity out of range (%d data points)!",
           nrange);

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

    /* Write info... */
    for (int ip = 0; ip < tbl->np[id][ig]; ip++)
      LOG(2,
          "p[%2d]= %.5e hPa | T[0:%2d]= %.2f ... %.2f K | u[0:%3d]= %.5e ... %.5e molec/cm^2 | eps[0:%3d]= %.5e ... %.5e",
```

```
            ip, tbl->p[id][ig][ip], tbl->nt[id][ig][ip] - 1,
            tbl->t[id][ig][ip][0],
            tbl->t[id][ig][ip][tbl->nt[id][ig][ip] - 1],
            tbl->nu[id][ig][ip][0] - 1, tbl->u[id][ig][ip][0][0],
            tbl->u[id][ig][ip][0][tbl->nu[id][ig][ip][0] - 1],
            tbl->nu[id][ig][ip][0] - 1, tbl->eps[id][ig][ip][0][0],
            tbl->eps[id][ig][ip][0][tbl->nu[id][ig][ip][0] - 1]);
    }
}

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

double scan_ctl(
  int argc,
  char *argv[],
  const char *varname,
  int arridx,
  const char *defvalue,
  char *value) {

  FILE *in = NULL;

  char dummy[LEN], fullname1[LEN], fullname2[LEN], line[LEN],
    rvarname[LEN], rval[LEN];

  int contain = 0;

  /* Open file... */
  if (argv[1][0] != '-')
    if (!(in = fopen(argv[1], "r")))
      ERRMSG("Cannot open file!");

  /* Set full variable name... */
  if (arridx >= 0) {
    sprintf(fullname1, "%s[%d]", varname, arridx);
    sprintf(fullname2, "%s[*]", varname);
  } else {
    sprintf(fullname1, "%s", varname);
    sprintf(fullname2, "%s", varname);
  }

  /* Read data... */
  if (in != NULL)
    while (fgets(line, LEN, in))
      if (sscanf(line, "%s %s %s", rvarname, dummy, rval) == 3)
        if (strcasecmp(rvarname, fullname1) == 0 ||
            strcasecmp(rvarname, fullname2) == 0) {
          contain = 1;
          break;
        }
  for (int i = 1; i < argc - 1; i++)
    if (strcasecmp(argv[i], fullname1) == 0 ||
        strcasecmp(argv[i], fullname2) == 0) {
      sprintf(rval, "%s", argv[i + 1]);
      contain = 1;
      break;
    }

  /* Close file... */
  if (in != NULL)
    fclose(in);

  /* Check for missing variables... */
  if (!contain) {
    if (strlen(defvalue) > 0)
      sprintf(rval, "%s", defvalue);
    else
      ERRMSG("Missing variable %s!\n", fullname1);
  }

  /* Write info... */
  LOG(1, "%s = %s", fullname1, rval);

  /* Return values... */
  if (value != NULL)
    sprintf(value, "%s", rval);
  return atof(rval);
}

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

double sza(
  const double sec,
  const double lon,
  const double lat) {

  /* Number of days and fraction with respect to 2000-01-01T12:00Z... */
```

```
  const double D = sec / 86400 - 0.5;

  /* Geocentric apparent ecliptic longitude [rad]... */
  const double g = DEG2RAD(357.529 + 0.98560028 * D);
  const double q = 280.459 + 0.98564736 * D;
  const double L = DEG2RAD(q + 1.915 * sin(g) + 0.020 * sin(2 * g));

  /* Mean obliquity of the ecliptic [rad]... */
  const double e = DEG2RAD(23.439 - 0.00000036 * D);

  /* Declination [rad]... */
  const double dec = asin(sin(e) * sin(L));

  /* Right ascension [rad]... */
  const double ra = atan2(cos(e) * sin(L), cos(L));

  /* Greenwich Mean Sidereal Time [h]... */
  const double GMST = 18.697374558 + 24.06570982441908 * D;

  /* Local Sidereal Time [h]... */
  const double LST = GMST + lon / 15;

  /* Hour angle [rad]... */
  const double h = LST / 12 * M_PI - ra;

  /* Convert latitude... */
  const double latr = DEG2RAD(lat);

  /* Return solar zenith angle [deg]... */
  return RAD2DEG(acos(sin(latr) * sin(dec) + cos(latr) * cos(dec) * cos(h)));
}

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

void tangent_point(
  const los_t *los,
  double *tpz,
  double *tplon,
  double *tplat) {

  double dummy, v[3], v0[3], v2[3];

  /* Find minimum altitude... */
  const size_t ip = gsl_stats_min_index(los->z, 1, (size_t) los->np);

  /* Nadir or zenith... */
  if (ip <= 0 || ip >= (size_t) los->np - 1) {
    *tpz = los->z[los->np - 1];
    *tplon = los->lon[los->np - 1];
    *tplat = los->lat[los->np - 1];
  }

  /* Limb... */
  else {

    /* Determine interpolating polynomial y=a*x^2+b*x+c... */
    const double yy0 = los->z[ip - 1];
    const double yy1 = los->z[ip];
    const double yy2 = los->z[ip + 1];
    const double x1 = sqrt(POW2(los->ds[ip]) - POW2(yy1 - yy0));
    const double x2 = x1 + sqrt(POW2(los->ds[ip + 1]) - POW2(yy2 - yy1));
    const double a = 1 / (x1 - x2) * (-(yy0 - yy1) / x1 + (yy0 - yy2) / x2);
    const double b = -(yy0 - yy1) / x1 - a * x1;
    const double c = yy0;

    /* Get tangent point location... */
    const double x = -b / (2 * a);
    *tpz = a * x * x + b * x + c;
    geo2cart(los->z[ip - 1], los->lon[ip - 1], los->lat[ip - 1], v0);
    geo2cart(los->z[ip + 1], los->lon[ip + 1], los->lat[ip + 1], v2);
    for (int i = 0; i < 3; i++)
      v[i] = LIN(0.0, v0[i], x2, v2[i], x);
    cart2geo(v, &dummy, tplon, tplat);
  }
}

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

void time2jsec(
  const int year,
  const int mon,
  const int day,
  const int hour,
  const int min,
  const int sec,
  const double remain,
  double *jsec) {
```

```
  struct tm t0, t1;

  t0.tm_year = 100;
  t0.tm_mon = 0;
  t0.tm_mday = 1;
  t0.tm_hour = 0;
  t0.tm_min = 0;
  t0.tm_sec = 0;

  t1.tm_year = year - 1900;
  t1.tm_mon = mon - 1;
  t1.tm_mday = day;
  t1.tm_hour = hour;
  t1.tm_min = min;
  t1.tm_sec = sec;

  *jsec = (double) timegm(&t1) - (double) timegm(&t0) + remain;
}

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

void timer(
  const char *name,
  const char *file,
  const char *func,
  int line,
  int mode) {

  static double w0[10];

  static int l0[10], nt;

  /* Start new timer... */
  if (mode == 1) {
    w0[nt] = omp_get_wtime();
    l0[nt] = line;
    if ((++nt) >= 10)
      ERRMSG("Too many timers!");
  }

  /* Write elapsed time... */
  else {

    /* Check timer index... */
    if (nt - 1 < 0)
      ERRMSG("Coding error!");

    /* Write elapsed time... */
    LOG(1, "Timer '%s' (%s, %s, l%d-%d): %.3f sec",
        name, file, func, l0[nt - 1], line, omp_get_wtime() - w0[nt - 1]);
  }

  /* Stop timer... */
  if (mode == 3)
    nt--;
}

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

void write_atm(
  const char *dirname,
  const char *filename,
  const ctl_t *ctl,
  const atm_t *atm) {

  FILE *out;

  char file[LEN];

  int n = 6;

  /* Set filename... */
  if (dirname != NULL)
    sprintf(file, "%s/%s", dirname, filename);
  else
    sprintf(file, "%s", filename);

  /* Write info... */
  LOG(1, "Write atmospheric data: %s", file);

  /* Create file... */
  if (!(out = fopen(file, "w")))
    ERRMSG("Cannot create file!");

  /* Write header... */
  fprintf(out,
```

```
          "# $1 = time (seconds since 2000-01-01T00:00Z)\n"
          "# $2 = altitude [km]\n"
          "# $3 = longitude [deg]\n"
          "# $4 = latitude [deg]\n"
          "# $5 = pressure [hPa]\n" "# $6 = temperature [K]\n");
  for (int ig = 0; ig < ctl->ng; ig++)
    fprintf(out, "# $%d = %s volume mixing ratio [ppv]\n",
            ++n, ctl->emitter[ig]);
  for (int iw = 0; iw < ctl->nw; iw++)
    fprintf(out, "# $%d = extinction (window %d) [km^-1]\n", ++n, iw);
  if (ctl->ncl > 0) {
    fprintf(out, "# $%d = cloud layer height [km]\n", ++n);
    fprintf(out, "# $%d = cloud layer depth [km]\n", ++n);
    for (int icl = 0; icl < ctl->ncl; icl++)
      fprintf(out, "# $%d = cloud layer extinction (%.4f cm^-1) [km^-1]\n",
              ++n, ctl->clnu[icl]);
  }
  if (ctl->nsf > 0) {
    fprintf(out, "# $%d = surface layer height [km]\n", ++n);
    fprintf(out, "# $%d = surface layer pressure [hPa]\n", ++n);
    fprintf(out, "# $%d = surface layer temperature [K]\n", ++n);
    for (int isf = 0; isf < ctl->nsf; isf++)
      fprintf(out, "# $%d = surface layer emissivity (%.4f cm^-1)\n",
              ++n, ctl->sfnu[isf]);
  }

  /* Write data... */
  for (int ip = 0; ip < atm->np; ip++) {
    if (ip == 0 || atm->time[ip] != atm->time[ip - 1])
      fprintf(out, "\n");
    fprintf(out, "%.2f %g %g %g %g %g", atm->time[ip], atm->z[ip],
            atm->lon[ip], atm->lat[ip], atm->p[ip], atm->t[ip]);
    for (int ig = 0; ig < ctl->ng; ig++)
      fprintf(out, " %g", atm->q[ig][ip]);
    for (int iw = 0; iw < ctl->nw; iw++)
      fprintf(out, " %g", atm->k[iw][ip]);
    if (ctl->ncl > 0) {
      fprintf(out, " %g %g", atm->clz, atm->cldz);
      for (int icl = 0; icl < ctl->ncl; icl++)
        fprintf(out, " %g", atm->clk[icl]);
    }
    if (ctl->nsf > 0) {
      fprintf(out, " %g %g %g", atm->sfz, atm->sfp, atm->sft);
      for (int isf = 0; isf < ctl->nsf; isf++)
        fprintf(out, " %g", atm->sfeps[isf]);
    }
    fprintf(out, "\n");
  }

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

  /* Write info... */
  double mini, maxi;
  LOG(2, "Number of data points: %d", atm->np);
  gsl_stats_minmax(&mini, &maxi, atm->time, 1, (size_t) atm->np);
  LOG(2, "Time range: %.2f ... %.2f s", mini, maxi);
  gsl_stats_minmax(&mini, &maxi, atm->z, 1, (size_t) atm->np);
  LOG(2, "Altitude range: %g ... %g km", mini, maxi);
  gsl_stats_minmax(&mini, &maxi, atm->lon, 1, (size_t) atm->np);
  LOG(2, "Longitude range: %g ... %g deg", mini, maxi);
  gsl_stats_minmax(&mini, &maxi, atm->lat, 1, (size_t) atm->np);
  LOG(2, "Latitude range: %g ... %g deg", mini, maxi);
  gsl_stats_minmax(&mini, &maxi, atm->p, 1, (size_t) atm->np);
  LOG(2, "Pressure range: %g ... %g hPa", maxi, mini);
  gsl_stats_minmax(&mini, &maxi, atm->t, 1, (size_t) atm->np);
  LOG(2, "Temperature range: %g ... %g K", mini, maxi);
  for (int ig = 0; ig < ctl->ng; ig++) {
    gsl_stats_minmax(&mini, &maxi, atm->q[ig], 1, (size_t) atm->np);
    LOG(2, "Emitter %s range: %g ... %g ppv", ctl->emitter[ig], mini, maxi);
  }
  for (int iw = 0; iw < ctl->nw; iw++) {
    gsl_stats_minmax(&mini, &maxi, atm->k[iw], 1, (size_t) atm->np);
    LOG(2, "Extinction range (window %d): %g ... %g km^-1", iw, mini, maxi);
  }
  if (ctl->ncl > 0 && atm->np == 0) {
    LOG(2, "Cloud layer: z= %g km | dz= %g km | k= %g ... %g km^-1",
        atm->clz, atm->cldz, atm->clk[0], atm->clk[ctl->ncl - 1]);
  } else
    LOG(2, "Cloud layer: none");
  if (ctl->nsf > 0 && atm->np == 0) {
    LOG(2,
        "Surface layer: z_s= %g km | p_s= %g hPa | T_s = %g K | eps= %g ... %g",
        atm->sfz, atm->sfp, atm->sft, atm->sfeps[0],
        atm->sfeps[ctl->nsf - 1]);
  } else
    LOG(2, "Surface layer: none");
```

```
}

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

void write_atm_rfm(
  const char *filename,
  const ctl_t *ctl,
  const atm_t *atm) {

  FILE *out;

  /* Write info... */
  LOG(1, "Write RFM data: %s", filename);

  /* Create file... */
  if (!(out = fopen(filename, "w")))
    ERRMSG("Cannot create file!");

  /* Write data... */
  fprintf(out, "%d\n", atm->np);
  fprintf(out, "*HGT [km]\n");
  for (int ip = 0; ip < atm->np; ip++)
    fprintf(out, "%g\n", atm->z[ip]);
  fprintf(out, "*PRE [mb]\n");
  for (int ip = 0; ip < atm->np; ip++)
    fprintf(out, "%g\n", atm->p[ip]);
  fprintf(out, "*TEM [K]\n");
  for (int ip = 0; ip < atm->np; ip++)
    fprintf(out, "%g\n", atm->t[ip]);
  for (int ig = 0; ig < ctl->ng; ig++) {
    fprintf(out, "*%s [ppmv]\n", ctl->emitter[ig]);
    for (int ip = 0; ip < atm->np; ip++)
      fprintf(out, "%g\n", atm->q[ig][ip] * 1e6);
  }
  fprintf(out, "*END\n");

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

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

void write_matrix(
  const char *dirname,
  const char *filename,
  const ctl_t *ctl,
  const gsl_matrix *matrix,
  const atm_t *atm,
  const obs_t *obs,
  const char *rowspace,
  const char *colspace,
  const char *sort) {

  FILE *out;

  char file[LEN], quantity[LEN];

  int *cida, *ciqa, *cipa, *cira, *rida, *riqa, *ripa, *rira;

  size_t i, j, nc, nr;

  /* Check output flag... */
  if (!ctl->write_matrix)
    return;

  /* Allocate... */
  ALLOC(cida, int,
        M);
  ALLOC(ciqa, int,
        N);
  ALLOC(cipa, int,
        N);
  ALLOC(cira, int,
        M);
  ALLOC(rida, int,
        M);
  ALLOC(riqa, int,
        N);
  ALLOC(ripa, int,
        N);
  ALLOC(rira, int,
        M);

  /* Set filename... */
  if (dirname != NULL)
    sprintf(file, "%s/%s", dirname, filename);
  else
```

```
  sprintf(file, "%s", filename);

/* Write info... */
LOG(1, "Write matrix: %s", file);

/* Create file... */
if (!(out = fopen(file, "w")))
  ERRMSG("Cannot create file!");

/* Write header (row space)... */
if (rowspace[0] == 'y') {

  fprintf(out,
          "# $1 = Row: index (measurement space)\n"
          "# $2 = Row: channel wavenumber [cm^-1]\n"
          "# $3 = Row: time (seconds since 2000-01-01T00:00Z)\n"
          "# $4 = Row: view point altitude [km]\n"
          "# $5 = Row: view point longitude [deg]\n"
          "# $6 = Row: view point latitude [deg]\n");

  /* Get number of rows... */
  nr = obs2y(ctl, obs, NULL, rida, rira);

} else {

  fprintf(out,
          "# $1 = Row: index (state space)\n"
          "# $2 = Row: name of quantity\n"
          "# $3 = Row: time (seconds since 2000-01-01T00:00Z)\n"
          "# $4 = Row: altitude [km]\n"
          "# $5 = Row: longitude [deg]\n" "# $6 = Row: latitude [deg]\n");

  /* Get number of rows... */
  nr = atm2x(ctl, atm, NULL, riqa, ripa);
}

/* Write header (column space)... */
if (colspace[0] == 'y') {

  fprintf(out,
          "# $7 = Col: index (measurement space)\n"
          "# $8 = Col: channel wavenumber [cm^-1]\n"
          "# $9 = Col: time (seconds since 2000-01-01T00:00Z)\n"
          "# $10 = Col: view point altitude [km]\n"
          "# $11 = Col: view point longitude [deg]\n"
          "# $12 = Col: view point latitude [deg]\n");

  /* Get number of columns... */
  nc = obs2y(ctl, obs, NULL, cida, cira);

} else {

  fprintf(out,
          "# $7 = Col: index (state space)\n"
          "# $8 = Col: name of quantity\n"
          "# $9 = Col: time (seconds since 2000-01-01T00:00Z)\n"
          "# $10 = Col: altitude [km]\n"
          "# $11 = Col: longitude [deg]\n" "# $12 = Col: latitude [deg]\n");

  /* Get number of columns... */
  nc = atm2x(ctl, atm, NULL, ciqa, cipa);
}

/* Write header entry... */
fprintf(out, "# $13 = Matrix element\n\n");

/* Write matrix data... */
i = j = 0;
while (i < nr && j < nc) {

  /* Write info about the row... */
  if (rowspace[0] == 'y')
    fprintf(out, "%d %.4f %.2f %g %g %g",
            (int) i, ctl->nu[rida[i]],
            obs->time[rira[i]], obs->vpz[rira[i]],
            obs->vplon[rira[i]], obs->vplat[rira[i]]);
  else {
    idx2name(ctl, riqa[i], quantity);
    fprintf(out, "%d %s %.2f %g %g %g", (int) i, quantity,
            atm->time[ripa[i]], atm->z[ripa[i]],
            atm->lon[ripa[i]], atm->lat[ripa[i]]);
  }

  /* Write info about the column... */
  if (colspace[0] == 'y')
    fprintf(out, " %d %.4f %.2f %g %g %g",
            (int) j, ctl->nu[cida[j]],
```

```
              obs->time[cira[j]], obs->vpz[cira[j]],
              obs->vplon[cira[j]], obs->vplat[cira[j]]);
    else {
      idx2name(ctl, ciqa[j], quantity);
      fprintf(out, " %d %s %.2f %g %g %g", (int) j, quantity,
              atm->time[cipa[j]], atm->z[cipa[j]],
              atm->lon[cipa[j]], atm->lat[cipa[j]]);
    }

    /* Write matrix entry... */
    fprintf(out, " %g\n", gsl_matrix_get(matrix, i, j));

    /* Set matrix indices... */
    if (sort[0] == 'r') {
      j++;
      if (j >= nc) {
        j = 0;
        i++;
        fprintf(out, "\n");
      }
    } else {
      i++;
      if (i >= nr) {
        i = 0;
        j++;
        fprintf(out, "\n");
      }
    }
  }

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

  /* Free... */
  free(cida);
  free(ciqa);
  free(cipa);
  free(cira);
  free(rida);
  free(riqa);
  free(ripa);
  free(rira);
}

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

void write_obs(
  const char *dirname,
  const char *filename,
  const ctl_t *ctl,
  const obs_t *obs) {

  FILE *out;

  char file[LEN];

  int n = 10;

  /* Set filename... */
  if (dirname != NULL)
    sprintf(file, "%s/%s", dirname, filename);
  else
    sprintf(file, "%s", filename);

  /* Write info... */
  LOG(1, "Write observation data: %s", file);

  /* Create file... */
  if (!(out = fopen(file, "w")))
    ERRMSG("Cannot create file!");

  /* Write header... */
  fprintf(out,
          "# $1 = time (seconds since 2000-01-01T00:00Z)\n"
          "# $2 = observer altitude [km]\n"
          "# $3 = observer longitude [deg]\n"
          "# $4 = observer latitude [deg]\n"
          "# $5 = view point altitude [km]\n"
          "# $6 = view point longitude [deg]\n"
          "# $7 = view point latitude [deg]\n"
          "# $8 = tangent point altitude [km]\n"
          "# $9 = tangent point longitude [deg]\n"
          "# $10 = tangent point latitude [deg]\n");
  for (int id = 0; id < ctl->nd; id++)
    if (ctl->write_bbt)
      fprintf(out, "# $%d = brightness temperature (%.4f cm^-1) [K]\n",
              ++n, ctl->nu[id]);
```

```
    else
      fprintf(out, "# $%d = radiance (%.4f cm^-1) [W/(m^2 sr cm^-1)]\n",
              ++n, ctl->nu[id]);
  for (int id = 0; id < ctl->nd; id++)
    fprintf(out, "# $%d = transmittance (%.4f cm^-1) [-]\n", ++n,
            ctl->nu[id]);

  /* Write data... */
  for (int ir = 0; ir < obs->nr; ir++) {
    if (ir == 0 || obs->time[ir] != obs->time[ir - 1])
      fprintf(out, "\n");
    fprintf(out, "%.2f %g %g %g %g %g %g %g %g %g", obs->time[ir],
            obs->obsz[ir], obs->obslon[ir], obs->obslat[ir],
            obs->vpz[ir], obs->vplon[ir], obs->vplat[ir],
            obs->tpz[ir], obs->tplon[ir], obs->tplat[ir]);
    for (int id = 0; id < ctl->nd; id++)
      fprintf(out, " %g", obs->rad[id][ir]);
    for (int id = 0; id < ctl->nd; id++)
      fprintf(out, " %g", obs->tau[id][ir]);
    fprintf(out, "\n");
  }

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

  /* Write info... */
  double mini, maxi;
  LOG(2, "Number of ray paths: %d", obs->nr);
  gsl_stats_minmax(&mini, &maxi, obs->time, 1, (size_t) obs->nr);
  LOG(2, "Time range: %.2f ... %.2f s", mini, maxi);
  gsl_stats_minmax(&mini, &maxi, obs->obsz, 1, (size_t) obs->nr);
  LOG(2, "Observer altitude range: %g ... %g km", mini, maxi);
  gsl_stats_minmax(&mini, &maxi, obs->obslon, 1, (size_t) obs->nr);
  LOG(2, "Observer longitude range: %g ... %g deg", mini, maxi);
  gsl_stats_minmax(&mini, &maxi, obs->obslat, 1, (size_t) obs->nr);
  LOG(2, "Observer latitude range: %g ... %g deg", mini, maxi);
  gsl_stats_minmax(&mini, &maxi, obs->vpz, 1, (size_t) obs->nr);
  LOG(2, "View point altitude range: %g ... %g km", mini, maxi);
  gsl_stats_minmax(&mini, &maxi, obs->vplon, 1, (size_t) obs->nr);
  LOG(2, "View point longitude range: %g ... %g deg", mini, maxi);
  gsl_stats_minmax(&mini, &maxi, obs->vplat, 1, (size_t) obs->nr);
  LOG(2, "View point latitude range: %g ... %g deg", mini, maxi);
  gsl_stats_minmax(&mini, &maxi, obs->tpz, 1, (size_t) obs->nr);
  LOG(2, "Tangent point altitude range: %g ... %g km", mini, maxi);
  gsl_stats_minmax(&mini, &maxi, obs->tplon, 1, (size_t) obs->nr);
  LOG(2, "Tangent point longitude range: %g ... %g deg", mini, maxi);
  gsl_stats_minmax(&mini, &maxi, obs->tplat, 1, (size_t) obs->nr);
  LOG(2, "Tangent point latitude range: %g ... %g deg", mini, maxi);
  for (int id = 0; id < ctl->nd; id++) {
    gsl_stats_minmax(&mini, &maxi, obs->rad[id], 1, (size_t) obs->nr);
    if (ctl->write_bbt) {
      LOG(2, "Brightness temperature (%.4f cm^-1) range: %g ... %g K",
          ctl->nu[id], mini, maxi);
    } else {
      LOG(2, "Radiance (%.4f cm^-1) range: %g ... %g W/(m^2 sr cm^-1)",
          ctl->nu[id], mini, maxi);
    }
  }
  for (int id = 0; id < ctl->nd; id++) {
    gsl_stats_minmax(&mini, &maxi, obs->tau[id], 1, (size_t) obs->nr);
    if (ctl->write_bbt) {
      LOG(2, "Transmittance (%.4f cm^-1) range: %g ... %g",
          ctl->nu[id], mini, maxi);
    }
  }
}

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

void write_shape(
  const char *filename,
  const double *x,
  const double *y,
  const int n) {

  FILE *out;

  /* Write info... */
  LOG(1, "Write shape function: %s", filename);

  /* Create file... */
  if (!(out = fopen(filename, "w")))
    ERRMSG("Cannot create file!");

  /* Write header... */
  fprintf(out,
          "# $1 = shape function x-value [-]\n"
```

```
          "# $2 = shape function y-value [-]\n\n");

  /* Write data... */
  for (int i = 0; i < n; i++)
    fprintf(out, "%.10g %.10g\n", x[i], y[i]);

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

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

void write_tbl(
  const ctl_t *ctl,
  const tbl_t *tbl) {

  FILE *out;

  char filename[2 * LEN];

  /* Loop over emitters and detectors... */
  for (int ig = 0; ig < ctl->ng; ig++)
    for (int id = 0; id < ctl->nd; id++) {

      /* Set filename... */
      sprintf(filename, "%s_%.4f_%s.%s", ctl->tblbase,
              ctl->nu[id], ctl->emitter[ig],
              ctl->tblfmt == 1 ? "tab" : "bin");

      /* Write info... */
      LOG(1, "Write emissivity table: %s", filename);

      /* Create file... */
      if (!(out = fopen(filename, "w")))
        ERRMSG("Cannot create file!");

      /* Write ASCII data... */
      if (ctl->tblfmt == 1) {

        /* Write header... */
        fprintf(out,
                "# $1 = pressure [hPa]\n"
                "# $2 = temperature [K]\n"
                "# $3 = column density [molecules/cm^2]\n"
                "# $4 = emissivity [-]\n");

        /* Save table file... */
        for (int ip = 0; ip < tbl->np[id][ig]; ip++)
          for (int it = 0; it < tbl->nt[id][ig][ip]; it++) {
            fprintf(out, "\n");
            for (int iu = 0; iu < tbl->nu[id][ig][ip][it]; iu++)
              fprintf(out, "%g %g %e %e\n",
                      tbl->p[id][ig][ip], tbl->t[id][ig][ip][it],
                      tbl->u[id][ig][ip][it][iu],
                      tbl->eps[id][ig][ip][it][iu]);
          }
      }

      /* Write binary data... */
      else if (ctl->tblfmt == 2) {
        FWRITE(&tbl->np[id][ig], int,
               1,
               out);
        FWRITE(tbl->p[id][ig], double,
                 (size_t) tbl->np[id][ig],
               out);
        for (int ip = 0; ip < tbl->np[id][ig]; ip++) {
          FWRITE(&tbl->nt[id][ig][ip], int,
                 1,
                 out);
          FWRITE(tbl->t[id][ig][ip], double,
                   (size_t) tbl->nt[id][ig][ip],
                 out);
          for (int it = 0; it < tbl->nt[id][ig][ip]; it++) {
            FWRITE(&tbl->nu[id][ig][ip][it], int,
                   1,
                   out);
            FWRITE(tbl->u[id][ig][ip][it], float,
                     (size_t) tbl->nu[id][ig][ip][it],
                   out);
            FWRITE(tbl->eps[id][ig][ip][it], float,
                     (size_t) tbl->nu[id][ig][ip][it],
                   out);
          }
        }
      }

```

```
      /* Error message... */
      else
        ERRMSG("Unknown look-up table format!");

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

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

void x2atm(
  const ctl_t *ctl,
  const gsl_vector *x,
  atm_t *atm) {

  size_t n = 0;

  /* Get pressure... */
  for (int ip = 0; ip < atm->np; ip++)
    if (atm->z[ip] >= ctl->retp_zmin && atm->z[ip] <= ctl->retp_zmax)
      x2atm_help(&atm->p[ip], x, &n);

  /* Get temperature... */
  for (int ip = 0; ip < atm->np; ip++)
    if (atm->z[ip] >= ctl->rett_zmin && atm->z[ip] <= ctl->rett_zmax)
      x2atm_help(&atm->t[ip], x, &n);

  /* Get volume mixing ratio... */
  for (int ig = 0; ig < ctl->ng; ig++)
    for (int ip = 0; ip < atm->np; ip++)
      if (atm->z[ip] >= ctl->retq_zmin[ig]
          && atm->z[ip] <= ctl->retq_zmax[ig])
        x2atm_help(&atm->q[ig][ip], x, &n);

  /* Get extinction... */
  for (int iw = 0; iw < ctl->nw; iw++)
    for (int ip = 0; ip < atm->np; ip++)
      if (atm->z[ip] >= ctl->retk_zmin[iw]
          && atm->z[ip] <= ctl->retk_zmax[iw])
        x2atm_help(&atm->k[iw][ip], x, &n);

  /* Get cloud data... */
  if (ctl->ret_clz)
    x2atm_help(&atm->clz, x, &n);
  if (ctl->ret_cldz)
    x2atm_help(&atm->cldz, x, &n);
  if (ctl->ret_clk)
    for (int icl = 0; icl < ctl->ncl; icl++)
      x2atm_help(&atm->clk[icl], x, &n);

  /* Get surface data... */
  if (ctl->ret_sfz)
    x2atm_help(&atm->sfz, x, &n);
  if (ctl->ret_sfp)
    x2atm_help(&atm->sfp, x, &n);
  if (ctl->ret_sft)
    x2atm_help(&atm->sft, x, &n);
  if (ctl->ret_sfeps)
    for (int isf = 0; isf < ctl->nsf; isf++)
      x2atm_help(&atm->sfeps[isf], x, &n);
}

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

void x2atm_help(
  double *value,
  const gsl_vector *x,
  size_t *n) {

  /* Get state vector element... */
  *value = gsl_vector_get(x, *n);
  (*n)++;
}

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

void y2obs(
  const ctl_t *ctl,
  const gsl_vector *y,
  obs_t *obs) {

  size_t m = 0;

  /* Decompose measurement vector... */
  for (int ir = 0; ir < obs->nr; ir++)
    for (int id = 0; id < ctl->nd; id++)
```

```
      if (isfinite(obs->rad[id][ir])) {
        obs->rad[id][ir] = gsl_vector_get(y, m);
        m++;
      }
}
```

5.7 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 (const ctl_t ∗ctl, const atm_t ∗atm, gsl_vector ∗x, int ∗iqa, int ∗ipa)

  Compose state vector or parameter vector.
- void atm2x_help (const double value, const int value_iqa, const int value_ip, gsl_vector ∗x, int ∗iqa, int ∗ipa, size_t ∗n)

  Add element to state vector.
- void cart2geo (const double ∗x, double ∗z, double ∗lon, double ∗lat)

  Convert Cartesian coordinates to geolocation.
- void climatology (const ctl_t ∗ctl, atm_t ∗atm_mean)

  Interpolate climatological data.
- double ctmco2 (const double nu, const double p, const double t, const double u)

  Compute carbon dioxide continuum (optical depth).
- double ctmh2o (const double nu, const double p, const double t, const double q, const double u)

  Compute water vapor continuum (optical depth).
- double ctmn2 (const double nu, const double p, const double t)

  Compute nitrogen continuum (absorption coefficient).
- double ctmo2 (const double nu, const double p, const double t)

  Compute oxygen continuum (absorption coefficient).
- void copy_atm (const ctl_t ∗ctl, atm_t ∗atm_dest, const atm_t ∗atm_src, const int init)

  Copy and initialize atmospheric data.
- void copy_obs (const ctl_t ∗ctl, obs_t ∗obs_dest, const obs_t ∗obs_src, const int init)

  Copy and initialize observation data.
- int find_emitter (const ctl_t ∗ctl, const char ∗emitter)

  Find index of an emitter.
- void formod (const ctl_t ∗ctl, atm_t ∗atm, obs_t ∗obs)

  Determine ray paths and compute radiative transfer.

- void formod_continua (const ctl_t ∗ctl, const los_t ∗los, const int ip, double ∗beta)

    Compute absorption coefficient of continua.
- void formod_fov (const ctl_t ∗ctl, obs_t ∗obs)

    Apply field of view convolution.
- void formod_pencil (const ctl_t ∗ctl, const atm_t ∗atm, obs_t ∗obs, const int ir)

    Compute radiative transfer for a pencil beam.
- void formod_rfm (const ctl_t ∗ctl, const atm_t ∗atm, obs_t ∗obs)

    Apply RFM for radiative transfer calculations.
- void formod_srcfunc (const ctl_t ∗ctl, const tbl_t ∗tbl, const double t, double ∗src)

    Compute Planck source function.
- void geo2cart (const double z, const double lon, const double lat, double ∗x)

    Convert geolocation to Cartesian coordinates.
- void hydrostatic (const ctl_t ∗ctl, atm_t ∗atm)

    Set hydrostatic equilibrium.
- void idx2name (const ctl_t ∗ctl, const int idx, char ∗quantity)

    Determine name of state vector quantity for given index.
- void init_srcfunc (const ctl_t ∗ctl, tbl_t ∗tbl)

    Initialize source function table.
- void intpol_atm (const ctl_t ∗ctl, const atm_t ∗atm, const double z, double ∗p, double ∗t, double ∗q, double ∗k)

    Interpolate atmospheric data.
- void intpol_tbl_cga (const ctl_t ∗ctl, const tbl_t ∗tbl, const los_t ∗los, const int ip, double tau_path[ND][NG], double tau_seg[ND])

    Get transmittance from look-up tables (CGA method).
- void intpol_tbl_ega (const ctl_t ∗ctl, const tbl_t ∗tbl, const los_t ∗los, const int ip, double tau_path[ND][NG], double tau_seg[ND])

    Get transmittance from look-up tables (EGA method).
- double intpol_tbl_eps (const tbl_t ∗tbl, const int ig, const int id, const int ip, const int it, const double u)

    Interpolate emissivity from look-up tables.
- double intpol_tbl_u (const tbl_t ∗tbl, const int ig, const int id, const int ip, const int it, const double eps)

    Interpolate column density from look-up tables.
- void jsec2time (const 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 (const double ∗xx, const int n, const double x)

    Find array index for irregular grid.
- int locate_reg (const double ∗xx, const int n, const double x)

    Find array index for regular grid.
- int locate_tbl (const float ∗xx, const int n, const double x)

    Find array index in float array.
- size_t obs2y (const ctl_t ∗ctl, const obs_t ∗obs, gsl_vector ∗y, int ∗ida, int ∗ira)

    Compose measurement vector.
- void raytrace (const ctl_t ∗ctl, const atm_t ∗atm, obs_t ∗obs, los_t ∗los, const int ir)

    Do ray-tracing to determine LOS.
- void read_atm (const char ∗dirname, const char ∗filename, const 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, const ctl_t *ctl, obs_t *obs)

  Read observation data.
- double read_obs_rfm (const char *basename, const double z, double *nu, double *f, int n)

  Read observation data in RFM format.
- void read_rfm_spec (const char *filename, double *nu, double *rad, int *npts)

  Read RFM spectrum.
- void read_shape (const char *filename, double *x, double *y, int *n)

  Read shape function.
- void read_tbl (const ctl_t *ctl, tbl_t *tbl)

  Read look-up table data.
- double scan_ctl (int argc, char *argv[ ], const char *varname, int arridx, const char *defvalue, char *value)

  Search control parameter file for variable entry.
- double sza (double sec, double lon, double lat)

  Calculate solar zenith angle.
- void tangent_point (const los_t *los, double *tpz, double *tplon, double *tplat)

  Find tangent point of a given LOS.
- void time2jsec (const int year, const int mon, const int day, const int hour, const int min, const int sec, const 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, const ctl_t *ctl, const atm_t *atm)

  Write atmospheric data.
- void write_atm_rfm (const char *filename, const ctl_t *ctl, const atm_t *atm)

  Write atmospheric data in RFM format.
- void write_matrix (const char *dirname, const char *filename, const ctl_t *ctl, const gsl_matrix *matrix, const atm_t *atm, const obs_t *obs, const char *rowspace, const char *colspace, const char *sort)

  Write matrix.
- void write_obs (const char *dirname, const char *filename, const ctl_t *ctl, const obs_t *obs)

  Write observation data.
- void write_shape (const char *filename, const double *x, const double *y, const int n)

  Write shape function.
- void write_tbl (const ctl_t *ctl, const tbl_t *tbl)

  Write look-up table data.
- void x2atm (const ctl_t *ctl, const gsl_vector *x, atm_t *atm)

  Decompose parameter vector or state vector.
- void x2atm_help (double *value, const gsl_vector *x, size_t *n)

  Get element from state vector.
- void y2obs (const ctl_t *ctl, const gsl_vector *y, obs_t *obs)

  Decompose measurement vector.

5.7.1 Detailed Description

JURASSIC library declarations.

Definition in file jurassic.h.

5.7.2 Function Documentation

atm2x()

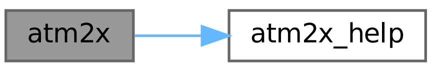
```
size_t atm2x (
            const ctl_t * ctl,
            const atm_t * atm,
            gsl_vector * x,
            int * iqa,
            int * ipa )
```

Compose state vector or parameter vector.

Definition at line 29 of file jurassic.c.

```
                {

  size_t n = 0;

  /* Add pressure... */
  for (int ip = 0; ip < atm->np; ip++)
    if (atm->z[ip] >= ctl->retp_zmin && atm->z[ip] <= ctl->retp_zmax)
      atm2x_help(atm->p[ip], IDXP, ip, x, iqa, ipa, &n);

  /* Add temperature... */
  for (int ip = 0; ip < atm->np; ip++)
    if (atm->z[ip] >= ctl->rett_zmin && atm->z[ip] <= ctl->rett_zmax)
      atm2x_help(atm->t[ip], IDXT, ip, x, iqa, ipa, &n);

  /* Add volume mixing ratios... */
  for (int ig = 0; ig < ctl->ng; ig++)
    for (int ip = 0; ip < atm->np; ip++)
      if (atm->z[ip] >= ctl->retq_zmin[ig]
          && atm->z[ip] <= ctl->retq_zmax[ig])
        atm2x_help(atm->q[ig][ip], IDXQ(ig), ip, x, iqa, ipa, &n);

  /* Add extinction... */
  for (int iw = 0; iw < ctl->nw; iw++)
    for (int ip = 0; ip < atm->np; ip++)
      if (atm->z[ip] >= ctl->retk_zmin[iw]
          && atm->z[ip] <= ctl->retk_zmax[iw])
        atm2x_help(atm->k[iw][ip], IDXK(iw), ip, x, iqa, ipa, &n);

  /* Add cloud parameters... */
  if (ctl->ret_clz)
    atm2x_help(atm->clz, IDXCLZ, 0, x, iqa, ipa, &n);
  if (ctl->ret_cldz)
    atm2x_help(atm->cldz, IDXCLDZ, 0, x, iqa, ipa, &n);
  if (ctl->ret_clk)
    for (int icl = 0; icl < ctl->ncl; icl++)
      atm2x_help(atm->clk[icl], IDXCLK(icl), 0, x, iqa, ipa, &n);

  /* Add surface parameters... */
  if (ctl->ret_sfz)
    atm2x_help(atm->sfz, IDXSFZ, 0, x, iqa, ipa, &n);
  if (ctl->ret_sfp)
    atm2x_help(atm->sfp, IDXSFP, 0, x, iqa, ipa, &n);
  if (ctl->ret_sft)
    atm2x_help(atm->sft, IDXSFT, 0, x, iqa, ipa, &n);
  if (ctl->ret_sfeps)
    for (int isf = 0; isf < ctl->nsf; isf++)
      atm2x_help(atm->sfeps[isf], IDXSFEPS(isf), 0, x, iqa, ipa, &n);

  return n;
}
```

Here is the call graph for this function:



atm2x_help()

```
void atm2x_help (
            const double value,
            const int value_iqa,
            const int value_ip,
            gsl_vector * x,
            int * iqa,
            int * ipa,
            size_t * n )
```

Add element to state vector.

Definition at line 87 of file jurassic.c.

```
                {

  /* Add element to state vector... */
  if (x != NULL)
    gsl_vector_set(x, *n, value);
  if (iqa != NULL)
    iqa[*n] = value_iqa;
  if (ipa != NULL)
    ipa[*n] = value_ip;
  (*n)++;
}
```

cart2geo()

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

Convert Cartesian coordinates to geolocation.

Definition at line 108 of file jurassic.c.

```
                  {

  const double radius = NORM(x);

  *lat = RAD2DEG(asin(x[2] / radius));
  *lon = RAD2DEG(atan2(x[1], x[0]));
  *z = radius - RE;
}
```

climatology()

```
void climatology (
            const ctl_t * ctl,
            atm_t * atm_mean )
```

Interpolate climatological data.

Definition at line 123 of file jurassic.c.

```
                 {

  static double z[121] = {
    0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19,
    20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37,
    38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55,
    56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73,
    74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91,
    92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107,
```

```
  108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 120
};

static double pre[121] = {
  1017, 901.083, 796.45, 702.227, 617.614, 541.644, 473.437, 412.288,
  357.603, 308.96, 265.994, 228.348, 195.619, 167.351, 143.039, 122.198,
  104.369, 89.141, 76.1528, 65.0804, 55.641, 47.591, 40.7233, 34.8637,
  29.8633, 25.5956, 21.9534, 18.8445, 16.1909, 13.9258, 11.9913,
  10.34, 8.92988, 7.72454, 6.6924, 5.80701, 5.04654, 4.39238, 3.82902,
  3.34337, 2.92413, 2.56128, 2.2464, 1.97258, 1.73384, 1.52519, 1.34242,
  1.18197, 1.04086, 0.916546, 0.806832, 0.709875, 0.624101, 0.548176,
  0.480974, 0.421507, 0.368904, 0.322408, 0.281386, 0.245249, 0.213465,
  0.185549, 0.161072, 0.139644, 0.120913, 0.104568, 0.0903249, 0.0779269,
  0.0671493, 0.0577962, 0.0496902, 0.0426736, 0.0366093, 0.0313743,
  0.0268598, 0.0229699, 0.0196206, 0.0167399, 0.0142646, 0.0121397,
  0.0103181, 0.00875775, 0.00742226, 0.00628076, 0.00530519, 0.00447183,
  0.00376124, 0.00315632, 0.00264248, 0.00220738, 0.00184003, 0.00153095,
  0.00127204, 0.00105608, 0.000876652, 0.00072798, 0.00060492,
  0.000503201, 0.000419226, 0.000349896, 0.000292659, 0.000245421,
  0.000206394, 0.000174125, 0.000147441, 0.000125333, 0.000106985,
  9.173e-05, 7.90172e-05, 6.84172e-05, 5.95574e-05, 5.21183e-05,
  4.58348e-05, 4.05127e-05, 3.59987e-05, 3.21583e-05, 2.88718e-05,
  2.60322e-05, 2.35687e-05, 2.14263e-05, 1.95489e-05
};

static double tem[121] = {
  285.14, 279.34, 273.91, 268.3, 263.24, 256.55, 250.2, 242.82, 236.17,
  229.87, 225.04, 221.19, 218.85, 217.19, 216.2, 215.68, 215.42, 215.55,
  215.92, 216.4, 216.93, 217.45, 218, 218.68, 219.39, 220.25, 221.3,
  222.41, 223.88, 225.42, 227.2, 229.52, 231.89, 234.51, 236.85, 239.42,
  241.94, 244.57, 247.36, 250.32, 253.34, 255.82, 258.27, 260.39,
  262.03, 263.45, 264.2, 264.78, 264.67, 264.38, 263.24, 262.03, 260.02,
  258.09, 255.63, 253.28, 250.43, 247.81, 245.26, 242.77, 240.38,
  237.94, 235.79, 233.53, 231.5, 229.53, 227.6, 225.62, 223.77, 222.06,
  220.33, 218.69, 217.18, 215.64, 214.13, 212.52, 210.86, 209.25,
  207.49, 205.81, 204.11, 202.22, 200.32, 198.39, 195.92, 193.46,
  190.94, 188.31, 185.82, 183.57, 181.43, 179.74, 178.64, 178.1, 178.25,
  178.7, 179.41, 180.67, 182.31, 184.18, 186.6, 189.53, 192.66, 196.54,
  201.13, 205.93, 211.73, 217.86, 225, 233.53, 242.57, 252.14, 261.48,
  272.97, 285.26, 299.12, 312.2, 324.17, 338.34, 352.56, 365.28
};

static double c2h2[121] = {
  1.352e-09, 2.83e-10, 1.269e-10, 6.926e-11, 4.346e-11, 2.909e-11,
  2.014e-11, 1.363e-11, 8.71e-12, 5.237e-12, 2.718e-12, 1.375e-12,
  5.786e-13, 2.16e-13, 7.317e-14, 2.551e-14, 1.055e-14, 4.758e-15,
  2.056e-15, 7.703e-16, 2.82e-16, 1.035e-16, 4.382e-17, 1.946e-17,
  9.638e-18, 5.2e-18, 2.811e-18, 1.494e-18, 7.925e-19, 4.213e-19,
  1.998e-19, 8.78e-20, 3.877e-20, 1.728e-20, 7.743e-21, 3.536e-21,
  1.623e-21, 7.508e-22, 3.508e-22, 1.65e-22, 7.837e-23, 3.733e-23,
  1.808e-23, 8.77e-24, 4.285e-24, 2.095e-24, 1.032e-24, 5.082e-25,
  2.506e-25, 1.236e-25, 6.088e-26, 2.996e-26, 1.465e-26, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
};

static double c2h6[121] = {
  2.667e-09, 2.02e-09, 1.658e-09, 1.404e-09, 1.234e-09, 1.109e-09,
  1.012e-09, 9.262e-10, 8.472e-10, 7.71e-10, 6.932e-10, 6.216e-10,
  5.503e-10, 4.87e-10, 4.342e-10, 3.861e-10, 3.347e-10, 2.772e-10,
  2.209e-10, 1.672e-10, 1.197e-10, 8.536e-11, 5.783e-11, 3.846e-11,
  2.495e-11, 1.592e-11, 1.017e-11, 6.327e-12, 3.895e-12, 2.403e-12,
  1.416e-12, 8.101e-13, 4.649e-13, 2.686e-13, 1.557e-13, 9.14e-14,
  5.386e-14, 3.19e-14, 1.903e-14, 1.14e-14, 6.875e-15, 4.154e-15,
  2.538e-15, 1.553e-15, 9.548e-16, 5.872e-16, 3.63e-16, 2.244e-16,
  1.388e-16, 8.587e-17, 5.308e-17, 3.279e-17, 2.017e-17, 1.238e-17,
  7.542e-18, 4.585e-18, 2.776e-18, 1.671e-18, 9.985e-19, 5.937e-19,
  3.518e-19, 2.07e-19, 1.215e-19, 7.06e-20, 4.097e-20, 2.37e-20,
  1.363e-20, 7.802e-21, 4.441e-21, 2.523e-21, 1.424e-21, 8.015e-22,
  4.497e-22, 2.505e-22, 1.391e-22, 7.691e-23, 4.238e-23, 2.331e-23,
  1.274e-23, 6.929e-24, 3.752e-24, 2.02e-24, 1.083e-24, 5.774e-25,
  3.041e-25, 1.593e-25, 8.308e-26, 4.299e-26, 2.195e-26, 1.112e-26,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0
};

static double ccl4[121] = {
  1.075e-10, 1.075e-10, 1.075e-10, 1.075e-10, 1.075e-10, 1.075e-10,
  1.075e-10, 1.075e-10, 1.075e-10, 1.06e-10, 1.024e-10, 9.69e-11,
  8.93e-11, 8.078e-11, 7.213e-11, 6.307e-11, 5.383e-11, 4.49e-11,
  3.609e-11, 2.705e-11, 1.935e-11, 1.385e-11, 8.35e-12, 5.485e-12,
  3.853e-12, 2.22e-12, 5.875e-13, 3.445e-13, 1.015e-13, 6.075e-14,
  4.383e-14, 2.692e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14,
  1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14,
  1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14,
  1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14,
```

```
  1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14,
  1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14,
  1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14,
  1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14,
  1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14, 1e-14,
  1e-14, 1e-14, 1e-14
};

static double ch4[121] = {
  1.864e-06, 1.835e-06, 1.819e-06, 1.805e-06, 1.796e-06, 1.788e-06,
  1.782e-06, 1.776e-06, 1.769e-06, 1.761e-06, 1.749e-06, 1.734e-06,
  1.716e-06, 1.692e-06, 1.654e-06, 1.61e-06, 1.567e-06, 1.502e-06,
  1.433e-06, 1.371e-06, 1.323e-06, 1.277e-06, 1.232e-06, 1.188e-06,
  1.147e-06, 1.108e-06, 1.07e-06, 1.027e-06, 9.854e-07, 9.416e-07,
  8.933e-07, 8.478e-07, 7.988e-07, 7.515e-07, 7.07e-07, 6.64e-07,
  6.239e-07, 5.864e-07, 5.512e-07, 5.184e-07, 4.87e-07, 4.571e-07,
  4.296e-07, 4.04e-07, 3.802e-07, 3.578e-07, 3.383e-07, 3.203e-07,
  3.032e-07, 2.889e-07, 2.76e-07, 2.635e-07, 2.519e-07, 2.409e-07,
  2.302e-07, 2.219e-07, 2.144e-07, 2.071e-07, 1.999e-07, 1.93e-07,
  1.862e-07, 1.795e-07, 1.731e-07, 1.668e-07, 1.607e-07, 1.548e-07,
  1.49e-07, 1.434e-07, 1.38e-07, 1.328e-07, 1.277e-07, 1.227e-07,
  1.18e-07, 1.134e-07, 1.089e-07, 1.046e-07, 1.004e-07, 9.635e-08,
  9.245e-08, 8.867e-08, 8.502e-08, 8.15e-08, 7.809e-08, 7.48e-08,
  7.159e-08, 6.849e-08, 6.55e-08, 6.262e-08, 5.98e-08, 5.708e-08,
  5.448e-08, 5.194e-08, 4.951e-08, 4.72e-08, 4.5e-08, 4.291e-08,
  4.093e-08, 3.905e-08, 3.729e-08, 3.563e-08, 3.408e-08, 3.265e-08,
  3.128e-08, 2.996e-08, 2.87e-08, 2.76e-08, 2.657e-08, 2.558e-08,
  2.467e-08, 2.385e-08, 2.307e-08, 2.234e-08, 2.168e-08, 2.108e-08,
  2.05e-08, 1.998e-08, 1.947e-08, 1.902e-08, 1.86e-08, 1.819e-08,
  1.782e-08
};

static double clo[121] = {
  7.419e-15, 1.061e-14, 1.518e-14, 2.195e-14, 3.175e-14, 4.666e-14,
  6.872e-14, 1.03e-13, 1.553e-13, 2.375e-13, 3.664e-13, 5.684e-13,
  8.915e-13, 1.402e-12, 2.269e-12, 4.125e-12, 7.501e-12, 1.257e-11,
  2.048e-11, 3.338e-11, 5.44e-11, 8.846e-11, 1.008e-10, 1.082e-10,
  1.157e-10, 1.232e-10, 1.312e-10, 1.539e-10, 1.822e-10, 2.118e-10,
  2.387e-10, 2.687e-10, 2.875e-10, 3.031e-10, 3.23e-10, 3.648e-10,
  4.117e-10, 4.477e-10, 4.633e-10, 4.794e-10, 4.95e-10, 5.104e-10,
  5.259e-10, 5.062e-10, 4.742e-10, 4.443e-10, 4.051e-10, 3.659e-10,
  3.305e-10, 2.911e-10, 2.54e-10, 2.215e-10, 1.927e-10, 1.675e-10,
  1.452e-10, 1.259e-10, 1.09e-10, 9.416e-11, 8.119e-11, 6.991e-11,
  6.015e-11, 5.163e-11, 4.43e-11, 3.789e-11, 3.24e-11, 2.769e-11,
  2.361e-11, 2.011e-11, 1.71e-11, 1.453e-11, 1.233e-11, 1.045e-11,
  8.851e-12, 7.48e-12, 6.316e-12, 5.326e-12, 4.487e-12, 3.778e-12,
  3.176e-12, 2.665e-12, 2.234e-12, 1.87e-12, 1.563e-12, 1.304e-12,
  1.085e-12, 9.007e-13, 7.468e-13, 6.179e-13, 5.092e-13, 4.188e-13,
  3.442e-13, 2.816e-13, 2.304e-13, 1.885e-13, 1.542e-13, 1.263e-13,
  1.035e-13, 8.5e-14, 7.004e-14, 5.783e-14, 4.795e-14, 4.007e-14,
  3.345e-14, 2.792e-14, 2.33e-14, 1.978e-14, 1.686e-14, 1.438e-14,
  1.234e-14, 1.07e-14, 9.312e-15, 8.131e-15, 7.164e-15, 6.367e-15,
  5.67e-15, 5.088e-15, 4.565e-15, 4.138e-15, 3.769e-15, 3.432e-15,
  3.148e-15
};

static double clono2[121] = {
  1.011e-13, 1.515e-13, 2.272e-13, 3.446e-13, 5.231e-13, 8.085e-13,
  1.253e-12, 1.979e-12, 3.149e-12, 5.092e-12, 8.312e-12, 1.366e-11,
  2.272e-11, 3.791e-11, 6.209e-11, 9.101e-11, 1.334e-10, 1.951e-10,
  2.853e-10, 3.94e-10, 4.771e-10, 5.771e-10, 6.675e-10, 7.665e-10,
  8.504e-10, 8.924e-10, 9.363e-10, 8.923e-10, 8.411e-10, 7.646e-10,
  6.525e-10, 5.576e-10, 4.398e-10, 3.403e-10, 2.612e-10, 1.915e-10,
  1.407e-10, 1.028e-10, 7.455e-11, 5.42e-11, 3.708e-11, 2.438e-11,
  1.618e-11, 1.075e-11, 7.17e-12, 4.784e-12, 3.205e-12, 2.147e-12,
  1.44e-12, 9.654e-13, 6.469e-13, 4.332e-13, 2.891e-13, 1.926e-13,
  1.274e-13, 8.422e-14, 5.547e-14, 3.636e-14, 2.368e-14, 1.536e-14,
  9.937e-15, 6.39e-15, 4.101e-15, 2.61e-15, 1.659e-15, 1.052e-15,
  6.638e-16, 4.172e-16, 2.61e-16, 1.63e-16, 1.013e-16, 6.275e-17,
  3.879e-17, 2.383e-17, 1.461e-17, 8.918e-18, 5.43e-18, 3.301e-18,
  1.997e-18, 1.203e-18, 7.216e-19, 4.311e-19, 2.564e-19, 1.519e-19,
  8.911e-20, 5.203e-20, 3.026e-20, 1.748e-20, 9.99e-21, 5.673e-21,
  3.215e-21, 1.799e-21, 1.006e-21, 5.628e-22, 3.146e-22, 1.766e-22,
  9.94e-23, 5.614e-23, 3.206e-23, 1.841e-23, 1.071e-23, 6.366e-24,
  3.776e-24, 2.238e-24, 1.326e-24, 8.253e-25, 5.201e-25, 3.279e-25,
  2.108e-25, 1.395e-25, 9.326e-26, 6.299e-26, 4.365e-26, 3.104e-26,
  2.219e-26, 1.621e-26, 1.185e-26, 8.92e-27, 6.804e-27, 5.191e-27,
  4.041e-27
};

static double co[121] = {
  1.907e-07, 1.553e-07, 1.362e-07, 1.216e-07, 1.114e-07, 1.036e-07,
  9.737e-08, 9.152e-08, 8.559e-08, 7.966e-08, 7.277e-08, 6.615e-08,
  5.884e-08, 5.22e-08, 4.699e-08, 4.284e-08, 3.776e-08, 3.274e-08,
  2.845e-08, 2.479e-08, 2.246e-08, 2.054e-08, 1.991e-08, 1.951e-08,
  1.94e-08, 2.009e-08, 2.1e-08, 2.201e-08, 2.322e-08, 2.45e-08,
  2.602e-08, 2.73e-08, 2.867e-08, 2.998e-08, 3.135e-08, 3.255e-08,
```

```
  3.352e-08, 3.426e-08, 3.484e-08, 3.53e-08, 3.593e-08, 3.671e-08,
  3.759e-08, 3.945e-08, 4.192e-08, 4.49e-08, 5.03e-08, 5.703e-08,
  6.538e-08, 7.878e-08, 9.644e-08, 1.196e-07, 1.498e-07, 1.904e-07,
  2.422e-07, 3.055e-07, 3.804e-07, 4.747e-07, 5.899e-07, 7.272e-07,
  8.91e-07, 1.071e-06, 1.296e-06, 1.546e-06, 1.823e-06, 2.135e-06,
  2.44e-06, 2.714e-06, 2.967e-06, 3.189e-06, 3.391e-06, 3.58e-06,
  3.773e-06, 4.022e-06, 4.346e-06, 4.749e-06, 5.199e-06, 5.668e-06,
  6.157e-06, 6.688e-06, 7.254e-06, 7.867e-06, 8.539e-06, 9.26e-06,
  1.009e-05, 1.119e-05, 1.228e-05, 1.365e-05, 1.506e-05, 1.641e-05,
  1.784e-05, 1.952e-05, 2.132e-05, 2.323e-05, 2.531e-05, 2.754e-05,
  3.047e-05, 3.459e-05, 3.922e-05, 4.439e-05, 4.825e-05, 5.077e-05,
  5.34e-05, 5.618e-05, 5.909e-05, 6.207e-05, 6.519e-05, 6.845e-05,
  6.819e-05, 6.726e-05, 6.622e-05, 6.512e-05, 6.671e-05, 6.862e-05,
  7.048e-05, 7.264e-05, 7.3e-05, 7.3e-05, 7.3e-05, 7.3e-05, 7.3e-05
};

static double cof2[121] = {
  7.5e-14, 1.055e-13, 1.485e-13, 2.111e-13, 3.001e-13, 4.333e-13,
  6.269e-13, 9.221e-13, 1.364e-12, 2.046e-12, 3.093e-12, 4.703e-12,
  7.225e-12, 1.113e-11, 1.66e-11, 2.088e-11, 2.626e-11, 3.433e-11,
  4.549e-11, 5.886e-11, 7.21e-11, 8.824e-11, 1.015e-10, 1.155e-10,
  1.288e-10, 1.388e-10, 1.497e-10, 1.554e-10, 1.606e-10, 1.639e-10,
  1.64e-10, 1.64e-10, 1.596e-10, 1.542e-10, 1.482e-10, 1.382e-10,
  1.289e-10, 1.198e-10, 1.109e-10, 1.026e-10, 9.484e-11, 8.75e-11,
  8.086e-11, 7.49e-11, 6.948e-11, 6.446e-11, 5.961e-11, 5.505e-11,
  5.085e-11, 4.586e-11, 4.1e-11, 3.665e-11, 3.235e-11, 2.842e-11,
  2.491e-11, 2.11e-11, 1.769e-11, 1.479e-11, 1.197e-11, 9.631e-12,
  7.74e-12, 6.201e-12, 4.963e-12, 3.956e-12, 3.151e-12, 2.507e-12,
  1.99e-12, 1.576e-12, 1.245e-12, 9.83e-13, 7.742e-13, 6.088e-13,
  4.782e-13, 3.745e-13, 2.929e-13, 2.286e-13, 1.782e-13, 1.388e-13,
  1.079e-13, 8.362e-14, 6.471e-14, 4.996e-14, 3.85e-14, 2.96e-14,
  2.265e-14, 1.729e-14, 1.317e-14, 9.998e-15, 7.549e-15, 5.683e-15,
  4.273e-15, 3.193e-15, 2.385e-15, 1.782e-15, 1.331e-15, 9.957e-16,
  7.461e-16, 5.601e-16, 4.228e-16, 3.201e-16, 2.438e-16, 1.878e-16,
  1.445e-16, 1.111e-16, 8.544e-17, 6.734e-17, 5.341e-17, 4.237e-17,
  3.394e-17, 2.759e-17, 2.254e-17, 1.851e-17, 1.54e-17, 1.297e-17,
  1.096e-17, 9.365e-18, 8e-18, 6.938e-18, 6.056e-18, 5.287e-18,
  4.662e-18
};

static double f11[121] = {
  2.65e-10, 2.65e-10, 2.65e-10, 2.65e-10, 2.65e-10, 2.65e-10, 2.65e-10,
  2.65e-10, 2.65e-10, 2.65e-10, 2.65e-10, 2.65e-10, 2.635e-10, 2.536e-10,
  2.44e-10, 2.348e-10, 2.258e-10, 2.153e-10, 2.046e-10, 1.929e-10,
  1.782e-10, 1.648e-10, 1.463e-10, 1.291e-10, 1.1e-10, 8.874e-11,
  7.165e-11, 5.201e-11, 3.744e-11, 2.577e-11, 1.64e-11, 1.048e-11,
  5.993e-12, 3.345e-12, 1.839e-12, 9.264e-13, 4.688e-13, 2.329e-13,
  1.129e-13, 5.505e-14, 2.825e-14, 1.492e-14, 7.997e-15, 5.384e-15,
  3.988e-15, 2.955e-15, 2.196e-15, 1.632e-15, 1.214e-15, 9.025e-16,
  6.708e-16, 4.984e-16, 3.693e-16, 2.733e-16, 2.013e-16, 1.481e-16,
  1.087e-16, 7.945e-17, 5.782e-17, 4.195e-17, 3.038e-17, 2.19e-17,
  1.577e-17, 1.128e-17, 8.063e-18, 5.753e-18, 4.09e-18, 2.899e-18,
  2.048e-18, 1.444e-18, 1.015e-18, 7.12e-19, 4.985e-19, 3.474e-19,
  2.417e-19, 1.677e-19, 1.161e-19, 8.029e-20, 5.533e-20, 3.799e-20,
  2.602e-20, 1.776e-20, 1.209e-20, 8.202e-21, 5.522e-21, 3.707e-21,
  2.48e-21, 1.652e-21, 1.091e-21, 7.174e-22, 4.709e-22, 3.063e-22,
  1.991e-22, 1.294e-22, 8.412e-23, 5.483e-23, 3.581e-23, 2.345e-23,
  1.548e-23, 1.027e-23, 6.869e-24, 4.673e-24, 3.173e-24, 2.153e-24,
  1.461e-24, 1.028e-24, 7.302e-25, 5.188e-25, 3.739e-25, 2.753e-25,
  2.043e-25, 1.528e-25, 1.164e-25, 9.041e-26, 7.051e-26, 5.587e-26,
  4.428e-26, 3.588e-26, 2.936e-26, 2.402e-26, 1.995e-26
};

static double f12[121] = {
  5.45e-10, 5.45e-10, 5.45e-10, 5.45e-10, 5.45e-10, 5.45e-10, 5.45e-10,
  5.45e-10, 5.45e-10, 5.45e-10, 5.45e-10, 5.45e-10, 5.429e-10, 5.291e-10,
  5.155e-10, 5.022e-10, 4.893e-10, 4.772e-10, 4.655e-10, 4.497e-10,
  4.249e-10, 4.015e-10, 3.632e-10, 3.261e-10, 2.858e-10, 2.408e-10,
  2.03e-10, 1.685e-10, 1.4e-10, 1.163e-10, 9.65e-11, 8.02e-11, 6.705e-11,
  5.624e-11, 4.764e-11, 4.249e-11, 3.792e-11, 3.315e-11, 2.819e-11,
  2.4e-11, 1.999e-11, 1.64e-11, 1.352e-11, 1.14e-11, 9.714e-12,
  8.28e-12, 7.176e-12, 6.251e-12, 5.446e-12, 4.72e-12, 4.081e-12,
  3.528e-12, 3.08e-12, 2.699e-12, 2.359e-12, 2.111e-12, 1.901e-12,
  1.709e-12, 1.534e-12, 1.376e-12, 1.233e-12, 1.103e-12, 9.869e-13,
  8.808e-13, 7.859e-13, 7.008e-13, 6.241e-13, 5.553e-13, 4.935e-13,
  4.383e-13, 3.889e-13, 3.447e-13, 3.054e-13, 2.702e-13, 2.389e-13,
  2.11e-13, 1.862e-13, 1.643e-13, 1.448e-13, 1.274e-13, 1.121e-13,
  9.844e-14, 8.638e-14, 7.572e-14, 6.62e-14, 5.782e-14, 5.045e-14,
  4.394e-14, 3.817e-14, 3.311e-14, 2.87e-14, 2.48e-14, 2.142e-14,
  1.851e-14, 1.599e-14, 1.383e-14, 1.196e-14, 1.036e-14, 9e-15,
  7.828e-15, 6.829e-15, 5.992e-15, 5.254e-15, 4.606e-15, 4.037e-15,
  3.583e-15, 3.19e-15, 2.841e-15, 2.542e-15, 2.291e-15, 2.07e-15,
  1.875e-15, 1.71e-15, 1.57e-15, 1.442e-15, 1.333e-15, 1.232e-15,
  1.147e-15, 1.071e-15, 1.001e-15, 9.396e-16
};

static double f14[121] = {
```

```
  9e-11, 9e-11, 9e-11, 9e-11, 9e-11, 9e-11, 9e-11, 9e-11, 9e-11, 9e-11,
  9e-11, 9e-11, 9e-11, 9e-11, 9e-11, 8.91e-11, 8.73e-11, 8.46e-11,
  8.19e-11, 7.92e-11, 7.74e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11,
  7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11,
  7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11,
  7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11,
  7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11,
  7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11,
  7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11,
  7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11,
  7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11,
  7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11,
  7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11,
  7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11,
  7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11,
  7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11,
  7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11, 7.65e-11
};

static double f22[121] = {
  1.4e-10, 1.4e-10, 1.4e-10, 1.4e-10, 1.4e-10, 1.4e-10, 1.4e-10,
  1.4e-10, 1.4e-10, 1.4e-10, 1.372e-10, 1.317e-10, 1.235e-10, 1.153e-10,
  1.075e-10, 1.002e-10, 9.332e-11, 8.738e-11, 8.194e-11, 7.7e-11,
  7.165e-11, 6.753e-11, 6.341e-11, 5.971e-11, 5.6e-11, 5.229e-11,
  4.859e-11, 4.488e-11, 4.118e-11, 3.83e-11, 3.568e-11, 3.308e-11,
  3.047e-11, 2.82e-11, 2.594e-11, 2.409e-11, 2.237e-11, 2.065e-11,
  1.894e-11, 1.771e-11, 1.647e-11, 1.532e-11, 1.416e-11, 1.332e-11,
  1.246e-11, 1.161e-11, 1.087e-11, 1.017e-11, 9.471e-12, 8.853e-12,
  8.235e-12, 7.741e-12, 7.247e-12, 6.836e-12, 6.506e-12, 6.176e-12,
  5.913e-12, 5.65e-12, 5.419e-12, 5.221e-12, 5.024e-12, 4.859e-12,
  4.694e-12, 4.546e-12, 4.414e-12, 4.282e-12, 4.15e-12, 4.019e-12,
  3.903e-12, 3.805e-12, 3.706e-12, 3.607e-12, 3.508e-12, 3.41e-12,
  3.31e-12, 3.212e-12, 3.129e-12, 3.047e-12, 2.964e-12, 2.882e-12,
  2.8e-12, 2.734e-12, 2.668e-12, 2.602e-12, 2.537e-12, 2.471e-12,
  2.421e-12, 2.372e-12, 2.322e-12, 2.273e-12, 2.224e-12, 2.182e-12,
  2.141e-12, 2.1e-12, 2.059e-12, 2.018e-12, 1.977e-12, 1.935e-12,
  1.894e-12, 1.853e-12, 1.812e-12, 1.77e-12, 1.73e-12, 1.688e-12,
  1.647e-12, 1.606e-12, 1.565e-12, 1.524e-12, 1.483e-12, 1.441e-12,
  1.4e-12, 1.359e-12, 1.317e-12, 1.276e-12, 1.235e-12, 1.194e-12,
  1.153e-12, 1.112e-12, 1.071e-12, 1.029e-12, 9.883e-13
};

static double h2o[121] = {
  0.01166, 0.008269, 0.005742, 0.003845, 0.00277, 0.001897, 0.001272,
  0.000827, 0.000539, 0.0003469, 0.0001579, 3.134e-05, 1.341e-05,
  6.764e-06, 4.498e-06, 3.703e-06, 3.724e-06, 3.899e-06, 4.002e-06,
  4.122e-06, 4.277e-06, 4.438e-06, 4.558e-06, 4.673e-06, 4.763e-06,
  4.809e-06, 4.856e-06, 4.936e-06, 5.021e-06, 5.114e-06, 5.222e-06,
  5.331e-06, 5.414e-06, 5.488e-06, 5.563e-06, 5.633e-06, 5.704e-06,
  5.767e-06, 5.819e-06, 5.872e-06, 5.914e-06, 5.949e-06, 5.984e-06,
  6.015e-06, 6.044e-06, 6.073e-06, 6.104e-06, 6.136e-06, 6.167e-06,
  6.189e-06, 6.208e-06, 6.226e-06, 6.212e-06, 6.185e-06, 6.158e-06,
  6.114e-06, 6.066e-06, 6.018e-06, 5.877e-06, 5.728e-06, 5.582e-06,
  5.437e-06, 5.296e-06, 5.156e-06, 5.02e-06, 4.886e-06, 4.754e-06,
  4.625e-06, 4.498e-06, 4.374e-06, 4.242e-06, 4.096e-06, 3.955e-06,
  3.817e-06, 3.683e-06, 3.491e-06, 3.204e-06, 2.94e-06, 2.696e-06,
  2.47e-06, 2.252e-06, 2.019e-06, 1.808e-06, 1.618e-06, 1.445e-06,
  1.285e-06, 1.105e-06, 9.489e-07, 8.121e-07, 6.938e-07, 5.924e-07,
  5.04e-07, 4.288e-07, 3.648e-07, 3.103e-07, 2.642e-07, 2.252e-07,
  1.921e-07, 1.643e-07, 1.408e-07, 1.211e-07, 1.048e-07, 9.063e-08,
  7.835e-08, 6.774e-08, 5.936e-08, 5.221e-08, 4.592e-08, 4.061e-08,
  3.62e-08, 3.236e-08, 2.902e-08, 2.62e-08, 2.383e-08, 2.171e-08,
  1.989e-08, 1.823e-08, 1.684e-08, 1.562e-08, 1.449e-08, 1.351e-08
};

static double h2o2[121] = {
  1.779e-10, 7.938e-10, 8.953e-10, 8.032e-10, 6.564e-10, 5.159e-10,
  4.003e-10, 3.026e-10, 2.222e-10, 1.58e-10, 1.044e-10, 6.605e-11,
  3.413e-11, 1.453e-11, 1.062e-11, 1.009e-11, 9.597e-12, 1.175e-11,
  1.572e-11, 2.091e-11, 2.746e-11, 3.603e-11, 4.791e-11, 6.387e-11,
  8.239e-11, 1.007e-10, 1.23e-10, 1.363e-10, 1.489e-10, 1.585e-10,
  1.608e-10, 1.632e-10, 1.576e-10, 1.502e-10, 1.423e-10, 1.302e-10,
  1.192e-10, 1.085e-10, 9.795e-11, 8.854e-11, 8.057e-11, 7.36e-11,
  6.736e-11, 6.362e-11, 6.087e-11, 5.825e-11, 5.623e-11, 5.443e-11,
  5.27e-11, 5.098e-11, 4.931e-11, 4.769e-11, 4.611e-11, 4.458e-11,
  4.308e-11, 4.102e-11, 3.887e-11, 3.682e-11, 3.521e-11, 3.369e-11,
  3.224e-11, 3.082e-11, 2.946e-11, 2.814e-11, 2.687e-11, 2.566e-11,
  2.449e-11, 2.336e-11, 2.227e-11, 2.123e-11, 2.023e-11, 1.927e-11,
  1.835e-11, 1.746e-11, 1.661e-11, 1.58e-11, 1.502e-11, 1.428e-11,
  1.357e-11, 1.289e-11, 1.224e-11, 1.161e-11, 1.102e-11, 1.045e-11,
  9.895e-12, 9.369e-12, 8.866e-12, 8.386e-12, 7.922e-12, 7.479e-12,
  7.06e-12, 6.656e-12, 6.274e-12, 5.914e-12, 5.575e-12, 5.257e-12,
  4.959e-12, 4.679e-12, 4.42e-12, 4.178e-12, 3.954e-12, 3.75e-12,
  3.557e-12, 3.372e-12, 3.198e-12, 3.047e-12, 2.908e-12, 2.775e-12,
  2.653e-12, 2.544e-12, 2.442e-12, 2.346e-12, 2.26e-12, 2.183e-12,
  2.11e-12, 2.044e-12, 1.98e-12, 1.924e-12, 1.871e-12, 1.821e-12,
  1.775e-12
```

```
};

static double hcn[121] = {
  5.5e-10, 5.5e-10, 5.5e-10, 5.5e-10, 5.5e-10, 5.5e-10, 5.5e-10,
  5.5e-10, 5.5e-10, 5.5e-10, 5.5e-10, 5.498e-10, 5.495e-10, 5.493e-10,
  5.49e-10, 5.488e-10, 4.717e-10, 3.946e-10, 3.174e-10, 2.4e-10,
  1.626e-10, 1.619e-10, 1.612e-10, 1.602e-10, 1.593e-10, 1.582e-10,
  1.572e-10, 1.56e-10, 1.549e-10, 1.539e-10, 1.53e-10, 1.519e-10,
  1.506e-10, 1.487e-10, 1.467e-10, 1.449e-10, 1.43e-10, 1.413e-10,
  1.397e-10, 1.382e-10, 1.368e-10, 1.354e-10, 1.337e-10, 1.315e-10,
  1.292e-10, 1.267e-10, 1.241e-10, 1.215e-10, 1.19e-10, 1.165e-10,
  1.141e-10, 1.118e-10, 1.096e-10, 1.072e-10, 1.047e-10, 1.021e-10,
  9.968e-11, 9.739e-11, 9.539e-11, 9.339e-11, 9.135e-11, 8.898e-11,
  8.664e-11, 8.439e-11, 8.249e-11, 8.075e-11, 7.904e-11, 7.735e-11,
  7.565e-11, 7.399e-11, 7.245e-11, 7.109e-11, 6.982e-11, 6.863e-11,
  6.755e-11, 6.657e-11, 6.587e-11, 6.527e-11, 6.476e-11, 6.428e-11,
  6.382e-11, 6.343e-11, 6.307e-11, 6.272e-11, 6.238e-11, 6.205e-11,
  6.17e-11, 6.137e-11, 6.102e-11, 6.072e-11, 6.046e-11, 6.03e-11,
  6.018e-11, 6.01e-11, 6.001e-11, 5.992e-11, 5.984e-11, 5.975e-11,
  5.967e-11, 5.958e-11, 5.95e-11, 5.941e-11, 5.933e-11, 5.925e-11,
  5.916e-11, 5.908e-11, 5.899e-11, 5.891e-11, 5.883e-11, 5.874e-11,
  5.866e-11, 5.858e-11, 5.85e-11, 5.841e-11, 5.833e-11, 5.825e-11,
  5.817e-11, 5.808e-11, 5.8e-11, 5.792e-11, 5.784e-11
};

static double hno3[121] = {
  1.809e-10, 7.234e-10, 5.899e-10, 4.342e-10, 3.277e-10, 2.661e-10,
  2.35e-10, 2.267e-10, 2.389e-10, 2.651e-10, 3.255e-10, 4.099e-10,
  5.42e-10, 6.978e-10, 8.807e-10, 1.112e-09, 1.405e-09, 2.04e-09,
  3.111e-09, 4.5e-09, 5.762e-09, 7.37e-09, 7.852e-09, 8.109e-09,
  8.067e-09, 7.554e-09, 7.076e-09, 6.268e-09, 5.524e-09, 4.749e-09,
  3.909e-09, 3.223e-09, 2.517e-09, 1.942e-09, 1.493e-09, 1.122e-09,
  8.449e-10, 6.361e-10, 4.787e-10, 3.611e-10, 2.804e-10, 2.215e-10,
  1.758e-10, 1.441e-10, 1.197e-10, 9.953e-11, 8.505e-11, 7.334e-11,
  6.325e-11, 5.625e-11, 5.058e-11, 4.548e-11, 4.122e-11, 3.748e-11,
  3.402e-11, 3.088e-11, 2.8e-11, 2.536e-11, 2.293e-11, 2.072e-11,
  1.871e-11, 1.687e-11, 1.52e-11, 1.368e-11, 1.23e-11, 1.105e-11,
  9.922e-12, 8.898e-12, 7.972e-12, 7.139e-12, 6.385e-12, 5.708e-12,
  5.099e-12, 4.549e-12, 4.056e-12, 3.613e-12, 3.216e-12, 2.862e-12,
  2.544e-12, 2.259e-12, 2.004e-12, 1.776e-12, 1.572e-12, 1.391e-12,
  1.227e-12, 1.082e-12, 9.528e-13, 8.379e-13, 7.349e-13, 6.436e-13,
  5.634e-13, 4.917e-13, 4.291e-13, 3.745e-13, 3.267e-13, 2.854e-13,
  2.494e-13, 2.181e-13, 1.913e-13, 1.68e-13, 1.479e-13, 1.31e-13,
  1.159e-13, 1.025e-13, 9.067e-14, 8.113e-14, 7.281e-14, 6.535e-14,
  5.892e-14, 5.348e-14, 4.867e-14, 4.439e-14, 4.073e-14, 3.76e-14,
  3.476e-14, 3.229e-14, 3e-14, 2.807e-14, 2.635e-14, 2.473e-14,
  2.332e-14
};

static double hno4[121] = {
  6.118e-12, 3.594e-12, 2.807e-12, 3.04e-12, 4.458e-12, 7.986e-12,
  1.509e-11, 2.661e-11, 3.738e-11, 4.652e-11, 4.429e-11, 3.992e-11,
  3.347e-11, 3.005e-11, 3.173e-11, 4.055e-11, 5.812e-11, 8.489e-11,
  1.19e-10, 1.482e-10, 1.766e-10, 2.103e-10, 2.35e-10, 2.598e-10,
  2.801e-10, 2.899e-10, 3e-10, 2.817e-10, 2.617e-10, 2.332e-10,
  1.933e-10, 1.605e-10, 1.232e-10, 9.285e-11, 6.941e-11, 4.951e-11,
  3.539e-11, 2.402e-11, 1.522e-11, 9.676e-12, 6.056e-12, 3.745e-12,
  2.34e-12, 1.463e-12, 9.186e-13, 5.769e-13, 3.322e-13, 1.853e-13,
  1.035e-13, 7.173e-14, 5.382e-14, 4.036e-14, 3.401e-14, 2.997e-14,
  2.635e-14, 2.316e-14, 2.034e-14, 1.783e-14, 1.56e-14, 1.363e-14,
  1.19e-14, 1.037e-14, 9.032e-15, 7.846e-15, 6.813e-15, 5.912e-15,
  5.121e-15, 4.431e-15, 3.829e-15, 3.306e-15, 2.851e-15, 2.456e-15,
  2.114e-15, 1.816e-15, 1.559e-15, 1.337e-15, 1.146e-15, 9.811e-16,
  8.389e-16, 7.162e-16, 6.109e-16, 5.203e-16, 4.425e-16, 3.76e-16,
  3.184e-16, 2.692e-16, 2.274e-16, 1.917e-16, 1.61e-16, 1.35e-16,
  1.131e-16, 9.437e-17, 7.874e-17, 6.57e-17, 5.481e-17, 4.579e-17,
  3.828e-17, 3.204e-17, 2.691e-17, 2.264e-17, 1.912e-17, 1.626e-17,
  1.382e-17, 1.174e-17, 9.972e-18, 8.603e-18, 7.45e-18, 6.453e-18,
  5.623e-18, 4.944e-18, 4.361e-18, 3.859e-18, 3.443e-18, 3.096e-18,
  2.788e-18, 2.528e-18, 2.293e-18, 2.099e-18, 1.929e-18, 1.773e-18,
  1.64e-18
};

static double hocl[121] = {
  1.056e-12, 1.194e-12, 1.35e-12, 1.531e-12, 1.737e-12, 1.982e-12,
  2.263e-12, 2.599e-12, 2.991e-12, 3.459e-12, 4.012e-12, 4.662e-12,
  5.438e-12, 6.35e-12, 7.425e-12, 8.686e-12, 1.016e-11, 1.188e-11,
  1.389e-11, 1.659e-11, 2.087e-11, 2.621e-11, 3.265e-11, 4.064e-11,
  4.859e-11, 5.441e-11, 6.09e-11, 6.373e-11, 6.611e-11, 6.94e-11,
  7.44e-11, 7.97e-11, 8.775e-11, 9.722e-11, 1.064e-10, 1.089e-10,
  1.114e-10, 1.106e-10, 1.053e-10, 1.004e-10, 9.006e-11, 7.778e-11,
  6.739e-11, 5.636e-11, 4.655e-11, 3.845e-11, 3.042e-11, 2.368e-11,
  1.845e-11, 1.442e-11, 1.127e-11, 8.814e-12, 6.544e-12, 4.763e-12,
  3.449e-12, 2.612e-12, 1.999e-12, 1.526e-12, 1.16e-12, 8.793e-13,
  6.655e-13, 5.017e-13, 3.778e-13, 2.829e-13, 2.117e-13, 1.582e-13,
  1.178e-13, 8.755e-14, 6.486e-14, 4.799e-14, 3.54e-14, 2.606e-14,
  1.916e-14, 1.403e-14, 1.026e-14, 7.48e-15, 5.446e-15, 3.961e-15,
```

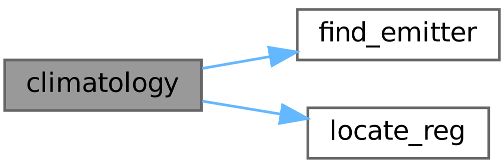
```
  2.872e-15, 2.076e-15, 1.498e-15, 1.077e-15, 7.726e-16, 5.528e-16,
  3.929e-16, 2.785e-16, 1.969e-16, 1.386e-16, 9.69e-17, 6.747e-17,
  4.692e-17, 3.236e-17, 2.232e-17, 1.539e-17, 1.061e-17, 7.332e-18,
  5.076e-18, 3.522e-18, 2.461e-18, 1.726e-18, 1.22e-18, 8.75e-19,
  6.264e-19, 4.482e-19, 3.207e-19, 2.368e-19, 1.762e-19, 1.312e-19,
  9.891e-20, 7.595e-20, 5.87e-20, 4.567e-20, 3.612e-20, 2.904e-20,
  2.343e-20, 1.917e-20, 1.568e-20, 1.308e-20, 1.1e-20, 9.25e-21,
  7.881e-21
};

static double n2o[121] = {
  3.17e-07, 3.17e-07, 3.17e-07, 3.17e-07, 3.17e-07, 3.17e-07, 3.17e-07,
  3.17e-07, 3.17e-07, 3.17e-07, 3.124e-07, 3.077e-07, 3.03e-07,
  2.984e-07, 2.938e-07, 2.892e-07, 2.847e-07, 2.779e-07, 2.705e-07,
  2.631e-07, 2.557e-07, 2.484e-07, 2.345e-07, 2.201e-07, 2.01e-07,
  1.754e-07, 1.532e-07, 1.329e-07, 1.154e-07, 1.003e-07, 8.735e-08,
  7.617e-08, 6.512e-08, 5.547e-08, 4.709e-08, 3.915e-08, 3.259e-08,
  2.738e-08, 2.327e-08, 1.98e-08, 1.711e-08, 1.493e-08, 1.306e-08,
  1.165e-08, 1.049e-08, 9.439e-09, 8.375e-09, 7.391e-09, 6.525e-09,
  5.759e-09, 5.083e-09, 4.485e-09, 3.953e-09, 3.601e-09, 3.27e-09,
  2.975e-09, 2.757e-09, 2.556e-09, 2.37e-09, 2.195e-09, 2.032e-09,
  1.912e-09, 1.79e-09, 1.679e-09, 1.572e-09, 1.482e-09, 1.402e-09,
  1.326e-09, 1.254e-09, 1.187e-09, 1.127e-09, 1.071e-09, 1.02e-09,
  9.673e-10, 9.193e-10, 8.752e-10, 8.379e-10, 8.017e-10, 7.66e-10,
  7.319e-10, 7.004e-10, 6.721e-10, 6.459e-10, 6.199e-10, 5.942e-10,
  5.703e-10, 5.488e-10, 5.283e-10, 5.082e-10, 4.877e-10, 4.696e-10,
  4.52e-10, 4.355e-10, 4.198e-10, 4.039e-10, 3.888e-10, 3.754e-10,
  3.624e-10, 3.499e-10, 3.381e-10, 3.267e-10, 3.163e-10, 3.058e-10,
  2.959e-10, 2.864e-10, 2.77e-10, 2.686e-10, 2.604e-10, 2.534e-10,
  2.462e-10, 2.386e-10, 2.318e-10, 2.247e-10, 2.189e-10, 2.133e-10,
  2.071e-10, 2.014e-10, 1.955e-10, 1.908e-10, 1.86e-10, 1.817e-10
};

static double n2o5[121] = {
  1.231e-11, 3.035e-12, 1.702e-12, 9.877e-13, 8.081e-13, 9.039e-13,
  1.169e-12, 1.474e-12, 1.651e-12, 1.795e-12, 1.998e-12, 2.543e-12,
  4.398e-12, 7.698e-12, 1.28e-11, 2.131e-11, 3.548e-11, 5.894e-11,
  7.645e-11, 1.089e-10, 1.391e-10, 1.886e-10, 2.386e-10, 2.986e-10,
  3.487e-10, 3.994e-10, 4.5e-10, 4.6e-10, 4.591e-10, 4.1e-10, 3.488e-10,
  2.846e-10, 2.287e-10, 1.696e-10, 1.011e-10, 6.428e-11, 4.324e-11,
  2.225e-11, 6.214e-12, 3.608e-12, 8.793e-13, 4.491e-13, 1.04e-13,
  6.1e-14, 3.436e-14, 6.671e-15, 1.171e-15, 5.848e-16, 1.212e-16,
  1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16,
  1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16,
  1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16,
  1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16,
  1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16,
  1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16,
  1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16, 1e-16,
  1e-16, 1e-16
};

static double nh3[121] = {
  1e-10, 1e-10, 1e-10, 1e-10, 1e-10, 1e-10, 1e-10, 1e-10, 1e-10, 1e-10,
  1e-10, 1e-10, 1e-10, 1e-10, 9.444e-11, 8.488e-11, 7.241e-11, 5.785e-11,
  4.178e-11, 3.018e-11, 2.18e-11, 1.574e-11, 1.137e-11, 8.211e-12,
  5.973e-12, 4.327e-12, 3.118e-12, 2.234e-12, 1.573e-12, 1.04e-12,
  6.762e-13, 4.202e-13, 2.406e-13, 1.335e-13, 6.938e-14, 3.105e-14,
  1.609e-14, 1.033e-14, 6.432e-15, 4.031e-15, 2.555e-15, 1.656e-15,
  1.115e-15, 7.904e-16, 5.63e-16, 4.048e-16, 2.876e-16, 2.004e-16,
  1.356e-16, 9.237e-17, 6.235e-17, 4.223e-17, 3.009e-17, 2.328e-17,
  2.002e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17,
  1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17,
  1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17,
  1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17,
  1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17,
  1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17,
  1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17,
  1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17,
  1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17,
  1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17,
  1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17, 1.914e-17,
  1.914e-17
};

static double no[121] = {
  2.586e-10, 4.143e-11, 1.566e-11, 9.591e-12, 8.088e-12, 8.462e-12,
  1.013e-11, 1.328e-11, 1.855e-11, 2.678e-11, 3.926e-11, 5.464e-11,
  7.012e-11, 8.912e-11, 1.127e-10, 1.347e-10, 1.498e-10, 1.544e-10,
  1.602e-10, 1.824e-10, 2.078e-10, 2.366e-10, 2.691e-10, 5.141e-10,
  8.259e-10, 1.254e-09, 1.849e-09, 2.473e-09, 3.294e-09, 4.16e-09,
  5.095e-09, 6.11e-09, 6.93e-09, 7.888e-09, 8.903e-09, 9.713e-09,
  1.052e-08, 1.115e-08, 1.173e-08, 1.21e-08, 1.228e-08, 1.239e-08,
  1.231e-08, 1.213e-08, 1.192e-08, 1.138e-08, 1.085e-08, 1.008e-08,
  9.224e-09, 8.389e-09, 7.262e-09, 6.278e-09, 5.335e-09, 4.388e-09,
  3.589e-09, 2.761e-09, 2.129e-09, 1.633e-09, 1.243e-09, 9.681e-10,
  8.355e-10, 7.665e-10, 7.442e-10, 8.584e-10, 9.732e-10, 1.063e-09,
```

```
  1.163e-09, 1.286e-09, 1.472e-09, 1.707e-09, 2.032e-09, 2.474e-09,
  2.977e-09, 3.506e-09, 4.102e-09, 5.013e-09, 6.493e-09, 8.414e-09,
  1.077e-08, 1.367e-08, 1.777e-08, 2.625e-08, 3.926e-08, 5.545e-08,
  7.195e-08, 9.464e-08, 1.404e-07, 2.183e-07, 3.329e-07, 4.535e-07,
  6.158e-07, 8.187e-07, 1.075e-06, 1.422e-06, 1.979e-06, 2.71e-06,
  3.58e-06, 4.573e-06, 5.951e-06, 7.999e-06, 1.072e-05, 1.372e-05,
  1.697e-05, 2.112e-05, 2.643e-05, 3.288e-05, 3.994e-05, 4.794e-05,
  5.606e-05, 6.383e-05, 7.286e-05, 8.156e-05, 8.883e-05, 9.469e-05,
  9.848e-05, 0.0001023, 0.0001066, 0.0001115, 0.0001145, 0.0001142,
  0.0001133
};

static double no2[121] = {
  3.036e-09, 2.945e-10, 9.982e-11, 5.069e-11, 3.485e-11, 2.982e-11,
  2.947e-11, 3.164e-11, 3.714e-11, 4.586e-11, 6.164e-11, 8.041e-11,
  9.982e-11, 1.283e-10, 1.73e-10, 2.56e-10, 3.909e-10, 5.959e-10,
  9.081e-10, 1.384e-09, 1.788e-09, 2.189e-09, 2.686e-09, 3.091e-09,
  3.49e-09, 3.796e-09, 4.2e-09, 5.103e-09, 6.005e-09, 6.3e-09, 6.706e-09,
  7.07e-09, 7.434e-09, 7.663e-09, 7.788e-09, 7.8e-09, 7.597e-09,
  7.482e-09, 7.227e-09, 6.403e-09, 5.585e-09, 4.606e-09, 3.703e-09,
  2.984e-09, 2.183e-09, 1.48e-09, 8.441e-10, 5.994e-10, 3.799e-10,
  2.751e-10, 1.927e-10, 1.507e-10, 1.102e-10, 6.971e-11, 5.839e-11,
  3.904e-11, 3.087e-11, 2.176e-11, 1.464e-11, 1.209e-11, 8.497e-12,
  6.477e-12, 4.371e-12, 2.914e-12, 2.424e-12, 1.753e-12, 1.35e-12,
  9.417e-13, 6.622e-13, 5.148e-13, 3.841e-13, 3.446e-13, 3.01e-13,
  2.551e-13, 2.151e-13, 1.829e-13, 1.64e-13, 1.475e-13, 1.352e-13,
  1.155e-13, 9.963e-14, 9.771e-14, 9.577e-14, 9.384e-14, 9.186e-14,
  9e-14, 9e-14, 9e-14, 9e-14, 9e-14, 9e-14, 9e-14, 9e-14, 9e-14, 9e-14,
  9e-14, 9e-14, 9e-14, 9e-14, 9e-14, 9e-14, 9e-14, 9e-14, 9e-14, 9e-14,
  9e-14, 9e-14, 9e-14, 9e-14, 9e-14, 9e-14, 9e-14, 9e-14, 9e-14, 9e-14,
  9e-14, 9e-14, 9e-14, 9e-14, 9e-14, 9e-14
};

static double o3[121] = {
  2.218e-08, 3.394e-08, 3.869e-08, 4.219e-08, 4.501e-08, 4.778e-08,
  5.067e-08, 5.402e-08, 5.872e-08, 6.521e-08, 7.709e-08, 9.461e-08,
  1.269e-07, 1.853e-07, 2.723e-07, 3.964e-07, 5.773e-07, 8.2e-07,
  1.155e-06, 1.59e-06, 2.076e-06, 2.706e-06, 3.249e-06, 3.848e-06,
  4.459e-06, 4.986e-06, 5.573e-06, 5.958e-06, 6.328e-06, 6.661e-06,
  6.9e-06, 7.146e-06, 7.276e-06, 7.374e-06, 7.447e-06, 7.383e-06,
  7.321e-06, 7.161e-06, 6.879e-06, 6.611e-06, 6.216e-06, 5.765e-06,
  5.355e-06, 4.905e-06, 4.471e-06, 4.075e-06, 3.728e-06, 3.413e-06,
  3.125e-06, 2.856e-06, 2.607e-06, 2.379e-06, 2.17e-06, 1.978e-06,
  1.8e-06, 1.646e-06, 1.506e-06, 1.376e-06, 1.233e-06, 1.102e-06,
  9.839e-07, 8.771e-07, 7.814e-07, 6.947e-07, 6.102e-07, 5.228e-07,
  4.509e-07, 3.922e-07, 3.501e-07, 3.183e-07, 2.909e-07, 2.686e-07,
  2.476e-07, 2.284e-07, 2.109e-07, 2.003e-07, 2.013e-07, 2.022e-07,
  2.032e-07, 2.042e-07, 2.097e-07, 2.361e-07, 2.656e-07, 2.989e-07,
  3.37e-07, 3.826e-07, 4.489e-07, 5.26e-07, 6.189e-07, 7.312e-07,
  8.496e-07, 8.444e-07, 8.392e-07, 8.339e-07, 8.286e-07, 8.234e-07,
  8.181e-07, 8.129e-07, 8.077e-07, 8.026e-07, 6.918e-07, 5.176e-07,
  3.865e-07, 2.885e-07, 2.156e-07, 1.619e-07, 1.219e-07, 9.161e-08,
  6.972e-08, 5.399e-08, 3.498e-08, 2.111e-08, 1.322e-08, 8.482e-09,
  5.527e-09, 3.423e-09, 2.071e-09, 1.314e-09, 8.529e-10, 5.503e-10,
  3.665e-10
};

static double ocs[121] = {
  6e-10, 6e-10, 6e-10, 6e-10, 6e-10, 6e-10, 6e-10, 6e-10, 6e-10, 5.997e-10,
  5.989e-10, 5.881e-10, 5.765e-10, 5.433e-10, 5.074e-10, 4.567e-10,
  4.067e-10, 3.601e-10, 3.093e-10, 2.619e-10, 2.232e-10, 1.805e-10,
  1.46e-10, 1.187e-10, 8.03e-11, 5.435e-11, 3.686e-11, 2.217e-11,
  1.341e-11, 8.756e-12, 4.511e-12, 2.37e-12, 1.264e-12, 8.28e-13,
  5.263e-13, 3.209e-13, 1.717e-13, 9.068e-14, 4.709e-14, 2.389e-14,
  1.236e-14, 1.127e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14,
  1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14,
  1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14,
  1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14,
  1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14,
  1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14,
  1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14,
  1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14,
  1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14,
  1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14,
  1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14,
  1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14,
  1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14, 1.091e-14,
  1.091e-14, 1.091e-14, 1.091e-14
};

static double sf6[121] = {
  4.103e-12, 4.103e-12, 4.103e-12, 4.103e-12, 4.103e-12, 4.103e-12,
  4.103e-12, 4.103e-12, 4.103e-12, 4.087e-12, 4.064e-12, 4.023e-12,
  3.988e-12, 3.941e-12, 3.884e-12, 3.755e-12, 3.622e-12, 3.484e-12,
  3.32e-12, 3.144e-12, 2.978e-12, 2.811e-12, 2.653e-12, 2.489e-12,
  2.332e-12, 2.199e-12, 2.089e-12, 2.013e-12, 1.953e-12, 1.898e-12,
  1.859e-12, 1.826e-12, 1.798e-12, 1.776e-12, 1.757e-12, 1.742e-12,
```

```
    1.728e-12, 1.717e-12, 1.707e-12, 1.698e-12, 1.691e-12, 1.685e-12,
    1.679e-12, 1.675e-12, 1.671e-12, 1.668e-12, 1.665e-12, 1.663e-12,
    1.661e-12, 1.659e-12, 1.658e-12, 1.657e-12, 1.656e-12, 1.655e-12,
    1.654e-12, 1.653e-12, 1.653e-12, 1.652e-12, 1.652e-12, 1.652e-12,
    1.651e-12, 1.651e-12, 1.651e-12, 1.651e-12, 1.651e-12, 1.651e-12,
    1.651e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12,
    1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12,
    1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12,
    1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12,
    1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12,
    1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12,
    1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12,
    1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12, 1.65e-12
  };

  static double so2[121] = {
    1e-10, 1e-10, 1e-10, 1e-10, 1e-10, 1e-10, 1e-10, 1e-10, 1e-10, 1e-10,
    1e-10, 1e-10, 9.867e-11, 9.537e-11, 9e-11, 8.404e-11, 7.799e-11,
    7.205e-11, 6.616e-11, 6.036e-11, 5.475e-11, 5.007e-11, 4.638e-11,
    4.346e-11, 4.055e-11, 3.763e-11, 3.471e-11, 3.186e-11, 2.905e-11,
    2.631e-11, 2.358e-11, 2.415e-11, 2.949e-11, 3.952e-11, 5.155e-11,
    6.76e-11, 8.741e-11, 1.099e-10, 1.278e-10, 1.414e-10, 1.512e-10,
    1.607e-10, 1.699e-10, 1.774e-10, 1.832e-10, 1.871e-10, 1.907e-10,
    1.943e-10, 1.974e-10, 1.993e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10,
    2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10,
    2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10,
    2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10,
    2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10,
    2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10,
    2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10,
    2e-10, 2e-10, 2e-10, 2e-10, 2e-10, 2e-10
  };

  static int ig_co2 = -999;

  double *q[NG] = { NULL };

  /* Find emitter index of CO2... */
  if (ig_co2 == -999)
    ig_co2 = find_emitter(ctl, "CO2");

  /* Identify variable... */
  for (int ig = 0; ig < ctl->ng; ig++) {
    q[ig] = NULL;
    if (strcasecmp(ctl->emitter[ig], "C2H2") == 0)
      q[ig] = c2h2;
    if (strcasecmp(ctl->emitter[ig], "C2H6") == 0)
      q[ig] = c2h6;
    if (strcasecmp(ctl->emitter[ig], "CCl4") == 0)
      q[ig] = ccl4;
    if (strcasecmp(ctl->emitter[ig], "CH4") == 0)
      q[ig] = ch4;
    if (strcasecmp(ctl->emitter[ig], "ClO") == 0)
      q[ig] = clo;
    if (strcasecmp(ctl->emitter[ig], "ClONO2") == 0)
      q[ig] = clono2;
    if (strcasecmp(ctl->emitter[ig], "CO") == 0)
      q[ig] = co;
    if (strcasecmp(ctl->emitter[ig], "COF2") == 0)
      q[ig] = cof2;
    if (strcasecmp(ctl->emitter[ig], "F11") == 0)
      q[ig] = f11;
    if (strcasecmp(ctl->emitter[ig], "F12") == 0)
      q[ig] = f12;
    if (strcasecmp(ctl->emitter[ig], "F14") == 0)
      q[ig] = f14;
    if (strcasecmp(ctl->emitter[ig], "F22") == 0)
      q[ig] = f22;
    if (strcasecmp(ctl->emitter[ig], "H2O") == 0)
      q[ig] = h2o;
    if (strcasecmp(ctl->emitter[ig], "H2O2") == 0)
      q[ig] = h2o2;
    if (strcasecmp(ctl->emitter[ig], "HCN") == 0)
      q[ig] = hcn;
    if (strcasecmp(ctl->emitter[ig], "HNO3") == 0)
      q[ig] = hno3;
    if (strcasecmp(ctl->emitter[ig], "HNO4") == 0)
      q[ig] = hno4;
    if (strcasecmp(ctl->emitter[ig], "HOCl") == 0)
      q[ig] = hocl;
    if (strcasecmp(ctl->emitter[ig], "N2O") == 0)
      q[ig] = n2o;
    if (strcasecmp(ctl->emitter[ig], "N2O5") == 0)
      q[ig] = n2o5;
    if (strcasecmp(ctl->emitter[ig], "NH3") == 0)
      q[ig] = nh3;
    if (strcasecmp(ctl->emitter[ig], "NO") == 0)
```

```
      q[ig] = no;
    if (strcasecmp(ctl->emitter[ig], "NO2") == 0)
      q[ig] = no2;
    if (strcasecmp(ctl->emitter[ig], "O3") == 0)
      q[ig] = o3;
    if (strcasecmp(ctl->emitter[ig], "OCS") == 0)
      q[ig] = ocs;
    if (strcasecmp(ctl->emitter[ig], "SF6") == 0)
      q[ig] = sf6;
    if (strcasecmp(ctl->emitter[ig], "SO2") == 0)
      q[ig] = so2;
  }

  /* Loop over atmospheric data points... */
  for (int ip = 0; ip < atm->np; ip++) {

    /* Get altitude index... */
    const int iz = locate_reg(z, 121, atm->z[ip]);

    /* Interpolate pressure... */
    atm->p[ip] = LOGY(z[iz], pre[iz], z[iz + 1], pre[iz + 1], atm->z[ip]);

    /* Interpolate temperature... */
    atm->t[ip] = LIN(z[iz], tem[iz], z[iz + 1], tem[iz + 1], atm->z[ip]);

    /* Interpolate trace gases... */
    for (int ig = 0; ig < ctl->ng; ig++)
      if (q[ig] != NULL)
        atm->q[ig][ip] =
          LIN(z[iz], q[ig][iz], z[iz + 1], q[ig][iz + 1], atm->z[ip]);
      else
        atm->q[ig][ip] = 0;

    /* Set CO2... */
    if (ig_co2 >= 0)
      atm->q[ig_co2][ip] =
        371.789948e-6 + 2.026214e-6 * (atm->time[ip] - 63158400.) / 31557600.;

    /* Set extinction to zero... */
    for (int iw = 0; iw < ctl->nw; iw++)
      atm->k[iw][ip] = 0;

    /* Set cloud layer... */
    atm->clz = atm->cldz = 0;
    for (int icl = 0; icl < ctl->ncl; icl++)
      atm->clk[icl] = 0;

    /* Set surface layer... */
    atm->sfz = atm->sfp = atm->sft = 0;
    for (int isf = 0; isf < ctl->nsf; isf++)
      atm->sfeps[isf] = 1;
  }
}
```

Here is the call graph for this function:



ctmco2()

```
double ctmco2 (
        const double nu,
```

```
const double p,
const double t,
const double u )
```

Compute carbon dioxide continuum (optical depth).

Definition at line 886 of file jurassic.c.

```
                    {

  static double co2296[2001] = { 9.3388e-5, 9.7711e-5, 1.0224e-4, 1.0697e-4,
    1.1193e-4, 1.1712e-4, 1.2255e-4, 1.2824e-4, 1.3419e-4, 1.4043e-4,
    1.4695e-4, 1.5378e-4, 1.6094e-4, 1.6842e-4, 1.7626e-4, 1.8447e-4,
    1.9307e-4, 2.0207e-4, 2.1149e-4, 2.2136e-4, 2.3169e-4, 2.4251e-4,
    2.5384e-4, 2.657e-4, 2.7813e-4, 2.9114e-4, 3.0477e-4, 3.1904e-4,
    3.3399e-4, 3.4965e-4, 3.6604e-4, 3.8322e-4, 4.0121e-4, 4.2006e-4,
    4.398e-4, 4.6047e-4, 4.8214e-4, 5.0483e-4, 5.286e-4, 5.535e-4,
    5.7959e-4, 6.0693e-4, 6.3557e-4, 6.6558e-4, 6.9702e-4, 7.2996e-4,
    7.6449e-4, 8.0066e-4, 8.3856e-4, 8.7829e-4, 9.1991e-4, 9.6354e-4,
    .0010093, .0010572, .0011074, .00116, .0012152, .001273,
    .0013336, .0013972, .0014638, .0015336, .0016068, .0016835,
    .001764, .0018483, .0019367, .0020295, .0021267, .0022286,
    .0023355, .0024476, .0025652, .0026885, .0028178, .0029534,
    .0030956, .0032448, .0034012, .0035654, .0037375, .0039181,
    .0041076, .0043063, .0045148, .0047336, .0049632, .005204,
    .0054567, .0057219, .0060002, .0062923, .0065988, .0069204,
    .007258, .0076123, .0079842, .0083746, .0087844, .0092146,
    .0096663, .01014, .010638, .011161, .01171, .012286, .012891,
    .013527, .014194, .014895, .015631, .016404, .017217, .01807,
    .018966, .019908, .020897, .021936, .023028, .024176, .025382,
    .026649, .027981, .02938, .030851, .032397, .034023, .035732,
    .037528, .039416, .041402, .04349, .045685, .047994, .050422,
    .052975, .055661, .058486, .061458, .064584, .067873, .071334,
    .074975, .078807, .082839, .087082, .091549, .096249, .1012,
    .10641, .11189, .11767, .12375, .13015, .13689, .14399, .15147,
    .15935, .16765, .17639, .18561, .19531, .20554, .21632, .22769,
    .23967, .25229, .2656, .27964, .29443, .31004, .3265, .34386,
    .36218, .3815, .40188, .42339, .44609, .47004, .49533, .52202,
    .5502, .57995, .61137, .64455, .6796, .71663, .75574, .79707,
    .84075, .88691, .9357, .98728, 1.0418, 1.0995, 1.1605, 1.225,
    1.2932, 1.3654, 1.4418, 1.5227, 1.6083, 1.6989, 1.7948, 1.8964,
    2.004, 2.118, 2.2388, 2.3668, 2.5025, 2.6463, 2.7988, 2.9606,
    3.1321, 3.314, 3.5071, 3.712, 3.9296, 4.1605, 4.4058, 4.6663,
    4.9431, 5.2374, 5.5501, 5.8818, 6.2353, 6.6114, 7.0115, 7.4372,
    7.8905, 8.3731, 8.8871, 9.4349, 10.019, 10.641, 11.305, 12.013,
    12.769, 13.576, 14.437, 15.358, 16.342, 17.39, 18.513, 19.716,
    21.003, 22.379, 23.854, 25.436, 27.126, 28.942, 30.89, 32.973,
    35.219, 37.634, 40.224, 43.021, 46.037, 49.29, 52.803, 56.447,
    60.418, 64.792, 69.526, 74.637, 80.182, 86.193, 92.713, 99.786,
    107.47, 115.84, 124.94, 134.86, 145.69, 157.49, 170.3, 184.39,
    199.83, 216.4, 234.55, 254.72, 276.82, 299.85, 326.16, 354.99,
    386.51, 416.68, 449.89, 490.12, 534.35, 578.25, 632.26, 692.61,
    756.43, 834.75, 924.11, 1016.9, 996.96, 1102.7, 1219.2, 1351.9,
    1494.3, 1654.1, 1826.5, 2027.9, 2249., 2453.8, 2714.4, 2999.4,
    3209.5, 3509., 3840.4, 3907.5, 4190.7, 4533.5, 4648.3, 5059.1,
    5561.6, 6191.4, 6820.8, 7905.9, 9362.2, 2431.3, 2211.3, 2046.8,
    2023.8, 1985.9, 1905.9, 1491.1, 1369.8, 1262.2, 1200.7, 887.74,
    820.25, 885.23, 887.21, 816.73, 1126.9, 1216.2, 1272.4, 1579.5,
    1634.2, 1656.3, 1657.9, 1789.5, 1670.8, 1509.5, 8474.6, 7489.2,
    6793.6, 6117., 5574.1, 5141.2, 5084.6, 4745.1, 4413.2, 4102.8,
    4024.7, 3715., 3398.6, 3100.8, 2900.4, 2629.2, 2374., 2144.7,
    1955.8, 1760.8, 1591.2, 1435.2, 1296.2, 1174., 1065.1, 967.76,
    999.48, 897.45, 809.23, 732.77, 670.26, 611.93, 560.11, 518.77,
    476.84, 438.8, 408.48, 380.21, 349.24, 322.71, 296.65, 272.85,
    251.96, 232.04, 213.88, 197.69, 182.41, 168.41, 155.79, 144.05,
    133.31, 123.48, 114.5, 106.21, 98.591, 91.612, 85.156, 79.204,
    73.719, 68.666, 63.975, 59.637, 56.35, 52.545, 49.042, 45.788,
    42.78, 39.992, 37.441, 35.037, 32.8, 30.744, 28.801, 26.986,
    25.297, 23.731, 22.258, 20.883, 19.603, 18.403, 17.295, 16.249,
    15.271, 14.356, 13.501, 12.701, 11.954, 11.254, 10.6, 9.9864,
    9.4118, 8.8745, 8.3714, 7.8997, 7.4578, 7.0446, 6.6573, 6.2949,
    5.9577, 5.6395, 5.3419, 5.063, 4.8037, 4.5608, 4.3452, 4.1364,
    3.9413, 3.7394, 3.562, 3.3932, 3.2325, 3.0789, 2.9318, 2.7898,
    2.6537, 2.5225, 2.3958, 2.2305, 2.1215, 2.0245, 1.9427, 1.8795,
    1.8336, 1.7604, 1.7016, 1.6419, 1.5282, 1.4611, 1.3443, 1.27,
    1.1675, 1.0824, 1.0534, .99833, .95854, .92981, .90887, .89346,
    .88113, .87068, .86102, .85096, .88262, .86151, .83565, .80518,
    .77045, .73736, .74744, .74954, .75773, .82267, .83493, .89402,
    .89725, .93426, .95564, .94045, .94174, .93404, .92035, .90456,
    .88621, .86673, .78117, .7515, .72056, .68822, .65658, .62764,
    .55984, .55598, .57407, .60963, .63763, .66198, .61132, .60972,
    .52496, .50649, .41872, .3964, .32422, .27276, .24048, .23772,
    .2286, .22711, .23999, .32038, .34371, .36621, .38561, .39953,
    .40636, .44913, .42716, .3919, .35477, .33935, .3351, .39746,
    .40993, .49398, .49956, .56157, .54742, .57295, .57386, .55417,
```

```
.50745, .471, .43446, .39102, .34993, .31269, .27888, .24912,
.22291, .19994, .17972, .16197, .14633, .13252, .12029, .10942,
.099745, .091118, .083404, .076494, .070292, .064716, .059697,
.055173, .051093, .047411, .044089, .041092, .038392, .035965,
.033789, .031846, .030122, .028607, .02729, .026169, .025209,
.024405, .023766, .023288, .022925, .022716, .022681, .022685,
.022768, .023133, .023325, .023486, .024004, .024126, .024083,
.023785, .024023, .023029, .021649, .021108, .019454, .017809,
.017292, .016635, .017037, .018068, .018977, .018756, .017847,
.016557, .016142, .014459, .012869, .012381, .010875, .0098701,
.009285, .0091698, .0091701, .0096145, .010553, .01106, .012613,
.014362, .015017, .016507, .017741, .01768, .017784, .0171,
.016357, .016172, .017257, .018978, .020935, .021741, .023567,
.025183, .025589, .026732, .027648, .028278, .028215, .02856,
.029015, .029062, .028851, .028497, .027825, .027801, .026523,
.02487, .022967, .022168, .020194, .018605, .017903, .018439,
.019697, .020311, .020855, .020057, .018608, .016738, .015963,
.013844, .011801, .011134, .0097573, .0086007, .0086226,
.0083721, .0090978, .0097616, .0098426, .011317, .012853, .01447,
.014657, .015771, .016351, .016079, .014829, .013431, .013185,
.013207, .01448, .016176, .017971, .018265, .019526, .020455,
.019797, .019802, .0194, .018176, .017505, .016197, .015339,
.014401, .013213, .012203, .011186, .010236, .0093288, .0084854,
.0076837, .0069375, .0062614, .0056628, .0051153, .0046015,
.0041501, .003752, .0033996, .0030865, .0028077, .0025586,
.0023355, .0021353, .0019553, .0017931, .0016466, .0015141,
.0013941, .0012852, .0011862, .0010962, .0010142, 9.3935e-4,
8.71e-4, 8.0851e-4, 7.5132e-4, 6.9894e-4, 6.5093e-4, 6.0689e-4,
5.6647e-4, 5.2935e-4, 4.9525e-4, 4.6391e-4, 4.3509e-4, 4.086e-4,
3.8424e-4, 3.6185e-4, 3.4126e-4, 3.2235e-4, 3.0498e-4, 2.8904e-4,
2.7444e-4, 2.6106e-4, 2.4883e-4, 2.3766e-4, 2.275e-4, 2.1827e-4,
2.0992e-4, 2.0239e-4, 1.9563e-4, 1.896e-4, 1.8427e-4, 1.796e-4,
1.7555e-4, 1.7209e-4, 1.692e-4, 1.6687e-4, 1.6505e-4, 1.6375e-4,
1.6294e-4, 1.6261e-4, 1.6274e-4, 1.6334e-4, 1.6438e-4, 1.6587e-4,
1.678e-4, 1.7017e-4, 1.7297e-4, 1.762e-4, 1.7988e-4, 1.8399e-4,
1.8855e-4, 1.9355e-4, 1.9902e-4, 2.0494e-4, 2.1134e-4, 2.1823e-4,
2.2561e-4, 2.335e-4, 2.4192e-4, 2.5088e-4, 2.604e-4, 2.705e-4,
2.8119e-4, 2.9251e-4, 3.0447e-4, 3.171e-4, 3.3042e-4, 3.4447e-4,
3.5927e-4, 3.7486e-4, 3.9127e-4, 4.0854e-4, 4.267e-4, 4.4579e-4,
4.6586e-4, 4.8696e-4, 5.0912e-4, 5.324e-4, 5.5685e-4, 5.8253e-4,
6.0949e-4, 6.378e-4, 6.6753e-4, 6.9873e-4, 7.3149e-4, 7.6588e-4,
8.0198e-4, 8.3987e-4, 8.7964e-4, 9.2139e-4, 9.6522e-4, .0010112,
.0010595, .0011102, .0011634, .0012193, .001278, .0013396,
.0014043, .0014722, .0015436, .0016185, .0016972, .0017799,
.0018668, .001958, .0020539, .0021547, .0022606, .0023719,
.002489, .002612, .0027414, .0028775, .0030206, .0031712,
.0033295, .0034962, .0036716, .0038563, .0040506, .0042553,
.0044709, .004698, .0049373, .0051894, .0054552, .0057354,
.006031, .0063427, .0066717, .0070188, .0073854, .0077726,
.0081816, .0086138, .0090709, .0095543, .010066, .010607,
.011181, .011789, .012433, .013116, .013842, .014613, .015432,
.016304, .017233, .018224, .019281, .020394, .021574, .022836,
.024181, .025594, .027088, .028707, .030401, .032245, .034219,
.036262, .038539, .040987, .043578, .04641, .04949, .052726,
.056326, .0602, .064093, .068521, .073278, .077734, .083064,
.088731, .093885, .1003, .1072, .11365, .12187, .13078, .13989,
.15095, .16299, .17634, .19116, .20628, .22419, .24386, .26587,
.28811, .31399, .34321, .36606, .39675, .42742, .44243, .47197,
.49993, .49027, .51147, .52803, .48931, .49729, .5026, .43854,
.441, .44766, .43414, .46151, .50029, .55247, .43855, .32115,
.32607, .3431, .36119, .38029, .41179, .43996, .47144, .51853,
.55362, .59122, .66338, .69877, .74001, .82923, .86907, .90361,
1.0025, 1.031, 1.0559, 1.104, 1.1178, 1.1341, 1.1547, 1.351,
1.4772, 1.4812, 1.4907, 1.512, 1.5442, 1.5853, 1.6358, 1.6963,
1.7674, 1.8474, 1.9353, 2.0335, 2.143, 2.2592, 2.3853, 2.5217,
2.6686, 2.8273, 2.9998, 3.183, 3.3868, 3.6109, 3.8564, 4.1159,
4.4079, 4.7278, 5.0497, 5.3695, 5.758, 6.0834, 6.4976, 6.9312,
7.38, 7.5746, 7.9833, 8.3791, 8.3956, 8.7501, 9.1067, 9.072,
9.4649, 9.9112, 10.402, 10.829, 11.605, 12.54, 12.713, 10.443,
10.825, 11.375, 11.955, 12.623, 13.326, 14.101, 15.041, 15.547,
16.461, 17.439, 18.716, 19.84, 21.036, 22.642, 23.901, 25.244,
27.03, 28.411, 29.871, 31.403, 33.147, 34.744, 36.456, 39.239,
43.605, 45.162, 47.004, 49.093, 51.391, 53.946, 56.673, 59.629,
63.167, 66.576, 70.254, 74.222, 78.477, 83.034, 87.914, 93.18,
98.77, 104.74, 111.15, 117.95, 125.23, 133.01, 141.33, 150.21,
159.71, 169.89, 180.93, 192.54, 204.99, 218.34, 232.65, 248.,
264.47, 282.14, 301.13, 321.53, 343.48, 367.08, 392.5, 419.88,
449.4, 481.26, 515.64, 552.79, 592.99, 636.48, 683.61, 734.65,
789.99, 850.02, 915.14, 985.81, 1062.5, 1147.1, 1237.8, 1336.4,
1443.2, 1558.9, 1684.2, 1819.2, 1965.2, 2122.6, 2291.7, 2470.8,
2665.7, 2874.9, 3099.4, 3337.9, 3541., 3813.3, 4111.9, 4439.3,
4798.9, 5196., 5639.2, 6087.5, 6657.7, 7306.7, 8040.7, 8845.5,
9702.2, 10670., 11739., 12842., 14141., 15498., 17068., 18729.,
20557., 22559., 25248., 27664., 30207., 32915., 35611., 38081.,
40715., 43191., 41651., 42750., 43785., 44353., 44366., 44189.,
43618., 42862., 41878., 35133., 35215., 36383., 39420., 44055.,
44155., 45850., 46853., 39197., 38274., 29942., 28553., 21792.,
```

```
21228., 17106., 14955., 18181., 19557., 21427., 23728., 26301.,
28584., 30775., 32536., 33867., 40089., 39204., 37329., 34452.,
31373., 33921., 34800., 36043., 44415., 45162., 52181., 50895.,
54140., 50840., 50468., 48302., 44915., 40910., 36754., 32755.,
29093., 25860., 22962., 20448., 18247., 16326., 14645., 13165.,
11861., 10708., 9686.9, 8779.7, 7971.9, 7250.8, 6605.7, 6027.2,
5507.3, 5039.1, 4616.6, 4234.8, 3889., 3575.4, 3290.5, 3031.3,
2795.2, 2579.9, 2383.1, 2203.3, 2038.6, 1887.6, 1749.1, 1621.9,
1505., 1397.4, 1298.3, 1207., 1122.8, 1045., 973.1, 906.64,
845.16, 788.22, 735.48, 686.57, 641.21, 599.1, 559.99, 523.64,
489.85, 458.42, 429.16, 401.92, 376.54, 352.88, 330.82, 310.24,
291.03, 273.09, 256.34, 240.69, 226.05, 212.37, 199.57, 187.59,
176.37, 165.87, 156.03, 146.82, 138.17, 130.07, 122.47, 115.34,
108.65, 102.37, 96.473, 90.934, 85.73, 80.84, 76.243, 71.922,
67.858, 64.034, 60.438, 57.052, 53.866, 50.866, 48.04, 45.379,
42.872, 40.51, 38.285, 36.188, 34.211, 32.347, 30.588, 28.929,
27.362, 25.884, 24.489, 23.171, 21.929, 20.755, 19.646, 18.599,
17.61, 16.677, 15.795, 14.961, 14.174, 13.43, 12.725, 12.06,
11.431, 10.834, 10.27, 9.7361, 9.2302, 8.7518, 8.2997, 7.8724,
7.4674, 7.0848, 6.7226, 6.3794, 6.054, 5.745, 5.4525, 5.1752,
4.9121, 4.6625, 4.4259, 4.2015, 3.9888, 3.7872, 3.5961, 3.4149,
3.2431, 3.0802, 2.9257, 2.7792, 2.6402, 2.5084, 2.3834, 2.2648,
2.1522, 2.0455, 1.9441, 1.848, 1.7567, 1.6701, 1.5878, 1.5097,
1.4356, 1.3651, 1.2981, 1.2345, 1.174, 1.1167, 1.062, 1.0101,
.96087, .91414, .86986, .82781, .78777, .74971, .71339, .67882,
.64604, .61473, .58507, .55676, .52987, .5044, .48014, .45715,
.43527, .41453, .3948, .37609, .35831, .34142, .32524, .30995,
.29536, .28142, .26807, .25527, .24311, .23166, .22077, .21053,
.20081, .19143, .18261, .17407, .16603, .15833, .15089, .14385,
.13707, .13065, .12449, .11865, .11306, .10774, .10266, .097818,
.093203, .088815, .084641, .080671, .076892, .073296, .069873,
.066613, .06351, .060555, .05774, .055058, .052504, .050071,
.047752, .045543, .043438, .041432, .039521, .037699, .035962,
.034307, .032729, .031225, .029791, .028423, .02712, .025877,
.024692, .023563, .022485, .021458, .020478, .019543, .018652,
.017802, .016992, .016219, .015481, .014778, .014107, .013467,
.012856, .012274, .011718, .011188, .010682, .0102, .0097393,
.0093001, .008881, .0084812, .0080997, .0077358, .0073885,
.0070571, .0067409, .0064393, .0061514, .0058768, .0056147,
.0053647, .0051262, .0048987, .0046816, .0044745, .0042769,
.0040884, .0039088, .0037373, .0035739, .003418, .0032693,
.0031277, .0029926, .0028639, .0027413, .0026245, .0025133,
.0024074, .0023066, .0022108, .0021196, .002033, .0019507,
.0018726, .0017985, .0017282, .0016617, .0015988, .0015394,
.0014834, .0014306, .0013811, .0013346, .0012911, .0012506,
.0012131, .0011784, .0011465, .0011175, .0010912, .0010678,
.0010472, .0010295, .0010147, .001003, 9.9428e-4, 9.8883e-4,
9.8673e-4, 9.8821e-4, 9.9343e-4, .0010027, .0010164, .0010348,
.0010586, .0010882, .0011245, .0011685, .0012145, .0012666,
.0013095, .0013688, .0014048, .0014663, .0015309, .0015499,
.0016144, .0016312, .001705, .0017892, .0018499, .0019715,
.0021102, .0022442, .0024284, .0025893, .0027703, .0029445,
.0031193, .003346, .0034552, .0036906, .0037584, .0040084,
.0041934, .0044587, .0047093, .0049759, .0053421, .0055134,
.0059048, .0058663, .0061036, .0063259, .0059657, .0060653,
.0060972, .0055539, .0055653, .0055772, .005331, .0054953,
.0055919, .0058684, .006183, .0066675, .0069808, .0075142,
.0078536, .0084282, .0089454, .0094625, .0093703, .0095857,
.0099283, .010063, .010521, .0097778, .0098175, .010379, .010447,
.0105, .010617, .010706, .01078, .011177, .011212, .011304,
.011446, .011603, .011816, .012165, .012545, .013069, .013539,
.01411, .014776, .016103, .017016, .017994, .018978, .01998,
.021799, .022745, .023681, .024627, .025562, .026992, .027958,
.029013, .030154, .031402, .03228, .033651, .035272, .037088,
.039021, .041213, .043597, .045977, .04877, .051809, .054943,
.058064, .061528, .06537, .069309, .071928, .075752, .079589,
.083352, .084096, .087497, .090817, .091198, .094966, .099045,
.10429, .10867, .11518, .12269, .13126, .14087, .15161, .16388,
.16423, .1759, .18721, .19994, .21275, .22513, .23041, .24231,
.25299, .25396, .26396, .27696, .27929, .2908, .30595, .31433,
.3282, .3429, .35944, .37467, .39277, .41245, .43326, .45649,
.48152, .51897, .54686, .57877, .61263, .64962, .68983, .73945,
.78619, .83537, .89622, .95002, 1.0067, 1.0742, 1.1355, 1.2007,
1.2738, 1.347, 1.4254, 1.5094, 1.6009, 1.6976, 1.8019, 1.9148,
2.0357, 2.166, 2.3066, 2.4579, 2.6208, 2.7966, 2.986, 3.188,
3.4081, 3.6456, 3.9, 4.1747, 4.4712, 4.7931, 5.1359, 5.5097,
5.9117, 6.3435, 6.8003, 7.3001, 7.8385, 8.3945, 9.011, 9.6869,
10.392, 11.18, 12.036, 12.938, 13.944, 14.881, 16.029, 17.255,
18.574, 19.945, 21.38, 22.9, 24.477, 26.128, 27.87, 29.037,
30.988, 33.145, 35.506, 37.76, 40.885, 44.487, 48.505, 52.911,
57.56, 61.964, 67.217, 72.26, 78.343, 85.08, 91.867, 99.435,
107.68, 116.97, 127.12, 138.32, 150.26, 163.04, 174.81, 189.26,
205.61, 224.68, 240.98, 261.88, 285.1, 307.58, 334.35, 363.53,
394.68, 427.85, 458.85, 489.25, 472.87, 486.93, 496.27, 501.52,
501.57, 497.14, 488.09, 476.32, 393.76, 388.51, 393.42, 414.45,
455.12, 514.62, 520.38, 547.42, 562.6, 487.47, 480.83, 391.06,
376.92, 303.7, 295.91, 256.03, 236.73, 280.38, 310.71, 335.53,
```

```
  367.88, 401.94, 435.52, 469.13, 497.94, 588.82, 597.94, 597.2,
  588.28, 571.2, 555.75, 603.56, 638.15, 680.75, 801.72, 848.01,
  962.15, 990.06, 1068.1, 1076.2, 1115.3, 1134.2, 1136.6, 1119.1,
  1108.9, 1090.6, 1068.7, 1041.9, 1005.4, 967.98, 927.08, 780.1,
  751.41, 733.12, 742.65, 785.56, 855.16, 852.45, 878.1, 784.59,
  777.81, 765.13, 622.93, 498.09, 474.89, 386.9, 378.48, 336.17,
  322.04, 329.57, 350.5, 383.38, 420.02, 462.39, 499.71, 531.98,
  654.99, 653.43, 639.99, 605.16, 554.16, 504.42, 540.64, 552.33,
  679.46, 699.51, 713.91, 832.17, 919.91, 884.96, 907.57, 846.56,
  818.56, 768.93, 706.71, 642.17, 575.95, 515.38, 459.07, 409.02,
  364.61, 325.46, 291.1, 260.89, 234.39, 211.01, 190.38, 172.11,
  155.91, 141.49, 128.63, 117.13, 106.84, 97.584, 89.262, 81.756,
  74.975, 68.842, 63.28, 58.232, 53.641, 49.46, 45.649, 42.168,
  38.991, 36.078, 33.409, 30.96, 28.71, 26.642, 24.737, 22.985,
  21.37, 19.882, 18.512, 17.242, 16.073, 14.987, 13.984, 13.05,
  12.186, 11.384, 10.637, 9.9436, 9.2988, 8.6991, 8.141, 7.6215,
  7.1378, 6.6872, 6.2671, 5.8754, 5.51, 5.1691, 4.851, 4.5539,
  4.2764, 4.0169, 3.7742, 3.5472, 3.3348, 3.1359, 2.9495, 2.7749,
  2.6113, 2.4578, 2.3139, 2.1789, 2.0523, 1.9334, 1.8219, 1.7171,
  1.6188, 1.5263, 1.4395, 1.3579, 1.2812, 1.209, 1.1411, 1.0773,
  1.0171, .96048, .90713, .85684, .80959, .76495, .72282, .68309,
  .64563, .61035, .57707, .54573, .51622, .48834, .46199, .43709,
  .41359, .39129, .37034, .35064, .33198, .31442, .29784, .28218,
  .26732, .25337, .24017, .22774, .21601, .20479, .19426
};

static double co2260[2001] = { 5.7971e-5, 6.0733e-5, 6.3628e-5, 6.6662e-5,
  6.9843e-5, 7.3176e-5, 7.6671e-5, 8.0334e-5, 8.4175e-5, 8.8201e-5,
  9.2421e-5, 9.6846e-5, 1.0149e-4, 1.0635e-4, 1.1145e-4, 1.1679e-4,
  1.224e-4, 1.2828e-4, 1.3444e-4, 1.409e-4, 1.4768e-4, 1.5479e-4,
  1.6224e-4, 1.7006e-4, 1.7826e-4, 1.8685e-4, 1.9587e-4, 2.0532e-4,
  2.1524e-4, 2.2565e-4, 2.3656e-4, 2.48e-4, 2.6001e-4, 2.7261e-4,
  2.8582e-4, 2.9968e-4, 3.1422e-4, 3.2948e-4, 3.4548e-4, 3.6228e-4,
  3.799e-4, 3.9838e-4, 4.1778e-4, 4.3814e-4, 4.595e-4, 4.8191e-4,
  5.0543e-4, 5.3012e-4, 5.5603e-4, 5.8321e-4, 6.1175e-4, 6.417e-4,
  6.7314e-4, 7.0614e-4, 7.4078e-4, 7.7714e-4, 8.1531e-4, 8.5538e-4,
  8.9745e-4, 9.4162e-4, 9.8798e-4, .0010367, .0010878, .0011415,
  .0011978, .001257, .0013191, .0013844, .001453, .0015249,
  .0016006, .00168, .0017634, .001851, .001943, .0020397, .0021412,
  .0022479, .00236, .0024778, .0026015, .0027316, .0028682,
  .0030117, .0031626, .0033211, .0034877, .0036628, .0038469,
  .0040403, .0042436, .0044574, .004682, .0049182, .0051665,
  .0054276, .0057021, .0059907, .0062942, .0066133, .0069489,
  .0073018, .0076729, .0080632, .0084738, .0089056, .0093599,
  .0098377, .01034, .010869, .011426, .012011, .012627, .013276,
  .013958, .014676, .015431, .016226, .017063, .017944, .018872,
  .019848, .020876, .021958, .023098, .024298, .025561, .026892,
  .028293, .029769, .031323, .032961, .034686, .036503, .038418,
  .040435, .042561, .044801, .047161, .049649, .052271, .055035,
  .057948, .061019, .064256, .06767, .07127, .075066, .079069,
  .083291, .087744, .092441, .097396, .10262, .10814, .11396,
  .1201, .12658, .13342, .14064, .14826, .1563, .1648, .17376,
  .18323, .19324, .2038, .21496, .22674, .23919, .25234, .26624,
  .28093, .29646, .31287, .33021, .34855, .36794, .38844, .41012,
  .43305, .45731, .48297, .51011, .53884, .56924, .60141, .63547,
  .67152, .70969, .75012, .79292, .83826, .8863, .93718, .99111,
  1.0482, 1.1088, 1.173, 1.2411, 1.3133, 1.3898, 1.471, 1.5571,
  1.6485, 1.7455, 1.8485, 1.9577, 2.0737, 2.197, 2.3278, 2.4668,
  2.6145, 2.7715, 2.9383, 3.1156, 3.3042, 3.5047, 3.7181, 3.9451,
  4.1866, 4.4437, 4.7174, 5.0089, 5.3192, 5.65, 6.0025, 6.3782,
  6.7787, 7.206, 7.6617, 8.1479, 8.6669, 9.221, 9.8128, 10.445,
  11.12, 11.843, 12.615, 13.441, 14.325, 15.271, 16.283, 17.367,
  18.529, 19.776, 21.111, 22.544, 24.082, 25.731, 27.504, 29.409,
  31.452, 33.654, 36.024, 38.573, 41.323, 44.29, 47.492, 50.951,
  54.608, 58.588, 62.929, 67.629, 72.712, 78.226, 84.207, 90.699,
  97.749, 105.42, 113.77, 122.86, 132.78, 143.61, 155.44, 168.33,
  182.48, 198.01, 214.87, 233.39, 253.86, 276.34, 300.3, 327.28,
  356.89, 389.48, 422.29, 458.99, 501.39, 548.13, 595.62, 652.74,
  716.54, 784.57, 866.78, 960.59, 1062.8, 1072.5, 1189.5, 1319.4,
  1467.6, 1630.2, 1813.7, 2016.9, 2253., 2515.3, 2773.5, 3092.8,
  3444.4, 3720.4, 4104.3, 4527.5, 4645.9, 5021.7, 5462.2, 5597.,
  6110.6, 6732.5, 7513.8, 8270.6, 9640.6, 11487., 2796.1, 2680.1,
  2441.6, 2404.2, 2334.8, 2215.2, 1642.5, 1477.9, 1328.1, 1223.5,
  843.34, 766.96, 831.65, 834.84, 774.85, 1156.3, 1275.6, 1366.1,
  1795.6, 1885., 1936.5, 1953.4, 2154.4, 2002.7, 1789.8, 10381.,
  9040., 8216.5, 7384.7, 6721.9, 6187.7, 6143.8, 5703.9, 5276.6,
  4873.1, 4736., 4325.3, 3927., 3554.1, 3286.1, 2950.1, 2642.4,
  2368.7, 2138.9, 1914., 1719.6, 1543.9, 1388.6, 1252.1, 1132.2,
  1024.1, 1025.4, 920.58, 829.59, 750.54, 685.01, 624.25, 570.14,
  525.81, 481.85, 441.95, 408.71, 377.23, 345.86, 318.51, 292.26,
  268.34, 247.04, 227.14, 209.02, 192.69, 177.59, 163.78, 151.26,
  139.73, 129.19, 119.53, 110.7, 102.57, 95.109, 88.264, 81.948,
  76.13, 70.768, 65.827, 61.251, 57.022, 53.495, 49.824, 46.443,
  43.307, 40.405, 37.716, 35.241, 32.923, 30.77, 28.78, 26.915,
  25.177, 23.56, 22.059, 20.654, 19.345, 18.126, 16.988, 15.93,
  14.939, 14.014, 13.149, 12.343, 11.589, 10.884, 10.225, 9.6093,
  9.0327, 8.4934, 7.9889, 7.5166, 7.0744, 6.6604, 6.2727, 5.9098,
```

```
5.5701, 5.2529, 4.955, 4.676, 4.4148, 4.171, 3.9426, 3.7332,
3.5347, 3.3493, 3.1677, 3.0025, 2.8466, 2.6994, 2.5601, 2.4277,
2.3016, 2.1814, 2.0664, 1.9564, 1.8279, 1.7311, 1.6427, 1.5645,
1.4982, 1.443, 1.374, 1.3146, 1.2562, 1.17, 1.1105, 1.0272,
.96863, .89718, .83654, .80226, .75908, .72431, .69573, .67174,
.65126, .63315, .61693, .60182, .58715, .59554, .57649, .55526,
.53177, .50622, .48176, .4813, .47642, .47492, .50273, .50293,
.52687, .52239, .53419, .53814, .52626, .52211, .51492, .50622,
.49746, .48841, .4792, .43534, .41999, .40349, .38586, .36799,
.35108, .31089, .30803, .3171, .33599, .35041, .36149, .32924,
.32462, .27309, .25961, .20922, .19504, .15683, .13098, .11588,
.11478, .11204, .11363, .12135, .16423, .17785, .19094, .20236,
.21084, .2154, .24108, .22848, .20871, .18797, .17963, .17834,
.21552, .22284, .26945, .27052, .30108, .28977, .29772, .29224,
.27658, .24956, .22777, .20654, .18392, .16338, .1452, .12916,
.1152, .10304, .092437, .083163, .075031, .067878, .061564,
.055976, .051018, .046609, .042679, .03917, .036032, .033223,
.030706, .02845, .026428, .024617, .022998, .021554, .02027,
.019136, .018141, .017278, .016541, .015926, .015432, .015058,
.014807, .014666, .014635, .014728, .014947, .01527, .015728,
.016345, .017026, .017798, .018839, .019752, .020636, .021886,
.022695, .02327, .023478, .024292, .023544, .022222, .021932,
.020052, .018143, .017722, .017031, .017782, .01938, .020734,
.020476, .019255, .017477, .016878, .014617, .012489, .011765,
.0099077, .0086446, .0079446, .0078644, .0079763, .008671,
.01001, .0108, .012933, .015349, .016341, .018484, .020254,
.020254, .020478, .019591, .018595, .018385, .019913, .022254,
.024847, .025809, .028053, .029924, .030212, .031367, .03222,
.032739, .032537, .03286, .033344, .033507, .033499, .033339,
.032809, .033041, .031723, .029837, .027511, .026603, .024032,
.021914, .020948, .021701, .023425, .024259, .024987, .023818,
.021768, .019223, .018144, .015282, .012604, .01163, .0097907,
.008336, .0082473, .0079582, .0088077, .009779, .010129, .012145,
.014378, .016761, .01726, .018997, .019998, .019809, .01819,
.016358, .016099, .01617, .017939, .020223, .022521, .02277,
.024279, .025247, .024222, .023989, .023224, .021493, .020362,
.018596, .017309, .015975, .014466, .013171, .011921, .01078,
.0097229, .0087612, .0078729, .0070682, .0063494, .0057156,
.0051459, .0046273, .0041712, .0037686, .0034119, .003095,
.0028126, .0025603, .0023342, .0021314, .0019489, .0017845,
.001636, .0015017, .00138, .0012697, .0011694, .0010782,
9.9507e-4, 9.1931e-4, 8.5013e-4, 7.869e-4, 7.2907e-4, 6.7611e-4,
6.2758e-4, 5.8308e-4, 5.4223e-4, 5.0473e-4, 4.7027e-4, 4.3859e-4,
4.0946e-4, 3.8265e-4, 3.5798e-4, 3.3526e-4, 3.1436e-4, 2.9511e-4,
2.7739e-4, 2.6109e-4, 2.4609e-4, 2.3229e-4, 2.1961e-4, 2.0797e-4,
1.9729e-4, 1.875e-4, 1.7855e-4, 1.7038e-4, 1.6294e-4, 1.5619e-4,
1.5007e-4, 1.4456e-4, 1.3961e-4, 1.3521e-4, 1.3131e-4, 1.2789e-4,
1.2494e-4, 1.2242e-4, 1.2032e-4, 1.1863e-4, 1.1733e-4, 1.1641e-4,
1.1585e-4, 1.1565e-4, 1.158e-4, 1.1629e-4, 1.1712e-4, 1.1827e-4,
1.1976e-4, 1.2158e-4, 1.2373e-4, 1.262e-4, 1.2901e-4, 1.3214e-4,
1.3562e-4, 1.3944e-4, 1.4361e-4, 1.4814e-4, 1.5303e-4, 1.5829e-4,
1.6394e-4, 1.6999e-4, 1.7644e-4, 1.8332e-4, 1.9063e-4, 1.984e-4,
2.0663e-4, 2.1536e-4, 2.246e-4, 2.3436e-4, 2.4468e-4, 2.5558e-4,
2.6708e-4, 2.7921e-4, 2.92e-4, 3.0548e-4, 3.1968e-4, 3.3464e-4,
3.5039e-4, 3.6698e-4, 3.8443e-4, 4.0281e-4, 4.2214e-4, 4.4248e-4,
4.6389e-4, 4.864e-4, 5.1009e-4, 5.3501e-4, 5.6123e-4, 5.888e-4,
6.1781e-4, 6.4833e-4, 6.8043e-4, 7.142e-4, 7.4973e-4, 7.8711e-4,
8.2644e-4, 8.6783e-4, 9.1137e-4, 9.5721e-4, .0010054, .0010562,
.0011096, .0011659, .0012251, .0012875, .0013532, .0014224,
.0014953, .001572, .0016529, .0017381, .0018279, .0019226,
.0020224, .0021277, .0022386, .0023557, .0024792, .0026095,
.002747, .0028921, .0030453, .0032071, .003378, .0035586,
.0037494, .003951, .0041642, .0043897, .0046282, .0048805,
.0051476, .0054304, .00573, .0060473, .0063837, .0067404,
.0071188, .0075203, .0079466, .0083994, .0088806, .0093922,
.0099366, .010516, .011134, .011792, .012494, .013244, .014046,
.014898, .015808, .016781, .017822, .018929, .020108, .02138,
.022729, .02419, .02576, .027412, .029233, .031198, .033301,
.035594, .038092, .040767, .04372, .046918, .050246, .053974,
.058009, .061976, .066586, .071537, .076209, .081856, .087998,
.093821, .10113, .10913, .11731, .12724, .13821, .15025, .1639,
.17807, .19472, .21356, .23496, .25758, .28387, .31389, .34104,
.37469, .40989, .43309, .46845, .5042, .5023, .52981, .55275,
.51075, .51976, .52457, .44779, .44721, .4503, .4243, .45244,
.49491, .55399, .39021, .24802, .2501, .2618, .27475, .28879,
.31317, .33643, .36257, .4018, .43275, .46525, .53333, .56599,
.60557, .70142, .74194, .77736, .88567, .91182, .93294, .98407,
.98772, .99176, .9995, 1.2405, 1.3602, 1.338, 1.3255, 1.3267,
1.3404, 1.3634, 1.3967, 1.4407, 1.4961, 1.5603, 1.6328, 1.7153,
1.8094, 1.9091, 2.018, 2.1367, 2.264, 2.4035, 2.5562, 2.7179,
2.9017, 3.1052, 3.3304, 3.5731, 3.8488, 4.1553, 4.4769, 4.7818,
5.1711, 5.5204, 5.9516, 6.4097, 6.8899, 7.1118, 7.5469, 7.9735,
7.9511, 8.3014, 8.6418, 8.4757, 8.8256, 9.2294, 9.6923, 10.033,
10.842, 11.851, 11.78, 8.8435, 9.1381, 9.5956, 10.076, 10.629,
11.22, 11.883, 12.69, 13.163, 13.974, 14.846, 16.027, 17.053,
18.148, 19.715, 20.907, 22.163, 23.956, 25.235, 26.566, 27.94,
29.576, 30.956, 32.432, 35.337, 39.911, 41.128, 42.625, 44.386,
```

```
46.369, 48.619, 51.031, 53.674, 56.825, 59.921, 63.286, 66.929,
70.859, 75.081, 79.618, 84.513, 89.739, 95.335, 101.35, 107.76,
114.63, 121.98, 129.87, 138.3, 147.34, 157.04, 167.56, 178.67,
190.61, 203.43, 217.19, 231.99, 247.88, 264.98, 283.37, 303.17,
324.49, 347.47, 372.25, 398.98, 427.85, 459.06, 492.8, 529.31,
568.89, 611.79, 658.35, 708.91, 763.87, 823.65, 888.72, 959.58,
1036.8, 1121.8, 1213.9, 1314.3, 1423.8, 1543., 1672.8, 1813.4,
1966.1, 2131.4, 2309.5, 2499.3, 2705., 2925.7, 3161.6, 3411.3,
3611.5, 3889.2, 4191.1, 4519.3, 4877.9, 5272.9, 5712.9, 6142.7,
6719.6, 7385., 8145., 8977.7, 9831.9, 10827., 11934., 13063.,
14434., 15878., 17591., 19435., 21510., 23835., 26835., 29740.,
32878., 36305., 39830., 43273., 46931., 50499., 49586., 51598.,
53429., 54619., 55081., 55102., 54485., 53487., 52042., 42689.,
42607., 44020., 47994., 54169., 53916., 55808., 56642., 46049.,
44243., 32929., 30658., 21963., 20835., 15962., 13679., 17652.,
19680., 22388., 25625., 29184., 32520., 35720., 38414., 40523.,
49228., 48173., 45678., 41768., 37600., 41313., 42654., 44465.,
55736., 56630., 65409., 63308., 66572., 61845., 60379., 56777.,
51920., 46601., 41367., 36529., 32219., 28470., 25192., 22362.,
19907., 17772., 15907., 14273., 12835., 11567., 10445., 9450.2,
8565.1, 7776., 7070.8, 6439.2, 5872.3, 5362.4, 4903., 4488.3,
4113.4, 3773.8, 3465.8, 3186.1, 2931.7, 2700.1, 2488.8, 2296.,
2119.8, 1958.6, 1810.9, 1675.6, 1551.4, 1437.3, 1332.4, 1236.,
1147.2, 1065.3, 989.86, 920.22, 855.91, 796.48, 741.53, 690.69,
643.62, 600.02, 559.6, 522.13, 487.35, 455.06, 425.08, 397.21,
371.3, 347.2, 324.78, 303.9, 284.46, 266.34, 249.45, 233.7,
219.01, 205.3, 192.5, 180.55, 169.38, 158.95, 149.2, 140.07,
131.54, 123.56, 116.09, 109.09, 102.54, 96.405, 90.655, 85.266,
80.213, 75.475, 71.031, 66.861, 62.948, 59.275, 55.827, 52.587,
49.544, 46.686, 43.998, 41.473, 39.099, 36.867, 34.768, 32.795,
30.939, 29.192, 27.546, 25.998, 24.539, 23.164, 21.869, 20.65,
19.501, 18.419, 17.399, 16.438, 15.532, 14.678, 13.874, 13.115,
12.4, 11.726, 11.088, 10.488, 9.921, 9.3846, 8.8784, 8.3996,
7.9469, 7.5197, 7.1174, 6.738, 6.379, 6.0409, 5.7213, 5.419,
5.1327, 4.8611, 4.6046, 4.3617, 4.1316, 3.9138, 3.7077, 3.5125,
3.3281, 3.1536, 2.9885, 2.8323, 2.6846, 2.5447, 2.4124, 2.2871,
2.1686, 2.0564, 1.9501, 1.8495, 1.7543, 1.6641, 1.5787, 1.4978,
1.4212, 1.3486, 1.2799, 1.2147, 1.1529, 1.0943, 1.0388, .98602,
.93596, .8886, .84352, .80078, .76029, .722, .68585, .65161,
.61901, .58808, .55854, .53044, .5039, .47853, .45459, .43173,
.41008, .38965, .37021, .35186, .33444, .31797, .30234, .28758,
.2736, .26036, .24764, .2357, .22431, .21342, .20295, .19288,
.18334, .17444, .166, .15815, .15072, .14348, .13674, .13015,
.12399, .11807, .11231, .10689, .10164, .096696, .091955,
.087476, .083183, .079113, .075229, .071536, .068026, .064698,
.06154, .058544, .055699, .052997, .050431, .047993, .045676,
.043475, .041382, .039392, .037501, .035702, .033991, .032364,
.030817, .029345, .027945, .026613, .025345, .024139, .022991,
.021899, .02086, .019871, .018929, .018033, .01718, .016368,
.015595, .014859, .014158, .013491, .012856, .012251, .011675,
.011126, .010604, .010107, .0096331, .009182, .0087523, .0083431,
.0079533, .0075821, .0072284, .0068915, .0065706, .0062649,
.0059737, .0056963, .005432, .0051802, .0049404, .0047118,
.0044941, .0042867, .0040891, .0039009, .0037216, .0035507,
.003388, .0032329, .0030852, .0029445, .0028105, .0026829,
.0025613, .0024455, .0023353, .0022303, .0021304, .0020353,
.0019448, .0018587, .0017767, .0016988, .0016247, .0015543,
.0014874, .0014238, .0013635, .0013062, .0012519, .0012005,
.0011517, .0011057, .0010621, .001021, 9.8233e-4, 9.4589e-4,
9.1167e-4, 8.7961e-4, 8.4964e-4, 8.2173e-4, 7.9582e-4, 7.7189e-4,
7.499e-4, 7.2983e-4, 7.1167e-4, 6.9542e-4, 6.8108e-4, 6.6866e-4,
6.5819e-4, 6.4971e-4, 6.4328e-4, 6.3895e-4, 6.3681e-4, 6.3697e-4,
6.3956e-4, 6.4472e-4, 6.5266e-4, 6.6359e-4, 6.778e-4, 6.9563e-4,
7.1749e-4, 7.4392e-4, 7.7556e-4, 8.1028e-4, 8.4994e-4, 8.8709e-4,
9.3413e-4, 9.6953e-4, .0010202, .0010738, .0010976, .0011507,
.0011686, .0012264, .001291, .0013346, .0014246, .0015293,
.0016359, .0017824, .0019255, .0020854, .002247, .0024148,
.0026199, .0027523, .0029704, .0030702, .0033047, .0035013,
.0037576, .0040275, .0043089, .0046927, .0049307, .0053486,
.0053809, .0056699, .0059325, .0055488, .005634, .0056392,
.004946, .0048855, .0048208, .0044386, .0045498, .0046377,
.0048939, .0052396, .0057324, .0060859, .0066906, .0071148,
.0077224, .0082687, .008769, .0084471, .008572, .0087729,
.008775, .0090742, .0080704, .0080288, .0085747, .0086087,
.0086408, .0088752, .0089381, .0089757, .0093532, .0092824,
.0092566, .0092645, .0092735, .009342, .0095806, .0097991,
.010213, .010611, .011129, .011756, .013237, .01412, .015034,
.015936, .01682, .018597, .019315, .019995, .020658, .021289,
.022363, .022996, .023716, .024512, .025434, .026067, .027118,
.028396, .029865, .031442, .033253, .03525, .037296, .039701,
.042356, .045154, .048059, .051294, .054893, .058636, .061407,
.065172, .068974, .072676, .073379, .076547, .079556, .079134,
.082308, .085739, .090192, .09359, .099599, .10669, .11496,
.1244, .13512, .14752, .14494, .15647, .1668, .17863, .19029,
.20124, .20254, .21179, .21982, .21625, .22364, .23405, .23382,
.2434, .25708, .26406, .27621, .28909, .30395, .31717, .33271,
.3496, .36765, .38774, .40949, .446, .46985, .49846, .5287, .562,
```

```
    .59841, .64598, .68834, .7327, .78978, .8373, .88708, .94744,
    1.0006, 1.0574, 1.1215, 1.1856, 1.2546, 1.3292, 1.4107, 1.4974,
    1.5913, 1.6931, 1.8028, 1.9212, 2.0492, 2.1874, 2.3365, 2.4978,
    2.6718, 2.8588, 3.062, 3.2818, 3.5188, 3.7752, 4.0527, 4.3542,
    4.6782, 5.0312, 5.4123, 5.8246, 6.2639, 6.7435, 7.2636, 7.8064,
    8.4091, 9.0696, 9.7677, 10.548, 11.4, 12.309, 13.324, 14.284,
    15.445, 16.687, 18.019, 19.403, 20.847, 22.366, 23.925, 25.537,
    27.213, 28.069, 29.864, 31.829, 33.988, 35.856, 38.829, 42.321,
    46.319, 50.606, 55.126, 59.126, 64.162, 68.708, 74.615, 81.176,
    87.739, 95.494, 103.83, 113.38, 123.99, 135.8, 148.7, 162.58,
    176.32, 192.6, 211.47, 232.7, 252.64, 277.41, 305.38, 333.44,
    366.42, 402.66, 442.14, 484.53, 526.42, 568.15, 558.78, 582.6,
    600.98, 613.94, 619.44, 618.24, 609.84, 595.96, 484.86, 475.59,
    478.49, 501.56, 552.19, 628.44, 630.39, 658.92, 671.96, 562.7,
    545.88, 423.43, 400.14, 306.59, 294.13, 246.8, 226.51, 278.21,
    314.39, 347.22, 389.13, 433.16, 477.48, 521.67, 560.54, 683.6,
    696.37, 695.91, 683.1, 658.24, 634.89, 698.85, 742.87, 796.66,
    954.49, 1009.5, 1150.5, 1179.1, 1267.9, 1272.4, 1312.7, 1330.4,
    1331.6, 1315.8, 1308.3, 1293.3, 1274.6, 1249.5, 1213.2, 1172.1,
    1124.4, 930.33, 893.36, 871.27, 883.54, 940.76, 1036., 1025.6,
    1053.1, 914.51, 894.15, 865.03, 670.63, 508.41, 475.15, 370.85,
    361.06, 319.38, 312.75, 331.87, 367.13, 415., 467.94, 525.49,
    578.41, 624.66, 794.82, 796.97, 780.29, 736.49, 670.18, 603.75,
    659.67, 679.8, 857.12, 884.05, 900.65, 1046.1, 1141.9, 1083.,
    1089.2, 1e3, 947.08, 872.31, 787.91, 704.75, 624.93, 553.68,
    489.91, 434.21, 385.64, 343.3, 306.42, 274.18, 245.94, 221.11,
    199.23, 179.88, 162.73, 147.48, 133.88, 121.73, 110.86, 101.1,
    92.323, 84.417, 77.281, 70.831, 64.991, 59.694, 54.884, 50.509,
    46.526, 42.893, 39.58, 36.549, 33.776, 31.236, 28.907, 26.77,
    24.805, 23., 21.339, 19.81, 18.404, 17.105, 15.909, 14.801,
    13.778, 12.83, 11.954, 11.142, 10.389, 9.691, 9.0434, 8.4423,
    7.8842, 7.3657, 6.8838, 6.4357, 6.0189, 5.6308, 5.2696, 4.9332,
    4.6198, 4.3277, 4.0553, 3.8012, 3.5639, 3.3424, 3.1355, 2.9422,
    2.7614, 2.5924, 2.4343, 2.2864, 2.148, 2.0184, 1.8971, 1.7835,
    1.677, 1.5773, 1.4838, 1.3961, 1.3139, 1.2369, 1.1645, 1.0966,
    1.0329, .97309, .91686, .86406, .81439, .76767, .72381, .68252,
    .64359, .60695, .57247, .54008, .50957, .48092, .45401, .42862,
    .40465, .38202, .36072, .34052, .3216, .30386, .28711, .27135,
    .25651, .24252, .2293, .21689, .20517, .19416, .18381, .17396,
    .16469
  };

  static double co2230[2001] = { 2.743e-5, 2.8815e-5, 3.027e-5, 3.1798e-5,
    3.3405e-5, 3.5094e-5, 3.6869e-5, 3.8734e-5, 4.0694e-5, 4.2754e-5,
    4.492e-5, 4.7196e-5, 4.9588e-5, 5.2103e-5, 5.4747e-5, 5.7525e-5,
    6.0446e-5, 6.3516e-5, 6.6744e-5, 7.0137e-5, 7.3704e-5, 7.7455e-5,
    8.1397e-5, 8.5543e-5, 8.9901e-5, 9.4484e-5, 9.9302e-5, 1.0437e-4,
    1.097e-4, 1.153e-4, 1.2119e-4, 1.2738e-4, 1.3389e-4, 1.4074e-4,
    1.4795e-4, 1.5552e-4, 1.6349e-4, 1.7187e-4, 1.8068e-4, 1.8995e-4,
    1.997e-4, 2.0996e-4, 2.2075e-4, 2.321e-4, 2.4403e-4, 2.5659e-4,
    2.698e-4, 2.837e-4, 2.9832e-4, 3.137e-4, 3.2988e-4, 3.4691e-4,
    3.6483e-4, 3.8368e-4, 4.0351e-4, 4.2439e-4, 4.4635e-4, 4.6947e-4,
    4.9379e-4, 5.1939e-4, 5.4633e-4, 5.7468e-4, 6.0452e-4, 6.3593e-4,
    6.69e-4, 7.038e-4, 7.4043e-4, 7.79e-4, 8.1959e-4, 8.6233e-4,
    9.0732e-4, 9.5469e-4, .0010046, .0010571, .0011124, .0011706,
    .0012319, .0012964, .0013644, .001436, .0015114, .0015908,
    .0016745, .0017625, .0018553, .0019531, .002056, .0021645,
    .0022788, .0023992, .002526, .0026596, .0028004, .0029488,
    .0031052, .0032699, .0034436, .0036265, .0038194, .0040227,
    .0042369, .0044628, .0047008, .0049518, .0052164, .0054953,
    .0057894, .0060995, .0064265, .0067713, .007135, .0075184,
    .0079228, .0083494, .0087993, .0092738, .0097745, .010303,
    .01086, .011448, .012068, .012722, .013413, .014142, .014911,
    .015723, .01658, .017484, .018439, .019447, .020511, .021635,
    .022821, .024074, .025397, .026794, .02827, .029829, .031475,
    .033215, .035052, .036994, .039045, .041213, .043504, .045926,
    .048485, .05119, .05405, .057074, .060271, .063651, .067225,
    .071006, .075004, .079233, .083708, .088441, .093449, .098749,
    .10436, .11029, .11657, .12322, .13026, .13772, .14561, .15397,
    .16282, .1722, .18214, .19266, .20381, .21563, .22816, .24143,
    .2555, .27043, .28625, .30303, .32082, .3397, .35972, .38097,
    .40352, .42746, .45286, .47983, .50847, .53888, .57119, .6055,
    .64196, .6807, .72187, .76564, .81217, .86165, .91427, .97025,
    1.0298, 1.0932, 1.1606, 1.2324, 1.3088, 1.3902, 1.477, 1.5693,
    1.6678, 1.7727, 1.8845, 2.0038, 2.131, 2.2666, 2.4114, 2.5659,
    2.7309, 2.907, 3.0951, 3.2961, 3.5109, 3.7405, 3.986, 4.2485,
    4.5293, 4.8299, 5.1516, 5.4961, 5.8651, 6.2605, 6.6842, 7.1385,
    7.6256, 8.1481, 8.7089, 9.3109, 9.9573, 10.652, 11.398, 12.2,
    13.063, 13.992, 14.99, 16.064, 17.222, 18.469, 19.813, 21.263,
    22.828, 24.516, 26.34, 28.31, 30.437, 32.738, 35.226, 37.914,
    40.824, 43.974, 47.377, 51.061, 55.011, 59.299, 63.961, 69.013,
    74.492, 80.444, 86.919, 93.836, 101.23, 109.25, 117.98, 127.47,
    137.81, 149.07, 161.35, 174.75, 189.42, 205.49, 223.02, 242.26,
    263.45, 286.75, 311.94, 340.01, 370.86, 404.92, 440.44, 480.27,
    525.17, 574.71, 626.22, 686.8, 754.38, 827.07, 913.38, 1011.7,
    1121.5, 1161.6, 1289.5, 1432.2, 1595.4, 1777., 1983.3, 2216.1,
    2485.7, 2788.3, 3101.5, 3481., 3902.1, 4257.1, 4740., 5272.8,
```

```
5457.9, 5946.2, 6505.3, 6668.4, 7302.4, 8061.6, 9015.8, 9908.3,
11613., 13956., 3249.6, 3243., 2901.5, 2841.3, 2729.6, 2558.2,
1797.8, 1583.2, 1386., 1233.5, 787.74, 701.46, 761.66, 767.21,
722.83, 1180.6, 1332.1, 1461.6, 2032.9, 2166., 2255.9, 2294.7,
2587.2, 2396.5, 2122.4, 12553., 10784., 9832.5, 8827.3, 8029.1,
7377.9, 7347.1, 6783.8, 6239.1, 5721.1, 5503., 4975.1, 4477.8,
4021.3, 3676.8, 3275.3, 2914.9, 2597.4, 2328.2, 2075.4, 1857.6,
1663.6, 1493.3, 1343.8, 1213.3, 1095.6, 1066.5, 958.91, 865.15,
783.31, 714.35, 650.77, 593.98, 546.2, 499.9, 457.87, 421.75,
387.61, 355.25, 326.62, 299.7, 275.21, 253.17, 232.83, 214.31,
197.5, 182.08, 167.98, 155.12, 143.32, 132.5, 122.58, 113.48,
105.11, 97.415, 90.182, 83.463, 77.281, 71.587, 66.341, 61.493,
57.014, 53.062, 49.21, 45.663, 42.38, 39.348, 36.547, 33.967,
31.573, 29.357, 27.314, 25.415, 23.658, 22.03, 20.524, 19.125,
17.829, 16.627, 15.511, 14.476, 13.514, 12.618, 11.786, 11.013,
10.294, 9.6246, 9.0018, 8.4218, 7.8816, 7.3783, 6.9092, 6.4719,
6.0641, 5.6838, 5.3289, 4.998, 4.6893, 4.4014, 4.1325, 3.8813,
3.6469, 3.4283, 3.2241, 3.035, 2.8576, 2.6922, 2.5348, 2.3896,
2.2535, 2.1258, 2.0059, 1.8929, 1.7862, 1.6854, 1.5898, 1.4992,
1.4017, 1.3218, 1.2479, 1.1809, 1.1215, 1.0693, 1.0116, .96016,
.9105, .84859, .80105, .74381, .69982, .65127, .60899, .57843,
.54592, .51792, .49336, .47155, .45201, .43426, .41807, .40303,
.38876, .3863, .37098, .35492, .33801, .32032, .30341, .29874,
.29193, .28689, .29584, .29155, .29826, .29195, .29287, .2904,
.28199, .27709, .27162, .26622, .26133, .25676, .25235, .23137,
.22365, .21519, .20597, .19636, .18699, .16485, .16262, .16643,
.17542, .18198, .18631, .16759, .16338, .13505, .1267, .10053,
.092554, .074093, .062159, .055523, .054849, .05401, .05528,
.058982, .07952, .08647, .093244, .099285, .10393, .10661,
.12072, .11417, .10396, .093265, .089137, .088909, .10902,
.11277, .13625, .13565, .14907, .14167, .1428, .13744, .12768,
.11382, .10244, .091686, .08109, .071739, .063616, .056579,
.050504, .045251, .040689, .036715, .033237, .030181, .027488,
.025107, .022998, .021125, .01946, .017979, .016661, .015489,
.014448, .013526, .012712, .011998, .011375, .010839, .010384,
.010007, .0097053, .0094783, .0093257, .0092489, .0092504,
.0093346, .0095077, .0097676, .01012, .01058, .011157, .011844,
.012672, .013665, .014766, .015999, .017509, .018972, .020444,
.022311, .023742, .0249, .025599, .026981, .026462, .025143,
.025066, .022814, .020458, .020026, .019142, .020189, .022371,
.024163, .023728, .02199, .019506, .018591, .015576, .012784,
.011744, .0094777, .0079148, .0070652, .006986, .0071758,
.008086, .0098025, .01087, .013609, .016764, .018137, .021061,
.023498, .023576, .023965, .022828, .021519, .021283, .023364,
.026457, .029782, .030856, .033486, .035515, .035543, .036558,
.037198, .037472, .037045, .037284, .03777, .038085, .038366,
.038526, .038282, .038915, .037697, .035667, .032941, .031959,
.028692, .025918, .024596, .025592, .027873, .028935, .02984,
.028148, .025305, .021912, .020454, .016732, .013357, .01205,
.009731, .0079881, .0077704, .0074387, .0083895, .0096776,
.010326, .01293, .015955, .019247, .020145, .02267, .024231,
.024184, .022131, .019784, .01955, .01971, .022119, .025116,
.027978, .028107, .029808, .030701, .029164, .028551, .027286,
.024946, .023259, .020982, .019221, .017471, .015643, .014074,
.01261, .011301, .010116, .0090582, .0081036, .0072542, .0065034,
.0058436, .0052571, .0047321, .0042697, .0038607, .0034977,
.0031747, .0028864, .0026284, .002397, .002189, .0020017,
.0018326, .0016798, .0015414, .0014159, .0013019, .0011983,
.0011039, .0010177, 9.391e-4, 8.6717e-4, 8.0131e-4, 7.4093e-4,
6.8553e-4, 6.3464e-4, 5.8787e-4, 5.4487e-4, 5.0533e-4, 4.69e-4,
4.3556e-4, 4.0474e-4, 3.7629e-4, 3.5e-4, 3.2569e-4, 3.032e-4,
2.8239e-4, 2.6314e-4, 2.4535e-4, 2.2891e-4, 2.1374e-4, 1.9975e-4,
1.8685e-4, 1.7498e-4, 1.6406e-4, 1.5401e-4, 1.4479e-4, 1.3633e-4,
1.2858e-4, 1.2148e-4, 1.1499e-4, 1.0907e-4, 1.0369e-4, 9.8791e-5,
9.4359e-5, 9.0359e-5, 8.6766e-5, 8.3555e-5, 8.0703e-5, 7.8192e-5,
7.6003e-5, 7.4119e-5, 7.2528e-5, 7.1216e-5, 7.0171e-5, 6.9385e-5,
6.8848e-5, 6.8554e-5, 6.8496e-5, 6.8669e-5, 6.9069e-5, 6.9694e-5,
7.054e-5, 7.1608e-5, 7.2896e-5, 7.4406e-5, 7.6139e-5, 7.8097e-5,
8.0283e-5, 8.2702e-5, 8.5357e-5, 8.8255e-5, 9.1402e-5, 9.4806e-5,
9.8473e-5, 1.0241e-4, 1.0664e-4, 1.1115e-4, 1.1598e-4, 1.2112e-4,
1.2659e-4, 1.3241e-4, 1.3859e-4, 1.4515e-4, 1.521e-4, 1.5947e-4,
1.6728e-4, 1.7555e-4, 1.8429e-4, 1.9355e-4, 2.0334e-4, 2.1369e-4,
2.2463e-4, 2.3619e-4, 2.4841e-4, 2.6132e-4, 2.7497e-4, 2.8938e-4,
3.0462e-4, 3.2071e-4, 3.3771e-4, 3.5567e-4, 3.7465e-4, 3.947e-4,
4.1588e-4, 4.3828e-4, 4.6194e-4, 4.8695e-4, 5.1338e-4, 5.4133e-4,
5.7087e-4, 6.0211e-4, 6.3515e-4, 6.701e-4, 7.0706e-4, 7.4617e-4,
7.8756e-4, 8.3136e-4, 8.7772e-4, 9.2681e-4, 9.788e-4, .0010339,
.0010922, .001154, .0012195, .0012889, .0013626, .0014407,
.0015235, .0016114, .0017048, .0018038, .001909, .0020207,
.0021395, .0022657, .0023998, .0025426, .0026944, .002856,
.0030281, .0032114, .0034068, .003615, .0038371, .004074,
.004327, .0045971, .0048857, .0051942, .0055239, .0058766,
.0062538, .0066573, .0070891, .007551, .0080455, .0085747,
.0091412, .0097481, .010397, .011092, .011837, .012638, .013495,
.014415, .01541, .016475, .017621, .018857, .020175, .02162,
.023185, .024876, .02672, .028732, .030916, .033319, .035939,
.038736, .041847, .04524, .048715, .052678, .056977, .061203,
```

```
.066184, .07164, .076952, .083477, .090674, .098049, .10697,
.1169, .1277, .14011, .15323, .1684, .18601, .20626, .22831,
.25417, .28407, .31405, .34957, .38823, .41923, .46026, .50409,
.51227, .54805, .57976, .53818, .55056, .557, .46741, .46403,
.4636, .42265, .45166, .49852, .56663, .34306, .17779, .17697,
.18346, .19129, .20014, .21778, .23604, .25649, .28676, .31238,
.33856, .39998, .4288, .46568, .56654, .60786, .64473, .76466,
.7897, .80778, .86443, .85736, .84798, .84157, 1.1385, 1.2446,
1.1923, 1.1552, 1.1338, 1.1266, 1.1292, 1.1431, 1.1683, 1.2059,
1.2521, 1.3069, 1.3712, 1.4471, 1.5275, 1.6165, 1.7145, 1.8189,
1.9359, 2.065, 2.2007, 2.3591, 2.5362, 2.7346, 2.9515, 3.2021,
3.4851, 3.7935, 4.0694, 4.4463, 4.807, 5.2443, 5.7178, 6.2231,
6.4796, 6.9461, 7.4099, 7.3652, 7.7182, 8.048, 7.7373, 8.0363,
8.3855, 8.8044, 9.0257, 9.8574, 10.948, 10.563, 6.8979, 7.0744,
7.4121, 7.7663, 8.1768, 8.6243, 9.1437, 9.7847, 10.182, 10.849,
11.572, 12.602, 13.482, 14.431, 15.907, 16.983, 18.11, 19.884,
21.02, 22.18, 23.355, 24.848, 25.954, 27.13, 30.186, 34.893,
35.682, 36.755, 38.111, 39.703, 41.58, 43.606, 45.868, 48.573,
51.298, 54.291, 57.559, 61.116, 64.964, 69.124, 73.628, 78.471,
83.683, 89.307, 95.341, 101.84, 108.83, 116.36, 124.46, 133.18,
142.57, 152.79, 163.69, 175.43, 188.11, 201.79, 216.55, 232.51,
249.74, 268.38, 288.54, 310.35, 333.97, 359.55, 387.26, 417.3,
449.88, 485.2, 523.54, 565.14, 610.28, 659.31, 712.56, 770.43,
833.36, 901.82, 976.36, 1057.6, 1146.8, 1243.8, 1350., 1466.3,
1593.6, 1732.7, 1884.1, 2049.1, 2228.2, 2421.9, 2629.4, 2853.7,
3094.4, 3351.1, 3622.3, 3829.8, 4123.1, 4438.3, 4777.2, 5144.1,
5545.4, 5990.5, 6404.5, 6996.8, 7687.6, 8482.9, 9349.4, 10203.,
11223., 12358., 13493., 14916., 16416., 18236., 20222., 22501.,
25102., 28358., 31707., 35404., 39538., 43911., 48391., 53193.,
58028., 58082., 61276., 64193., 66294., 67480., 67921., 67423.,
66254., 64341., 51737., 51420., 53072., 58145., 66195., 65358.,
67377., 67869., 53509., 50553., 35737., 32425., 21704., 19974.,
14457., 12142., 16798., 19489., 23049., 27270., 31910., 36457.,
40877., 44748., 47876., 59793., 58626., 55454., 50337., 44893.,
50228., 52216., 54747., 69541., 70455., 81014., 77694., 80533.,
73953., 70927., 65539., 59002., 52281., 45953., 40292., 35360.,
31124., 27478., 24346., 21647., 19308., 17271., 15491., 13927.,
12550., 11331., 10250., 9288.8, 8431.4, 7664.9, 6978.3, 6361.8,
5807.4, 5307.7, 4856.8, 4449., 4079.8, 3744.9, 3440.8, 3164.2,
2912.3, 2682.7, 2473., 2281.4, 2106., 1945.3, 1797.9, 1662.5,
1538.1, 1423.6, 1318.1, 1221., 1131.5, 1049., 972.99, 902.87,
838.01, 777.95, 722.2, 670.44, 622.35, 577.68, 536.21, 497.76,
462.12, 429.13, 398.61, 370.39, 344.29, 320.16, 297.85, 277.2,
258.08, 240.38, 223.97, 208.77, 194.66, 181.58, 169.43, 158.15,
147.67, 137.92, 128.86, 120.44, 112.6, 105.3, 98.499, 92.166,
86.264, 80.763, 75.632, 70.846, 66.381, 62.213, 58.321, 54.685,
51.288, 48.114, 45.145, 42.368, 39.772, 37.341, 35.065, 32.937,
30.943, 29.077, 27.33, 25.693, 24.158, 22.717, 21.367, 20.099,
18.909, 17.792, 16.744, 15.761, 14.838, 13.971, 13.157, 12.393,
11.676, 11.003, 10.369, 9.775, 9.2165, 8.6902, 8.1963, 7.7314,
7.2923, 6.8794, 6.4898, 6.122, 5.7764, 5.4525, 5.1484, 4.8611,
4.5918, 4.3379, 4.0982, 3.8716, 3.6567, 3.4545, 3.2634, 3.0828,
2.9122, 2.7512, 2.5993, 2.4561, 2.3211, 2.1938, 2.0737, 1.9603,
1.8534, 1.7525, 1.6572, 1.5673, 1.4824, 1.4022, 1.3265, 1.2551,
1.1876, 1.1239, 1.0637, 1.0069, .9532, .90248, .85454, .80921,
.76631, .72569, .6872, .65072, .61635, .5836, .55261, .52336,
.49581, .46998, .44559, .42236, .40036, .37929, .35924, .34043,
.32238, .30547, .28931, .27405, .25975, .24616, .23341, .22133,
.20997, .19924, .18917, .17967, .17075, .16211, .15411, .14646,
.13912, .13201, .12509, .11857, .11261, .10698, .10186, .097039,
.092236, .087844, .083443, .07938, .075452, .071564, .067931,
.064389, .061078, .057901, .054921, .052061, .049364, .046789,
.04435, .042044, .039866, .037808, .035863, .034023, .032282,
.030634, .029073, .027595, .026194, .024866, .023608, .022415,
.021283, .02021, .019193, .018228, .017312, .016443, .015619,
.014837, .014094, .01339, .012721, .012086, .011483, .010911,
.010368, .009852, .0093623, .0088972, .0084556, .0080362,
.0076379, .0072596, .0069003, .006559, .0062349, .0059269,
.0056344, .0053565, .0050925, .0048417, .0046034, .004377,
.0041618, .0039575, .0037633, .0035788, .0034034, .0032368,
.0030785, .002928, .0027851, .0026492, .0025201, .0023975,
.0022809, .0021701, .0020649, .0019649, .0018699, .0017796,
.0016938, .0016122, .0015348, .0014612, .0013913, .001325,
.0012619, .0012021, .0011452, .0010913, .0010401, 9.9149e-4,
9.454e-4, 9.0169e-4, 8.6024e-4, 8.2097e-4, 7.8377e-4, 7.4854e-4,
7.1522e-4, 6.8371e-4, 6.5393e-4, 6.2582e-4, 5.9932e-4, 5.7435e-4,
5.5087e-4, 5.2882e-4, 5.0814e-4, 4.8881e-4, 4.7076e-4, 4.5398e-4,
4.3843e-4, 4.2407e-4, 4.109e-4, 3.9888e-4, 3.88e-4, 3.7826e-4,
3.6963e-4, 3.6213e-4, 3.5575e-4, 3.505e-4, 3.464e-4, 3.4346e-4,
3.4173e-4, 3.4125e-4, 3.4206e-4, 3.4424e-4, 3.4787e-4, 3.5303e-4,
3.5986e-4, 3.6847e-4, 3.7903e-4, 3.9174e-4, 4.0681e-4, 4.2455e-4,
4.4527e-4, 4.6942e-4, 4.9637e-4, 5.2698e-4, 5.5808e-4, 5.9514e-4,
6.2757e-4, 6.689e-4, 7.1298e-4, 7.3955e-4, 7.8403e-4, 8.0449e-4,
8.5131e-4, 9.0256e-4, 9.3692e-4, .0010051, .0010846, .0011678,
.001282, .0014016, .0015355, .0016764, .0018272, .0020055,
.0021455, .0023421, .0024615, .0026786, .0028787, .0031259,
.0034046, .0036985, .0040917, .0043902, .0048349, .0049531,
```

```
    .0052989, .0056148, .0052452, .0053357, .005333, .0045069,
    .0043851, .004253, .003738, .0038084, .0039013, .0041505,
    .0045372, .0050569, .0054507, .0061267, .0066122, .0072449,
    .0078012, .0082651, .0076538, .0076573, .0076806, .0075227,
    .0076269, .0063758, .006254, .0067749, .0067909, .0068231,
    .0072143, .0072762, .0072954, .007679, .0075107, .0073658,
    .0072441, .0071074, .0070378, .007176, .0072472, .0075844,
    .0079291, .008412, .0090165, .010688, .011535, .012375, .013166,
    .013895, .015567, .016011, .016392, .016737, .017043, .017731,
    .018031, .018419, .018877, .019474, .019868, .020604, .021538,
    .022653, .023869, .025288, .026879, .028547, .030524, .03274,
    .035132, .03769, .040567, .043793, .047188, .049962, .053542,
    .057205, .060776, .061489, .064419, .067124, .065945, .068487,
    .071209, .074783, .077039, .082444, .08902, .09692, .10617,
    .11687, .12952, .12362, .13498, .14412, .15492, .16519, .1744,
    .17096, .17714, .18208, .17363, .17813, .18564, .18295, .19045,
    .20252, .20815, .21844, .22929, .24229, .25321, .26588, .2797,
    .29465, .31136, .32961, .36529, .38486, .41027, .43694, .4667,
    .49943, .54542, .58348, .62303, .67633, .71755, .76054, .81371,
    .85934, .90841, .96438, 1.0207, 1.0821, 1.1491, 1.2226, 1.3018,
    1.388, 1.4818, 1.5835, 1.6939, 1.8137, 1.9435, 2.0843, 2.237,
    2.4026, 2.5818, 2.7767, 2.9885, 3.2182, 3.4679, 3.7391, 4.0349,
    4.3554, 4.7053, 5.0849, 5.4986, 5.9436, 6.4294, 6.9598, 7.5203,
    8.143, 8.8253, 9.5568, 10.371, 11.267, 12.233, 13.31, 14.357,
    15.598, 16.93, 18.358, 19.849, 21.408, 23.04, 24.706, 26.409,
    28.153, 28.795, 30.549, 32.43, 34.49, 36.027, 38.955, 42.465,
    46.565, 50.875, 55.378, 59.002, 63.882, 67.949, 73.693, 80.095,
    86.403, 94.264, 102.65, 112.37, 123.3, 135.54, 149.14, 163.83,
    179.17, 196.89, 217.91, 240.94, 264.13, 292.39, 324.83, 358.21,
    397.16, 440.5, 488.6, 541.04, 595.3, 650.43, 652.03, 688.74,
    719.47, 743.54, 757.68, 762.35, 756.43, 741.42, 595.43, 580.97,
    580.83, 605.68, 667.88, 764.49, 759.93, 789.12, 798.17, 645.66,
    615.65, 455.05, 421.09, 306.45, 289.14, 235.7, 215.52, 274.57,
    316.53, 357.73, 409.89, 465.06, 521.84, 579.02, 630.64, 794.46,
    813., 813.56, 796.25, 761.57, 727.97, 812.14, 866.75, 932.5,
    1132.8, 1194.8, 1362.2, 1387.2, 1482.3, 1479.7, 1517.9, 1533.1,
    1534.2, 1523.3, 1522.5, 1515.5, 1505.2, 1486.5, 1454., 1412.,
    1358.8, 1107.8, 1060.9, 1033.5, 1048.2, 1122.4, 1248.9, 1227.1,
    1255.4, 1058.9, 1020.7, 970.59, 715.24, 512.56, 468.47, 349.3,
    338.26, 299.22, 301.26, 332.38, 382.08, 445.49, 515.87, 590.85,
    662.3, 726.05, 955.59, 964.11, 945.17, 891.48, 807.11, 720.9,
    803.36, 834.46, 1073.9, 1107.1, 1123.6, 1296., 1393.7, 1303.1,
    1284.3, 1161.8, 1078.8, 976.13, 868.72, 767.4, 674.72, 593.73,
    523.12, 462.24, 409.75, 364.34, 325., 290.73, 260.76, 234.46,
    211.28, 190.78, 172.61, 156.44, 142.01, 129.12, 117.57, 107.2,
    97.877, 89.47, 81.882, 75.021, 68.807, 63.171, 58.052, 53.396,
    49.155, 45.288, 41.759, 38.531, 35.576, 32.868, 30.384, 28.102,
    26.003, 24.071, 22.293, 20.655, 19.147, 17.756, 16.476, 15.292,
    14.198, 13.183, 12.241, 11.367, 10.554, 9.7989, 9.0978, 8.4475,
    7.845, 7.2868, 6.7704, 6.2927, 5.8508, 5.4421, 5.064, 4.714,
    4.3902, 4.0902, 3.8121, 3.5543, 3.315, 3.093, 2.8869, 2.6953,
    2.5172, 2.3517, 2.1977, 2.0544, 1.9211, 1.7969, 1.6812, 1.5735,
    1.4731, 1.3794, 1.2921, 1.2107, 1.1346, 1.0637, .99744, .93554,
    .87771, .82368, .77313, .72587, .6816, .64014, .60134, .565,
    .53086, .49883, .46881, .44074, .4144, .38979, .36679, .34513,
    .32474, .30552, .28751, .27045, .25458, .23976, .22584, .21278,
    .20051, .18899, .17815, .16801, .15846, .14954, .14117, .13328,
    .12584
  };

  /* Get CO2 continuum absorption... */
  const double xw = nu / 2 + 1;
  if (xw >= 1 && xw < 2001) {
    const int iw = (int) xw;
    const double dw = xw - iw;
    const double ew = 1 - dw;
    const double cw296 = ew * co2296[iw - 1] + dw * co2296[iw];
    const double cw260 = ew * co2260[iw - 1] + dw * co2260[iw];
    const double cw230 = ew * co2230[iw - 1] + dw * co2230[iw];
    const double dt230 = t - 230;
    const double dt260 = t - 260;
    const double dt296 = t - 296;
    const double ctw =
      dt260 * 5.050505e-4 * dt296 * cw230 -
      dt230 * 9.259259e-4 * dt296 * cw260 +
      dt230 * 4.208754e-4 * dt260 * cw296;
    return u / NA / 1000 * p / P0 * ctw;
  } else
    return 0;
}
```

ctmh2o()

```
double ctmh2o (
```

```
const double nu,
const double p,
const double t,
const double q,
const double u )
```

Compute water vapor continuum (optical depth).

Definition at line 1746 of file jurassic.c.

```
                  {

  static double h2o296[2001] = { .17, .1695, .172, .168, .1687, .1624, .1606,
    .1508, .1447, .1344, .1214, .1133, .1009, .09217, .08297, .06989,
    .06513, .05469, .05056, .04417, .03779, .03484, .02994, .0272,
    .02325, .02063, .01818, .01592, .01405, .01251, .0108, .009647,
    .008424, .007519, .006555, .00588, .005136, .004511, .003989,
    .003509, .003114, .00274, .002446, .002144, .001895, .001676,
    .001486, .001312, .001164, .001031, 9.129e-4, 8.106e-4, 7.213e-4,
    6.4e-4, 5.687e-4, 5.063e-4, 4.511e-4, 4.029e-4, 3.596e-4,
    3.22e-4, 2.889e-4, 2.597e-4, 2.337e-4, 2.108e-4, 1.907e-4,
    1.728e-4, 1.57e-4, 1.43e-4, 1.305e-4, 1.195e-4, 1.097e-4,
    1.009e-4, 9.307e-5, 8.604e-5, 7.971e-5, 7.407e-5, 6.896e-5,
    6.433e-5, 6.013e-5, 5.631e-5, 5.283e-5, 4.963e-5, 4.669e-5,
    4.398e-5, 4.148e-5, 3.917e-5, 3.702e-5, 3.502e-5, 3.316e-5,
    3.142e-5, 2.978e-5, 2.825e-5, 2.681e-5, 2.546e-5, 2.419e-5,
    2.299e-5, 2.186e-5, 2.079e-5, 1.979e-5, 1.884e-5, 1.795e-5,
    1.711e-5, 1.633e-5, 1.559e-5, 1.49e-5, 1.426e-5, 1.367e-5,
    1.312e-5, 1.263e-5, 1.218e-5, 1.178e-5, 1.143e-5, 1.112e-5,
    1.088e-5, 1.07e-5, 1.057e-5, 1.05e-5, 1.051e-5, 1.059e-5,
    1.076e-5, 1.1e-5, 1.133e-5, 1.18e-5, 1.237e-5, 1.308e-5,
    1.393e-5, 1.483e-5, 1.614e-5, 1.758e-5, 1.93e-5, 2.123e-5,
    2.346e-5, 2.647e-5, 2.93e-5, 3.279e-5, 3.745e-5, 4.152e-5,
    4.813e-5, 5.477e-5, 6.203e-5, 7.331e-5, 8.056e-5, 9.882e-5,
    1.05e-4, 1.21e-4, 1.341e-4, 1.572e-4, 1.698e-4, 1.968e-4,
    2.175e-4, 2.431e-4, 2.735e-4, 2.867e-4, 3.19e-4, 3.371e-4,
    3.554e-4, 3.726e-4, 3.837e-4, 3.878e-4, 3.864e-4, 3.858e-4,
    3.841e-4, 3.852e-4, 3.815e-4, 3.762e-4, 3.618e-4, 3.579e-4,
    3.45e-4, 3.202e-4, 3.018e-4, 2.785e-4, 2.602e-4, 2.416e-4,
    2.097e-4, 1.939e-4, 1.689e-4, 1.498e-4, 1.308e-4, 1.17e-4,
    1.011e-4, 9.237e-5, 7.909e-5, 7.006e-5, 6.112e-5, 5.401e-5,
    4.914e-5, 4.266e-5, 3.963e-5, 3.316e-5, 3.037e-5, 2.598e-5,
    2.294e-5, 2.066e-5, 1.813e-5, 1.583e-5, 1.423e-5, 1.247e-5,
    1.116e-5, 9.76e-6, 8.596e-6, 7.72e-6, 6.825e-6, 6.108e-6,
    5.366e-6, 4.733e-6, 4.229e-6, 3.731e-6, 3.346e-6, 2.972e-6,
    2.628e-6, 2.356e-6, 2.102e-6, 1.878e-6, 1.678e-6, 1.507e-6,
    1.348e-6, 1.21e-6, 1.089e-6, 9.806e-7, 8.857e-7, 8.004e-7,
    7.261e-7, 6.599e-7, 6.005e-7, 5.479e-7, 5.011e-7, 4.595e-7,
    4.219e-7, 3.885e-7, 3.583e-7, 3.314e-7, 3.071e-7, 2.852e-7,
    2.654e-7, 2.474e-7, 2.311e-7, 2.162e-7, 2.026e-7, 1.902e-7,
    1.788e-7, 1.683e-7, 1.587e-7, 1.497e-7, 1.415e-7, 1.338e-7,
    1.266e-7, 1.2e-7, 1.138e-7, 1.08e-7, 1.027e-7, 9.764e-8,
    9.296e-8, 8.862e-8, 8.458e-8, 8.087e-8, 7.744e-8, 7.429e-8,
    7.145e-8, 6.893e-8, 6.664e-8, 6.468e-8, 6.322e-8, 6.162e-8,
    6.07e-8, 5.992e-8, 5.913e-8, 5.841e-8, 5.796e-8, 5.757e-8,
    5.746e-8, 5.731e-8, 5.679e-8, 5.577e-8, 5.671e-8, 5.656e-8,
    5.594e-8, 5.593e-8, 5.602e-8, 5.62e-8, 5.693e-8, 5.725e-8,
    5.858e-8, 6.037e-8, 6.249e-8, 6.535e-8, 6.899e-8, 7.356e-8,
    7.918e-8, 8.618e-8, 9.385e-8, 1.039e-7, 1.158e-7, 1.29e-7,
    1.437e-7, 1.65e-7, 1.871e-7, 2.121e-7, 2.427e-7, 2.773e-7,
    3.247e-7, 3.677e-7, 4.037e-7, 4.776e-7, 5.101e-7, 6.214e-7,
    6.936e-7, 7.581e-7, 8.486e-7, 9.355e-7, 9.942e-7, 1.063e-6,
    1.123e-6, 1.191e-6, 1.215e-6, 1.247e-6, 1.26e-6, 1.271e-6,
    1.284e-6, 1.317e-6, 1.323e-6, 1.349e-6, 1.353e-6, 1.362e-6,
    1.344e-6, 1.329e-6, 1.336e-6, 1.327e-6, 1.325e-6, 1.359e-6,
    1.374e-6, 1.415e-6, 1.462e-6, 1.526e-6, 1.619e-6, 1.735e-6,
    1.863e-6, 2.034e-6, 2.265e-6, 2.482e-6, 2.756e-6, 3.103e-6,
    3.466e-6, 3.832e-6, 4.378e-6, 4.913e-6, 5.651e-6, 6.311e-6,
    7.169e-6, 8.057e-6, 9.253e-6, 1.047e-5, 1.212e-5, 1.36e-5,
    1.569e-5, 1.776e-5, 2.02e-5, 2.281e-5, 2.683e-5, 2.994e-5,
    3.488e-5, 3.896e-5, 4.499e-5, 5.175e-5, 6.035e-5, 6.34e-5,
    7.281e-5, 7.923e-5, 8.348e-5, 9.631e-5, 1.044e-4, 1.102e-4,
    1.176e-4, 1.244e-4, 1.283e-4, 1.326e-4, 1.4e-4, 1.395e-4,
    1.387e-4, 1.363e-4, 1.314e-4, 1.241e-4, 1.228e-4, 1.148e-4,
    1.086e-4, 1.018e-4, 8.89e-5, 8.316e-5, 7.292e-5, 6.452e-5,
    5.625e-5, 5.045e-5, 4.38e-5, 3.762e-5, 3.29e-5, 2.836e-5,
    2.485e-5, 2.168e-5, 1.895e-5, 1.659e-5, 1.453e-5, 1.282e-5,
    1.132e-5, 1.001e-5, 8.836e-6, 7.804e-6, 6.922e-6, 6.116e-6,
    5.429e-6, 4.824e-6, 4.278e-6, 3.788e-6, 3.371e-6, 2.985e-6,
    2.649e-6, 2.357e-6, 2.09e-6, 1.858e-6, 1.647e-6, 1.462e-6,
    1.299e-6, 1.155e-6, 1.028e-6, 9.142e-7, 8.132e-7, 7.246e-7,
    6.451e-7, 5.764e-7, 5.151e-7, 4.603e-7, 4.121e-7, 3.694e-7,
    3.318e-7, 2.985e-7, 2.69e-7, 2.428e-7, 2.197e-7, 1.992e-7,
    1.81e-7, 1.649e-7, 1.506e-7, 1.378e-7, 1.265e-7, 1.163e-7,
```

```
1.073e-7, 9.918e-8, 9.191e-8, 8.538e-8, 7.949e-8, 7.419e-8,
6.94e-8, 6.508e-8, 6.114e-8, 5.761e-8, 5.437e-8, 5.146e-8,
4.89e-8, 4.636e-8, 4.406e-8, 4.201e-8, 4.015e-8, 3.84e-8,
3.661e-8, 3.51e-8, 3.377e-8, 3.242e-8, 3.13e-8, 3.015e-8,
2.918e-8, 2.83e-8, 2.758e-8, 2.707e-8, 2.656e-8, 2.619e-8,
2.609e-8, 2.615e-8, 2.63e-8, 2.675e-8, 2.745e-8, 2.842e-8,
2.966e-8, 3.125e-8, 3.318e-8, 3.565e-8, 3.85e-8, 4.191e-8,
4.59e-8, 5.059e-8, 5.607e-8, 6.239e-8, 6.958e-8, 7.796e-8,
8.773e-8, 9.88e-8, 1.114e-7, 1.258e-7, 1.422e-7, 1.61e-7,
1.822e-7, 2.06e-7, 2.337e-7, 2.645e-7, 2.996e-7, 3.393e-7,
3.843e-7, 4.363e-7, 4.935e-7, 5.607e-7, 6.363e-7, 7.242e-7,
8.23e-7, 9.411e-7, 1.071e-6, 1.232e-6, 1.402e-6, 1.6e-6, 1.82e-6,
2.128e-6, 2.386e-6, 2.781e-6, 3.242e-6, 3.653e-6, 4.323e-6,
4.747e-6, 5.321e-6, 5.919e-6, 6.681e-6, 7.101e-6, 7.983e-6,
8.342e-6, 8.741e-6, 9.431e-6, 9.952e-6, 1.026e-5, 1.055e-5,
1.095e-5, 1.095e-5, 1.087e-5, 1.056e-5, 1.026e-5, 9.715e-6,
9.252e-6, 8.452e-6, 7.958e-6, 7.268e-6, 6.295e-6, 6.003e-6, 5e-6,
4.591e-6, 3.983e-6, 3.479e-6, 3.058e-6, 2.667e-6, 2.293e-6,
1.995e-6, 1.747e-6, 1.517e-6, 1.335e-6, 1.165e-6, 1.028e-6,
9.007e-7, 7.956e-7, 7.015e-7, 6.192e-7, 5.491e-7, 4.859e-7,
4.297e-7, 3.799e-7, 3.38e-7, 3.002e-7, 2.659e-7, 2.366e-7,
2.103e-7, 1.861e-7, 1.655e-7, 1.469e-7, 1.309e-7, 1.162e-7,
1.032e-7, 9.198e-8, 8.181e-8, 7.294e-8, 6.516e-8, 5.787e-8,
5.163e-8, 4.612e-8, 4.119e-8, 3.695e-8, 3.308e-8, 2.976e-8,
2.67e-8, 2.407e-8, 2.171e-8, 1.965e-8, 1.78e-8, 1.617e-8,
1.47e-8, 1.341e-8, 1.227e-8, 1.125e-8, 1.033e-8, 9.524e-9,
8.797e-9, 8.162e-9, 7.565e-9, 7.04e-9, 6.56e-9, 6.129e-9,
5.733e-9, 5.376e-9, 5.043e-9, 4.75e-9, 4.466e-9, 4.211e-9,
3.977e-9, 3.759e-9, 3.558e-9, 3.373e-9, 3.201e-9, 3.043e-9,
2.895e-9, 2.76e-9, 2.635e-9, 2.518e-9, 2.411e-9, 2.314e-9,
2.23e-9, 2.151e-9, 2.087e-9, 2.035e-9, 1.988e-9, 1.946e-9,
1.927e-9, 1.916e-9, 1.916e-9, 1.933e-9, 1.966e-9, 2.018e-9,
2.09e-9, 2.182e-9, 2.299e-9, 2.442e-9, 2.623e-9, 2.832e-9,
3.079e-9, 3.368e-9, 3.714e-9, 4.104e-9, 4.567e-9, 5.091e-9,
5.701e-9, 6.398e-9, 7.194e-9, 8.127e-9, 9.141e-9, 1.035e-8,
1.177e-8, 1.338e-8, 1.508e-8, 1.711e-8, 1.955e-8, 2.216e-8,
2.534e-8, 2.871e-8, 3.291e-8, 3.711e-8, 4.285e-8, 4.868e-8,
5.509e-8, 6.276e-8, 7.262e-8, 8.252e-8, 9.4e-8, 1.064e-7,
1.247e-7, 1.411e-7, 1.626e-7, 1.827e-7, 2.044e-7, 2.284e-7,
2.452e-7, 2.854e-7, 3.026e-7, 3.278e-7, 3.474e-7, 3.693e-7,
3.93e-7, 4.104e-7, 4.22e-7, 4.439e-7, 4.545e-7, 4.778e-7,
4.812e-7, 5.018e-7, 4.899e-7, 5.075e-7, 5.073e-7, 5.171e-7,
5.131e-7, 5.25e-7, 5.617e-7, 5.846e-7, 6.239e-7, 6.696e-7,
7.398e-7, 8.073e-7, 9.15e-7, 1.009e-6, 1.116e-6, 1.264e-6,
1.439e-6, 1.644e-6, 1.856e-6, 2.147e-6, 2.317e-6, 2.713e-6,
2.882e-6, 2.99e-6, 3.489e-6, 3.581e-6, 4.033e-6, 4.26e-6,
4.543e-6, 4.84e-6, 4.826e-6, 5.013e-6, 5.252e-6, 5.277e-6,
5.306e-6, 5.236e-6, 5.123e-6, 5.171e-6, 4.843e-6, 4.615e-6,
4.385e-6, 3.97e-6, 3.693e-6, 3.231e-6, 2.915e-6, 2.495e-6,
2.144e-6, 1.91e-6, 1.639e-6, 1.417e-6, 1.226e-6, 1.065e-6,
9.29e-7, 8.142e-7, 7.161e-7, 6.318e-7, 5.581e-7, 4.943e-7,
4.376e-7, 3.884e-7, 3.449e-7, 3.06e-7, 2.712e-7, 2.412e-7,
2.139e-7, 1.903e-7, 1.689e-7, 1.499e-7, 1.331e-7, 1.183e-7,
1.05e-7, 9.362e-8, 8.306e-8, 7.403e-8, 6.578e-8, 5.853e-8,
5.216e-8, 4.632e-8, 4.127e-8, 3.678e-8, 3.279e-8, 2.923e-8,
2.612e-8, 2.339e-8, 2.094e-8, 1.877e-8, 1.686e-8, 1.516e-8,
1.366e-8, 1.234e-8, 1.114e-8, 1.012e-8, 9.182e-9, 8.362e-9,
7.634e-9, 6.981e-9, 6.406e-9, 5.888e-9, 5.428e-9, 5.021e-9,
4.65e-9, 4.326e-9, 4.033e-9, 3.77e-9, 3.536e-9, 3.327e-9,
3.141e-9, 2.974e-9, 2.825e-9, 2.697e-9, 2.584e-9, 2.488e-9,
2.406e-9, 2.34e-9, 2.292e-9, 2.259e-9, 2.244e-9, 2.243e-9,
2.272e-9, 2.31e-9, 2.378e-9, 2.454e-9, 2.618e-9, 2.672e-9,
2.831e-9, 3.05e-9, 3.225e-9, 3.425e-9, 3.677e-9, 3.968e-9,
4.221e-9, 4.639e-9, 4.96e-9, 5.359e-9, 5.649e-9, 6.23e-9,
6.716e-9, 7.218e-9, 7.746e-9, 7.988e-9, 8.627e-9, 8.999e-9,
9.442e-9, 9.82e-9, 1.015e-8, 1.06e-8, 1.079e-8, 1.109e-8,
1.137e-8, 1.186e-8, 1.18e-8, 1.187e-8, 1.194e-8, 1.192e-8,
1.224e-8, 1.245e-8, 1.246e-8, 1.318e-8, 1.377e-8, 1.471e-8,
1.582e-8, 1.713e-8, 1.853e-8, 2.063e-8, 2.27e-8, 2.567e-8,
2.891e-8, 3.264e-8, 3.744e-8, 4.286e-8, 4.915e-8, 5.623e-8,
6.336e-8, 7.293e-8, 8.309e-8, 9.319e-8, 1.091e-7, 1.243e-7,
1.348e-7, 1.449e-7, 1.62e-7, 1.846e-7, 1.937e-7, 2.04e-7,
2.179e-7, 2.298e-7, 2.433e-7, 2.439e-7, 2.464e-7, 2.611e-7,
2.617e-7, 2.582e-7, 2.453e-7, 2.401e-7, 2.349e-7, 2.203e-7,
2.066e-7, 1.939e-7, 1.78e-7, 1.558e-7, 1.391e-7, 1.203e-7,
1.048e-7, 9.464e-8, 8.306e-8, 7.239e-8, 6.317e-8, 5.52e-8,
4.847e-8, 4.282e-8, 3.796e-8, 3.377e-8, 2.996e-8, 2.678e-8,
2.4e-8, 2.134e-8, 1.904e-8, 1.705e-8, 1.523e-8, 1.35e-8,
1.204e-8, 1.07e-8, 9.408e-9, 8.476e-9, 7.47e-9, 6.679e-9,
5.929e-9, 5.267e-9, 4.711e-9, 4.172e-9, 3.761e-9, 3.288e-9,
2.929e-9, 2.609e-9, 2.315e-9, 2.042e-9, 1.844e-9, 1.64e-9,
1.47e-9, 1.31e-9, 1.176e-9, 1.049e-9, 9.377e-10, 8.462e-10,
7.616e-10, 6.854e-10, 6.191e-10, 5.596e-10, 5.078e-10, 4.611e-10,
4.197e-10, 3.83e-10, 3.505e-10, 3.215e-10, 2.956e-10, 2.726e-10,
2.521e-10, 2.338e-10, 2.173e-10, 2.026e-10, 1.895e-10, 1.777e-10,
1.672e-10, 1.579e-10, 1.496e-10, 1.423e-10, 1.358e-10, 1.302e-10,
1.254e-10, 1.216e-10, 1.187e-10, 1.163e-10, 1.147e-10, 1.145e-10,
```

```
1.15e-10, 1.17e-10, 1.192e-10, 1.25e-10, 1.298e-10, 1.345e-10,
1.405e-10, 1.538e-10, 1.648e-10, 1.721e-10, 1.872e-10, 1.968e-10,
2.089e-10, 2.172e-10, 2.317e-10, 2.389e-10, 2.503e-10, 2.585e-10,
2.686e-10, 2.8e-10, 2.895e-10, 3.019e-10, 3.037e-10, 3.076e-10,
3.146e-10, 3.198e-10, 3.332e-10, 3.397e-10, 3.54e-10, 3.667e-10,
3.895e-10, 4.071e-10, 4.565e-10, 4.983e-10, 5.439e-10, 5.968e-10,
6.676e-10, 7.456e-10, 8.405e-10, 9.478e-10, 1.064e-9, 1.218e-9,
1.386e-9, 1.581e-9, 1.787e-9, 2.032e-9, 2.347e-9, 2.677e-9,
3.008e-9, 3.544e-9, 4.056e-9, 4.687e-9, 5.331e-9, 6.227e-9,
6.854e-9, 8.139e-9, 8.945e-9, 9.865e-9, 1.125e-8, 1.178e-8,
1.364e-8, 1.436e-8, 1.54e-8, 1.672e-8, 1.793e-8, 1.906e-8,
2.036e-8, 2.144e-8, 2.292e-8, 2.371e-8, 2.493e-8, 2.606e-8,
2.706e-8, 2.866e-8, 3.036e-8, 3.136e-8, 3.405e-8, 3.665e-8,
3.837e-8, 4.229e-8, 4.748e-8, 5.32e-8, 5.763e-8, 6.677e-8,
7.216e-8, 7.716e-8, 8.958e-8, 9.419e-8, 1.036e-7, 1.108e-7,
1.189e-7, 1.246e-7, 1.348e-7, 1.31e-7, 1.361e-7, 1.364e-7,
1.363e-7, 1.343e-7, 1.293e-7, 1.254e-7, 1.235e-7, 1.158e-7,
1.107e-7, 9.961e-8, 9.011e-8, 7.91e-8, 6.916e-8, 6.338e-8,
5.564e-8, 4.827e-8, 4.198e-8, 3.695e-8, 3.276e-8, 2.929e-8,
2.633e-8, 2.391e-8, 2.192e-8, 2.021e-8, 1.89e-8, 1.772e-8,
1.667e-8, 1.603e-8, 1.547e-8, 1.537e-8, 1.492e-8, 1.515e-8,
1.479e-8, 1.45e-8, 1.513e-8, 1.495e-8, 1.529e-8, 1.565e-8,
1.564e-8, 1.553e-8, 1.569e-8, 1.584e-8, 1.57e-8, 1.538e-8,
1.513e-8, 1.472e-8, 1.425e-8, 1.349e-8, 1.328e-8, 1.249e-8,
1.17e-8, 1.077e-8, 9.514e-9, 8.614e-9, 7.46e-9, 6.621e-9,
5.775e-9, 5.006e-9, 4.308e-9, 3.747e-9, 3.24e-9, 2.84e-9,
2.481e-9, 2.184e-9, 1.923e-9, 1.71e-9, 1.504e-9, 1.334e-9,
1.187e-9, 1.053e-9, 9.367e-10, 8.306e-10, 7.419e-10, 6.63e-10,
5.918e-10, 5.277e-10, 4.717e-10, 4.222e-10, 3.783e-10, 3.39e-10,
3.036e-10, 2.729e-10, 2.455e-10, 2.211e-10, 1.995e-10, 1.804e-10,
1.635e-10, 1.485e-10, 1.355e-10, 1.24e-10, 1.139e-10, 1.051e-10,
9.757e-11, 9.114e-11, 8.577e-11, 8.139e-11, 7.792e-11, 7.52e-11,
7.39e-11, 7.311e-11, 7.277e-11, 7.482e-11, 7.698e-11, 8.162e-11,
8.517e-11, 8.968e-11, 9.905e-11, 1.075e-10, 1.187e-10, 1.291e-10,
1.426e-10, 1.573e-10, 1.734e-10, 1.905e-10, 2.097e-10, 2.28e-10,
2.473e-10, 2.718e-10, 2.922e-10, 3.128e-10, 3.361e-10, 3.641e-10,
3.91e-10, 4.196e-10, 4.501e-10, 4.932e-10, 5.258e-10, 5.755e-10,
6.253e-10, 6.664e-10, 7.344e-10, 7.985e-10, 8.877e-10, 1.005e-9,
1.118e-9, 1.251e-9, 1.428e-9, 1.61e-9, 1.888e-9, 2.077e-9,
2.331e-9, 2.751e-9, 3.061e-9, 3.522e-9, 3.805e-9, 4.181e-9,
4.575e-9, 5.167e-9, 5.634e-9, 6.007e-9, 6.501e-9, 6.829e-9,
7.211e-9, 7.262e-9, 7.696e-9, 7.832e-9, 7.799e-9, 7.651e-9,
7.304e-9, 7.15e-9, 6.977e-9, 6.603e-9, 6.209e-9, 5.69e-9,
5.432e-9, 4.764e-9, 4.189e-9, 3.64e-9, 3.203e-9, 2.848e-9,
2.51e-9, 2.194e-9, 1.946e-9, 1.75e-9, 1.567e-9, 1.426e-9,
1.302e-9, 1.197e-9, 1.109e-9, 1.035e-9, 9.719e-10, 9.207e-10,
8.957e-10, 8.578e-10, 8.262e-10, 8.117e-10, 7.987e-10, 7.875e-10,
7.741e-10, 7.762e-10, 7.537e-10, 7.424e-10, 7.474e-10, 7.294e-10,
7.216e-10, 7.233e-10, 7.075e-10, 6.892e-10, 6.618e-10, 6.314e-10,
6.208e-10, 5.689e-10, 5.55e-10, 4.984e-10, 4.6e-10, 4.078e-10,
3.879e-10, 3.459e-10, 2.982e-10, 2.626e-10, 2.329e-10, 1.988e-10,
1.735e-10, 1.487e-10, 1.297e-10, 1.133e-10, 9.943e-11, 8.736e-11,
7.726e-11, 6.836e-11, 6.053e-11, 5.384e-11, 4.789e-11, 4.267e-11,
3.804e-11, 3.398e-11, 3.034e-11, 2.71e-11, 2.425e-11, 2.173e-11,
1.95e-11, 1.752e-11, 1.574e-11, 1.418e-11, 1.278e-11, 1.154e-11,
1.044e-11, 9.463e-12, 8.602e-12, 7.841e-12, 7.171e-12, 6.584e-12,
6.073e-12, 5.631e-12, 5.254e-12, 4.937e-12, 4.679e-12, 4.476e-12,
4.328e-12, 4.233e-12, 4.194e-12, 4.211e-12, 4.286e-12, 4.424e-12,
4.628e-12, 4.906e-12, 5.262e-12, 5.708e-12, 6.254e-12, 6.914e-12,
7.714e-12, 8.677e-12, 9.747e-12, 1.101e-11, 1.256e-11, 1.409e-11,
1.597e-11, 1.807e-11, 2.034e-11, 2.316e-11, 2.622e-11, 2.962e-11,
3.369e-11, 3.819e-11, 4.329e-11, 4.932e-11, 5.589e-11, 6.364e-11,
7.284e-11, 8.236e-11, 9.447e-11, 1.078e-10, 1.229e-10, 1.417e-10,
1.614e-10, 1.843e-10, 2.107e-10, 2.406e-10, 2.728e-10, 3.195e-10,
3.595e-10, 4.153e-10, 4.736e-10, 5.41e-10, 6.088e-10, 6.769e-10,
7.691e-10, 8.545e-10, 9.621e-10, 1.047e-9, 1.161e-9, 1.296e-9,
1.424e-9, 1.576e-9, 1.739e-9, 1.893e-9, 2.08e-9, 2.336e-9,
2.604e-9, 2.76e-9, 3.001e-9, 3.365e-9, 3.55e-9, 3.895e-9,
4.183e-9, 4.614e-9, 4.846e-9, 5.068e-9, 5.427e-9, 5.541e-9,
5.864e-9, 5.997e-9, 5.997e-9, 6.061e-9, 5.944e-9, 5.855e-9,
5.661e-9, 5.523e-9, 5.374e-9, 4.94e-9, 4.688e-9, 4.17e-9,
3.913e-9, 3.423e-9, 2.997e-9, 2.598e-9, 2.253e-9, 1.946e-9,
1.71e-9, 1.507e-9, 1.336e-9, 1.19e-9, 1.068e-9, 9.623e-10,
8.772e-10, 8.007e-10, 7.42e-10, 6.884e-10, 6.483e-10, 6.162e-10,
5.922e-10, 5.688e-10, 5.654e-10, 5.637e-10, 5.701e-10, 5.781e-10,
5.874e-10, 6.268e-10, 6.357e-10, 6.525e-10, 7.137e-10, 7.441e-10,
8.024e-10, 8.485e-10, 9.143e-10, 9.536e-10, 9.717e-10, 1.018e-9,
1.042e-9, 1.054e-9, 1.092e-9, 1.079e-9, 1.064e-9, 1.043e-9,
1.02e-9, 9.687e-10, 9.273e-10, 9.208e-10, 9.068e-10, 7.687e-10,
7.385e-10, 6.595e-10, 5.87e-10, 5.144e-10, 4.417e-10, 3.804e-10,
3.301e-10, 2.866e-10, 2.509e-10, 2.202e-10, 1.947e-10, 1.719e-10,
1.525e-10, 1.361e-10, 1.21e-10, 1.084e-10, 9.8e-11, 8.801e-11,
7.954e-11, 7.124e-11, 6.335e-11, 5.76e-11, 5.132e-11, 4.601e-11,
4.096e-11, 3.657e-11, 3.25e-11, 2.909e-11, 2.587e-11, 2.297e-11,
2.05e-11, 1.828e-11, 1.632e-11, 1.462e-11, 1.314e-11, 1.185e-11,
1.073e-11, 9.76e-12, 8.922e-12, 8.206e-12, 7.602e-12, 7.1e-12,
6.694e-12, 6.378e-12, 6.149e-12, 6.004e-12, 5.941e-12, 5.962e-12,
```

```
    6.069e-12, 6.265e-12, 6.551e-12, 6.935e-12, 7.457e-12, 8.074e-12,
    8.811e-12, 9.852e-12, 1.086e-11, 1.207e-11, 1.361e-11, 1.553e-11,
    1.737e-11, 1.93e-11, 2.175e-11, 2.41e-11, 2.706e-11, 3.023e-11,
    3.313e-11, 3.657e-11, 4.118e-11, 4.569e-11, 5.025e-11, 5.66e-11,
    6.231e-11, 6.881e-11, 7.996e-11, 8.526e-11, 9.694e-11, 1.106e-10,
    1.222e-10, 1.355e-10, 1.525e-10, 1.775e-10, 1.924e-10, 2.181e-10,
    2.379e-10, 2.662e-10, 2.907e-10, 3.154e-10, 3.366e-10, 3.579e-10,
    3.858e-10, 4.046e-10, 4.196e-10, 4.166e-10, 4.457e-10, 4.466e-10,
    4.404e-10, 4.337e-10, 4.15e-10, 4.083e-10, 3.91e-10, 3.723e-10,
    3.514e-10, 3.303e-10, 2.847e-10, 2.546e-10, 2.23e-10, 1.994e-10,
    1.733e-10, 1.488e-10, 1.297e-10, 1.144e-10, 1.004e-10, 8.741e-11,
    7.928e-11, 7.034e-11, 6.323e-11, 5.754e-11, 5.25e-11, 4.85e-11,
    4.502e-11, 4.286e-11, 4.028e-11, 3.899e-11, 3.824e-11, 3.761e-11,
    3.804e-11, 3.839e-11, 3.845e-11, 4.244e-11, 4.382e-11, 4.582e-11,
    4.847e-11, 5.209e-11, 5.384e-11, 5.887e-11, 6.371e-11, 6.737e-11,
    7.168e-11, 7.415e-11, 7.827e-11, 8.037e-11, 8.12e-11, 8.071e-11,
    8.008e-11, 7.851e-11, 7.544e-11, 7.377e-11, 7.173e-11, 6.801e-11,
    6.267e-11, 5.727e-11, 5.288e-11, 4.853e-11, 4.082e-11, 3.645e-11,
    3.136e-11, 2.672e-11, 2.304e-11, 1.986e-11, 1.725e-11, 1.503e-11,
    1.315e-11, 1.153e-11, 1.014e-11, 8.942e-12, 7.901e-12, 6.993e-12,
    6.199e-12, 5.502e-12, 4.89e-12, 4.351e-12, 3.878e-12, 3.461e-12,
    3.094e-12, 2.771e-12, 2.488e-12, 2.241e-12, 2.025e-12, 1.838e-12,
    1.677e-12, 1.541e-12, 1.427e-12, 1.335e-12, 1.262e-12, 1.209e-12,
    1.176e-12, 1.161e-12, 1.165e-12, 1.189e-12, 1.234e-12, 1.3e-12,
    1.389e-12, 1.503e-12, 1.644e-12, 1.814e-12, 2.017e-12, 2.255e-12,
    2.534e-12, 2.858e-12, 3.231e-12, 3.661e-12, 4.153e-12, 4.717e-12,
    5.36e-12, 6.094e-12, 6.93e-12, 7.882e-12, 8.966e-12, 1.02e-11,
    1.162e-11, 1.324e-11, 1.51e-11, 1.72e-11, 1.965e-11, 2.237e-11,
    2.56e-11, 2.927e-11, 3.371e-11, 3.842e-11, 4.429e-11, 5.139e-11,
    5.798e-11, 6.697e-11, 7.626e-11, 8.647e-11, 1.022e-10, 1.136e-10,
    1.3e-10, 1.481e-10, 1.672e-10, 1.871e-10, 2.126e-10, 2.357e-10,
    2.583e-10, 2.997e-10, 3.289e-10, 3.702e-10, 4.012e-10, 4.319e-10,
    4.527e-10, 5.001e-10, 5.448e-10, 5.611e-10, 5.76e-10, 5.965e-10,
    6.079e-10, 6.207e-10, 6.276e-10, 6.222e-10, 6.137e-10, 6e-10,
    5.814e-10, 5.393e-10, 5.35e-10, 4.947e-10, 4.629e-10, 4.117e-10,
    3.712e-10, 3.372e-10, 2.923e-10, 2.55e-10, 2.232e-10, 1.929e-10,
    1.679e-10, 1.46e-10, 1.289e-10, 1.13e-10, 9.953e-11, 8.763e-11,
    7.76e-11, 6.9e-11, 6.16e-11, 5.525e-11, 4.958e-11, 4.489e-11,
    4.072e-11, 3.728e-11, 3.438e-11, 3.205e-11, 3.006e-11, 2.848e-11,
    2.766e-11, 2.688e-11, 2.664e-11, 2.67e-11, 2.696e-11, 2.786e-11,
    2.861e-11, 3.009e-11, 3.178e-11, 3.389e-11, 3.587e-11, 3.819e-11,
    4.054e-11, 4.417e-11, 4.703e-11, 5.137e-11, 5.46e-11, 6.055e-11,
    6.333e-11, 6.773e-11, 7.219e-11, 7.717e-11, 8.131e-11, 8.491e-11,
    8.574e-11, 9.01e-11, 9.017e-11, 8.999e-11, 8.959e-11, 8.838e-11,
    8.579e-11, 8.162e-11, 8.098e-11, 7.472e-11, 7.108e-11, 6.559e-11,
    5.994e-11, 5.172e-11, 4.424e-11, 3.951e-11, 3.34e-11, 2.902e-11,
    2.541e-11, 2.215e-11, 1.945e-11, 1.716e-11, 1.503e-11, 1.339e-11,
    1.185e-11, 1.05e-11, 9.336e-12, 8.307e-12, 7.312e-12, 6.55e-12,
    5.836e-12, 5.178e-12, 4.6e-12, 4.086e-12, 3.639e-12, 3.247e-12,
    2.904e-12, 2.604e-12, 2.341e-12, 2.112e-12, 1.914e-12, 1.744e-12,
    1.598e-12, 1.476e-12, 1.374e-12, 1.293e-12, 1.23e-12, 1.185e-12,
    1.158e-12, 1.147e-12, 1.154e-12, 1.177e-12, 1.219e-12, 1.28e-12,
    1.36e-12, 1.463e-12, 1.591e-12, 1.75e-12, 1.94e-12, 2.156e-12,
    2.43e-12, 2.748e-12, 3.052e-12, 3.533e-12, 3.967e-12, 4.471e-12,
    5.041e-12, 5.86e-12, 6.664e-12, 7.522e-12, 8.342e-12, 9.412e-12,
    1.072e-11, 1.213e-11, 1.343e-11, 1.496e-11, 1.664e-11, 1.822e-11,
    2.029e-11, 2.233e-11, 2.457e-11, 2.709e-11, 2.928e-11, 3.115e-11,
    3.356e-11, 3.592e-11, 3.818e-11, 3.936e-11, 4.061e-11, 4.149e-11,
    4.299e-11, 4.223e-11, 4.251e-11, 4.287e-11, 4.177e-11, 4.094e-11,
    3.942e-11, 3.772e-11, 3.614e-11, 3.394e-11, 3.222e-11, 2.791e-11,
    2.665e-11, 2.309e-11, 2.032e-11, 1.74e-11, 1.535e-11, 1.323e-11,
    1.151e-11, 9.803e-12, 8.65e-12, 7.54e-12, 6.619e-12, 5.832e-12,
    5.113e-12, 4.503e-12, 3.975e-12, 3.52e-12, 3.112e-12, 2.797e-12,
    2.5e-12, 2.24e-12, 2.013e-12, 1.819e-12, 1.653e-12, 1.513e-12,
    1.395e-12, 1.299e-12, 1.225e-12, 1.168e-12, 1.124e-12, 1.148e-12,
    1.107e-12, 1.128e-12, 1.169e-12, 1.233e-12, 1.307e-12, 1.359e-12,
    1.543e-12, 1.686e-12, 1.794e-12, 2.028e-12, 2.21e-12, 2.441e-12,
    2.653e-12, 2.828e-12, 3.093e-12, 3.28e-12, 3.551e-12, 3.677e-12,
    3.803e-12, 3.844e-12, 4.068e-12, 4.093e-12, 4.002e-12, 3.904e-12,
    3.624e-12, 3.633e-12, 3.622e-12, 3.443e-12, 3.184e-12, 2.934e-12,
    2.476e-12, 2.212e-12, 1.867e-12, 1.594e-12, 1.37e-12, 1.192e-12,
    1.045e-12, 9.211e-13, 8.17e-13, 7.29e-13, 6.55e-13, 5.929e-13,
    5.415e-13, 4.995e-13, 4.661e-13, 4.406e-13, 4.225e-13, 4.116e-13,
    4.075e-13, 4.102e-13, 4.198e-13, 4.365e-13, 4.606e-13, 4.925e-13,
    5.326e-13, 5.818e-13, 6.407e-13, 7.104e-13, 7.92e-13, 8.868e-13,
    9.964e-13, 1.123e-12, 1.268e-12, 1.434e-12, 1.626e-12, 1.848e-12,
    2.107e-12, 2.422e-12, 2.772e-12, 3.145e-12, 3.704e-12, 4.27e-12,
    4.721e-12, 5.361e-12, 6.083e-12, 7.095e-12, 7.968e-12, 9.228e-12,
    1.048e-11, 1.187e-11, 1.336e-11, 1.577e-11, 1.772e-11, 2.017e-11,
    2.25e-11, 2.63e-11, 2.911e-11, 3.356e-11, 3.82e-11, 4.173e-11,
    4.811e-11, 5.254e-11, 5.839e-11, 6.187e-11, 6.805e-11, 7.118e-11,
    7.369e-11, 7.664e-11, 7.794e-11, 7.947e-11, 8.036e-11, 7.954e-11,
    7.849e-11, 7.518e-11, 7.462e-11, 6.926e-11, 6.531e-11, 6.197e-11,
    5.421e-11, 4.777e-11, 4.111e-11, 3.679e-11, 3.166e-11, 2.786e-11,
    2.436e-11, 2.144e-11, 1.859e-11, 1.628e-11, 1.414e-11, 1.237e-11,
    1.093e-11, 9.558e-12
};
```

```
static double h2o260[2001] = { .2752, .2732, .2749, .2676, .2667, .2545,
  .2497, .2327, .2218, .2036, .1825, .1694, .1497, .1353, .121,
  .1014, .09405, .07848, .07195, .06246, .05306, .04853, .04138,
  .03735, .03171, .02785, .02431, .02111, .01845, .0164, .01405,
  .01255, .01098, .009797, .008646, .007779, .006898, .006099,
  .005453, .004909, .004413, .003959, .003581, .003199, .002871,
  .002583, .00233, .002086, .001874, .001684, .001512, .001361,
  .001225, .0011, 9.89e-4, 8.916e-4, 8.039e-4, 7.256e-4, 6.545e-4,
  5.918e-4, 5.359e-4, 4.867e-4, 4.426e-4, 4.033e-4, 3.682e-4,
  3.366e-4, 3.085e-4, 2.833e-4, 2.605e-4, 2.403e-4, 2.221e-4,
  2.055e-4, 1.908e-4, 1.774e-4, 1.653e-4, 1.544e-4, 1.443e-4,
  1.351e-4, 1.267e-4, 1.19e-4, 1.119e-4, 1.053e-4, 9.922e-5,
  9.355e-5, 8.831e-5, 8.339e-5, 7.878e-5, 7.449e-5, 7.043e-5,
  6.664e-5, 6.307e-5, 5.969e-5, 5.654e-5, 5.357e-5, 5.075e-5,
  4.81e-5, 4.56e-5, 4.322e-5, 4.102e-5, 3.892e-5, 3.696e-5,
  3.511e-5, 3.339e-5, 3.177e-5, 3.026e-5, 2.886e-5, 2.756e-5,
  2.636e-5, 2.527e-5, 2.427e-5, 2.337e-5, 2.257e-5, 2.185e-5,
  2.127e-5, 2.08e-5, 2.041e-5, 2.013e-5, 2e-5, 1.997e-5, 2.009e-5,
  2.031e-5, 2.068e-5, 2.124e-5, 2.189e-5, 2.267e-5, 2.364e-5,
  2.463e-5, 2.618e-5, 2.774e-5, 2.937e-5, 3.144e-5, 3.359e-5,
  3.695e-5, 4.002e-5, 4.374e-5, 4.947e-5, 5.431e-5, 6.281e-5,
  7.169e-5, 8.157e-5, 9.728e-5, 1.079e-4, 1.337e-4, 1.442e-4,
  1.683e-4, 1.879e-4, 2.223e-4, 2.425e-4, 2.838e-4, 3.143e-4,
  3.527e-4, 4.012e-4, 4.237e-4, 4.747e-4, 5.057e-4, 5.409e-4,
  5.734e-4, 5.944e-4, 6.077e-4, 6.175e-4, 6.238e-4, 6.226e-4,
  6.248e-4, 6.192e-4, 6.098e-4, 5.818e-4, 5.709e-4, 5.465e-4,
  5.043e-4, 4.699e-4, 4.294e-4, 3.984e-4, 3.672e-4, 3.152e-4,
  2.883e-4, 2.503e-4, 2.211e-4, 1.92e-4, 1.714e-4, 1.485e-4,
  1.358e-4, 1.156e-4, 1.021e-4, 8.887e-5, 7.842e-5, 7.12e-5,
  6.186e-5, 5.73e-5, 4.792e-5, 4.364e-5, 3.72e-5, 3.28e-5,
  2.946e-5, 2.591e-5, 2.261e-5, 2.048e-5, 1.813e-5, 1.63e-5,
  1.447e-5, 1.282e-5, 1.167e-5, 1.041e-5, 9.449e-6, 8.51e-6,
  7.596e-6, 6.961e-6, 6.272e-6, 5.728e-6, 5.198e-6, 4.667e-6,
  4.288e-6, 3.897e-6, 3.551e-6, 3.235e-6, 2.952e-6, 2.688e-6,
  2.449e-6, 2.241e-6, 2.05e-6, 1.879e-6, 1.722e-6, 1.582e-6,
  1.456e-6, 1.339e-6, 1.236e-6, 1.144e-6, 1.06e-6, 9.83e-7,
  9.149e-7, 8.535e-7, 7.973e-7, 7.466e-7, 6.999e-7, 6.574e-7,
  6.18e-7, 5.821e-7, 5.487e-7, 5.18e-7, 4.896e-7, 4.631e-7,
  4.386e-7, 4.16e-7, 3.945e-7, 3.748e-7, 3.562e-7, 3.385e-7,
  3.222e-7, 3.068e-7, 2.922e-7, 2.788e-7, 2.659e-7, 2.539e-7,
  2.425e-7, 2.318e-7, 2.219e-7, 2.127e-7, 2.039e-7, 1.958e-7,
  1.885e-7, 1.818e-7, 1.758e-7, 1.711e-7, 1.662e-7, 1.63e-7,
  1.605e-7, 1.58e-7, 1.559e-7, 1.545e-7, 1.532e-7, 1.522e-7,
  1.51e-7, 1.495e-7, 1.465e-7, 1.483e-7, 1.469e-7, 1.448e-7,
  1.444e-7, 1.436e-7, 1.426e-7, 1.431e-7, 1.425e-7, 1.445e-7,
  1.477e-7, 1.515e-7, 1.567e-7, 1.634e-7, 1.712e-7, 1.802e-7,
  1.914e-7, 2.024e-7, 2.159e-7, 2.295e-7, 2.461e-7, 2.621e-7,
  2.868e-7, 3.102e-7, 3.394e-7, 3.784e-7, 4.223e-7, 4.864e-7,
  5.501e-7, 6.039e-7, 7.193e-7, 7.728e-7, 9.514e-7, 1.073e-6,
  1.18e-6, 1.333e-6, 1.472e-6, 1.566e-6, 1.677e-6, 1.784e-6,
  1.904e-6, 1.953e-6, 2.02e-6, 2.074e-6, 2.128e-6, 2.162e-6,
  2.219e-6, 2.221e-6, 2.249e-6, 2.239e-6, 2.235e-6, 2.185e-6,
  2.141e-6, 2.124e-6, 2.09e-6, 2.068e-6, 2.1e-6, 2.104e-6,
  2.142e-6, 2.181e-6, 2.257e-6, 2.362e-6, 2.5e-6, 2.664e-6,
  2.884e-6, 3.189e-6, 3.48e-6, 3.847e-6, 4.313e-6, 4.79e-6,
  5.25e-6, 5.989e-6, 6.692e-6, 7.668e-6, 8.52e-6, 9.606e-6,
  1.073e-5, 1.225e-5, 1.377e-5, 1.582e-5, 1.761e-5, 2.029e-5,
  2.284e-5, 2.602e-5, 2.94e-5, 3.483e-5, 3.928e-5, 4.618e-5,
  5.24e-5, 6.132e-5, 7.183e-5, 8.521e-5, 9.111e-5, 1.07e-4,
  1.184e-4, 1.264e-4, 1.475e-4, 1.612e-4, 1.704e-4, 1.818e-4,
  1.924e-4, 1.994e-4, 2.061e-4, 2.18e-4, 2.187e-4, 2.2e-4,
  2.196e-4, 2.131e-4, 2.015e-4, 1.988e-4, 1.847e-4, 1.729e-4,
  1.597e-4, 1.373e-4, 1.262e-4, 1.087e-4, 9.439e-5, 8.061e-5,
  7.093e-5, 6.049e-5, 5.12e-5, 4.435e-5, 3.817e-5, 3.34e-5,
  2.927e-5, 2.573e-5, 2.291e-5, 2.04e-5, 1.827e-5, 1.636e-5,
  1.463e-5, 1.309e-5, 1.17e-5, 1.047e-5, 9.315e-6, 8.328e-6,
  7.458e-6, 6.665e-6, 5.94e-6, 5.316e-6, 4.752e-6, 4.252e-6,
  3.825e-6, 3.421e-6, 3.064e-6, 2.746e-6, 2.465e-6, 2.216e-6,
  1.99e-6, 1.79e-6, 1.609e-6, 1.449e-6, 1.306e-6, 1.177e-6,
  1.063e-6, 9.607e-7, 8.672e-7, 7.855e-7, 7.118e-7, 6.46e-7,
  5.871e-7, 5.34e-7, 4.868e-7, 4.447e-7, 4.068e-7, 3.729e-7,
  3.423e-7, 3.151e-7, 2.905e-7, 2.686e-7, 2.484e-7, 2.306e-7,
  2.142e-7, 1.995e-7, 1.86e-7, 1.738e-7, 1.626e-7, 1.522e-7,
  1.427e-7, 1.338e-7, 1.258e-7, 1.183e-7, 1.116e-7, 1.056e-7,
  9.972e-8, 9.46e-8, 9.007e-8, 8.592e-8, 8.195e-8, 7.816e-8,
  7.483e-8, 7.193e-8, 6.892e-8, 6.642e-8, 6.386e-8, 6.154e-8,
  5.949e-8, 5.764e-8, 5.622e-8, 5.479e-8, 5.364e-8, 5.301e-8,
  5.267e-8, 5.263e-8, 5.313e-8, 5.41e-8, 5.55e-8, 5.745e-8,
  6.003e-8, 6.311e-8, 6.713e-8, 7.173e-8, 7.724e-8, 8.368e-8,
  9.121e-8, 9.986e-8, 1.097e-7, 1.209e-7, 1.338e-7, 1.486e-7,
  1.651e-7, 1.837e-7, 2.048e-7, 2.289e-7, 2.557e-7, 2.857e-7,
  3.195e-7, 3.587e-7, 4.015e-7, 4.497e-7, 5.049e-7, 5.665e-7,
  6.366e-7, 7.121e-7, 7.996e-7, 8.946e-7, 1.002e-6, 1.117e-6,
  1.262e-6, 1.416e-6, 1.611e-6, 1.807e-6, 2.056e-6, 2.351e-6,
  2.769e-6, 3.138e-6, 3.699e-6, 4.386e-6, 5.041e-6, 6.074e-6,
  6.812e-6, 7.79e-6, 8.855e-6, 1.014e-5, 1.095e-5, 1.245e-5,
```

```
1.316e-5, 1.39e-5, 1.504e-5, 1.583e-5, 1.617e-5, 1.652e-5,
1.713e-5, 1.724e-5, 1.715e-5, 1.668e-5, 1.629e-5, 1.552e-5,
1.478e-5, 1.34e-5, 1.245e-5, 1.121e-5, 9.575e-6, 8.956e-6,
7.345e-6, 6.597e-6, 5.612e-6, 4.818e-6, 4.165e-6, 3.579e-6,
3.041e-6, 2.623e-6, 2.29e-6, 1.984e-6, 1.748e-6, 1.534e-6,
1.369e-6, 1.219e-6, 1.092e-6, 9.8e-7, 8.762e-7, 7.896e-7,
7.104e-7, 6.364e-7, 5.691e-7, 5.107e-7, 4.575e-7, 4.09e-7,
3.667e-7, 3.287e-7, 2.931e-7, 2.633e-7, 2.356e-7, 2.111e-7,
1.895e-7, 1.697e-7, 1.525e-7, 1.369e-7, 1.233e-7, 1.114e-7,
9.988e-8, 9.004e-8, 8.149e-8, 7.352e-8, 6.662e-8, 6.03e-8,
5.479e-8, 4.974e-8, 4.532e-8, 4.129e-8, 3.781e-8, 3.462e-8,
3.176e-8, 2.919e-8, 2.687e-8, 2.481e-8, 2.292e-8, 2.119e-8,
1.967e-8, 1.828e-8, 1.706e-8, 1.589e-8, 1.487e-8, 1.393e-8,
1.307e-8, 1.228e-8, 1.156e-8, 1.089e-8, 1.028e-8, 9.696e-9,
9.159e-9, 8.658e-9, 8.187e-9, 7.746e-9, 7.34e-9, 6.953e-9,
6.594e-9, 6.259e-9, 5.948e-9, 5.66e-9, 5.386e-9, 5.135e-9,
4.903e-9, 4.703e-9, 4.515e-9, 4.362e-9, 4.233e-9, 4.117e-9,
4.017e-9, 3.962e-9, 3.924e-9, 3.905e-9, 3.922e-9, 3.967e-9,
4.046e-9, 4.165e-9, 4.32e-9, 4.522e-9, 4.769e-9, 5.083e-9,
5.443e-9, 5.872e-9, 6.366e-9, 6.949e-9, 7.601e-9, 8.371e-9,
9.22e-9, 1.02e-8, 1.129e-8, 1.251e-8, 1.393e-8, 1.542e-8,
1.72e-8, 1.926e-8, 2.152e-8, 2.392e-8, 2.678e-8, 3.028e-8,
3.39e-8, 3.836e-8, 4.309e-8, 4.9e-8, 5.481e-8, 6.252e-8,
7.039e-8, 7.883e-8, 8.849e-8, 1.012e-7, 1.142e-7, 1.3e-7,
1.475e-7, 1.732e-7, 1.978e-7, 2.304e-7, 2.631e-7, 2.988e-7,
3.392e-7, 3.69e-7, 4.355e-7, 4.672e-7, 5.11e-7, 5.461e-7,
5.828e-7, 6.233e-7, 6.509e-7, 6.672e-7, 6.969e-7, 7.104e-7,
7.439e-7, 7.463e-7, 7.708e-7, 7.466e-7, 7.668e-7, 7.549e-7,
7.586e-7, 7.384e-7, 7.439e-7, 7.785e-7, 7.915e-7, 8.31e-7,
8.745e-7, 9.558e-7, 1.038e-6, 1.173e-6, 1.304e-6, 1.452e-6,
1.671e-6, 1.931e-6, 2.239e-6, 2.578e-6, 3.032e-6, 3.334e-6,
3.98e-6, 4.3e-6, 4.518e-6, 5.321e-6, 5.508e-6, 6.211e-6, 6.59e-6,
7.046e-6, 7.555e-6, 7.558e-6, 7.875e-6, 8.319e-6, 8.433e-6,
8.59e-6, 8.503e-6, 8.304e-6, 8.336e-6, 7.739e-6, 7.301e-6,
6.827e-6, 6.078e-6, 5.551e-6, 4.762e-6, 4.224e-6, 3.538e-6,
2.984e-6, 2.619e-6, 2.227e-6, 1.923e-6, 1.669e-6, 1.462e-6,
1.294e-6, 1.155e-6, 1.033e-6, 9.231e-7, 8.238e-7, 7.36e-7,
6.564e-7, 5.869e-7, 5.236e-7, 4.673e-7, 4.174e-7, 3.736e-7,
3.33e-7, 2.976e-7, 2.657e-7, 2.367e-7, 2.106e-7, 1.877e-7,
1.671e-7, 1.494e-7, 1.332e-7, 1.192e-7, 1.065e-7, 9.558e-8,
8.586e-8, 7.717e-8, 6.958e-8, 6.278e-8, 5.666e-8, 5.121e-8,
4.647e-8, 4.213e-8, 3.815e-8, 3.459e-8, 3.146e-8, 2.862e-8,
2.604e-8, 2.375e-8, 2.162e-8, 1.981e-8, 1.817e-8, 1.67e-8,
1.537e-8, 1.417e-8, 1.31e-8, 1.215e-8, 1.128e-8, 1.05e-8,
9.793e-9, 9.158e-9, 8.586e-9, 8.068e-9, 7.595e-9, 7.166e-9,
6.778e-9, 6.427e-9, 6.108e-9, 5.826e-9, 5.571e-9, 5.347e-9,
5.144e-9, 4.968e-9, 4.822e-9, 4.692e-9, 4.589e-9, 4.506e-9,
4.467e-9, 4.44e-9, 4.466e-9, 4.515e-9, 4.718e-9, 4.729e-9,
4.937e-9, 5.249e-9, 5.466e-9, 5.713e-9, 6.03e-9, 6.436e-9,
6.741e-9, 7.33e-9, 7.787e-9, 8.414e-9, 8.908e-9, 9.868e-9,
1.069e-8, 1.158e-8, 1.253e-8, 1.3e-8, 1.409e-8, 1.47e-8,
1.548e-8, 1.612e-8, 1.666e-8, 1.736e-8, 1.763e-8, 1.812e-8,
1.852e-8, 1.923e-8, 1.897e-8, 1.893e-8, 1.888e-8, 1.868e-8,
1.895e-8, 1.899e-8, 1.876e-8, 1.96e-8, 2.02e-8, 2.121e-8,
2.239e-8, 2.379e-8, 2.526e-8, 2.766e-8, 2.994e-8, 3.332e-8,
3.703e-8, 4.158e-8, 4.774e-8, 5.499e-8, 6.355e-8, 7.349e-8,
8.414e-8, 9.846e-8, 1.143e-7, 1.307e-7, 1.562e-7, 1.817e-7,
2.011e-7, 2.192e-7, 2.485e-7, 2.867e-7, 3.035e-7, 3.223e-7,
3.443e-7, 3.617e-7, 3.793e-7, 3.793e-7, 3.839e-7, 4.081e-7,
4.117e-7, 4.085e-7, 3.92e-7, 3.851e-7, 3.754e-7, 3.49e-7,
3.229e-7, 2.978e-7, 2.691e-7, 2.312e-7, 2.029e-7, 1.721e-7,
1.472e-7, 1.308e-7, 1.132e-7, 9.736e-8, 8.458e-8, 7.402e-8,
6.534e-8, 5.811e-8, 5.235e-8, 4.762e-8, 4.293e-8, 3.896e-8,
3.526e-8, 3.165e-8, 2.833e-8, 2.551e-8, 2.288e-8, 2.036e-8,
1.82e-8, 1.626e-8, 1.438e-8, 1.299e-8, 1.149e-8, 1.03e-8,
9.148e-9, 8.122e-9, 7.264e-9, 6.425e-9, 5.777e-9, 5.06e-9,
4.502e-9, 4.013e-9, 3.567e-9, 3.145e-9, 2.864e-9, 2.553e-9,
2.311e-9, 2.087e-9, 1.886e-9, 1.716e-9, 1.556e-9, 1.432e-9,
1.311e-9, 1.202e-9, 1.104e-9, 1.013e-9, 9.293e-10, 8.493e-10,
7.79e-10, 7.185e-10, 6.642e-10, 6.141e-10, 5.684e-10, 5.346e-10,
5.032e-10, 4.725e-10, 4.439e-10, 4.176e-10, 3.93e-10, 3.714e-10,
3.515e-10, 3.332e-10, 3.167e-10, 3.02e-10, 2.887e-10, 2.769e-10,
2.665e-10, 2.578e-10, 2.503e-10, 2.436e-10, 2.377e-10, 2.342e-10,
2.305e-10, 2.296e-10, 2.278e-10, 2.321e-10, 2.355e-10, 2.402e-10,
2.478e-10, 2.67e-10, 2.848e-10, 2.982e-10, 3.263e-10, 3.438e-10,
3.649e-10, 3.829e-10, 4.115e-10, 4.264e-10, 4.473e-10, 4.63e-10,
4.808e-10, 4.995e-10, 5.142e-10, 5.313e-10, 5.318e-10, 5.358e-10,
5.452e-10, 5.507e-10, 5.698e-10, 5.782e-10, 5.983e-10, 6.164e-10,
6.532e-10, 6.811e-10, 7.624e-10, 8.302e-10, 9.067e-10, 9.937e-10,
1.104e-9, 1.221e-9, 1.361e-9, 1.516e-9, 1.675e-9, 1.883e-9,
2.101e-9, 2.349e-9, 2.614e-9, 2.92e-9, 3.305e-9, 3.724e-9,
4.142e-9, 4.887e-9, 5.614e-9, 6.506e-9, 7.463e-9, 8.817e-9,
9.849e-9, 1.187e-8, 1.321e-8, 1.474e-8, 1.698e-8, 1.794e-8,
2.09e-8, 2.211e-8, 2.362e-8, 2.556e-8, 2.729e-8, 2.88e-8,
3.046e-8, 3.167e-8, 3.367e-8, 3.457e-8, 3.59e-8, 3.711e-8,
3.826e-8, 4.001e-8, 4.211e-8, 4.315e-8, 4.661e-8, 5.01e-8,
5.249e-8, 5.84e-8, 6.628e-8, 7.512e-8, 8.253e-8, 9.722e-8,
```

```
1.067e-7, 1.153e-7, 1.347e-7, 1.428e-7, 1.577e-7, 1.694e-7,
1.833e-7, 1.938e-7, 2.108e-7, 2.059e-7, 2.157e-7, 2.185e-7,
2.208e-7, 2.182e-7, 2.093e-7, 2.014e-7, 1.962e-7, 1.819e-7,
1.713e-7, 1.51e-7, 1.34e-7, 1.154e-7, 9.89e-8, 8.88e-8, 7.673e-8,
6.599e-8, 5.73e-8, 5.081e-8, 4.567e-8, 4.147e-8, 3.773e-8,
3.46e-8, 3.194e-8, 2.953e-8, 2.759e-8, 2.594e-8, 2.442e-8,
2.355e-8, 2.283e-8, 2.279e-8, 2.231e-8, 2.279e-8, 2.239e-8,
2.21e-8, 2.309e-8, 2.293e-8, 2.352e-8, 2.415e-8, 2.43e-8,
2.426e-8, 2.465e-8, 2.5e-8, 2.496e-8, 2.465e-8, 2.445e-8,
2.383e-8, 2.299e-8, 2.165e-8, 2.113e-8, 1.968e-8, 1.819e-8,
1.644e-8, 1.427e-8, 1.27e-8, 1.082e-8, 9.428e-9, 8.091e-9,
6.958e-9, 5.988e-9, 5.246e-9, 4.601e-9, 4.098e-9, 3.664e-9,
3.287e-9, 2.942e-9, 2.656e-9, 2.364e-9, 2.118e-9, 1.903e-9,
1.703e-9, 1.525e-9, 1.365e-9, 1.229e-9, 1.107e-9, 9.96e-10,
8.945e-10, 8.08e-10, 7.308e-10, 6.616e-10, 5.994e-10, 5.422e-10,
4.929e-10, 4.478e-10, 4.07e-10, 3.707e-10, 3.379e-10, 3.087e-10,
2.823e-10, 2.592e-10, 2.385e-10, 2.201e-10, 2.038e-10, 1.897e-10,
1.774e-10, 1.667e-10, 1.577e-10, 1.502e-10, 1.437e-10, 1.394e-10,
1.358e-10, 1.324e-10, 1.329e-10, 1.324e-10, 1.36e-10, 1.39e-10,
1.424e-10, 1.544e-10, 1.651e-10, 1.817e-10, 1.984e-10, 2.195e-10,
2.438e-10, 2.7e-10, 2.991e-10, 3.322e-10, 3.632e-10, 3.957e-10,
4.36e-10, 4.701e-10, 5.03e-10, 5.381e-10, 5.793e-10, 6.19e-10,
6.596e-10, 7.004e-10, 7.561e-10, 7.934e-10, 8.552e-10, 9.142e-10,
9.57e-10, 1.027e-9, 1.097e-9, 1.193e-9, 1.334e-9, 1.47e-9,
1.636e-9, 1.871e-9, 2.122e-9, 2.519e-9, 2.806e-9, 3.203e-9,
3.846e-9, 4.362e-9, 5.114e-9, 5.643e-9, 6.305e-9, 6.981e-9,
7.983e-9, 8.783e-9, 9.419e-9, 1.017e-8, 1.063e-8, 1.121e-8,
1.13e-8, 1.201e-8, 1.225e-8, 1.232e-8, 1.223e-8, 1.177e-8,
1.151e-8, 1.116e-8, 1.047e-8, 9.698e-9, 8.734e-9, 8.202e-9,
7.041e-9, 6.074e-9, 5.172e-9, 4.468e-9, 3.913e-9, 3.414e-9,
2.975e-9, 2.65e-9, 2.406e-9, 2.173e-9, 2.009e-9, 1.861e-9,
1.727e-9, 1.612e-9, 1.514e-9, 1.43e-9, 1.362e-9, 1.333e-9,
1.288e-9, 1.249e-9, 1.238e-9, 1.228e-9, 1.217e-9, 1.202e-9,
1.209e-9, 1.177e-9, 1.157e-9, 1.165e-9, 1.142e-9, 1.131e-9,
1.138e-9, 1.117e-9, 1.1e-9, 1.069e-9, 1.023e-9, 1.005e-9,
9.159e-10, 8.863e-10, 7.865e-10, 7.153e-10, 6.247e-10, 5.846e-10,
5.133e-10, 4.36e-10, 3.789e-10, 3.335e-10, 2.833e-10, 2.483e-10,
2.155e-10, 1.918e-10, 1.709e-10, 1.529e-10, 1.374e-10, 1.235e-10,
1.108e-10, 9.933e-11, 8.932e-11, 8.022e-11, 7.224e-11, 6.52e-11,
5.896e-11, 5.328e-11, 4.813e-11, 4.365e-11, 3.961e-11, 3.594e-11,
3.266e-11, 2.967e-11, 2.701e-11, 2.464e-11, 2.248e-11, 2.054e-11,
1.878e-11, 1.721e-11, 1.579e-11, 1.453e-11, 1.341e-11, 1.241e-11,
1.154e-11, 1.078e-11, 1.014e-11, 9.601e-12, 9.167e-12, 8.838e-12,
8.614e-12, 8.493e-12, 8.481e-12, 8.581e-12, 8.795e-12, 9.131e-12,
9.601e-12, 1.021e-11, 1.097e-11, 1.191e-11, 1.303e-11, 1.439e-11,
1.601e-11, 1.778e-11, 1.984e-11, 2.234e-11, 2.474e-11, 2.766e-11,
3.085e-11, 3.415e-11, 3.821e-11, 4.261e-11, 4.748e-11, 5.323e-11,
5.935e-11, 6.619e-11, 7.418e-11, 8.294e-11, 9.26e-11, 1.039e-10,
1.156e-10, 1.297e-10, 1.46e-10, 1.641e-10, 1.858e-10, 2.1e-10,
2.383e-10, 2.724e-10, 3.116e-10, 3.538e-10, 4.173e-10, 4.727e-10,
5.503e-10, 6.337e-10, 7.32e-10, 8.298e-10, 9.328e-10, 1.059e-9,
1.176e-9, 1.328e-9, 1.445e-9, 1.593e-9, 1.77e-9, 1.954e-9,
2.175e-9, 2.405e-9, 2.622e-9, 2.906e-9, 3.294e-9, 3.713e-9,
3.98e-9, 4.384e-9, 4.987e-9, 5.311e-9, 5.874e-9, 6.337e-9,
7.027e-9, 7.39e-9, 7.769e-9, 8.374e-9, 8.605e-9, 9.165e-9,
9.415e-9, 9.511e-9, 9.704e-9, 9.588e-9, 9.45e-9, 9.086e-9,
8.798e-9, 8.469e-9, 7.697e-9, 7.168e-9, 6.255e-9, 5.772e-9,
4.97e-9, 4.271e-9, 3.653e-9, 3.154e-9, 2.742e-9, 2.435e-9,
2.166e-9, 1.936e-9, 1.731e-9, 1.556e-9, 1.399e-9, 1.272e-9,
1.157e-9, 1.066e-9, 9.844e-10, 9.258e-10, 8.787e-10, 8.421e-10,
8.083e-10, 8.046e-10, 8.067e-10, 8.181e-10, 8.325e-10, 8.517e-10,
9.151e-10, 9.351e-10, 9.677e-10, 1.071e-9, 1.126e-9, 1.219e-9,
1.297e-9, 1.408e-9, 1.476e-9, 1.517e-9, 1.6e-9, 1.649e-9,
1.678e-9, 1.746e-9, 1.742e-9, 1.728e-9, 1.699e-9, 1.655e-9,
1.561e-9, 1.48e-9, 1.451e-9, 1.411e-9, 1.171e-9, 1.106e-9,
9.714e-10, 8.523e-10, 7.346e-10, 6.241e-10, 5.371e-10, 4.704e-10,
4.144e-10, 3.683e-10, 3.292e-10, 2.942e-10, 2.62e-10, 2.341e-10,
2.104e-10, 1.884e-10, 1.7e-10, 1.546e-10, 1.394e-10, 1.265e-10,
1.14e-10, 1.019e-10, 9.279e-11, 8.283e-11, 7.458e-11, 6.668e-11,
5.976e-11, 5.33e-11, 4.794e-11, 4.289e-11, 3.841e-11, 3.467e-11,
3.13e-11, 2.832e-11, 2.582e-11, 2.356e-11, 2.152e-11, 1.97e-11,
1.808e-11, 1.664e-11, 1.539e-11, 1.434e-11, 1.344e-11, 1.269e-11,
1.209e-11, 1.162e-11, 1.129e-11, 1.108e-11, 1.099e-11, 1.103e-11,
1.119e-11, 1.148e-11, 1.193e-11, 1.252e-11, 1.329e-11, 1.421e-11,
1.555e-11, 1.685e-11, 1.839e-11, 2.054e-11, 2.317e-11, 2.571e-11,
2.839e-11, 3.171e-11, 3.49e-11, 3.886e-11, 4.287e-11, 4.645e-11,
5.047e-11, 5.592e-11, 6.109e-11, 6.628e-11, 7.381e-11, 8.088e-11,
8.966e-11, 1.045e-10, 1.12e-10, 1.287e-10, 1.486e-10, 1.662e-10,
1.866e-10, 2.133e-10, 2.524e-10, 2.776e-10, 3.204e-10, 3.559e-10,
4.028e-10, 4.448e-10, 4.882e-10, 5.244e-10, 5.605e-10, 6.018e-10,
6.328e-10, 6.579e-10, 6.541e-10, 7.024e-10, 7.074e-10, 7.068e-10,
7.009e-10, 6.698e-10, 6.545e-10, 6.209e-10, 5.834e-10, 5.412e-10,
5.001e-10, 4.231e-10, 3.727e-10, 3.211e-10, 2.833e-10, 2.447e-10,
2.097e-10, 1.843e-10, 1.639e-10, 1.449e-10, 1.27e-10, 1.161e-10,
1.033e-10, 9.282e-11, 8.407e-11, 7.639e-11, 7.023e-11, 6.474e-11,
6.142e-11, 5.76e-11, 5.568e-11, 5.472e-11, 5.39e-11, 5.455e-11,
5.54e-11, 5.587e-11, 6.23e-11, 6.49e-11, 6.868e-11, 7.382e-11,
```

```
    8.022e-11, 8.372e-11, 9.243e-11, 1.004e-10, 1.062e-10, 1.13e-10,
    1.176e-10, 1.244e-10, 1.279e-10, 1.298e-10, 1.302e-10, 1.312e-10,
    1.295e-10, 1.244e-10, 1.211e-10, 1.167e-10, 1.098e-10, 9.927e-11,
    8.854e-11, 8.011e-11, 7.182e-11, 5.923e-11, 5.212e-11, 4.453e-11,
    3.832e-11, 3.371e-11, 2.987e-11, 2.651e-11, 2.354e-11, 2.093e-11,
    1.863e-11, 1.662e-11, 1.486e-11, 1.331e-11, 1.193e-11, 1.071e-11,
    9.628e-12, 8.66e-12, 7.801e-12, 7.031e-12, 6.347e-12, 5.733e-12,
    5.182e-12, 4.695e-12, 4.26e-12, 3.874e-12, 3.533e-12, 3.235e-12,
    2.979e-12, 2.76e-12, 2.579e-12, 2.432e-12, 2.321e-12, 2.246e-12,
    2.205e-12, 2.196e-12, 2.223e-12, 2.288e-12, 2.387e-12, 2.525e-12,
    2.704e-12, 2.925e-12, 3.191e-12, 3.508e-12, 3.876e-12, 4.303e-12,
    4.793e-12, 5.347e-12, 5.978e-12, 6.682e-12, 7.467e-12, 8.34e-12,
    9.293e-12, 1.035e-11, 1.152e-11, 1.285e-11, 1.428e-11, 1.586e-11,
    1.764e-11, 1.972e-11, 2.214e-11, 2.478e-11, 2.776e-11, 3.151e-11,
    3.591e-11, 4.103e-11, 4.66e-11, 5.395e-11, 6.306e-11, 7.172e-11,
    8.358e-11, 9.67e-11, 1.11e-10, 1.325e-10, 1.494e-10, 1.736e-10,
    2.007e-10, 2.296e-10, 2.608e-10, 3.004e-10, 3.361e-10, 3.727e-10,
    4.373e-10, 4.838e-10, 5.483e-10, 6.006e-10, 6.535e-10, 6.899e-10,
    7.687e-10, 8.444e-10, 8.798e-10, 9.135e-10, 9.532e-10, 9.757e-10,
    9.968e-10, 1.006e-9, 9.949e-10, 9.789e-10, 9.564e-10, 9.215e-10,
    8.51e-10, 8.394e-10, 7.707e-10, 7.152e-10, 6.274e-10, 5.598e-10,
    5.028e-10, 4.3e-10, 3.71e-10, 3.245e-10, 2.809e-10, 2.461e-10,
    2.154e-10, 1.91e-10, 1.685e-10, 1.487e-10, 1.313e-10, 1.163e-10,
    1.031e-10, 9.172e-11, 8.221e-11, 7.382e-11, 6.693e-11, 6.079e-11,
    5.581e-11, 5.167e-11, 4.811e-11, 4.506e-11, 4.255e-11, 4.083e-11,
    3.949e-11, 3.881e-11, 3.861e-11, 3.858e-11, 3.951e-11, 4.045e-11,
    4.24e-11, 4.487e-11, 4.806e-11, 5.133e-11, 5.518e-11, 5.919e-11,
    6.533e-11, 7.031e-11, 7.762e-11, 8.305e-11, 9.252e-11, 9.727e-11,
    1.045e-10, 1.117e-10, 1.2e-10, 1.275e-10, 1.341e-10, 1.362e-10,
    1.438e-10, 1.45e-10, 1.455e-10, 1.455e-10, 1.434e-10, 1.381e-10,
    1.301e-10, 1.276e-10, 1.163e-10, 1.089e-10, 9.911e-11, 8.943e-11,
    7.618e-11, 6.424e-11, 5.717e-11, 4.866e-11, 4.257e-11, 3.773e-11,
    3.331e-11, 2.958e-11, 2.629e-11, 2.316e-11, 2.073e-11, 1.841e-11,
    1.635e-11, 1.464e-11, 1.31e-11, 1.16e-11, 1.047e-11, 9.408e-12,
    8.414e-12, 7.521e-12, 6.705e-12, 5.993e-12, 5.371e-12, 4.815e-12,
    4.338e-12, 3.921e-12, 3.567e-12, 3.265e-12, 3.01e-12, 2.795e-12,
    2.613e-12, 2.464e-12, 2.346e-12, 2.256e-12, 2.195e-12, 2.165e-12,
    2.166e-12, 2.198e-12, 2.262e-12, 2.364e-12, 2.502e-12, 2.682e-12,
    2.908e-12, 3.187e-12, 3.533e-12, 3.946e-12, 4.418e-12, 5.013e-12,
    5.708e-12, 6.379e-12, 7.43e-12, 8.39e-12, 9.51e-12, 1.078e-11,
    1.259e-11, 1.438e-11, 1.63e-11, 1.814e-11, 2.055e-11, 2.348e-11,
    2.664e-11, 2.956e-11, 3.3e-11, 3.677e-11, 4.032e-11, 4.494e-11,
    4.951e-11, 5.452e-11, 6.014e-11, 6.5e-11, 6.915e-11, 7.45e-11,
    7.971e-11, 8.468e-11, 8.726e-11, 8.995e-11, 9.182e-11, 9.509e-11,
    9.333e-11, 9.386e-11, 9.457e-11, 9.21e-11, 9.019e-11, 8.68e-11,
    8.298e-11, 7.947e-11, 7.46e-11, 7.082e-11, 6.132e-11, 5.855e-11,
    5.073e-11, 4.464e-11, 3.825e-11, 3.375e-11, 2.911e-11, 2.535e-11,
    2.16e-11, 1.907e-11, 1.665e-11, 1.463e-11, 1.291e-11, 1.133e-11,
    9.997e-12, 8.836e-12, 7.839e-12, 6.943e-12, 6.254e-12, 5.6e-12,
    5.029e-12, 4.529e-12, 4.102e-12, 3.737e-12, 3.428e-12, 3.169e-12,
    2.959e-12, 2.798e-12, 2.675e-12, 2.582e-12, 2.644e-12, 2.557e-12,
    2.614e-12, 2.717e-12, 2.874e-12, 3.056e-12, 3.187e-12, 3.631e-12,
    3.979e-12, 4.248e-12, 4.817e-12, 5.266e-12, 5.836e-12, 6.365e-12,
    6.807e-12, 7.47e-12, 7.951e-12, 8.636e-12, 8.972e-12, 9.314e-12,
    9.445e-12, 1.003e-11, 1.013e-11, 9.937e-12, 9.729e-12, 9.064e-12,
    9.119e-12, 9.124e-12, 8.704e-12, 8.078e-12, 7.47e-12, 6.329e-12,
    5.674e-12, 4.808e-12, 4.119e-12, 3.554e-12, 3.103e-12, 2.731e-12,
    2.415e-12, 2.15e-12, 1.926e-12, 1.737e-12, 1.578e-12, 1.447e-12,
    1.34e-12, 1.255e-12, 1.191e-12, 1.146e-12, 1.121e-12, 1.114e-12,
    1.126e-12, 1.156e-12, 1.207e-12, 1.278e-12, 1.372e-12, 1.49e-12,
    1.633e-12, 1.805e-12, 2.01e-12, 2.249e-12, 2.528e-12, 2.852e-12,
    3.228e-12, 3.658e-12, 4.153e-12, 4.728e-12, 5.394e-12, 6.176e-12,
    7.126e-12, 8.188e-12, 9.328e-12, 1.103e-11, 1.276e-11, 1.417e-11,
    1.615e-11, 1.84e-11, 2.155e-11, 2.429e-11, 2.826e-11, 3.222e-11,
    3.664e-11, 4.14e-11, 4.906e-11, 5.536e-11, 6.327e-11, 7.088e-11,
    8.316e-11, 9.242e-11, 1.07e-10, 1.223e-10, 1.341e-10, 1.553e-10,
    1.703e-10, 1.9e-10, 2.022e-10, 2.233e-10, 2.345e-10, 2.438e-10,
    2.546e-10, 2.599e-10, 2.661e-10, 2.703e-10, 2.686e-10, 2.662e-10,
    2.56e-10, 2.552e-10, 2.378e-10, 2.252e-10, 2.146e-10, 1.885e-10,
    1.668e-10, 1.441e-10, 1.295e-10, 1.119e-10, 9.893e-11, 8.687e-11,
    7.678e-11, 6.685e-11, 5.879e-11, 5.127e-11, 4.505e-11, 3.997e-11,
    3.511e-11
  };

  static double h2ofrn[2001] = { .01095, .01126, .01205, .01322, .0143,
    .01506, .01548, .01534, .01486, .01373, .01262, .01134, .01001,
    .008702, .007475, .006481, .00548, .0046, .003833, .00311,
    .002543, .002049, .00168, .001374, .001046, 8.193e-4, 6.267e-4,
    4.968e-4, 3.924e-4, 2.983e-4, 2.477e-4, 1.997e-4, 1.596e-4,
    1.331e-4, 1.061e-4, 8.942e-5, 7.168e-5, 5.887e-5, 4.848e-5,
    3.817e-5, 3.17e-5, 2.579e-5, 2.162e-5, 1.768e-5, 1.49e-5,
    1.231e-5, 1.013e-5, 8.555e-6, 7.328e-6, 6.148e-6, 5.207e-6,
    4.387e-6, 3.741e-6, 3.22e-6, 2.753e-6, 2.346e-6, 1.985e-6,
    1.716e-6, 1.475e-6, 1.286e-6, 1.122e-6, 9.661e-7, 8.284e-7,
    7.057e-7, 6.119e-7, 5.29e-7, 4.571e-7, 3.948e-7, 3.432e-7,
    2.983e-7, 2.589e-7, 2.265e-7, 1.976e-7, 1.704e-7, 1.456e-7,
    1.26e-7, 1.101e-7, 9.648e-8, 8.415e-8, 7.34e-8, 6.441e-8,
```

```
5.643e-8, 4.94e-8, 4.276e-8, 3.703e-8, 3.227e-8, 2.825e-8,
2.478e-8, 2.174e-8, 1.898e-8, 1.664e-8, 1.458e-8, 1.278e-8,
1.126e-8, 9.891e-9, 8.709e-9, 7.652e-9, 6.759e-9, 5.975e-9,
5.31e-9, 4.728e-9, 4.214e-9, 3.792e-9, 3.463e-9, 3.226e-9,
2.992e-9, 2.813e-9, 2.749e-9, 2.809e-9, 2.913e-9, 3.037e-9,
3.413e-9, 3.738e-9, 4.189e-9, 4.808e-9, 5.978e-9, 7.088e-9,
8.071e-9, 9.61e-9, 1.21e-8, 1.5e-8, 1.764e-8, 2.221e-8, 2.898e-8,
3.948e-8, 5.068e-8, 6.227e-8, 7.898e-8, 1.033e-7, 1.437e-7,
1.889e-7, 2.589e-7, 3.59e-7, 4.971e-7, 7.156e-7, 9.983e-7,
1.381e-6, 1.929e-6, 2.591e-6, 3.453e-6, 4.57e-6, 5.93e-6,
7.552e-6, 9.556e-6, 1.183e-5, 1.425e-5, 1.681e-5, 1.978e-5,
2.335e-5, 2.668e-5, 3.022e-5, 3.371e-5, 3.715e-5, 3.967e-5,
4.06e-5, 4.01e-5, 3.809e-5, 3.491e-5, 3.155e-5, 2.848e-5,
2.678e-5, 2.66e-5, 2.811e-5, 3.071e-5, 3.294e-5, 3.459e-5,
3.569e-5, 3.56e-5, 3.434e-5, 3.186e-5, 2.916e-5, 2.622e-5,
2.275e-5, 1.918e-5, 1.62e-5, 1.373e-5, 1.182e-5, 1.006e-5,
8.556e-6, 7.26e-6, 6.107e-6, 5.034e-6, 4.211e-6, 3.426e-6,
2.865e-6, 2.446e-6, 1.998e-6, 1.628e-6, 1.242e-6, 1.005e-6,
7.853e-7, 6.21e-7, 5.071e-7, 4.156e-7, 3.548e-7, 2.825e-7,
2.261e-7, 1.916e-7, 1.51e-7, 1.279e-7, 1.059e-7, 9.14e-8,
7.707e-8, 6.17e-8, 5.311e-8, 4.263e-8, 3.518e-8, 2.961e-8,
2.457e-8, 2.119e-8, 1.712e-8, 1.439e-8, 1.201e-8, 1.003e-8,
8.564e-9, 7.199e-9, 6.184e-9, 5.206e-9, 4.376e-9, 3.708e-9,
3.157e-9, 2.725e-9, 2.361e-9, 2.074e-9, 1.797e-9, 1.562e-9,
1.364e-9, 1.196e-9, 1.042e-9, 8.862e-10, 7.648e-10, 6.544e-10,
5.609e-10, 4.791e-10, 4.108e-10, 3.531e-10, 3.038e-10, 2.618e-10,
2.268e-10, 1.969e-10, 1.715e-10, 1.496e-10, 1.308e-10, 1.147e-10,
1.008e-10, 8.894e-11, 7.885e-11, 7.031e-11, 6.355e-11, 5.854e-11,
5.534e-11, 5.466e-11, 5.725e-11, 6.447e-11, 7.943e-11, 1.038e-10,
1.437e-10, 2.04e-10, 2.901e-10, 4.051e-10, 5.556e-10, 7.314e-10,
9.291e-10, 1.134e-9, 1.321e-9, 1.482e-9, 1.596e-9, 1.669e-9,
1.715e-9, 1.762e-9, 1.817e-9, 1.828e-9, 1.848e-9, 1.873e-9,
1.902e-9, 1.894e-9, 1.864e-9, 1.841e-9, 1.797e-9, 1.704e-9,
1.559e-9, 1.382e-9, 1.187e-9, 1.001e-9, 8.468e-10, 7.265e-10,
6.521e-10, 6.381e-10, 6.66e-10, 7.637e-10, 9.705e-10, 1.368e-9,
1.856e-9, 2.656e-9, 3.954e-9, 5.96e-9, 8.72e-9, 1.247e-8,
1.781e-8, 2.491e-8, 3.311e-8, 4.272e-8, 5.205e-8, 6.268e-8,
7.337e-8, 8.277e-8, 9.185e-8, 1.004e-7, 1.091e-7, 1.159e-7,
1.188e-7, 1.175e-7, 1.124e-7, 1.033e-7, 9.381e-8, 8.501e-8,
7.956e-8, 7.894e-8, 8.331e-8, 9.102e-8, 9.836e-8, 1.035e-7,
1.064e-7, 1.06e-7, 1.032e-7, 9.808e-8, 9.139e-8, 8.442e-8,
7.641e-8, 6.881e-8, 6.161e-8, 5.404e-8, 4.804e-8, 4.446e-8,
4.328e-8, 4.259e-8, 4.421e-8, 4.673e-8, 4.985e-8, 5.335e-8,
5.796e-8, 6.542e-8, 7.714e-8, 8.827e-8, 1.04e-7, 1.238e-7,
1.499e-7, 1.829e-7, 2.222e-7, 2.689e-7, 3.303e-7, 3.981e-7,
4.84e-7, 5.91e-7, 7.363e-7, 9.087e-7, 1.139e-6, 1.455e-6,
1.866e-6, 2.44e-6, 3.115e-6, 3.941e-6, 4.891e-6, 5.992e-6,
7.111e-6, 8.296e-6, 9.21e-6, 9.987e-6, 1.044e-5, 1.073e-5,
1.092e-5, 1.106e-5, 1.138e-5, 1.171e-5, 1.186e-5, 1.186e-5,
1.179e-5, 1.166e-5, 1.151e-5, 1.16e-5, 1.197e-5, 1.241e-5,
1.268e-5, 1.26e-5, 1.184e-5, 1.063e-5, 9.204e-6, 7.584e-6,
6.053e-6, 4.482e-6, 3.252e-6, 2.337e-6, 1.662e-6, 1.18e-6,
8.15e-7, 5.95e-7, 4.354e-7, 3.302e-7, 2.494e-7, 1.93e-7,
1.545e-7, 1.25e-7, 1.039e-7, 8.602e-8, 7.127e-8, 5.897e-8,
4.838e-8, 4.018e-8, 3.28e-8, 2.72e-8, 2.307e-8, 1.972e-8,
1.654e-8, 1.421e-8, 1.174e-8, 1.004e-8, 8.739e-9, 7.358e-9,
6.242e-9, 5.303e-9, 4.567e-9, 3.94e-9, 3.375e-9, 2.864e-9,
2.422e-9, 2.057e-9, 1.75e-9, 1.505e-9, 1.294e-9, 1.101e-9,
9.401e-10, 8.018e-10, 6.903e-10, 5.965e-10, 5.087e-10, 4.364e-10,
3.759e-10, 3.247e-10, 2.809e-10, 2.438e-10, 2.123e-10, 1.853e-10,
1.622e-10, 1.426e-10, 1.26e-10, 1.125e-10, 1.022e-10, 9.582e-11,
9.388e-11, 9.801e-11, 1.08e-10, 1.276e-10, 1.551e-10, 1.903e-10,
2.291e-10, 2.724e-10, 3.117e-10, 3.4e-10, 3.562e-10, 3.625e-10,
3.619e-10, 3.429e-10, 3.221e-10, 2.943e-10, 2.645e-10, 2.338e-10,
2.062e-10, 1.901e-10, 1.814e-10, 1.827e-10, 1.906e-10, 1.984e-10,
2.04e-10, 2.068e-10, 2.075e-10, 2.018e-10, 1.959e-10, 1.897e-10,
1.852e-10, 1.791e-10, 1.696e-10, 1.634e-10, 1.598e-10, 1.561e-10,
1.518e-10, 1.443e-10, 1.377e-10, 1.346e-10, 1.342e-10, 1.375e-10,
1.525e-10, 1.767e-10, 2.108e-10, 2.524e-10, 2.981e-10, 3.477e-10,
4.262e-10, 5.326e-10, 6.646e-10, 8.321e-10, 1.069e-9, 1.386e-9,
1.743e-9, 2.216e-9, 2.808e-9, 3.585e-9, 4.552e-9, 5.907e-9,
7.611e-9, 9.774e-9, 1.255e-8, 1.666e-8, 2.279e-8, 3.221e-8,
4.531e-8, 6.4e-8, 9.187e-8, 1.295e-7, 1.825e-7, 2.431e-7,
3.181e-7, 4.009e-7, 4.941e-7, 5.88e-7, 6.623e-7, 7.155e-7,
7.451e-7, 7.594e-7, 7.541e-7, 7.467e-7, 7.527e-7, 7.935e-7,
8.461e-7, 8.954e-7, 9.364e-7, 9.843e-7, 1.024e-6, 1.05e-6,
1.059e-6, 1.074e-6, 1.072e-6, 1.043e-6, 9.789e-7, 8.803e-7,
7.662e-7, 6.378e-7, 5.133e-7, 3.958e-7, 2.914e-7, 2.144e-7,
1.57e-7, 1.14e-7, 8.47e-8, 6.2e-8, 4.657e-8, 3.559e-8, 2.813e-8,
2.222e-8, 1.769e-8, 1.391e-8, 1.125e-8, 9.186e-9, 7.704e-9,
6.447e-9, 5.381e-9, 4.442e-9, 3.669e-9, 3.057e-9, 2.564e-9,
2.153e-9, 1.784e-9, 1.499e-9, 1.281e-9, 1.082e-9, 9.304e-10,
8.169e-10, 6.856e-10, 5.866e-10, 5.043e-10, 4.336e-10, 3.731e-10,
3.175e-10, 2.745e-10, 2.374e-10, 2.007e-10, 1.737e-10, 1.508e-10,
1.302e-10, 1.13e-10, 9.672e-11, 8.375e-11, 7.265e-11, 6.244e-11,
5.343e-11, 4.654e-11, 3.975e-11, 3.488e-11, 3.097e-11, 2.834e-11,
2.649e-11, 2.519e-11, 2.462e-11, 2.443e-11, 2.44e-11, 2.398e-11,
```

```
2.306e-11, 2.183e-11, 2.021e-11, 1.821e-11, 1.599e-11, 1.403e-11,
1.196e-11, 1.023e-11, 8.728e-12, 7.606e-12, 6.941e-12, 6.545e-12,
6.484e-12, 6.6e-12, 6.718e-12, 6.785e-12, 6.746e-12, 6.724e-12,
6.764e-12, 6.995e-12, 7.144e-12, 7.32e-12, 7.33e-12, 7.208e-12,
6.789e-12, 6.09e-12, 5.337e-12, 4.62e-12, 4.037e-12, 3.574e-12,
3.311e-12, 3.346e-12, 3.566e-12, 3.836e-12, 4.076e-12, 4.351e-12,
4.691e-12, 5.114e-12, 5.427e-12, 6.167e-12, 7.436e-12, 8.842e-12,
1.038e-11, 1.249e-11, 1.54e-11, 1.915e-11, 2.48e-11, 3.256e-11,
4.339e-11, 5.611e-11, 7.519e-11, 1.037e-10, 1.409e-10, 1.883e-10,
2.503e-10, 3.38e-10, 4.468e-10, 5.801e-10, 7.335e-10, 8.98e-10,
1.11e-9, 1.363e-9, 1.677e-9, 2.104e-9, 2.681e-9, 3.531e-9,
4.621e-9, 6.106e-9, 8.154e-9, 1.046e-8, 1.312e-8, 1.607e-8,
1.948e-8, 2.266e-8, 2.495e-8, 2.655e-8, 2.739e-8, 2.739e-8,
2.662e-8, 2.589e-8, 2.59e-8, 2.664e-8, 2.833e-8, 3.023e-8,
3.305e-8, 3.558e-8, 3.793e-8, 3.961e-8, 4.056e-8, 4.102e-8,
4.025e-8, 3.917e-8, 3.706e-8, 3.493e-8, 3.249e-8, 3.096e-8,
3.011e-8, 3.111e-8, 3.395e-8, 3.958e-8, 4.875e-8, 6.066e-8,
7.915e-8, 1.011e-7, 1.3e-7, 1.622e-7, 2.003e-7, 2.448e-7,
2.863e-7, 3.317e-7, 3.655e-7, 3.96e-7, 4.098e-7, 4.168e-7,
4.198e-7, 4.207e-7, 4.289e-7, 4.384e-7, 4.471e-7, 4.524e-7,
4.574e-7, 4.633e-7, 4.785e-7, 5.028e-7, 5.371e-7, 5.727e-7,
5.955e-7, 5.998e-7, 5.669e-7, 5.082e-7, 4.397e-7, 3.596e-7,
2.814e-7, 2.074e-7, 1.486e-7, 1.057e-7, 7.25e-8, 4.946e-8,
3.43e-8, 2.447e-8, 1.793e-8, 1.375e-8, 1.096e-8, 9.091e-9,
7.709e-9, 6.631e-9, 5.714e-9, 4.886e-9, 4.205e-9, 3.575e-9,
3.07e-9, 2.631e-9, 2.284e-9, 2.002e-9, 1.745e-9, 1.509e-9,
1.284e-9, 1.084e-9, 9.163e-10, 7.663e-10, 6.346e-10, 5.283e-10,
4.354e-10, 3.59e-10, 2.982e-10, 2.455e-10, 2.033e-10, 1.696e-10,
1.432e-10, 1.211e-10, 1.02e-10, 8.702e-11, 7.38e-11, 6.293e-11,
5.343e-11, 4.532e-11, 3.907e-11, 3.365e-11, 2.945e-11, 2.558e-11,
2.192e-11, 1.895e-11, 1.636e-11, 1.42e-11, 1.228e-11, 1.063e-11,
9.348e-12, 8.2e-12, 7.231e-12, 6.43e-12, 5.702e-12, 5.052e-12,
4.469e-12, 4e-12, 3.679e-12, 3.387e-12, 3.197e-12, 3.158e-12,
3.327e-12, 3.675e-12, 4.292e-12, 5.437e-12, 7.197e-12, 1.008e-11,
1.437e-11, 2.035e-11, 2.905e-11, 4.062e-11, 5.528e-11, 7.177e-11,
9.064e-11, 1.109e-10, 1.297e-10, 1.473e-10, 1.652e-10, 1.851e-10,
2.079e-10, 2.313e-10, 2.619e-10, 2.958e-10, 3.352e-10, 3.796e-10,
4.295e-10, 4.923e-10, 5.49e-10, 5.998e-10, 6.388e-10, 6.645e-10,
6.712e-10, 6.549e-10, 6.38e-10, 6.255e-10, 6.253e-10, 6.459e-10,
6.977e-10, 7.59e-10, 8.242e-10, 8.92e-10, 9.403e-10, 9.701e-10,
9.483e-10, 9.135e-10, 8.617e-10, 7.921e-10, 7.168e-10, 6.382e-10,
5.677e-10, 5.045e-10, 4.572e-10, 4.312e-10, 4.145e-10, 4.192e-10,
4.541e-10, 5.368e-10, 6.771e-10, 8.962e-10, 1.21e-9, 1.659e-9,
2.33e-9, 3.249e-9, 4.495e-9, 5.923e-9, 7.642e-9, 9.607e-9,
1.178e-8, 1.399e-8, 1.584e-8, 1.73e-8, 1.816e-8, 1.87e-8,
1.868e-8, 1.87e-8, 1.884e-8, 1.99e-8, 2.15e-8, 2.258e-8,
2.364e-8, 2.473e-8, 2.602e-8, 2.689e-8, 2.731e-8, 2.816e-8,
2.859e-8, 2.839e-8, 2.703e-8, 2.451e-8, 2.149e-8, 1.787e-8,
1.449e-8, 1.111e-8, 8.282e-9, 6.121e-9, 4.494e-9, 3.367e-9,
2.487e-9, 1.885e-9, 1.503e-9, 1.249e-9, 1.074e-9, 9.427e-10,
8.439e-10, 7.563e-10, 6.772e-10, 6.002e-10, 5.254e-10, 4.588e-10,
3.977e-10, 3.449e-10, 3.003e-10, 2.624e-10, 2.335e-10, 2.04e-10,
1.771e-10, 1.534e-10, 1.296e-10, 1.097e-10, 9.173e-11, 7.73e-11,
6.547e-11, 5.191e-11, 4.198e-11, 3.361e-11, 2.732e-11, 2.244e-11,
1.791e-11, 1.509e-11, 1.243e-11, 1.035e-11, 8.969e-12, 7.394e-12,
6.323e-12, 5.282e-12, 4.543e-12, 3.752e-12, 3.14e-12, 2.6e-12,
2.194e-12, 1.825e-12, 1.511e-12, 1.245e-12, 1.024e-12, 8.539e-13,
7.227e-13, 6.102e-13, 5.189e-13, 4.43e-13, 3.774e-13, 3.236e-13,
2.8e-13, 2.444e-13, 2.156e-13, 1.932e-13, 1.775e-13, 1.695e-13,
1.672e-13, 1.704e-13, 1.825e-13, 2.087e-13, 2.614e-13, 3.377e-13,
4.817e-13, 6.989e-13, 1.062e-12, 1.562e-12, 2.288e-12, 3.295e-12,
4.55e-12, 5.965e-12, 7.546e-12, 9.395e-12, 1.103e-11, 1.228e-11,
1.318e-11, 1.38e-11, 1.421e-11, 1.39e-11, 1.358e-11, 1.336e-11,
1.342e-11, 1.356e-11, 1.424e-11, 1.552e-11, 1.73e-11, 1.951e-11,
2.128e-11, 2.249e-11, 2.277e-11, 2.226e-11, 2.111e-11, 1.922e-11,
1.775e-11, 1.661e-11, 1.547e-11, 1.446e-11, 1.323e-11, 1.21e-11,
1.054e-11, 9.283e-12, 8.671e-12, 8.67e-12, 9.429e-12, 1.062e-11,
1.255e-11, 1.506e-11, 1.818e-11, 2.26e-11, 2.831e-11, 3.723e-11,
5.092e-11, 6.968e-11, 9.826e-11, 1.349e-10, 1.87e-10, 2.58e-10,
3.43e-10, 4.424e-10, 5.521e-10, 6.812e-10, 8.064e-10, 9.109e-10,
9.839e-10, 1.028e-9, 1.044e-9, 1.029e-9, 1.005e-9, 1.002e-9,
1.038e-9, 1.122e-9, 1.233e-9, 1.372e-9, 1.524e-9, 1.665e-9,
1.804e-9, 1.908e-9, 2.015e-9, 2.117e-9, 2.219e-9, 2.336e-9,
2.531e-9, 2.805e-9, 3.189e-9, 3.617e-9, 4.208e-9, 4.911e-9,
5.619e-9, 6.469e-9, 7.188e-9, 7.957e-9, 8.503e-9, 9.028e-9,
9.571e-9, 9.99e-9, 1.055e-8, 1.102e-8, 1.132e-8, 1.141e-8,
1.145e-8, 1.145e-8, 1.176e-8, 1.224e-8, 1.304e-8, 1.388e-8,
1.445e-8, 1.453e-8, 1.368e-8, 1.22e-8, 1.042e-8, 8.404e-9,
6.403e-9, 4.643e-9, 3.325e-9, 2.335e-9, 1.638e-9, 1.19e-9,
9.161e-10, 7.412e-10, 6.226e-10, 5.516e-10, 5.068e-10, 4.831e-10,
4.856e-10, 5.162e-10, 5.785e-10, 6.539e-10, 7.485e-10, 8.565e-10,
9.534e-10, 1.052e-9, 1.115e-9, 1.173e-9, 1.203e-9, 1.224e-9,
1.243e-9, 1.248e-9, 1.261e-9, 1.265e-9, 1.25e-9, 1.217e-9,
1.176e-9, 1.145e-9, 1.153e-9, 1.199e-9, 1.278e-9, 1.366e-9,
1.426e-9, 1.444e-9, 1.365e-9, 1.224e-9, 1.051e-9, 8.539e-10,
6.564e-10, 4.751e-10, 3.404e-10, 2.377e-10, 1.631e-10, 1.114e-10,
7.87e-11, 5.793e-11, 4.284e-11, 3.3e-11, 2.62e-11, 2.152e-11,
```

```
1.777e-11, 1.496e-11, 1.242e-11, 1.037e-11, 8.725e-12, 7.004e-12,
5.718e-12, 4.769e-12, 3.952e-12, 3.336e-12, 2.712e-12, 2.213e-12,
1.803e-12, 1.492e-12, 1.236e-12, 1.006e-12, 8.384e-13, 7.063e-13,
5.879e-13, 4.93e-13, 4.171e-13, 3.569e-13, 3.083e-13, 2.688e-13,
2.333e-13, 2.035e-13, 1.82e-13, 1.682e-13, 1.635e-13, 1.674e-13,
1.769e-13, 2.022e-13, 2.485e-13, 3.127e-13, 4.25e-13, 5.928e-13,
8.514e-13, 1.236e-12, 1.701e-12, 2.392e-12, 3.231e-12, 4.35e-12,
5.559e-12, 6.915e-12, 8.519e-12, 1.013e-11, 1.146e-11, 1.24e-11,
1.305e-11, 1.333e-11, 1.318e-11, 1.263e-11, 1.238e-11, 1.244e-11,
1.305e-11, 1.432e-11, 1.623e-11, 1.846e-11, 2.09e-11, 2.328e-11,
2.526e-11, 2.637e-11, 2.702e-11, 2.794e-11, 2.889e-11, 2.989e-11,
3.231e-11, 3.68e-11, 4.375e-11, 5.504e-11, 7.159e-11, 9.502e-11,
1.279e-10, 1.645e-10, 2.098e-10, 2.618e-10, 3.189e-10, 3.79e-10,
4.303e-10, 4.753e-10, 5.027e-10, 5.221e-10, 5.293e-10, 5.346e-10,
5.467e-10, 5.796e-10, 6.2e-10, 6.454e-10, 6.705e-10, 6.925e-10,
7.233e-10, 7.35e-10, 7.538e-10, 7.861e-10, 8.077e-10, 8.132e-10,
7.749e-10, 7.036e-10, 6.143e-10, 5.093e-10, 4.089e-10, 3.092e-10,
2.299e-10, 1.705e-10, 1.277e-10, 9.723e-11, 7.533e-11, 6.126e-11,
5.154e-11, 4.428e-11, 3.913e-11, 3.521e-11, 3.297e-11, 3.275e-11,
3.46e-11, 3.798e-11, 4.251e-11, 4.745e-11, 5.232e-11, 5.606e-11,
5.82e-11, 5.88e-11, 5.79e-11, 5.661e-11, 5.491e-11, 5.366e-11,
5.341e-11, 5.353e-11, 5.336e-11, 5.293e-11, 5.248e-11, 5.235e-11,
5.208e-11, 5.322e-11, 5.521e-11, 5.725e-11, 5.827e-11, 5.685e-11,
5.245e-11, 4.612e-11, 3.884e-11, 3.129e-11, 2.404e-11, 1.732e-11,
1.223e-11, 8.574e-12, 5.888e-12, 3.986e-12, 2.732e-12, 1.948e-12,
1.414e-12, 1.061e-12, 8.298e-13, 6.612e-13, 5.413e-13, 4.472e-13,
3.772e-13, 3.181e-13, 2.645e-13, 2.171e-13, 1.778e-13, 1.464e-13,
1.183e-13, 9.637e-14, 7.991e-14, 6.668e-14, 5.57e-14, 4.663e-14,
3.848e-14, 3.233e-14, 2.706e-14, 2.284e-14, 1.944e-14, 1.664e-14,
1.43e-14, 1.233e-14, 1.066e-14, 9.234e-15, 8.023e-15, 6.993e-15,
6.119e-15, 5.384e-15, 4.774e-15, 4.283e-15, 3.916e-15, 3.695e-15,
3.682e-15, 4.004e-15, 4.912e-15, 6.853e-15, 1.056e-14, 1.712e-14,
2.804e-14, 4.516e-14, 7.113e-14, 1.084e-13, 1.426e-13, 1.734e-13,
1.978e-13, 2.194e-13, 2.388e-13, 2.489e-13, 2.626e-13, 2.865e-13,
3.105e-13, 3.387e-13, 3.652e-13, 3.984e-13, 4.398e-13, 4.906e-13,
5.55e-13, 6.517e-13, 7.813e-13, 9.272e-13, 1.164e-12, 1.434e-12,
1.849e-12, 2.524e-12, 3.328e-12, 4.523e-12, 6.108e-12, 8.207e-12,
1.122e-11, 1.477e-11, 1.9e-11, 2.412e-11, 2.984e-11, 3.68e-11,
4.353e-11, 4.963e-11, 5.478e-11, 5.903e-11, 6.233e-11, 6.483e-11,
6.904e-11, 7.569e-11, 8.719e-11, 1.048e-10, 1.278e-10, 1.557e-10,
1.869e-10, 2.218e-10, 2.61e-10, 2.975e-10, 3.371e-10, 3.746e-10,
4.065e-10, 4.336e-10, 4.503e-10, 4.701e-10, 4.8e-10, 4.917e-10,
5.038e-10, 5.128e-10, 5.143e-10, 5.071e-10, 5.019e-10, 5.025e-10,
5.183e-10, 5.496e-10, 5.877e-10, 6.235e-10, 6.42e-10, 6.234e-10,
5.698e-10, 4.916e-10, 4.022e-10, 3.126e-10, 2.282e-10, 1.639e-10,
1.142e-10, 7.919e-11, 5.69e-11, 4.313e-11, 3.413e-11, 2.807e-11,
2.41e-11, 2.166e-11, 2.024e-11, 1.946e-11, 1.929e-11, 1.963e-11,
2.035e-11, 2.162e-11, 2.305e-11, 2.493e-11, 2.748e-11, 3.048e-11,
3.413e-11, 3.754e-11, 4.155e-11, 4.635e-11, 5.11e-11, 5.734e-11,
6.338e-11, 6.99e-11, 7.611e-11, 8.125e-11, 8.654e-11, 8.951e-11,
9.182e-11, 9.31e-11, 9.273e-11, 9.094e-11, 8.849e-11, 8.662e-11,
8.67e-11, 8.972e-11, 9.566e-11, 1.025e-10, 1.083e-10, 1.111e-10,
1.074e-10, 9.771e-11, 8.468e-11, 6.958e-11, 5.47e-11, 4.04e-11,
2.94e-11, 2.075e-11, 1.442e-11, 1.01e-11, 7.281e-12, 5.409e-12,
4.138e-12, 3.304e-12, 2.784e-12, 2.473e-12, 2.273e-12, 2.186e-12,
2.118e-12, 2.066e-12, 1.958e-12, 1.818e-12, 1.675e-12, 1.509e-12,
1.349e-12, 1.171e-12, 9.838e-13, 8.213e-13, 6.765e-13, 5.378e-13,
4.161e-13, 3.119e-13, 2.279e-13, 1.637e-13, 1.152e-13, 8.112e-14,
5.919e-14, 4.47e-14, 3.492e-14, 2.811e-14, 2.319e-14, 1.948e-14,
1.66e-14, 1.432e-14, 1.251e-14, 1.109e-14, 1.006e-14, 9.45e-15,
9.384e-15, 1.012e-14, 1.216e-14, 1.636e-14, 2.305e-14, 3.488e-14,
5.572e-14, 8.479e-14, 1.265e-13, 1.905e-13, 2.73e-13, 3.809e-13,
4.955e-13, 6.303e-13, 7.861e-13, 9.427e-13, 1.097e-12, 1.212e-12,
1.328e-12, 1.415e-12, 1.463e-12, 1.495e-12, 1.571e-12, 1.731e-12,
1.981e-12, 2.387e-12, 2.93e-12, 3.642e-12, 4.584e-12, 5.822e-12,
7.278e-12, 9.193e-12, 1.135e-11, 1.382e-11, 1.662e-11, 1.958e-11,
2.286e-11, 2.559e-11, 2.805e-11, 2.988e-11, 3.106e-11, 3.182e-11,
3.2e-11, 3.258e-11, 3.362e-11, 3.558e-11, 3.688e-11, 3.8e-11,
3.929e-11, 4.062e-11, 4.186e-11, 4.293e-11, 4.48e-11, 4.643e-11,
4.704e-11, 4.571e-11, 4.206e-11, 3.715e-11, 3.131e-11, 2.541e-11,
1.978e-11, 1.508e-11, 1.146e-11, 8.7e-12, 6.603e-12, 5.162e-12,
4.157e-12, 3.408e-12, 2.829e-12, 2.405e-12, 2.071e-12, 1.826e-12,
1.648e-12, 1.542e-12, 1.489e-12, 1.485e-12, 1.493e-12, 1.545e-12,
1.637e-12, 1.814e-12, 2.061e-12, 2.312e-12, 2.651e-12, 3.03e-12,
3.46e-12, 3.901e-12, 4.306e-12, 4.721e-12, 5.008e-12, 5.281e-12,
5.541e-12, 5.791e-12, 6.115e-12, 6.442e-12, 6.68e-12, 6.791e-12,
6.831e-12, 6.839e-12, 6.946e-12, 7.128e-12, 7.537e-12, 8.036e-12,
8.392e-12, 8.526e-12, 8.11e-12, 7.325e-12, 6.329e-12, 5.183e-12,
4.081e-12, 2.985e-12, 2.141e-12, 1.492e-12, 1.015e-12, 6.684e-13,
4.414e-13, 2.987e-13, 2.038e-13, 1.391e-13, 9.86e-14, 7.24e-14,
5.493e-14, 4.288e-14, 3.427e-14, 2.787e-14, 2.296e-14, 1.909e-14,
1.598e-14, 1.344e-14, 1.135e-14, 9.616e-15, 8.169e-15, 6.957e-15,
5.938e-15, 5.08e-15, 4.353e-15, 3.738e-15, 3.217e-15, 2.773e-15,
2.397e-15, 2.077e-15, 1.805e-15, 1.575e-15, 1.382e-15, 1.221e-15,
1.09e-15, 9.855e-16, 9.068e-16, 8.537e-16, 8.27e-16, 8.29e-16,
8.634e-16, 9.359e-16, 1.055e-15, 1.233e-15, 1.486e-15, 1.839e-15,
2.326e-15, 2.998e-15, 3.934e-15, 5.256e-15, 7.164e-15, 9.984e-15,
```

```
    1.427e-14, 2.099e-14, 3.196e-14, 5.121e-14, 7.908e-14, 1.131e-13,
    1.602e-13, 2.239e-13, 3.075e-13, 4.134e-13, 5.749e-13, 7.886e-13,
    1.071e-12, 1.464e-12, 2.032e-12, 2.8e-12, 3.732e-12, 4.996e-12,
    6.483e-12, 8.143e-12, 1.006e-11, 1.238e-11, 1.484e-11, 1.744e-11,
    2.02e-11, 2.274e-11, 2.562e-11, 2.848e-11, 3.191e-11, 3.617e-11,
    4.081e-11, 4.577e-11, 4.937e-11, 5.204e-11, 5.401e-11, 5.462e-11,
    5.507e-11, 5.51e-11, 5.605e-11, 5.686e-11, 5.739e-11, 5.766e-11,
    5.74e-11, 5.754e-11, 5.761e-11, 5.777e-11, 5.712e-11, 5.51e-11,
    5.088e-11, 4.438e-11, 3.728e-11, 2.994e-11, 2.305e-11, 1.715e-11,
    1.256e-11, 9.208e-12, 6.745e-12, 5.014e-12, 3.785e-12, 2.9e-12,
    2.239e-12, 1.757e-12, 1.414e-12, 1.142e-12, 9.482e-13, 8.01e-13,
    6.961e-13, 6.253e-13, 5.735e-13, 5.433e-13, 5.352e-13, 5.493e-13,
    5.706e-13, 6.068e-13, 6.531e-13, 7.109e-13, 7.767e-13, 8.59e-13,
    9.792e-13, 1.142e-12, 1.371e-12, 1.65e-12, 1.957e-12, 2.302e-12,
    2.705e-12, 3.145e-12, 3.608e-12, 4.071e-12, 4.602e-12, 5.133e-12,
    5.572e-12, 5.987e-12, 6.248e-12, 6.533e-12, 6.757e-12, 6.935e-12,
    7.224e-12, 7.422e-12, 7.538e-12, 7.547e-12, 7.495e-12, 7.543e-12,
    7.725e-12, 8.139e-12, 8.627e-12, 9.146e-12, 9.443e-12, 9.318e-12,
    8.649e-12, 7.512e-12, 6.261e-12, 4.915e-12, 3.647e-12, 2.597e-12,
    1.785e-12, 1.242e-12, 8.66e-13, 6.207e-13, 4.61e-13, 3.444e-13,
    2.634e-13, 2.1e-13, 1.725e-13, 1.455e-13, 1.237e-13, 1.085e-13,
    9.513e-14, 7.978e-14, 6.603e-14, 5.288e-14, 4.084e-14, 2.952e-14,
    2.157e-14, 1.593e-14, 1.199e-14, 9.267e-15, 7.365e-15, 6.004e-15,
    4.995e-15, 4.218e-15, 3.601e-15, 3.101e-15, 2.692e-15, 2.36e-15,
    2.094e-15, 1.891e-15, 1.755e-15, 1.699e-15, 1.755e-15, 1.987e-15,
    2.506e-15, 3.506e-15, 5.289e-15, 8.311e-15, 1.325e-14, 2.129e-14,
    3.237e-14, 4.595e-14, 6.441e-14, 8.433e-14, 1.074e-13, 1.383e-13,
    1.762e-13, 2.281e-13, 2.831e-13, 3.523e-13, 4.38e-13, 5.304e-13,
    6.29e-13, 7.142e-13, 8.032e-13, 8.934e-13, 9.888e-13, 1.109e-12,
    1.261e-12, 1.462e-12, 1.74e-12, 2.099e-12, 2.535e-12, 3.008e-12,
    3.462e-12, 3.856e-12, 4.098e-12, 4.239e-12, 4.234e-12, 4.132e-12,
    3.986e-12, 3.866e-12, 3.829e-12, 3.742e-12, 3.705e-12, 3.694e-12,
    3.765e-12, 3.849e-12, 3.929e-12, 4.056e-12, 4.092e-12, 4.047e-12,
    3.792e-12, 3.407e-12, 2.953e-12, 2.429e-12, 1.931e-12, 1.46e-12,
    1.099e-12, 8.199e-13, 6.077e-13, 4.449e-13, 3.359e-13, 2.524e-13,
    1.881e-13, 1.391e-13, 1.02e-13, 7.544e-14, 5.555e-14, 4.22e-14,
    3.321e-14, 2.686e-14, 2.212e-14, 1.78e-14, 1.369e-14, 1.094e-14,
    9.13e-15, 8.101e-15, 7.828e-15, 8.393e-15, 1.012e-14, 1.259e-14,
    1.538e-14, 1.961e-14, 2.619e-14, 3.679e-14, 5.049e-14, 6.917e-14,
    8.88e-14, 1.115e-13, 1.373e-13, 1.619e-13, 1.878e-13, 2.111e-13,
    2.33e-13, 2.503e-13, 2.613e-13, 2.743e-13, 2.826e-13, 2.976e-13,
    3.162e-13, 3.36e-13, 3.491e-13, 3.541e-13, 3.595e-13, 3.608e-13,
    3.709e-13, 3.869e-13, 4.12e-13, 4.366e-13, 4.504e-13, 4.379e-13,
    3.955e-13, 3.385e-13, 2.741e-13, 2.089e-13, 1.427e-13, 9.294e-14,
    5.775e-14, 3.565e-14, 2.21e-14, 1.398e-14, 9.194e-15, 6.363e-15,
    4.644e-15, 3.55e-15, 2.808e-15, 2.274e-15, 1.871e-15, 1.557e-15,
    1.308e-15, 1.108e-15, 9.488e-16, 8.222e-16, 7.238e-16, 6.506e-16,
    6.008e-16, 5.742e-16, 5.724e-16, 5.991e-16, 6.625e-16, 7.775e-16,
    9.734e-16, 1.306e-15, 1.88e-15, 2.879e-15, 4.616e-15, 7.579e-15,
    1.248e-14, 2.03e-14, 3.244e-14, 5.171e-14, 7.394e-14, 9.676e-14,
    1.199e-13, 1.467e-13, 1.737e-13, 2.02e-13, 2.425e-13, 3.016e-13,
    3.7e-13, 4.617e-13, 5.949e-13, 7.473e-13, 9.378e-13, 1.191e-12,
    1.481e-12, 1.813e-12, 2.232e-12, 2.722e-12, 3.254e-12, 3.845e-12,
    4.458e-12, 5.048e-12, 5.511e-12, 5.898e-12, 6.204e-12, 6.293e-12,
    6.386e-12, 6.467e-12, 6.507e-12, 6.466e-12, 6.443e-12, 6.598e-12,
    6.873e-12, 7.3e-12, 7.816e-12, 8.368e-12, 8.643e-12, 8.466e-12,
    7.871e-12, 6.853e-12, 5.714e-12, 4.482e-12, 3.392e-12, 2.613e-12,
    2.008e-12, 1.562e-12, 1.228e-12, 9.888e-13, 7.646e-13, 5.769e-13,
    4.368e-13, 3.324e-13, 2.508e-13, 1.916e-13
  };

  static double xfcrev[15] =
    { 1.003, 1.009, 1.015, 1.023, 1.029, 1.033, 1.037,
    1.039, 1.04, 1.046, 1.036, 1.027, 1.01, 1.002, 1.
  };

  double sfac;

  /* Get H2O continuum absorption... */
  const double xw = nu / 10 + 1;
  if (xw >= 1 && xw < 2001) {
    const int iw = (int) xw;
    const double dw = xw - iw;
    const double ew = 1 - dw;
    const double cw296 = ew * h2o296[iw - 1] + dw * h2o296[iw];
    const double cw260 = ew * h2o260[iw - 1] + dw * h2o260[iw];
    const double cwfrn = ew * h2ofrn[iw - 1] + dw * h2ofrn[iw];
    if (nu <= 820 || nu >= 960) {
      sfac = 1;
    } else {
      const double xx = (nu - 820) / 10;
      const int ix = (int) xx;
      const double dx = xx - ix;
      sfac = (1 - dx) * xfcrev[ix] + dx * xfcrev[ix + 1];
    }
    const double ctwslf =
      sfac * cw296 * pow(cw260 / cw296, (296 - t) / (296 - 260));
```

```
    const double vf2 = POW2(nu - 370);
    const double vf6 = POW3(vf2);
    const double fscal = 36100 / (vf2 + vf6 * 1e-8 + 36100) * -.25 + 1;
    const double ctwfrn = cwfrn * fscal;
    const double a1 = nu * u * tanh(.7193876 / t * nu);
    const double a2 = 296 / t;
    const double a3 = p / P0 * (q * ctwslf + (1 - q) * ctwfrn) * 1e-20;
    return a1 * a2 * a3;
  } else
    return 0;
}
```

ctmn2()

```
double ctmn2 (
            const double nu,
            const double p,
            const double t )
```

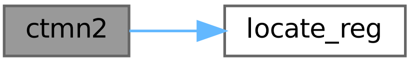
Compute nitrogen continuum (absorption coefficient).

Definition at line 2796 of file jurassic.c.

```
                    {

  static double ba[98] = { 0., 4.45e-8, 5.22e-8, 6.46e-8, 7.75e-8, 9.03e-8,
    1.06e-7, 1.21e-7, 1.37e-7, 1.57e-7, 1.75e-7, 2.01e-7, 2.3e-7,
    2.59e-7, 2.95e-7, 3.26e-7, 3.66e-7, 4.05e-7, 4.47e-7, 4.92e-7,
    5.34e-7, 5.84e-7, 6.24e-7, 6.67e-7, 7.14e-7, 7.26e-7, 7.54e-7,
    7.84e-7, 8.09e-7, 8.42e-7, 8.62e-7, 8.87e-7, 9.11e-7, 9.36e-7,
    9.76e-7, 1.03e-6, 1.11e-6, 1.23e-6, 1.39e-6, 1.61e-6, 1.76e-6,
    1.94e-6, 1.97e-6, 1.87e-6, 1.75e-6, 1.56e-6, 1.42e-6, 1.35e-6,
    1.32e-6, 1.29e-6, 1.29e-6, 1.29e-6, 1.3e-6, 1.32e-6, 1.33e-6,
    1.34e-6, 1.35e-6, 1.33e-6, 1.31e-6, 1.29e-6, 1.24e-6, 1.2e-6,
    1.16e-6, 1.1e-6, 1.04e-6, 9.96e-7, 9.38e-7, 8.63e-7, 7.98e-7,
    7.26e-7, 6.55e-7, 5.94e-7, 5.35e-7, 4.74e-7, 4.24e-7, 3.77e-7,
    3.33e-7, 2.96e-7, 2.63e-7, 2.34e-7, 2.08e-7, 1.85e-7, 1.67e-7,
    1.47e-7, 1.32e-7, 1.2e-7, 1.09e-7, 9.85e-8, 9.08e-8, 8.18e-8,
    7.56e-8, 6.85e-8, 6.14e-8, 5.83e-8, 5.77e-8, 5e-8, 4.32e-8, 0.
  };

  static double betaa[98] = { 802., 802., 761., 722., 679., 646., 609., 562.,
    511., 472., 436., 406., 377., 355., 338., 319., 299., 278., 255.,
    233., 208., 184., 149., 107., 66., 25., -13., -49., -82., -104.,
    -119., -130., -139., -144., -146., -146., -147., -148., -150.,
    -153., -160., -169., -181., -189., -195., -200., -205., -209.,
    -211., -210., -210., -209., -205., -199., -190., -180., -168.,
    -157., -143., -126., -108., -89., -63., -32., 1., 35., 65., 95.,
    121., 141., 152., 161., 164., 164., 161., 155., 148., 143., 137.,
    133., 131., 133., 139., 150., 165., 187., 213., 248., 284., 321.,
    372., 449., 514., 569., 609., 642., 673., 673.
  };

  static double nua[98] = { 2120., 2125., 2130., 2135., 2140., 2145., 2150.,
    2155., 2160., 2165., 2170., 2175., 2180., 2185., 2190., 2195.,
    2200., 2205., 2210., 2215., 2220., 2225., 2230., 2235., 2240.,
    2245., 2250., 2255., 2260., 2265., 2270., 2275., 2280., 2285.,
    2290., 2295., 2300., 2305., 2310., 2315., 2320., 2325., 2330.,
    2335., 2340., 2345., 2350., 2355., 2360., 2365., 2370., 2375.,
    2380., 2385., 2390., 2395., 2400., 2405., 2410., 2415., 2420.,
    2425., 2430., 2435., 2440., 2445., 2450., 2455., 2460., 2465.,
    2470., 2475., 2480., 2485., 2490., 2495., 2500., 2505., 2510.,
    2515., 2520., 2525., 2530., 2535., 2540., 2545., 2550., 2555.,
    2560., 2565., 2570., 2575., 2580., 2585., 2590., 2595., 2600., 2605.
  };

  const double q_n2 = 0.79, t0 = 273.0, tr = 296.0;

  /* Check wavenumber range... */
  if (nu < nua[0] || nu > nua[97])
    return 0;

  /* Interpolate B and beta... */
  const int idx = locate_reg(nua, 98, nu);
  const double b = LIN(nua[idx], ba[idx], nua[idx + 1], ba[idx + 1], nu);
  const double beta =
    LIN(nua[idx], betaa[idx], nua[idx + 1], betaa[idx + 1], nu);

  /* Compute absorption coefficient... */
  return 0.1 * POW2(p / P0 * t0 / t) * exp(beta * (1 / tr - 1 / t))
```

```
     * q_n2 * b * (q_n2 + (1 - q_n2) * (1.294 - 0.4545 * t / tr));
}
```

Here is the call graph for this function:



ctmo2()

```
double ctmo2 (
            const double nu,
            const double p,
            const double t )
```

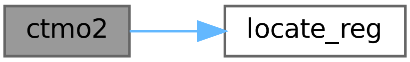
Compute oxygen continuum (absorption coefficient).

Definition at line 2861 of file jurassic.c.

```
                  {

  static double ba[90] = { 0., .061, .074, .084, .096, .12, .162, .208, .246,
    .285, .314, .38, .444, .5, .571, .673, .768, .853, .966, 1.097,
    1.214, 1.333, 1.466, 1.591, 1.693, 1.796, 1.922, 2.037, 2.154,
    2.264, 2.375, 2.508, 2.671, 2.847, 3.066, 3.417, 3.828, 4.204,
    4.453, 4.599, 4.528, 4.284, 3.955, 3.678, 3.477, 3.346, 3.29,
    3.251, 3.231, 3.226, 3.212, 3.192, 3.108, 3.033, 2.911, 2.798,
    2.646, 2.508, 2.322, 2.13, 1.928, 1.757, 1.588, 1.417, 1.253,
    1.109, .99, .888, .791, .678, .587, .524, .464, .403, .357, .32,
    .29, .267, .242, .215, .182, .16, .146, .128, .103, .087, .081,
    .071, .064, 0.
  };

  static double betaa[90] = { 467., 467., 400., 315., 379., 368., 475., 521.,
    531., 512., 442., 444., 430., 381., 335., 324., 296., 248., 215.,
    193., 158., 127., 101., 71., 31., -6., -26., -47., -63., -79.,
    -88., -88., -87., -90., -98., -99., -109., -134., -160., -167.,
    -164., -158., -153., -151., -156., -166., -168., -173., -170.,
    -161., -145., -126., -108., -84., -59., -29., 4., 41., 73., 97.,
    123., 159., 198., 220., 242., 256., 281., 311., 334., 319., 313.,
    321., 323., 310., 315., 320., 335., 361., 378., 373., 338., 319.,
    346., 322., 291., 290., 350., 371., 504., 504.
  };

  static double nua[90] = { 1360., 1365., 1370., 1375., 1380., 1385., 1390.,
    1395., 1400., 1405., 1410., 1415., 1420., 1425., 1430., 1435.,
    1440., 1445., 1450., 1455., 1460., 1465., 1470., 1475., 1480.,
    1485., 1490., 1495., 1500., 1505., 1510., 1515., 1520., 1525.,
    1530., 1535., 1540., 1545., 1550., 1555., 1560., 1565., 1570.,
    1575., 1580., 1585., 1590., 1595., 1600., 1605., 1610., 1615.,
    1620., 1625., 1630., 1635., 1640., 1645., 1650., 1655., 1660.,
    1665., 1670., 1675., 1680., 1685., 1690., 1695., 1700., 1705.,
    1710., 1715., 1720., 1725., 1730., 1735., 1740., 1745., 1750.,
    1755., 1760., 1765., 1770., 1775., 1780., 1785., 1790., 1795.,
    1800., 1805.
  };

  const double q_o2 = 0.21, t0 = 273, tr = 296;

  /* Check wavenumber range... */
  if (nu < nua[0] || nu > nua[89])
    return 0;

  /* Interpolate B and beta... */
  const int idx = locate_reg(nua, 90, nu);
```

```
  const double b = LIN(nua[idx], ba[idx], nua[idx + 1], ba[idx + 1], nu);
  const double beta =
    LIN(nua[idx], betaa[idx], nua[idx + 1], betaa[idx + 1], nu);

  /* Compute absorption coefficient... */
  return 0.1 * POW2(p / P0 * t0 / t) * exp(beta * (1 / tr - 1 / t)) * q_o2 *
    b;
}
```

Here is the call graph for this function:



copy_atm()

```
void copy_atm (
            const ctl_t * ctl,
            atm_t * atm_dest,
            const atm_t * atm_src,
            const int init )
```

Copy and initialize atmospheric data.

Definition at line 2921 of file jurassic.c.

```
                        {

  /* Data size... */
  const size_t s = (size_t) atm_src->np * sizeof(double);

  /* Copy data... */
  atm_dest->np = atm_src->np;
  memcpy(atm_dest->time, atm_src->time, s);
  memcpy(atm_dest->z, atm_src->z, s);
  memcpy(atm_dest->lon, atm_src->lon, s);
  memcpy(atm_dest->lat, atm_src->lat, s);
  memcpy(atm_dest->p, atm_src->p, s);
  memcpy(atm_dest->t, atm_src->t, s);
  for (int ig = 0; ig < ctl->ng; ig++)
    memcpy(atm_dest->q[ig], atm_src->q[ig], s);
  for (int iw = 0; iw < ctl->nw; iw++)
    memcpy(atm_dest->k[iw], atm_src->k[iw], s);
  atm_dest->clz = atm_src->clz;
  atm_dest->cldz = atm_src->cldz;
  for (int icl = 0; icl < ctl->ncl; icl++)
    atm_dest->clk[icl] = atm_src->clk[icl];
  atm_dest->sfz = atm_src->sfz;
  atm_dest->sfp = atm_src->sfp;
  atm_dest->sft = atm_src->sft;
  for (int isf = 0; isf < ctl->nsf; isf++)
    atm_dest->sfeps[isf] = atm_src->sfeps[isf];

  /* Initialize... */
  if (init)
    for (int ip = 0; ip < atm_dest->np; ip++) {
      atm_dest->p[ip] = 0;
      atm_dest->t[ip] = 0;
      for (int ig = 0; ig < ctl->ng; ig++)
        atm_dest->q[ig][ip] = 0;
      for (int iw = 0; iw < ctl->nw; iw++)
        atm_dest->k[iw][ip] = 0;
      atm_dest->clz = 0;
      atm_dest->cldz = 0;
      for (int icl = 0; icl < ctl->ncl; icl++)
```

```
        atm_dest->clk[icl] = 0;
      atm_dest->sfz = 0;
      atm_dest->sfp = 0;
      atm_dest->sft = 0;
      for (int isf = 0; isf < ctl->nsf; isf++)
        atm_dest->sfeps[isf] = 1;
    }
}
```

copy_obs()

```
void copy_obs (
            const ctl_t * ctl,
            obs_t * obs_dest,
            const obs_t * obs_src,
            const int init )
```

Copy and initialize observation data.

Definition at line 2975 of file jurassic.c.

```
                    {

  /* Data size... */
  const size_t s = (size_t) obs_src->nr * sizeof(double);

  /* Copy data... */
  obs_dest->nr = obs_src->nr;
  memcpy(obs_dest->time, obs_src->time, s);
  memcpy(obs_dest->obsz, obs_src->obsz, s);
  memcpy(obs_dest->obslon, obs_src->obslon, s);
  memcpy(obs_dest->obslat, obs_src->obslat, s);
  memcpy(obs_dest->vpz, obs_src->vpz, s);
  memcpy(obs_dest->vplon, obs_src->vplon, s);
  memcpy(obs_dest->vplat, obs_src->vplat, s);
  memcpy(obs_dest->tpz, obs_src->tpz, s);
  memcpy(obs_dest->tplon, obs_src->tplon, s);
  memcpy(obs_dest->tplat, obs_src->tplat, s);
  for (int id = 0; id < ctl->nd; id++)
    memcpy(obs_dest->rad[id], obs_src->rad[id], s);
  for (int id = 0; id < ctl->nd; id++)
    memcpy(obs_dest->tau[id], obs_src->tau[id], s);

  /* Initialize... */
  if (init)
    for (int id = 0; id < ctl->nd; id++)
      for (int ir = 0; ir < obs_dest->nr; ir++)
        if (isfinite(obs_dest->rad[id][ir])) {
          obs_dest->rad[id][ir] = 0;
          obs_dest->tau[id][ir] = 0;
        }
}
```

find_emitter()

```
int find_emitter (
            const ctl_t * ctl,
            const char * emitter )
```

Find index of an emitter.

Definition at line 3013 of file jurassic.c.

```
                         {

  for (int ig = 0; ig < ctl->ng; ig++)
    if (strcasecmp(ctl->emitter[ig], emitter) == 0)
      return ig;

  return -1;
}
```

formod()

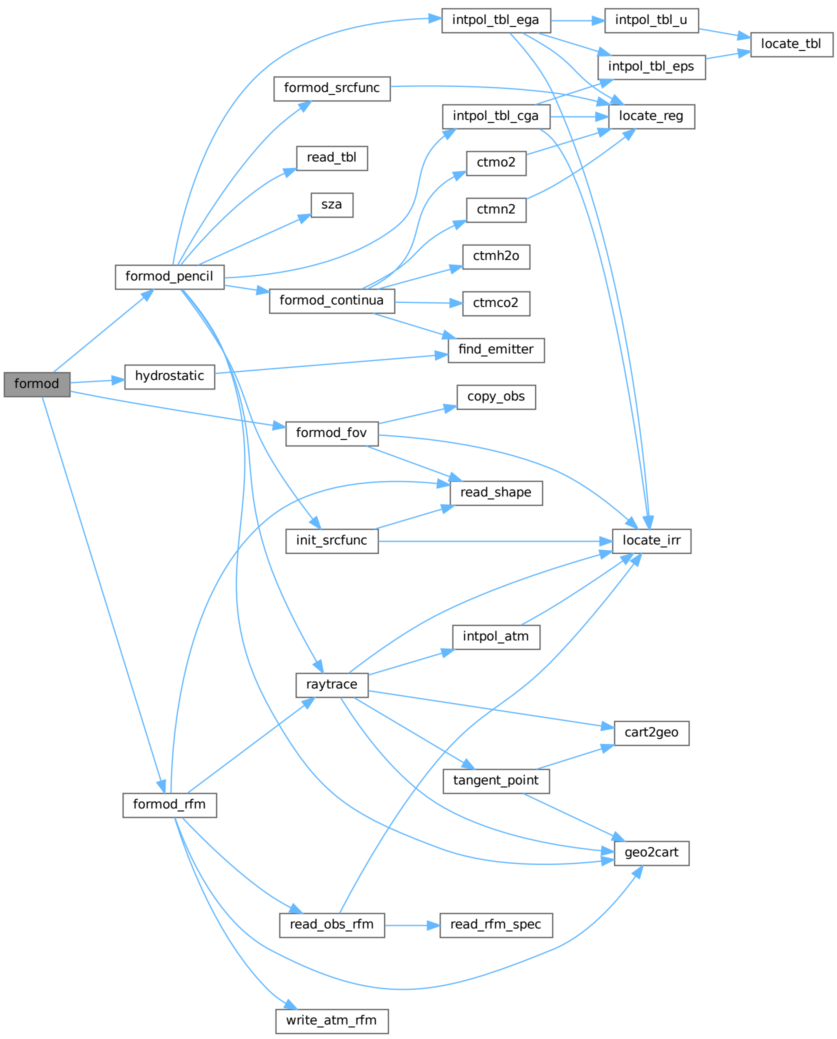
```
void formod (
            const ctl_t * ctl,
            atm_t * atm,
            obs_t * obs )
```

Determine ray paths and compute radiative transfer.

Definition at line 3026 of file jurassic.c.

```
                {

  int *mask;

  /* Allocate... */
  ALLOC(mask, int,
        ND * NR);

  /* Save observation mask... */
  for (int id = 0; id < ctl->nd; id++)
    for (int ir = 0; ir < obs->nr; ir++)
      mask[id * NR + ir] = !isfinite(obs->rad[id][ir]);

  /* Hydrostatic equilibrium... */
  hydrostatic(ctl, atm);

  /* CGA or EGA forward model... */
  if (ctl->formod == 0 || ctl->formod == 1)
    for (int ir = 0; ir < obs->nr; ir++)
      formod_pencil(ctl, atm, obs, ir);

  /* Call RFM... */
  else if (ctl->formod == 2)
    formod_rfm(ctl, atm, obs);

  /* Apply field-of-view convolution... */
  formod_fov(ctl, obs);

  /* Convert radiance to brightness temperature... */
  if (ctl->write_bbt)
    for (int id = 0; id < ctl->nd; id++)
      for (int ir = 0; ir < obs->nr; ir++)
        obs->rad[id][ir] = BRIGHT(obs->rad[id][ir], ctl->nu[id]);

  /* Apply observation mask... */
  for (int id = 0; id < ctl->nd; id++)
    for (int ir = 0; ir < obs->nr; ir++)
      if (mask[id * NR + ir])
        obs->rad[id][ir] = NAN;

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

Here is the call graph for this function:



formod_continua()

```
void formod_continua (
            const ctl_t * ctl,
            const los_t * los,
            const int ip,
            double * beta )
```

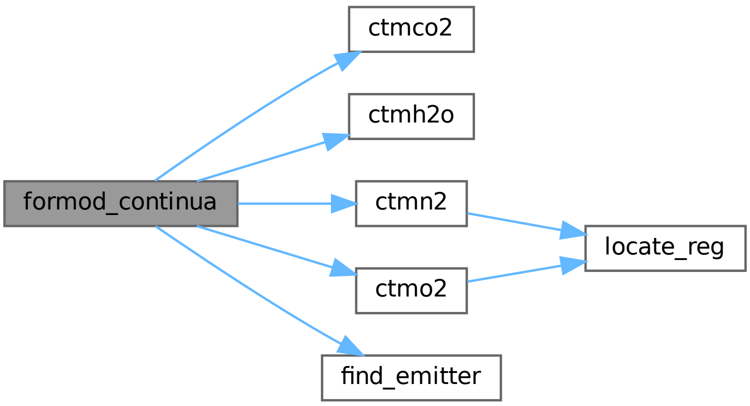
Compute absorption coefficient of continua.

Definition at line 3075 of file jurassic.c.

```
                  {

  static int ig_co2 = -999, ig_h2o = -999;

  /* Extinction... */
  for (int id = 0; id < ctl->nd; id++)
    beta[id] = los->k[ip][id];

```

```
  /* CO2 continuum... */
  if (ctl->ctm_co2) {
    if (ig_co2 == -999)
      ig_co2 = find_emitter(ctl, "CO2");
    if (ig_co2 >= 0)
      for (int id = 0; id < ctl->nd; id++)
        beta[id] += ctmco2(ctl->nu[id], los->p[ip], los->t[ip],
                           los->u[ip][ig_co2]) / los->ds[ip];
  }

  /* H2O continuum... */
  if (ctl->ctm_h2o) {
    if (ig_h2o == -999)
      ig_h2o = find_emitter(ctl, "H2O");
    if (ig_h2o >= 0)
      for (int id = 0; id < ctl->nd; id++)
        beta[id] += ctmh2o(ctl->nu[id], los->p[ip], los->t[ip],
                           los->q[ip][ig_h2o], los->u[ip][ig_h2o])
          / los->ds[ip];
  }

  /* N2 continuum... */
  if (ctl->ctm_n2)
    for (int id = 0; id < ctl->nd; id++)
      beta[id] += ctmn2(ctl->nu[id], los->p[ip], los->t[ip]);

  /* O2 continuum... */
  if (ctl->ctm_o2)
    for (int id = 0; id < ctl->nd; id++)
      beta[id] += ctmo2(ctl->nu[id], los->p[ip], los->t[ip]);
}
```

Here is the call graph for this function:



formod_fov()

```
void formod_fov (
            const ctl_t * ctl,
            obs_t * obs )
```

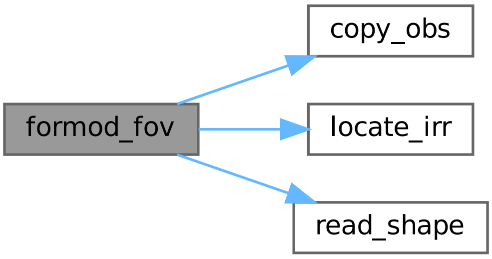
Apply field of view convolution.

Definition at line 3121 of file jurassic.c.

```
               {

  static double dz[NSHAPE], w[NSHAPE];

```

```
  static int init = 0, n;

  obs_t *obs2;

  double rad[ND][NR], tau[ND][NR], z[NR];

  /* Do not take into account FOV... */
  if (ctl->fov[0] == '-')
    return;

  /* Initialize FOV data... */
  if (!init) {
    init = 1;
    read_shape(ctl->fov, dz, w, &n);
  }

  /* Allocate... */
  ALLOC(obs2, obs_t, 1);

  /* Copy observation data... */
  copy_obs(ctl, obs2, obs, 0);

  /* Loop over ray paths... */
  for (int ir = 0; ir < obs->nr; ir++) {

    /* Get radiance and transmittance profiles... */
    int nz = 0;
    for (int ir2 = MAX(ir - NFOV, 0);
         ir2 < MIN(ir + 1 + NFOV, obs->nr); ir2++)
      if (obs->time[ir2] == obs->time[ir]) {
        z[nz] = obs2->vpz[ir2];
        for (int id = 0; id < ctl->nd; id++) {
          rad[id][nz] = obs2->rad[id][ir2];
          tau[id][nz] = obs2->tau[id][ir2];
        }
        nz++;
      }
    if (nz < 2)
      ERRMSG("Cannot apply FOV convolution!");

    /* Convolute profiles with FOV... */
    double wsum = 0;
    for (int id = 0; id < ctl->nd; id++) {
      obs->rad[id][ir] = 0;
      obs->tau[id][ir] = 0;
    }
    for (int i = 0; i < n; i++) {
      const double zfov = obs->vpz[ir] + dz[i];
      const int idx = locate_irr(z, nz, zfov);
      for (int id = 0; id < ctl->nd; id++) {
        obs->rad[id][ir] += w[i]
          * LIN(z[idx], rad[id][idx], z[idx + 1], rad[id][idx + 1], zfov);
        obs->tau[id][ir] += w[i]
          * LIN(z[idx], tau[id][idx], z[idx + 1], tau[id][idx + 1], zfov);
      }
      wsum += w[i];
    }
    for (int id = 0; id < ctl->nd; id++) {
      obs->rad[id][ir] /= wsum;
      obs->tau[id][ir] /= wsum;
    }
  }

  /* Free... */
  free(obs2);
}
```

Here is the call graph for this function:



formod_pencil()

```
void formod_pencil (
            const ctl_t * ctl,
            const atm_t * atm,
            obs_t * obs,
            const int ir )
```

Compute radiative transfer for a pencil beam.

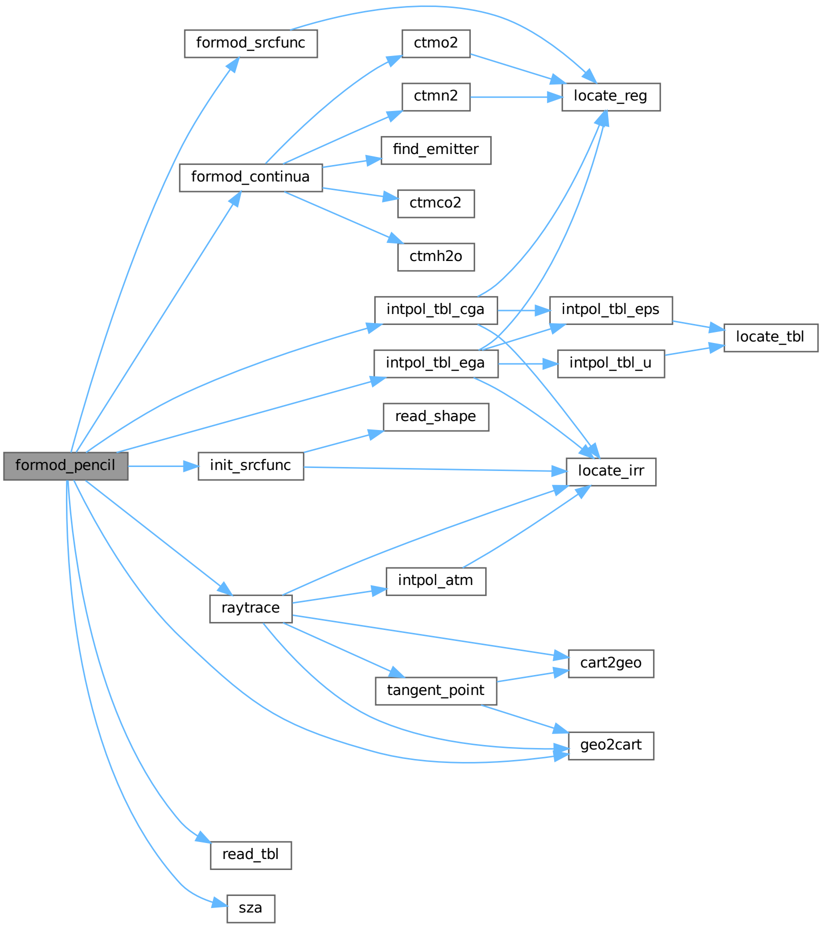
Definition at line 3196 of file jurassic.c.

```
                    {

  static tbl_t *tbl;

  static int init = 0;

  los_t *los;

  double beta_ctm[ND], rad[ND], tau[ND], tau_refl[ND],
    tau_path[ND][NG], tau_gas[ND], x0[3], x1[3];

  /* Initialize look-up tables... */
  if (!init) {
    init = 1;
    ALLOC(tbl, tbl_t, 1);
    read_tbl(ctl, tbl);
    init_srcfunc(ctl, tbl);
  }

  /* Allocate... */
  ALLOC(los, los_t, 1);

  /* Initialize... */
  for (int id = 0; id < ctl->nd; id++) {
    rad[id] = 0;
    tau[id] = 1;
    for (int ig = 0; ig < ctl->ng; ig++)
      tau_path[id][ig] = 1;
  }

  /* Raytracing... */
  raytrace(ctl, atm, obs, los, ir);

  /* Loop over LOS points... */
  for (int ip = 0; ip < los->np; ip++) {

    /* Get trace gas transmittance... */
    if (ctl->formod == 0)
      intpol_tbl_cga(ctl, tbl, los, ip, tau_path, tau_gas);
```

```
    else
      intpol_tbl_ega(ctl, tbl, los, ip, tau_path, tau_gas);

    /* Get continuum absorption... */
    formod_continua(ctl, los, ip, beta_ctm);

    /* Compute Planck function... */
    formod_srcfunc(ctl, tbl, los->t[ip], los->src[ip]);

    /* Loop over channels... */
    for (int id = 0; id < ctl->nd; id++)
      if (tau_gas[id] > 0) {

        /* Get segment emissivity... */
        los->eps[ip][id] = 1 - tau_gas[id] * exp(-beta_ctm[id] * los->ds[ip]);

        /* Compute radiance... */
        rad[id] += los->src[ip][id] * los->eps[ip][id] * tau[id];

        /* Compute path transmittance... */
        tau[id] *= (1 - los->eps[ip][id]);
      }
  }

  /* Check whether LOS hit the ground... */
  if (ctl->sftype >= 1 && los->sft > 0) {

    /* Add surface emissions... */
    double src_sf[ND];
    formod_srcfunc(ctl, tbl, los->sft, src_sf);
    for (int id = 0; id < ctl->nd; id++)
      rad[id] += los->sfeps[id] * src_sf[id] * tau[id];

    /* Check reflectivity... */
    int refl = 0;
    if (ctl->sftype >= 2)
      for (int id = 0; id < ctl->nd; id++)
        if (los->sfeps[id] < 1) {
          refl = 1;
          break;
        }

    /* Calculate reflection... */
    if (refl) {

      /* Initialize... */
      for (int id = 0; id < ctl->nd; id++)
        tau_refl[id] = 1;

      /* Add down-welling radiance... */
      for (int ip = los->np - 1; ip >= 0; ip--)
        for (int id = 0; id < ctl->nd; id++) {
          rad[id] += los->src[ip][id] * los->eps[ip][id] * tau_refl[id]
            * tau[id] * (1 - los->sfeps[id]);
          tau_refl[id] *= (1 - los->eps[ip][id]);
        }

      /* Add solar term... */
      if (ctl->sftype >= 3) {

        /* Get solar zenith angle... */
        double sza2;
        if (ctl->sfsza < 0)
          sza2 =
            sza(obs->time[ir], los->lon[los->np - 1], los->lat[los->np - 1]);
        else
          sza2 = ctl->sfsza;

        /* Check solar zenith angle... */
        if (sza2 < 89.999) {

          /* Get angle of incidence... */
          geo2cart(los->z[los->np - 1], los->lon[los->np - 1],
                   los->lat[los->np - 1], x0);
          geo2cart(los->z[0], los->lon[0], los->lat[0], x1);
          for (int i = 0; i < 3; i++)
            x1[i] -= x0[i];
          const double cosa = DOTP(x0, x1) / NORM(x0) / NORM(x1);

          /* Get ratio of SZA and incident radiation... */
          const double rcos = cosa / cos(DEG2RAD(sza2));

          /* Add solar radiation... */
          for (int id = 0; id < ctl->nd; id++)
            rad[id] += 6.764e-5 / (2. * M_PI) * PLANCK(TSUN, ctl->nu[id])
              * tau_refl[id] * (1 - los->sfeps[id]) * tau[id] * rcos;
        }
```

```
      }
    }
  }

  /* Copy results... */
  for (int id = 0; id < ctl->nd; id++) {
    obs->rad[id][ir] = rad[id];
    obs->tau[id][ir] = tau[id];
  }

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

Here is the call graph for this function:



formod_rfm()

```
void formod_rfm (
            const ctl_t * ctl,
            const atm_t * atm,
            obs_t * obs )
```

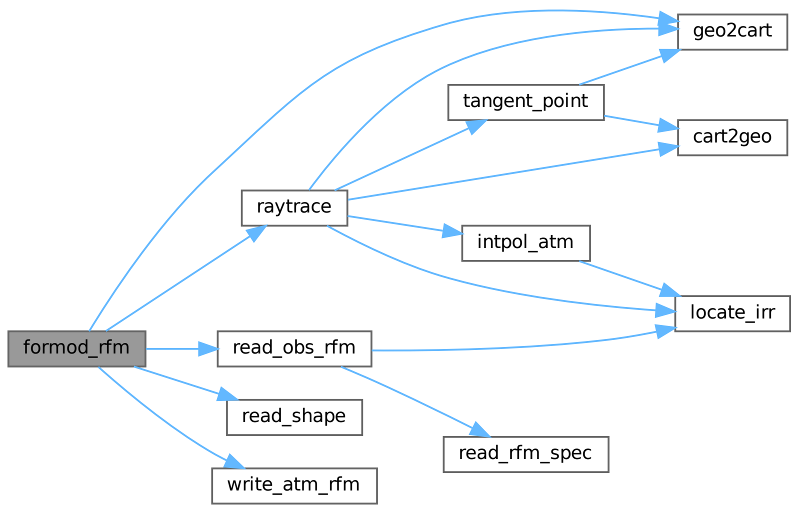
Apply RFM for radiative transfer calculations.

Definition at line 3342 of file jurassic.c.

```
          {

  los_t *los;

  FILE *out;

  char cmd[2 * LEN], filename[2 * LEN],
    rfmflg[LEN] = { "RAD TRA MIX LIN SFC" };

  double f[NSHAPE], nu[NSHAPE], nu0, nu1, obsz = -999, tsurf,
    xd[3], xo[3], xv[3], z[NR], zmin, zmax;

  int n, nadir = 0;

  /* Allocate... */
  ALLOC(los, los_t, 1);

  /* Check observer positions... */
  for (int ir = 1; ir < obs->nr; ir++)
    if (obs->obsz[ir] != obs->obsz[0]
        || obs->obslon[ir] != obs->obslon[0]
        || obs->obslat[ir] != obs->obslat[0])
      ERRMSG("RFM interface requires identical observer positions!");

  /* Check extinction data... */
  for (int iw = 0; iw < ctl->nw; iw++)
    for (int ip = 0; ip < atm->np; ip++)
      if (atm->k[iw][ip] != 0)
        ERRMSG("RFM interface cannot handle extinction data!");

  /* Get altitude range of atmospheric data... */
  gsl_stats_minmax(&zmin, &zmax, atm->z, 1, (size_t) atm->np);

  /* Observer within atmosphere? */
  if (obs->obsz[0] >= zmin && obs->obsz[0] <= zmax) {
    obsz = obs->obsz[0];
    strcat(rfmflg, " OBS");
  }

  /* Determine tangent altitude or air mass factor... */
  for (int ir = 0; ir < obs->nr; ir++) {

    /* Raytracing... */
    raytrace(ctl, atm, obs, los, ir);

    /* Nadir? */
    if (obs->tpz[ir] <= zmin) {
      geo2cart(obs->obsz[ir], obs->obslon[ir], obs->obslat[ir], xo);
      geo2cart(obs->vpz[ir], obs->vplon[ir], obs->vplat[ir], xv);
      for (int i = 0; i < 3; i++)
        xd[i] = xo[i] - xv[i];
      z[ir] = NORM(xo) * NORM(xd) / DOTP(xo, xd);
      nadir++;
    } else
      z[ir] = obs->tpz[ir];
  }
  if (nadir > 0 && nadir < obs->nr)
    ERRMSG("Limb and nadir not simultaneously possible!");

  /* Nadir? */
  if (nadir)
    strcat(rfmflg, " NAD");

  /* Get surface temperature... */
  tsurf = atm->t[gsl_stats_min_index(atm->z, 1, (size_t) atm->np)];

  /* Refraction? */
  if (!nadir && !ctl->refrac)
    strcat(rfmflg, " GEO");

  /* Continua? */
  if (ctl->ctm_co2 || ctl->ctm_h2o || ctl->ctm_n2 || ctl->ctm_o2)
    strcat(rfmflg, " CTM");

  /* Write atmospheric data file... */
  write_atm_rfm("rfm.atm", ctl, atm);

  /* Loop over channels... */
  for (int id = 0; id < ctl->nd; id++) {

    /* Read filter function... */
    sprintf(filename, "%s_%.4f.filt", ctl->tblbase, ctl->nu[id]);
    read_shape(filename, nu, f, &n);

    /* Set spectral range... */
    nu0 = nu[0];
    nu1 = nu[n - 1];
```

```
    /* Create RFM driver file... */
    if (!(out = fopen("rfm.drv", "w")))
      ERRMSG("Cannot create file!");
    fprintf(out, "*HDR\nRFM call by JURASSIC.\n");
    fprintf(out, "*FLG\n%s\n", rfmflg);
    fprintf(out, "*SPC\n%.4f %.4f 0.0005\n", nu0, nu1);
    fprintf(out, "*GAS\n");
    for (int ig = 0; ig < ctl->ng; ig++)
      fprintf(out, "%s\n", ctl->emitter[ig]);
    fprintf(out, "*ATM\nrfm.atm\n");
    fprintf(out, "*TAN\n");
    for (int ir = 0; ir < obs->nr; ir++)
      fprintf(out, "%g\n", z[ir]);
    fprintf(out, "*SFC\n%g 1.0\n", tsurf);
    if (obsz >= 0)
      fprintf(out, "*OBS\n%g\n", obsz);
    fprintf(out, "*HIT\n%s\n", ctl->rfmhit);
    fprintf(out, "*XSC\n");
    for (int ig = 0; ig < ctl->ng; ig++)
      if (ctl->rfmxsc[ig][0] != '-')
        fprintf(out, "%s\n", ctl->rfmxsc[ig]);
    fprintf(out, "*END\n");
    fclose(out);

    /* Remove temporary files... */
    if (system("rm -f rfm.runlog rad_*.asc tra_*.asc"))
      ERRMSG("Cannot remove temporary files!");

    /* Call RFM... */
    sprintf(cmd, "echo | %s", ctl->rfmbin);
    if (system(cmd))
      ERRMSG("Error while calling RFM!");

    /* Read data... */
    for (int ir = 0; ir < obs->nr; ir++) {
      obs->rad[id][ir] = read_obs_rfm("rad", z[ir], nu, f, n) * 1e-5;
      obs->tau[id][ir] = read_obs_rfm("tra", z[ir], nu, f, n);
    }
  }

  /* Remove temporary files... */
  if (system("rm -f rfm.drv rfm.atm rfm.runlog rad_*.asc tra_*.asc"))
    ERRMSG("Error while removing temporary files!");

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

Here is the call graph for this function:



formod_srcfunc()

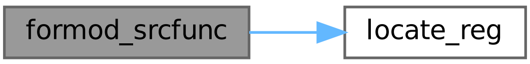
```
void formod_srcfunc (
            const ctl_t * ctl,
            const tbl_t * tbl,
            const double t,
            double * src )
```

Compute Planck source function.

Definition at line 3483 of file jurassic.c.

```
                {

  /* Determine index in temperature array... */
  const int it = locate_reg(tbl->st, TBLNS, t);

  /* Interpolate Planck function value... */
  for (int id = 0; id < ctl->nd; id++)
    src[id] = LIN(tbl->st[it], tbl->sr[it][id],
                  tbl->st[it + 1], tbl->sr[it + 1][id], t);
}
```

Here is the call graph for this function:



geo2cart()

```
void geo2cart (
            const double z,
            const double lon,
            const double lat,
            double * x )
```

Convert geolocation to Cartesian coordinates.

Definition at line 3500 of file jurassic.c.

```
              {

  const double radius = z + RE;

  const double latrad = lat / 180. * M_PI;
  const double lonrad = lon / 180. * M_PI;

  const double coslat = cos(latrad);

  x[0] = radius * coslat * cos(lonrad);
  x[1] = radius * coslat * sin(lonrad);
  x[2] = radius * sin(latrad);
}
```

hydrostatic()

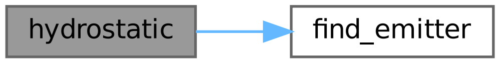
```
void hydrostatic (
            const ctl_t * ctl,
            atm_t * atm )
```

Set hydrostatic equilibrium.

Definition at line 3520 of file jurassic.c.

```
                    {

  const double mmair = 28.96456e-3, mmh2o = 18.0153e-3;

  const int ipts = 20;

  static int ig_h2o = -999;

  double dzmin = 1e99, e = 0;

  int ipref = 0;

  /* Check reference height... */
  if (ctl->hydz < 0)
    return;

  /* Determine emitter index of H2O... */
  if (ig_h2o == -999)
    ig_h2o = find_emitter(ctl, "H2O");

  /* Find air parcel next to reference height... */
  for (int ip = 0; ip < atm->np; ip++)
    if (fabs(atm->z[ip] - ctl->hydz) < dzmin) {
      dzmin = fabs(atm->z[ip] - ctl->hydz);
      ipref = ip;
    }

  /* Upper part of profile... */
  for (int ip = ipref + 1; ip < atm->np; ip++) {
    double mean = 0;
    for (int i = 0; i < ipts; i++) {
      if (ig_h2o >= 0)
        e = LIN(0.0, atm->q[ig_h2o][ip - 1],
                ipts - 1.0, atm->q[ig_h2o][ip], (double) i);
      mean += (e * mmh2o + (1 - e) * mmair)
        * G0 / RI
        / LIN(0.0, atm->t[ip - 1], ipts - 1.0, atm->t[ip], (double) i) / ipts;
    }

    /* Compute p(z,T)... */
    atm->p[ip] =
      exp(log(atm->p[ip - 1]) - mean * 1000 * (atm->z[ip] - atm->z[ip - 1]));
  }

  /* Lower part of profile... */
  for (int ip = ipref - 1; ip >= 0; ip--) {
    double mean = 0;
    for (int i = 0; i < ipts; i++) {
      if (ig_h2o >= 0)
        e = LIN(0.0, atm->q[ig_h2o][ip + 1],
                ipts - 1.0, atm->q[ig_h2o][ip], (double) i);
      mean += (e * mmh2o + (1 - e) * mmair)
        * G0 / RI
        / LIN(0.0, atm->t[ip + 1], ipts - 1.0, atm->t[ip], (double) i) / ipts;
    }

    /* Compute p(z,T)... */
    atm->p[ip] =
      exp(log(atm->p[ip + 1]) - mean * 1000 * (atm->z[ip] - atm->z[ip + 1]));
  }
}
```

Here is the call graph for this function:



idx2name()

```
void idx2name (
            const ctl_t * ctl,
            const int idx,
            char * quantity )
```

Determine name of state vector quantity for given index.

Definition at line 3586 of file jurassic.c.

```
                        {

  if (idx == IDXP)
    sprintf(quantity, "PRESSURE");

  if (idx == IDXT)
    sprintf(quantity, "TEMPERATURE");

  for (int ig = 0; ig < ctl->ng; ig++)
    if (idx == IDXQ(ig))
      sprintf(quantity, "%s", ctl->emitter[ig]);

  for (int iw = 0; iw < ctl->nw; iw++)
    if (idx == IDXK(iw))
      sprintf(quantity, "EXTINCT_WINDOW_%d", iw);

  if (idx == IDXCLZ)
    sprintf(quantity, "CLOUD_HEIGHT");

  if (idx == IDXCLDZ)
    sprintf(quantity, "CLOUD_DEPTH");

  for (int icl = 0; icl < ctl->ncl; icl++)
    if (idx == IDXCLK(icl))
      sprintf(quantity, "CLOUD_EXTINCT_%.4f", ctl->clnu[icl]);

  if (idx == IDXSFZ)
    sprintf(quantity, "SURFACE_HEIGHT");

  if (idx == IDXSFP)
    sprintf(quantity, "SURFACE_PRESSURE");

  if (idx == IDXSFT)
    sprintf(quantity, "SURFACE_TEMPERATURE");

  for (int isf = 0; isf < ctl->nsf; isf++)
    if (idx == IDXSFEPS(isf))
      sprintf(quantity, "SURFACE_EMISSIVITY_%.4f", ctl->sfnu[isf]);
}
```

init_srcfunc()

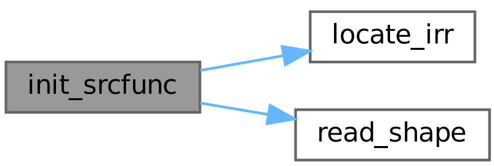
```
void init_srcfunc (
            const ctl_t * ctl,
            tbl_t * tbl )
```

Initialize source function table.

Definition at line 3631 of file jurassic.c.

```
                    {

  char filename[2 * LEN];

  double f[NSHAPE], nu[NSHAPE];

  int n;

  /* Write info... */
  LOG(1, "Initialize source function table...");
  LOG(2, "Number of data points: %d", TBLNS);

  /* Loop over channels... */
  for (int id = 0; id < ctl->nd; id++) {

    /* Read filter function... */
    sprintf(filename, "%s_%.4f.filt", ctl->tblbase, ctl->nu[id]);
    read_shape(filename, nu, f, &n);

    /* Get minimum grid spacing... */
    double dnu = 1.0;
    for (int i = 1; i < n; i++)
      dnu = MIN(dnu, nu[i] - nu[i - 1]);

    /* Compute source function table... */
#pragma omp parallel for default(none) shared(ctl,tbl,id,nu,f,n,dnu)
    for (int it = 0; it < TBLNS; it++) {

      /* Set temperature... */
      tbl->st[it] = LIN(0.0, TMIN, TBLNS - 1.0, TMAX, (double) it);

      /* Integrate Planck function... */
      double fsum = tbl->sr[it][id] = 0;
      for (double fnu = nu[0]; fnu <= nu[n - 1]; fnu += dnu) {
        const int i = locate_irr(nu, n, fnu);
        const double ff = LIN(nu[i], f[i], nu[i + 1], f[i + 1], fnu);
        fsum += ff;
        tbl->sr[it][id] += ff * PLANCK(tbl->st[it], fnu);
      }
      tbl->sr[it][id] /= fsum;
    }

    /* Write info... */
    LOG(2,
        "channel= %.4f cm^-1 | T= %g ... %g K | B= %g ... %g W/(m^2 sr cm^-1)",
        ctl->nu[id], tbl->st[0], tbl->st[TBLNS - 1], tbl->sr[0][id],
        tbl->sr[TBLNS - 1][id]);
  }
}
```

Here is the call graph for this function:



intpol_atm()

```
void intpol_atm (
            const ctl_t * ctl,
```

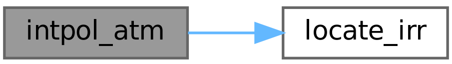
```
            const atm_t * atm,
            const double z,
            double * p,
            double * t,
            double * q,
            double * k )
```

Interpolate atmospheric data.

Definition at line 3685 of file jurassic.c.

```
                {

  /* Get array index... */
  const int ip = locate_irr(atm->z, atm->np, z);

  /* Interpolate... */
  *p = LOGY(atm->z[ip], atm->p[ip], atm->z[ip + 1], atm->p[ip + 1], z);
  *t = LIN(atm->z[ip], atm->t[ip], atm->z[ip + 1], atm->t[ip + 1], z);
  for (int ig = 0; ig < ctl->ng; ig++)
    q[ig] =
      LIN(atm->z[ip], atm->q[ig][ip], atm->z[ip + 1], atm->q[ig][ip + 1], z);
  for (int iw = 0; iw < ctl->nw; iw++)
    k[iw] =
      LIN(atm->z[ip], atm->k[iw][ip], atm->z[ip + 1], atm->k[iw][ip + 1], z);
}
```

Here is the call graph for this function:



intpol_tbl_cga()

```
void intpol_tbl_cga (
            const ctl_t * ctl,
            const tbl_t * tbl,
            const los_t * los,
            const int ip,
            double tau_path[ND][NG],
            double tau_seg[ND] )
```

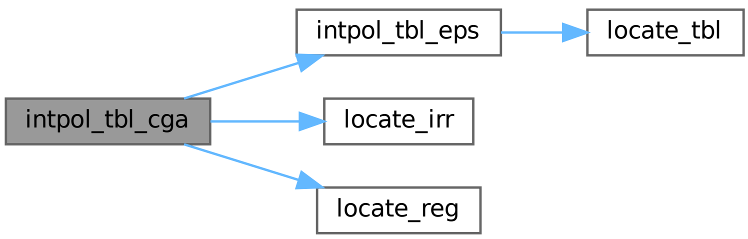
Get transmittance from look-up tables (CGA method).

Definition at line 3710 of file jurassic.c.

```
                            {

  double eps;

  /* Loop over channels... */
  for (int id = 0; id < ctl->nd; id++) {

    /* Initialize... */
    tau_seg[id] = 1;

    /* Loop over emitters.... */
    for (int ig = 0; ig < ctl->ng; ig++) {

      /* Check size of table (pressure)... */
```

```
      if (tbl->np[id][ig] < 30)
        eps = 0;

      /* Check transmittance... */
      else if (tau_path[id][ig] < 1e-9)
        eps = 1;

      /* Interpolate... */
      else {

        /* Determine pressure and temperature indices... */
        const int ipr =
          locate_irr(tbl->p[id][ig], tbl->np[id][ig], los->cgp[ip][ig]);
        const int it0 = locate_reg(tbl->t[id][ig][ipr], tbl->nt[id][ig][ipr],
                                   los->cgt[ip][ig]);
        const int it1 =
          locate_reg(tbl->t[id][ig][ipr + 1], tbl->nt[id][ig][ipr + 1],
                     los->cgt[ip][ig]);

        /* Check size of table (temperature and column density)... */
        if (tbl->nt[id][ig][ipr] < 2 || tbl->nt[id][ig][ipr + 1] < 2
            || tbl->nu[id][ig][ipr][it0] < 2
            || tbl->nu[id][ig][ipr][it0 + 1] < 2
            || tbl->nu[id][ig][ipr + 1][it1] < 2
            || tbl->nu[id][ig][ipr + 1][it1 + 1] < 2)
          eps = 0;

        else {

          /* Get emissivities of extended path... */
          double eps00
            = intpol_tbl_eps(tbl, ig, id, ipr, it0, los->cgu[ip][ig]);
          double eps01 =
            intpol_tbl_eps(tbl, ig, id, ipr, it0 + 1, los->cgu[ip][ig]);
          double eps10 =
            intpol_tbl_eps(tbl, ig, id, ipr + 1, it1, los->cgu[ip][ig]);
          double eps11 =
            intpol_tbl_eps(tbl, ig, id, ipr + 1, it1 + 1, los->cgu[ip][ig]);

          /* Interpolate with respect to temperature... */
          eps00 = LIN(tbl->t[id][ig][ipr][it0], eps00,
                      tbl->t[id][ig][ipr][it0 + 1], eps01, los->cgt[ip][ig]);
          eps11 = LIN(tbl->t[id][ig][ipr + 1][it1], eps10,
                      tbl->t[id][ig][ipr + 1][it1 + 1],
                      eps11, los->cgt[ip][ig]);

          /* Interpolate with respect to pressure... */
          eps00 = LOGX(tbl->p[id][ig][ipr], eps00,
                       tbl->p[id][ig][ipr + 1], eps11, los->cgp[ip][ig]);

          /* Check emssivity range... */
          eps00 = MAX(MIN(eps00, 1), 0);

          /* Determine segment emissivity... */
          eps = 1 - (1 - eps00) / tau_path[id][ig];
        }
      }

      /* Get transmittance of extended path... */
      tau_path[id][ig] *= (1 - eps);

      /* Get segment transmittance... */
      tau_seg[id] *= (1 - eps);
    }
  }
}
```

Here is the call graph for this function:



intpol_tbl_ega()

```
void intpol_tbl_ega (
            const ctl_t * ctl,
            const tbl_t * tbl,
            const los_t * los,
            const int ip,
            double tau_path[ND][NG],
            double tau_seg[ND] )
```

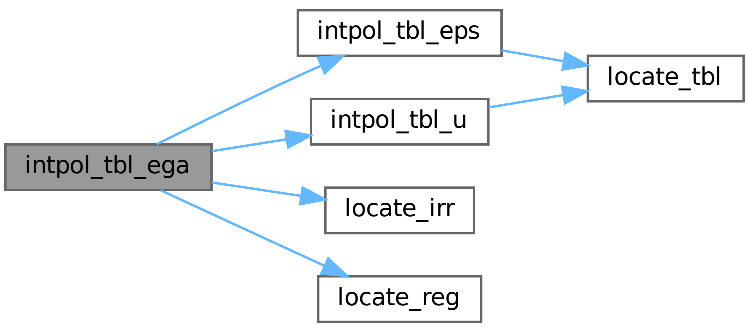
Get transmittance from look-up tables (EGA method).

Definition at line 3799 of file jurassic.c.

```
                          {

  double eps, u;

  /* Loop over channels... */
  for (int id = 0; id < ctl->nd; id++) {

    /* Initialize... */
    tau_seg[id] = 1;

    /* Loop over emitters.... */
    for (int ig = 0; ig < ctl->ng; ig++) {

      /* Check size of table (pressure)... */
      if (tbl->np[id][ig] < 30)
        eps = 0;

      /* Check transmittance... */
      else if (tau_path[id][ig] < 1e-9)
        eps = 1;

      /* Interpolate... */
      else {

        /* Determine pressure and temperature indices... */
        const int ipr =
          locate_irr(tbl->p[id][ig], tbl->np[id][ig], los->p[ip]);
        const int it0 =
          locate_reg(tbl->t[id][ig][ipr], tbl->nt[id][ig][ipr], los->t[ip]);
        const int it1 =
          locate_reg(tbl->t[id][ig][ipr + 1], tbl->nt[id][ig][ipr + 1],
                     los->t[ip]);

        /* Check size of table (temperature and column density)... */
        if (tbl->nt[id][ig][ipr] < 2 || tbl->nt[id][ig][ipr + 1] < 2
            || tbl->nu[id][ig][ipr][it0] < 2
            || tbl->nu[id][ig][ipr][it0 + 1] < 2
            || tbl->nu[id][ig][ipr + 1][it1] < 2
            || tbl->nu[id][ig][ipr + 1][it1 + 1] < 2)
```

```
          eps = 0;

        else {

          /* Get emissivities of extended path... */
          u = intpol_tbl_u(tbl, ig, id, ipr, it0, 1 - tau_path[id][ig]);
          double eps00
            = intpol_tbl_eps(tbl, ig, id, ipr, it0, u + los->u[ip][ig]);

          u = intpol_tbl_u(tbl, ig, id, ipr, it0 + 1, 1 - tau_path[id][ig]);
          double eps01 =
            intpol_tbl_eps(tbl, ig, id, ipr, it0 + 1, u + los->u[ip][ig]);

          u = intpol_tbl_u(tbl, ig, id, ipr + 1, it1, 1 - tau_path[id][ig]);
          double eps10 =
            intpol_tbl_eps(tbl, ig, id, ipr + 1, it1, u + los->u[ip][ig]);

          u =
            intpol_tbl_u(tbl, ig, id, ipr + 1, it1 + 1, 1 - tau_path[id][ig]);
          double eps11 =
            intpol_tbl_eps(tbl, ig, id, ipr + 1, it1 + 1, u + los->u[ip][ig]);

          /* Interpolate with respect to temperature... */
          eps00 = LIN(tbl->t[id][ig][ipr][it0], eps00,
                      tbl->t[id][ig][ipr][it0 + 1], eps01, los->t[ip]);
          eps11 = LIN(tbl->t[id][ig][ipr + 1][it1], eps10,
                      tbl->t[id][ig][ipr + 1][it1 + 1], eps11, los->t[ip]);

          /* Interpolate with respect to pressure... */
          eps00 = LIN(tbl->p[id][ig][ipr], eps00,
                      tbl->p[id][ig][ipr + 1], eps11, los->p[ip]);

          /* Check emssivity range... */
          eps00 = MAX(MIN(eps00, 1), 0);

          /* Determine segment emissivity... */
          eps = 1 - (1 - eps00) / tau_path[id][ig];
        }
      }

      /* Get transmittance of extended path... */
      tau_path[id][ig] *= (1 - eps);

      /* Get segment transmittance... */
      tau_seg[id] *= (1 - eps);
    }
  }
}
```

Here is the call graph for this function:



intpol_tbl_eps()

```
double intpol_tbl_eps (
            const tbl_t * tbl,
```

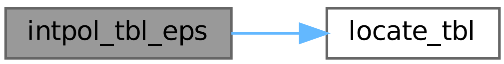
```
            const int ig,
            const int id,
            const int ip,
            const int it,
            const double u )
```

Interpolate emissivity from look-up tables.

Definition at line 3895 of file jurassic.c.

```
                    {

  /* Lower boundary... */
  if (u < tbl->u[id][ig][ip][it][0])
    return LIN(0, 0, tbl->u[id][ig][ip][it][0], tbl->eps[id][ig][ip][it][0],
               u);

  /* Upper boundary... */
  else if (u > tbl->u[id][ig][ip][it][tbl->nu[id][ig][ip][it] - 1]) {
    const double a =
      log(1 - tbl->eps[id][ig][ip][it][tbl->nu[id][ig][ip][it] - 1])
      / tbl->u[id][ig][ip][it][tbl->nu[id][ig][ip][it] - 1];
    return 1 - exp(a * u);
  }

  /* Interpolation... */
  else {

    /* Get index... */
    const int idx =
      locate_tbl(tbl->u[id][ig][ip][it], tbl->nu[id][ig][ip][it], u);

    /* Interpolate... */
    return
      LIN(tbl->u[id][ig][ip][it][idx], tbl->eps[id][ig][ip][it][idx],
          tbl->u[id][ig][ip][it][idx + 1], tbl->eps[id][ig][ip][it][idx + 1],
          u);
  }
}
```

Here is the call graph for this function:



intpol_tbl_u()

```
double intpol_tbl_u (
            const tbl_t * tbl,
            const int ig,
            const int id,
            const int ip,
            const int it,
            const double eps )
```

Interpolate column density from look-up tables.

Definition at line 3933 of file jurassic.c.

```
                      {

```

```
  /* Lower boundary... */
  if (eps < tbl->eps[id][ig][ip][it][0])
    return LIN(0, 0, tbl->eps[id][ig][ip][it][0], tbl->u[id][ig][ip][it][0],
               eps);

  /* Upper boundary... */
  else if (eps > tbl->eps[id][ig][ip][it][tbl->nu[id][ig][ip][it] - 1]) {
    const double a =
      log(1 - tbl->eps[id][ig][ip][it][tbl->nu[id][ig][ip][it] - 1])
      / tbl->u[id][ig][ip][it][tbl->nu[id][ig][ip][it] - 1];
    return log(1 - eps) / a;
  }

  /* Interpolation... */
  else {

    /* Get index... */
    const int idx =
      locate_tbl(tbl->eps[id][ig][ip][it], tbl->nu[id][ig][ip][it], eps);

    /* Interpolate... */
    return
      LIN(tbl->eps[id][ig][ip][it][idx], tbl->u[id][ig][ip][it][idx],
          tbl->eps[id][ig][ip][it][idx + 1], tbl->u[id][ig][ip][it][idx + 1],
          eps);
  }
}
```

Here is the call graph for this function:

jsec2time()

```
void jsec2time (
              const double jsec,
              int * year,
              int * mon,
              int * day,
              int * hour,
              int * min,
              int * sec,
              double * remain )
```

Convert seconds to date.

Definition at line 3971 of file jurassic.c.

```
                        {

  struct tm t0, *t1;

  t0.tm_year = 100;
  t0.tm_mon = 0;
  t0.tm_mday = 1;
  t0.tm_hour = 0;
  t0.tm_min = 0;
  t0.tm_sec = 0;

  time_t jsec0 = (time_t) jsec + timegm(&t0);
  t1 = gmtime(&jsec0);

```

```
  *year = t1->tm_year + 1900;
  *mon = t1->tm_mon + 1;
  *day = t1->tm_mday;
  *hour = t1->tm_hour;
  *min = t1->tm_min;
  *sec = t1->tm_sec;
  *remain = jsec - floor(jsec);
}
```

kernel()

```
void kernel (
            ctl_t * ctl,
            atm_t * atm,
            obs_t * obs,
            gsl_matrix * k )
```

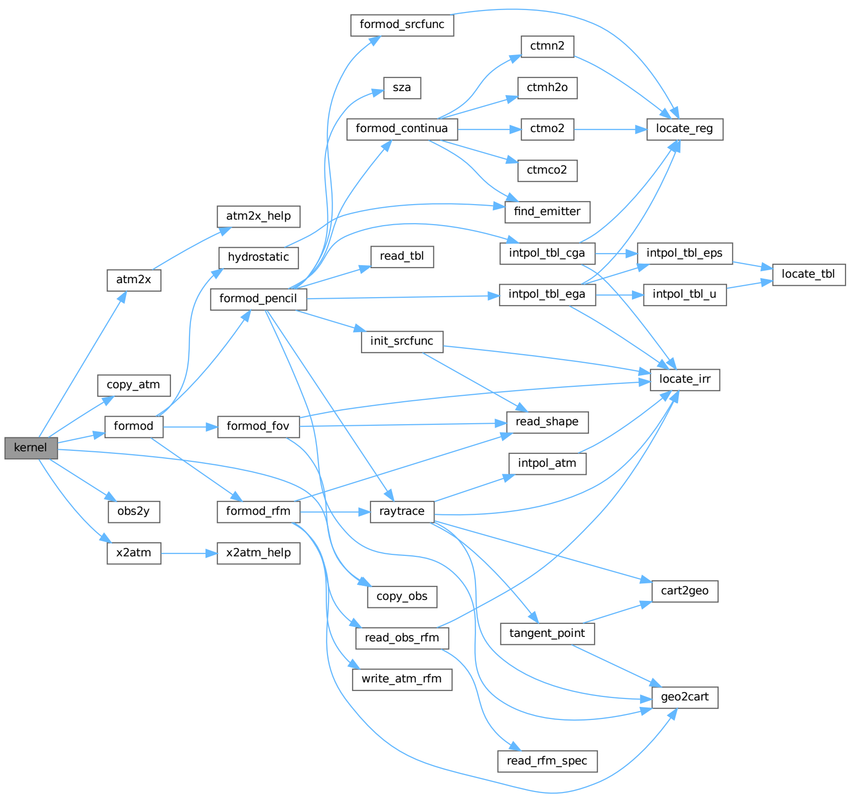
Compute Jacobians.

Definition at line 4004 of file jurassic.c.

```
                   {

  atm_t *atm1;
  obs_t *obs1;

  int *iqa;

  /* Get sizes... */
  const size_t m = k->size1;
  const size_t n = k->size2;

  /* Allocate... */
  gsl_vector *x0 = gsl_vector_alloc(n);
  gsl_vector *yy0 = gsl_vector_alloc(m);
  ALLOC(iqa, int,
        N);

  /* Compute radiance for undisturbed atmospheric data... */
  formod(ctl, atm, obs);

  /* Compose vectors... */
  atm2x(ctl, atm, x0, iqa, NULL);
  obs2y(ctl, obs, yy0, NULL, NULL);

  /* Initialize kernel matrix... */
  gsl_matrix_set_zero(k);

  /* Loop over state vector elements... */
#pragma omp parallel for default(none) shared(ctl,atm,obs,k,x0,yy0,n,m,iqa) private(atm1, obs1)
  for (size_t j = 0; j < n; j++) {

    /* Allocate... */
    gsl_vector *x1 = gsl_vector_alloc(n);
    gsl_vector *yy1 = gsl_vector_alloc(m);
    ALLOC(atm1, atm_t, 1);
    ALLOC(obs1, obs_t, 1);

    /* Set perturbation size... */
    double h;
    if (iqa[j] == IDXP)
      h = MAX(fabs(0.01 * gsl_vector_get(x0, j)), 1e-7);
    else if (iqa[j] == IDXT)
      h = 1.0;
    else if (iqa[j] >= IDXQ(0) && iqa[j] < IDXQ(ctl->ng))
      h = MAX(fabs(0.01 * gsl_vector_get(x0, j)), 1e-15);
    else if (iqa[j] >= IDXK(0) && iqa[j] < IDXK(ctl->nw))
      h = 1e-4;
    else if (iqa[j] == IDXCLZ || iqa[j] == IDXCLDZ)
      h = 1.0;
    else if (iqa[j] >= IDXCLK(0) && iqa[j] < IDXCLK(ctl->ncl))
      h = 1e-4;
    else if (iqa[j] == IDXSFZ)
      h = 0.1;
    else if (iqa[j] == IDXSFP)
      h = 10.0;
    else if (iqa[j] == IDXSFT)
      h = 1.0;
    else if (iqa[j] >= IDXSFEPS(0) && iqa[j] < IDXSFEPS(ctl->nsf))
      h = 1e-2;
```

```
    else
      ERRMSG("Cannot set perturbation size!");

    /* Disturb state vector element... */
    gsl_vector_memcpy(x1, x0);
    gsl_vector_set(x1, j, gsl_vector_get(x1, j) + h);
    copy_atm(ctl, atm1, atm, 0);
    copy_obs(ctl, obs1, obs, 0);
    x2atm(ctl, x1, atm1);

    /* Compute radiance for disturbed atmospheric data... */
    formod(ctl, atm1, obs1);

    /* Compose measurement vector for disturbed radiance data... */
    obs2y(ctl, obs1, yy1, NULL, NULL);

    /* Compute derivatives... */
    for (size_t i = 0; i < m; i++)
      gsl_matrix_set(k, i, j,
                     (gsl_vector_get(yy1, i) - gsl_vector_get(yy0, i)) / h);

    /* Free... */
    gsl_vector_free(x1);
    gsl_vector_free(yy1);
    free(atm1);
    free(obs1);
  }

  /* Free... */
  gsl_vector_free(x0);
  gsl_vector_free(yy0);
  free(iqa);
}
```

Here is the call graph for this function:



locate_irr()

```
int locate_irr (
```

```
const double * xx,
const int n,
const double x )
```

Find array index for irregular grid.

Definition at line 4103 of file jurassic.c.

```
                      {

  int ilo = 0;
  int ihi = n - 1;
  int i = (ihi + ilo) » 1;

  if (xx[i] < xx[i + 1])
    while (ihi > ilo + 1) {
      i = (ihi + ilo) » 1;
      if (xx[i] > x)
        ihi = i;
      else
        ilo = i;
  } else
    while (ihi > ilo + 1) {
      i = (ihi + ilo) » 1;
      if (xx[i] <= x)
        ihi = i;
      else
        ilo = i;
    }

  return ilo;
}
```

locate_reg()

```
int locate_reg (
              const double * xx,
              const int n,
              const double x )
```

Find array index for regular grid.

Definition at line 4133 of file jurassic.c.

```
                      {

  /* Calculate index... */
  const int i = (int) ((x - xx[0]) / (xx[1] - xx[0]));

  /* Check range... */
  if (i < 0)
    return 0;
  else if (i > n - 2)
    return n - 2;
  else
    return i;
}
```

locate_tbl()

```
int locate_tbl (
              const float * xx,
              const int n,
              const double x )
```

Find array index in float array.

Definition at line 4152 of file jurassic.c.

```
                      {

```

```
  int ilo = 0;
  int ihi = n - 1;
  int i = (ihi + ilo) » 1;

  while (ihi > ilo + 1) {
    i = (ihi + ilo) » 1;
    if (xx[i] > x)
      ihi = i;
    else
      ilo = i;
  }

  return ilo;
}
```

obs2y()

```
size_t obs2y (
              const ctl_t * ctl,
              const obs_t * obs,
              gsl_vector * y,
              int * ida,
              int * ira )
```

Compose measurement vector.

Definition at line 4174 of file jurassic.c.

```
              {

  size_t m = 0;

  /* Determine measurement vector... */
  for (int ir = 0; ir < obs->nr; ir++)
    for (int id = 0; id < ctl->nd; id++)
      if (isfinite(obs->rad[id][ir])) {
        if (y != NULL)
          gsl_vector_set(y, m, obs->rad[id][ir]);
        if (ida != NULL)
          ida[m] = id;
        if (ira != NULL)
          ira[m] = ir;
        m++;
      }

  return m;
}
```

raytrace()

```
void raytrace (
              const ctl_t * ctl,
              const atm_t * atm,
              obs_t * obs,
              los_t * los,
              const int ir )
```

Do ray-tracing to determine LOS.

Definition at line 4201 of file jurassic.c.

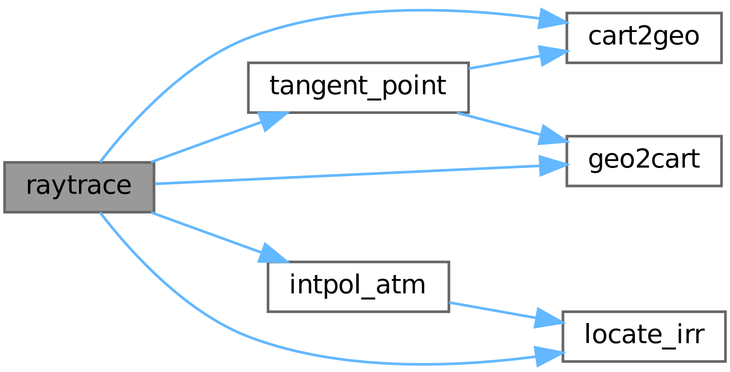
```
                  {

  const double h = 0.02, zrefrac = 60;

  double ex0[3], ex1[3], k[NW], lat, lon, n, ng[3], norm, p, q[NG], t,
    x[3], xh[3], xobs[3], xvp[3], z = 1e99, zmax, zmin;

  int stop = 0;

  /* Initialize... */
```

```
  los->np = 0;
  los->sft = -999;
  obs->tpz[ir] = obs->vpz[ir];
  obs->tplon[ir] = obs->vplon[ir];
  obs->tplat[ir] = obs->vplat[ir];

  /* Get altitude range of atmospheric data... */
  gsl_stats_minmax(&zmin, &zmax, atm->z, 1, (size_t) atm->np);
  if (ctl->nsf > 0) {
    zmin = MAX(atm->sfz, zmin);
    if (atm->sfp > 0) {
      const int ip = locate_irr(atm->p, atm->np, atm->sfp);
      const double zip =
        LIN(log(atm->p[ip]), atm->z[ip], log(atm->p[ip + 1]), atm->z[ip + 1],
            log(atm->sfp));
      zmin = MAX(zip, zmin);
    }
  }

  /* Check observer altitude... */
  if (obs->obsz[ir] < zmin)
    ERRMSG("Observer below surface!");

  /* Check view point altitude... */
  if (obs->vpz[ir] > zmax)
    return;

  /* Determine Cartesian coordinates for observer and view point... */
  geo2cart(obs->obsz[ir], obs->obslon[ir], obs->obslat[ir], xobs);
  geo2cart(obs->vpz[ir], obs->vplon[ir], obs->vplat[ir], xvp);

  /* Determine initial tangent vector... */
  for (int i = 0; i < 3; i++)
    ex0[i] = xvp[i] - xobs[i];
  norm = NORM(ex0);
  for (int i = 0; i < 3; i++)
    ex0[i] /= norm;

  /* Observer within atmosphere... */
  for (int i = 0; i < 3; i++)
    x[i] = xobs[i];

  /* Observer above atmosphere (search entry point)... */
  if (obs->obsz[ir] > zmax) {
    double dmax = norm, dmin = 0;
    while (fabs(dmin - dmax) > 0.001) {
      const double d = (dmax + dmin) / 2;
      for (int i = 0; i < 3; i++)
        x[i] = xobs[i] + d * ex0[i];
      cart2geo(x, &z, &lon, &lat);
      if (z <= zmax && z > zmax - 0.001)
        break;
      if (z < zmax - 0.0005)
        dmax = d;
      else
        dmin = d;
    }
  }

  /* Ray-tracing... */
  while (1) {

    /* Set step length... */
    double ds = ctl->rayds;
    if (ctl->raydz > 0) {
      norm = NORM(x);
      for (int i = 0; i < 3; i++)
        xh[i] = x[i] / norm;
      const double cosa = fabs(DOTP(ex0, xh));
      if (cosa != 0)
        ds = MIN(ctl->rayds, ctl->raydz / cosa);
    }

    /* Determine geolocation... */
    cart2geo(x, &z, &lon, &lat);

    /* Check if LOS hits the ground or has left atmosphere... */
    if (z < zmin || z > zmax) {
      stop = (z < zmin ? 2 : 1);
      const double frac =
        ((z <
          zmin ? zmin : zmax) - los->z[los->np - 1]) / (z - los->z[los->np -
                                                                    1]);
      geo2cart(los->z[los->np - 1], los->lon[los->np - 1],
               los->lat[los->np - 1], xh);
      for (int i = 0; i < 3; i++)
        x[i] = xh[i] + frac * (x[i] - xh[i]);
```

```
      cart2geo(x, &z, &lon, &lat);
      los->ds[los->np - 1] = ds * frac;
      ds = 0;
    }

    /* Interpolate atmospheric data... */
    intpol_atm(ctl, atm, z, &p, &t, q, k);

    /* Save data... */
    los->lon[los->np] = lon;
    los->lat[los->np] = lat;
    los->z[los->np] = z;
    los->p[los->np] = p;
    los->t[los->np] = t;
    for (int ig = 0; ig < ctl->ng; ig++)
      los->q[los->np][ig] = q[ig];
    for (int id = 0; id < ctl->nd; id++)
      los->k[los->np][id] = k[ctl->window[id]];
    los->ds[los->np] = ds;

    /* Add cloud extinction... */
    if (ctl->ncl > 0 && atm->cldz > 0) {
      const double aux = exp(-0.5 * POW2((z - atm->clz) / atm->cldz));
      for (int id = 0; id < ctl->nd; id++) {
        const int icl = locate_irr(ctl->clnu, ctl->ncl, ctl->nu[id]);
        los->k[los->np][id]
          += aux * LIN(ctl->clnu[icl], atm->clk[icl],
                       ctl->clnu[icl + 1], atm->clk[icl + 1], ctl->nu[id]);
      }
    }

    /* Increment and check number of LOS points... */
    if ((++los->np) > NLOS)
      ERRMSG("Too many LOS points!");

    /* Check stop flag... */
    if (stop) {

      /* Set surface temperature... */
      if (ctl->nsf > 0 && atm->sft > 0)
        t = atm->sft;
      los->sft = (stop == 2 ? t : -999);

      /* Set surface emissivity... */
      for (int id = 0; id < ctl->nd; id++) {
        los->sfeps[id] = 1.0;
        if (ctl->nsf > 0) {
          const int isf = locate_irr(ctl->sfnu, ctl->nsf, ctl->nu[id]);
          los->sfeps[id] = LIN(ctl->sfnu[isf], atm->sfeps[isf],
                               ctl->sfnu[isf + 1], atm->sfeps[isf + 1],
                               ctl->nu[id]);
        }
      }

      /* Leave raytracer... */
      break;
    }

    /* Determine refractivity... */
    if (ctl->refrac && z <= zrefrac)
      n = 1 + REFRAC(p, t);
    else
      n = 1;

    /* Construct new tangent vector (first term)... */
    for (int i = 0; i < 3; i++)
      ex1[i] = ex0[i] * n;

    /* Compute gradient of refractivity... */
    if (ctl->refrac && z <= zrefrac) {
      for (int i = 0; i < 3; i++)
        xh[i] = x[i] + 0.5 * ds * ex0[i];
      cart2geo(xh, &z, &lon, &lat);
      intpol_atm(ctl, atm, z, &p, &t, q, k);
      n = REFRAC(p, t);
      for (int i = 0; i < 3; i++) {
        xh[i] += h;
        cart2geo(xh, &z, &lon, &lat);
        intpol_atm(ctl, atm, z, &p, &t, q, k);
        ng[i] = (REFRAC(p, t) - n) / h;
        xh[i] -= h;
      }
    } else
      for (int i = 0; i < 3; i++)
        ng[i] = 0;

    /* Construct new tangent vector (second term)... */
```

```
    for (int i = 0; i < 3; i++)
      ex1[i] += ds * ng[i];

    /* Normalize new tangent vector... */
    norm = NORM(ex1);
    for (int i = 0; i < 3; i++)
      ex1[i] /= norm;

    /* Determine next point of LOS... */
    for (int i = 0; i < 3; i++)
      x[i] += 0.5 * ds * (ex0[i] + ex1[i]);

    /* Copy tangent vector... */
    for (int i = 0; i < 3; i++)
      ex0[i] = ex1[i];
  }

  /* Get tangent point (to be done before changing segment lengths!)... */
  tangent_point(los, &obs->tpz[ir], &obs->tplon[ir], &obs->tplat[ir]);

  /* Change segment lengths according to trapezoid rule... */
  for (int ip = los->np - 1; ip >= 1; ip--)
    los->ds[ip] = 0.5 * (los->ds[ip - 1] + los->ds[ip]);
  los->ds[0] *= 0.5;

  /* Compute column density... */
  for (int ip = 0; ip < los->np; ip++)
    for (int ig = 0; ig < ctl->ng; ig++)
      los->u[ip][ig] = 10 * los->q[ip][ig] * los->p[ip]
        / (KB * los->t[ip]) * los->ds[ip];

  /* Compute Curtis-Godson means... */
  for (int ig = 0; ig < ctl->ng; ig++) {
    los->cgu[0][ig] = los->u[0][ig];
    los->cgp[0][ig] = los->u[0][ig] * los->p[0];
    los->cgt[0][ig] = los->u[0][ig] * los->t[0];
  }
  for (int ip = 1; ip < los->np; ip++)
    for (int ig = 0; ig < ctl->ng; ig++) {
      los->cgu[ip][ig] = los->cgu[ip - 1][ig] + los->u[ip][ig];
      los->cgp[ip][ig] = los->cgp[ip - 1][ig] + los->u[ip][ig] * los->p[ip];
      los->cgt[ip][ig] = los->cgt[ip - 1][ig] + los->u[ip][ig] * los->t[ip];
    }
  for (int ip = 0; ip < los->np; ip++)
    for (int ig = 0; ig < ctl->ng; ig++) {
      los->cgp[ip][ig] /= los->cgu[ip][ig];
      los->cgt[ip][ig] /= los->cgu[ip][ig];
    }
}
```

Here is the call graph for this function:



read_atm()

```
void read_atm (
        const char * dirname,
        const char * filename,
        const ctl_t * ctl,
        atm_t * atm )
```

Read atmospheric data.

Definition at line 4442 of file jurassic.c.

```
                  {

  FILE *in;

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

  /* Init... */
  atm->np = 0;

  /* Set filename... */
  if (dirname != NULL)
    sprintf(file, "%s/%s", dirname, filename);
  else
    sprintf(file, "%s", filename);

  /* Write info... */
  LOG(1, "Read atmospheric data: %s", file);

  /* Open file... */
  if (!(in = fopen(file, "r")))
    ERRMSG("Cannot open file!");

  /* Read line... */
  while (fgets(line, LEN, in)) {

    /* Read data... */
    TOK(line, tok, "%lg", atm->time[atm->np]);
    TOK(NULL, tok, "%lg", atm->z[atm->np]);
    TOK(NULL, tok, "%lg", atm->lon[atm->np]);
    TOK(NULL, tok, "%lg", atm->lat[atm->np]);
    TOK(NULL, tok, "%lg", atm->p[atm->np]);
    TOK(NULL, tok, "%lg", atm->t[atm->np]);
    for (int ig = 0; ig < ctl->ng; ig++)
      TOK(NULL, tok, "%lg", atm->q[ig][atm->np]);
    for (int iw = 0; iw < ctl->nw; iw++)
      TOK(NULL, tok, "%lg", atm->k[iw][atm->np]);
    if (ctl->ncl > 0 && atm->np == 0) {
      TOK(NULL, tok, "%lg", atm->clz);
      TOK(NULL, tok, "%lg", atm->cldz);
      for (int icl = 0; icl < ctl->ncl; icl++)
        TOK(NULL, tok, "%lg", atm->clk[icl]);
    }
    if (ctl->nsf > 0 && atm->np == 0) {
      TOK(NULL, tok, "%lg", atm->sfz);
      TOK(NULL, tok, "%lg", atm->sfp);
      TOK(NULL, tok, "%lg", atm->sft);
      for (int isf = 0; isf < ctl->nsf; isf++)
        TOK(NULL, tok, "%lg", atm->sfeps[isf]);
    }

    /* Increment data point counter... */
    if ((++atm->np) > NP)
      ERRMSG("Too many data points!");
  }

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

  /* Check number of points... */
  if (atm->np < 1)
    ERRMSG("Could not read any data!");

  /* Write info... */
  double mini, maxi;
  LOG(2, "Number of data points: %d", atm->np);
  gsl_stats_minmax(&mini, &maxi, atm->time, 1, (size_t) atm->np);
  LOG(2, "Time range: %.2f ... %.2f s", mini, maxi);
  gsl_stats_minmax(&mini, &maxi, atm->z, 1, (size_t) atm->np);
  LOG(2, "Altitude range: %g ... %g km", mini, maxi);
  gsl_stats_minmax(&mini, &maxi, atm->lon, 1, (size_t) atm->np);
```

```
  LOG(2, "Longitude range: %g ... %g deg", mini, maxi);
  gsl_stats_minmax(&mini, &maxi, atm->lat, 1, (size_t) atm->np);
  LOG(2, "Latitude range: %g ... %g deg", mini, maxi);
  gsl_stats_minmax(&mini, &maxi, atm->p, 1, (size_t) atm->np);
  LOG(2, "Pressure range: %g ... %g hPa", maxi, mini);
  gsl_stats_minmax(&mini, &maxi, atm->t, 1, (size_t) atm->np);
  LOG(2, "Temperature range: %g ... %g K", mini, maxi);
  for (int ig = 0; ig < ctl->ng; ig++) {
    gsl_stats_minmax(&mini, &maxi, atm->q[ig], 1, (size_t) atm->np);
    LOG(2, "Emitter %s range: %g ... %g ppv", ctl->emitter[ig], mini, maxi);
  }
  for (int iw = 0; iw < ctl->nw; iw++) {
    gsl_stats_minmax(&mini, &maxi, atm->k[iw], 1, (size_t) atm->np);
    LOG(2, "Extinction range (window %d): %g ... %g km^-1", iw, mini, maxi);
  }
  if (ctl->ncl > 0 && atm->np == 0) {
    LOG(2, "Cloud layer: z= %g km | dz= %g km | k= %g ... %g km^-1",
        atm->clz, atm->cldz, atm->clk[0], atm->clk[ctl->ncl - 1]);
  } else
    LOG(2, "Cloud layer: none");
  if (ctl->nsf > 0 && atm->np == 0) {
    LOG(2,
        "Surface layer: z_s= %g km | p_s= %g hPa | T_s = %g K | eps= %g ... %g",
        atm->sfz, atm->sfp, atm->sft, atm->sfeps[0],
        atm->sfeps[ctl->nsf - 1]);
  } else
    LOG(2, "Surface layer: none");
}
```

read_ctl()

```
void read_ctl (
        int argc,
        char * argv[],
        ctl_t * ctl )
```

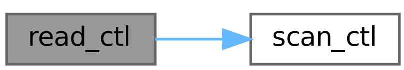
Read forward model control parameters.

Definition at line 4547 of file jurassic.c.

```
                {

  /* Write info... */
  LOG(1, "\nJuelich Rapid Spectral Simulation Code (JURASSIC)\n"
      "(executable: %s | version: %s | compiled: %s, %s)\n",
      argv[0], VERSION, __DATE__, __TIME__);

  /* Emitters... */
  ctl->ng = (int) scan_ctl(argc, argv, "NG", -1, "0", NULL);
  if (ctl->ng < 0 || ctl->ng > NG)
    ERRMSG("Set 0 <= NG <= MAX!");
  for (int ig = 0; ig < ctl->ng; ig++)
    scan_ctl(argc, argv, "EMITTER", ig, "", ctl->emitter[ig]);

  /* Radiance channels... */
  ctl->nd = (int) scan_ctl(argc, argv, "ND", -1, "0", NULL);
  if (ctl->nd < 0 || ctl->nd > ND)
    ERRMSG("Set 0 <= ND <= MAX!");
  for (int id = 0; id < ctl->nd; id++)
    ctl->nu[id] = scan_ctl(argc, argv, "NU", id, "", NULL);

  /* Spectral windows... */
  ctl->nw = (int) scan_ctl(argc, argv, "NW", -1, "1", NULL);
  if (ctl->nw < 0 || ctl->nw > NW)
    ERRMSG("Set 0 <= NW <= MAX!");
  for (int id = 0; id < ctl->nd; id++)
    ctl->window[id] = (int) scan_ctl(argc, argv, "WINDOW", id, "0", NULL);

  /* Cloud data... */
  ctl->ncl = (int) scan_ctl(argc, argv, "NCL", -1, "0", NULL);
  if (ctl->ncl < 0 || ctl->ncl > NCL)
    ERRMSG("Set 0 <= NCL <= MAX!");
  if (ctl->ncl == 1)
    ERRMSG("Set NCL > 1!");
  for (int icl = 0; icl < ctl->ncl; icl++)
    ctl->clnu[icl] = scan_ctl(argc, argv, "CLNU", icl, "", NULL);

  /* Surface data... */
  ctl->nsf = (int) scan_ctl(argc, argv, "NSF", -1, "0", NULL);
  if (ctl->nsf < 0 || ctl->nsf > NSF)
```

```
    ERRMSG("Set 0 <= NSF <= MAX!");
  if (ctl->nsf == 1)
    ERRMSG("Set NSF > 1!");
  for (int isf = 0; isf < ctl->nsf; isf++)
    ctl->sfnu[isf] = scan_ctl(argc, argv, "SFNU", isf, "", NULL);
  ctl->sftype = (int) scan_ctl(argc, argv, "SFTYPE", -1, "2", NULL);
  if (ctl->sftype < 0 || ctl->sftype > 3)
    ERRMSG("Set 0 <= SFTYPE <= 3!");
  ctl->sfsza = scan_ctl(argc, argv, "SFSZA", -1, "-999", NULL);

  /* Emissivity look-up tables... */
  scan_ctl(argc, argv, "TBLBASE", -1, "-", ctl->tblbase);
  ctl->tblfmt = (int) scan_ctl(argc, argv, "TBLFMT", -1, "1", NULL);

  /* Hydrostatic equilibrium... */
  ctl->hydz = scan_ctl(argc, argv, "HYDZ", -1, "-999", NULL);

  /* Continua... */
  ctl->ctm_co2 = (int) scan_ctl(argc, argv, "CTM_CO2", -1, "1", NULL);
  ctl->ctm_h2o = (int) scan_ctl(argc, argv, "CTM_H2O", -1, "1", NULL);
  ctl->ctm_n2 = (int) scan_ctl(argc, argv, "CTM_N2", -1, "1", NULL);
  ctl->ctm_o2 = (int) scan_ctl(argc, argv, "CTM_O2", -1, "1", NULL);

  /* Ray-tracing... */
  ctl->refrac = (int) scan_ctl(argc, argv, "REFRAC", -1, "1", NULL);
  ctl->rayds = scan_ctl(argc, argv, "RAYDS", -1, "10", NULL);
  ctl->raydz = scan_ctl(argc, argv, "RAYDZ", -1, "0.1", NULL);

  /* Field of view... */
  scan_ctl(argc, argv, "FOV", -1, "-", ctl->fov);

  /* Retrieval interface... */
  ctl->retp_zmin = scan_ctl(argc, argv, "RETP_ZMIN", -1, "-999", NULL);
  ctl->retp_zmax = scan_ctl(argc, argv, "RETP_ZMAX", -1, "-999", NULL);
  ctl->rett_zmin = scan_ctl(argc, argv, "RETT_ZMIN", -1, "-999", NULL);
  ctl->rett_zmax = scan_ctl(argc, argv, "RETT_ZMAX", -1, "-999", NULL);
  for (int ig = 0; ig < ctl->ng; ig++) {
    ctl->retq_zmin[ig] = scan_ctl(argc, argv, "RETQ_ZMIN", ig, "-999", NULL);
    ctl->retq_zmax[ig] = scan_ctl(argc, argv, "RETQ_ZMAX", ig, "-999", NULL);
  }
  for (int iw = 0; iw < ctl->nw; iw++) {
    ctl->retk_zmin[iw] = scan_ctl(argc, argv, "RETK_ZMIN", iw, "-999", NULL);
    ctl->retk_zmax[iw] = scan_ctl(argc, argv, "RETK_ZMAX", iw, "-999", NULL);
  }
  ctl->ret_clz = (int) scan_ctl(argc, argv, "RET_CLZ", -1, "0", NULL);
  ctl->ret_cldz = (int) scan_ctl(argc, argv, "RET_CLDZ", -1, "0", NULL);
  ctl->ret_clk = (int) scan_ctl(argc, argv, "RET_CLK", -1, "0", NULL);
  ctl->ret_sfz = (int) scan_ctl(argc, argv, "RET_SFZ", -1, "0", NULL);
  ctl->ret_sfp = (int) scan_ctl(argc, argv, "RET_SFP", -1, "0", NULL);
  ctl->ret_sft = (int) scan_ctl(argc, argv, "RET_SFT", -1, "0", NULL);
  ctl->ret_sfeps = (int) scan_ctl(argc, argv, "RET_SFEPS", -1, "0", NULL);

  /* Output flags... */
  ctl->write_bbt = (int) scan_ctl(argc, argv, "WRITE_BBT", -1, "0", NULL);
  ctl->write_matrix =
    (int) scan_ctl(argc, argv, "WRITE_MATRIX", -1, "0", NULL);

  /* External forward models... */
  ctl->formod = (int) scan_ctl(argc, argv, "FORMOD", -1, "1", NULL);
  scan_ctl(argc, argv, "RFMBIN", -1, "-", ctl->rfmbin);
  scan_ctl(argc, argv, "RFMHIT", -1, "-", ctl->rfmhit);
  for (int ig = 0; ig < ctl->ng; ig++)
    scan_ctl(argc, argv, "RFMXSC", ig, "-", ctl->rfmxsc[ig]);
}
```

Here is the call graph for this function:



read_matrix()

```
void read_matrix (
            const char * dirname,
            const char * filename,
            gsl_matrix * matrix )
```

Read matrix.

Definition at line 4657 of file jurassic.c.

```
                            {

  FILE *in;

  char dum[LEN], file[LEN], line[LEN];

  double value;

  int i, j;

  /* Set filename... */
  if (dirname != NULL)
    sprintf(file, "%s/%s", dirname, filename);
  else
    sprintf(file, "%s", filename);

  /* Write info... */
  LOG(1, "Read matrix: %s", file);

  /* Open file... */
  if (!(in = fopen(file, "r")))
    ERRMSG("Cannot open file!");

  /* Read data... */
  gsl_matrix_set_zero(matrix);
  while (fgets(line, LEN, in))
    if (sscanf(line, "%d %s %s %s %s %s %d %s %s %s %s %s %lg",
               &i, dum, dum, dum, dum, dum,
               &j, dum, dum, dum, dum, dum, &value) == 13)
      gsl_matrix_set(matrix, (size_t) i, (size_t) j, value);

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

read_obs()

```
void read_obs (
            const char * dirname,
            const char * filename,
            const ctl_t * ctl,
            obs_t * obs )
```

Read observation data.

Definition at line 4697 of file jurassic.c.

```
                    {

  FILE *in;

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

  /* Init... */
  obs->nr = 0;

  /* Set filename... */
  if (dirname != NULL)
    sprintf(file, "%s/%s", dirname, filename);
  else
    sprintf(file, "%s", filename);

  /* Write info... */
  LOG(1, "Read observation data: %s", file);
```

```
    /* Open file... */
    if (!(in = fopen(file, "r")))
      ERRMSG("Cannot open file!");

    /* Read line... */
    while (fgets(line, LEN, in)) {

      /* Read data... */
      TOK(line, tok, "%lg", obs->time[obs->nr]);
      TOK(NULL, tok, "%lg", obs->obsz[obs->nr]);
      TOK(NULL, tok, "%lg", obs->obslon[obs->nr]);
      TOK(NULL, tok, "%lg", obs->obslat[obs->nr]);
      TOK(NULL, tok, "%lg", obs->vpz[obs->nr]);
      TOK(NULL, tok, "%lg", obs->vplon[obs->nr]);
      TOK(NULL, tok, "%lg", obs->vplat[obs->nr]);
      TOK(NULL, tok, "%lg", obs->tpz[obs->nr]);
      TOK(NULL, tok, "%lg", obs->tplon[obs->nr]);
      TOK(NULL, tok, "%lg", obs->tplat[obs->nr]);
      for (int id = 0; id < ctl->nd; id++)
        TOK(NULL, tok, "%lg", obs->rad[id][obs->nr]);
      for (int id = 0; id < ctl->nd; id++)
        TOK(NULL, tok, "%lg", obs->tau[id][obs->nr]);

      /* Increment counter... */
      if ((++obs->nr) > NR)
        ERRMSG("Too many rays!");
    }

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

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

    /* Write info... */
    double mini, maxi;
    LOG(2, "Number of ray paths: %d", obs->nr);
    gsl_stats_minmax(&mini, &maxi, obs->time, 1, (size_t) obs->nr);
    LOG(2, "Time range: %.2f ... %.2f s", mini, maxi);
    gsl_stats_minmax(&mini, &maxi, obs->obsz, 1, (size_t) obs->nr);
    LOG(2, "Observer altitude range: %g ... %g km", mini, maxi);
    gsl_stats_minmax(&mini, &maxi, obs->obslon, 1, (size_t) obs->nr);
    LOG(2, "Observer longitude range: %g ... %g deg", mini, maxi);
    gsl_stats_minmax(&mini, &maxi, obs->obslat, 1, (size_t) obs->nr);
    LOG(2, "Observer latitude range: %g ... %g deg", mini, maxi);
    gsl_stats_minmax(&mini, &maxi, obs->vpz, 1, (size_t) obs->nr);
    LOG(2, "View point altitude range: %g ... %g km", mini, maxi);
    gsl_stats_minmax(&mini, &maxi, obs->vplon, 1, (size_t) obs->nr);
    LOG(2, "View point longitude range: %g ... %g deg", mini, maxi);
    gsl_stats_minmax(&mini, &maxi, obs->vplat, 1, (size_t) obs->nr);
    LOG(2, "View point latitude range: %g ... %g deg", mini, maxi);
    gsl_stats_minmax(&mini, &maxi, obs->tpz, 1, (size_t) obs->nr);
    LOG(2, "Tangent point altitude range: %g ... %g km", mini, maxi);
    gsl_stats_minmax(&mini, &maxi, obs->tplon, 1, (size_t) obs->nr);
    LOG(2, "Tangent point longitude range: %g ... %g deg", mini, maxi);
    gsl_stats_minmax(&mini, &maxi, obs->tplat, 1, (size_t) obs->nr);
    LOG(2, "Tangent point latitude range: %g ... %g deg", mini, maxi);
    for (int id = 0; id < ctl->nd; id++) {
      gsl_stats_minmax(&mini, &maxi, obs->rad[id], 1, (size_t) obs->nr);
      if (ctl->write_bbt) {
        LOG(2, "Brightness temperature (%.4f cm^-1) range: %g ... %g K",
            ctl->nu[id], mini, maxi);
      } else {
        LOG(2, "Radiance (%.4f cm^-1) range: %g ... %g W/(m^2 sr cm^-1)",
            ctl->nu[id], mini, maxi);
      }
    }
    for (int id = 0; id < ctl->nd; id++) {
      gsl_stats_minmax(&mini, &maxi, obs->tau[id], 1, (size_t) obs->nr);
      if (ctl->write_bbt) {
        LOG(2, "Transmittance (%.4f cm^-1) range: %g ... %g",
            ctl->nu[id], mini, maxi);
      }
    }
  }
```

read_obs_rfm()

```
double read_obs_rfm (
            const char * basename,
```

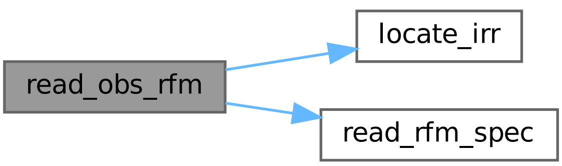
```
const double z,
double * nu,
double * f,
int n )
```

Read observation data in RFM format.

Definition at line 4798 of file jurassic.c.

```
        {

  FILE *in;

  char filename[LEN];

  double filt, fsum = 0, nu2[NSHAPE], *nurfm, *rad, radsum = 0;

  int npts;

  /* Allocate... */
  ALLOC(nurfm, double,
        RFMNPTS);
  ALLOC(rad, double,
        RFMNPTS);

  /* Search RFM spectrum... */
  sprintf(filename, "%s_%05d.asc", basename, (int) (z * 1000));
  if (!(in = fopen(filename, "r"))) {
    sprintf(filename, "%s_%05d.asc", basename, (int) (z * 1000) + 1);
    if (!(in = fopen(filename, "r")))
      ERRMSG("Cannot find RFM data file!");
  }
  fclose(in);

  /* Read RFM spectrum... */
  read_rfm_spec(filename, nurfm, rad, &npts);

  /* Set wavenumbers... */
  nu2[0] = nu[0];
  nu2[n - 1] = nu[n - 1];
  for (int i = 1; i < n - 1; i++)
    nu2[i] = LIN(0.0, nu2[0], n - 1.0, nu2[n - 1], i);

  /* Convolute... */
  for (int ipts = 0; ipts < npts; ipts++)
    if (nurfm[ipts] >= nu2[0] && nurfm[ipts] <= nu2[n - 1]) {
      const int idx = locate_irr(nu2, n, nurfm[ipts]);
      filt = LIN(nu2[idx], f[idx], nu2[idx + 1], f[idx + 1], nurfm[ipts]);
      fsum += filt;
      radsum += filt * rad[ipts];
    }

  /* Free... */
  free(nurfm);
  free(rad);

  /* Return radiance... */
  return radsum / fsum;
}
```

Here is the call graph for this function:



read_rfm_spec()

```
void read_rfm_spec (
            const char * filename,
            double * nu,
            double * rad,
            int * npts )
```

Read RFM spectrum.

Definition at line 4856 of file jurassic.c.

```
{

  FILE *in;

  char line[RFMLINE], *tok;

  double dnu, nu0, nu1;

  int ipts = 0;

  /* Write info... */
  LOG(1, "Read RFM data: %s", filename);

  /* Open file... */
  if (!(in = fopen(filename, "r")))
    ERRMSG("Cannot open file!");

  /* Read header...... */
  for (int i = 0; i < 4; i++)
    if (fgets(line, RFMLINE, in) == NULL)
      ERRMSG("Error while reading file header!");
  sscanf(line, "%d %lg %lg %lg", npts, &nu0, &dnu, &nu1);
  if (*npts > RFMNPTS)
    ERRMSG("Too many spectral grid points!");

  /* Read radiance data... */
  while (fgets(line, RFMLINE, in) && ipts < *npts) {
    if ((tok = strtok(line, " \t\n")) != NULL)
      if (sscanf(tok, "%lg", &rad[ipts]) == 1)
        ipts++;
    while ((tok = strtok(NULL, " \t\n")) != NULL)
      if (sscanf(tok, "%lg", &rad[ipts]) == 1)
        ipts++;
  }
  if (ipts != *npts)
    ERRMSG("Error while reading RFM data!");

  /* Compute wavenumbers... */
  for (ipts = 0; ipts < *npts; ipts++)
    nu[ipts] = LIN(0.0, nu0, (double) (*npts - 1), nu1, (double) ipts);

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

read_shape()

```
void read_shape (
            const char * filename,
            double * x,
            double * y,
            int * n )
```

Read shape function.

Definition at line 4907 of file jurassic.c.

```
{

  FILE *in;

  char line[LEN];

```

```
  /* Write info... */
  LOG(1, "Read shape function: %s", filename);

  /* Open file... */
  if (!(in = fopen(filename, "r")))
    ERRMSG("Cannot open file!");

  /* Read data... */
  *n = 0;
  while (fgets(line, LEN, in))
    if (sscanf(line, "%lg %lg", &x[*n], &y[*n]) == 2)
      if ((++(*n)) > NSHAPE)
        ERRMSG("Too many data points!");

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

  /* Check number of data points... */
  if (*n < 2)
    ERRMSG("Could not read any data!");

  /* Write info... */
  double mini, maxi;
  LOG(2, "Number of data points: %d", *n);
  gsl_stats_minmax(&mini, &maxi, x, 1, (size_t) *n);
  LOG(2, "Range of x values: %.4f ... %.4f", mini, maxi);
  gsl_stats_minmax(&mini, &maxi, y, 1, (size_t) *n);
  LOG(2, "Range of y values: %g ... %g", mini, maxi);
}
```

read_tbl()

```
void read_tbl (
            const ctl_t * ctl,
            tbl_t * tbl )
```

Read look-up table data.

Definition at line 4949 of file jurassic.c.

```
                {

  FILE *in;

  char filename[2 * LEN], line[LEN];

  double eps, press, temp, u;

  /* Loop over trace gases and channels... */
  for (int id = 0; id < ctl->nd; id++)
    for (int ig = 0; ig < ctl->ng; ig++) {

      /* Initialize... */
      tbl->np[id][ig] = -1;
      double eps_old = -999;
      double press_old = -999;
      double temp_old = -999;
      double u_old = -999;
      int nrange = 0;

      /* Set filename... */
      sprintf(filename, "%s_%.4f_%s.%s", ctl->tblbase,
              ctl->nu[id], ctl->emitter[ig],
              ctl->tblfmt == 1 ? "tab" : "bin");

      /* Write info... */
      LOG(1, "Read emissivity table: %s", filename);

      /* Try to open file... */
      if (!(in = fopen(filename, "r"))) {
        WARN("Missing emissivity table: %s", filename);
        continue;
      }

      /* Read ASCII tables... */
      if (ctl->tblfmt == 1) {

        /* Read data... */
        while (fgets(line, LEN, in)) {

          /* Parse line... */
```

```
          if (sscanf(line, "%lg %lg %lg %lg", &press, &temp, &u, &eps) != 4)
            continue;

          /* Check ranges... */
          if (u < UMIN || u > UMAX || eps < EPSMIN || eps > EPSMAX) {
            nrange++;
            continue;
          }

          /* Determine pressure index... */
          if (press != press_old) {
            press_old = press;
            if ((++tbl->np[id][ig]) >= TBLNP)
              ERRMSG("Too many pressure levels!");
            tbl->nt[id][ig][tbl->np[id][ig]] = -1;
          }

          /* Determine temperature index... */
          if (temp != temp_old) {
            temp_old = temp;
            if ((++tbl->nt[id][ig][tbl->np[id][ig]]) >= TBLNT)
              ERRMSG("Too many temperatures!");
            tbl->nu[id][ig][tbl->np[id][ig]]
              [tbl->nt[id][ig][tbl->np[id][ig]]] = -1;
          }

          /* Determine column density index... */
          if ((eps > eps_old && u > u_old) || tbl->nu[id][ig][tbl->np[id][ig]]
              [tbl->nt[id][ig][tbl->np[id][ig]]] < 0) {
            eps_old = eps;
            u_old = u;
            if ((++tbl->nu[id][ig][tbl->np[id][ig]]
                 [tbl->nt[id][ig][tbl->np[id][ig]]]) >= TBLNU)
              ERRMSG("Too many column densities!");
          }

          /* Store data... */
          tbl->p[id][ig][tbl->np[id][ig]] = press;
          tbl->t[id][ig][tbl->np[id][ig]][tbl->nt[id][ig][tbl->np[id][ig]]]
            = temp;
          tbl->u[id][ig][tbl->np[id][ig]][tbl->nt[id][ig][tbl->np[id][ig]]]
            [tbl->nu[id][ig][tbl->np[id][ig]]
             [tbl->nt[id][ig][tbl->np[id][ig]]]] = (float) u;
          tbl->eps[id][ig][tbl->np[id][ig]][tbl->nt[id][ig][tbl->np[id][ig]]]
            [tbl->nu[id][ig][tbl->np[id][ig]]
             [tbl->nt[id][ig][tbl->np[id][ig]]]] = (float) eps;
        }

        /* Increment counters... */
        tbl->np[id][ig]++;
        for (int ip = 0; ip < tbl->np[id][ig]; ip++) {
          tbl->nt[id][ig][ip]++;
          for (int it = 0; it < tbl->nt[id][ig][ip]; it++)
            tbl->nu[id][ig][ip][it]++;
        }
      }

      /* Read binary data... */
      else if (ctl->tblfmt == 2) {

        /* Read data... */
        FREAD(&tbl->np[id][ig], int,
              1,
              in);
        if (tbl->np[id][ig] > TBLNP)
          ERRMSG("Too many pressure levels!");
        FREAD(tbl->p[id][ig], double,
                (size_t) tbl->np[id][ig],
              in);
        for (int ip = 0; ip < tbl->np[id][ig]; ip++) {
          FREAD(&tbl->nt[id][ig][ip], int,
                1,
                in);
          if (tbl->nt[id][ig][ip] > TBLNT)
            ERRMSG("Too many temperatures!");
          FREAD(tbl->t[id][ig][ip], double,
                  (size_t) tbl->nt[id][ig][ip],
                in);
          for (int it = 0; it < tbl->nt[id][ig][ip]; it++) {
            FREAD(&tbl->nu[id][ig][ip][it], int,
                  1,
                  in);
            if (tbl->nu[id][ig][ip][it] > TBLNU)
              ERRMSG("Too many column densities!");
            FREAD(tbl->u[id][ig][ip][it], float,
                    (size_t) tbl->nu[id][ig][ip][it],
                  in);
```

```
            FREAD(tbl->eps[id][ig][ip][it], float,
                  (size_t) tbl->nu[id][ig][ip][it],
                  in);
          }
        }
      }

      /* Error message... */
      else
        ERRMSG("Unknown look-up table format!");

      /* Check ranges... */
      if (nrange > 0)
        WARN("Column density or emissivity out of range (%d data points)!",
             nrange);

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

      /* Write info... */
      for (int ip = 0; ip < tbl->np[id][ig]; ip++)
        LOG(2,
            "p[%2d]= %.5e hPa | T[0:%2d]= %.2f ... %.2f K | u[0:%3d]= %.5e ... %.5e molec/cm^2 | eps[0:%3d]= %.5e ... %.5e",
            ip, tbl->p[id][ig][ip], tbl->nt[id][ig][ip] - 1,
            tbl->t[id][ig][ip][0],
            tbl->t[id][ig][ip][tbl->nt[id][ig][ip] - 1],
            tbl->nu[id][ig][ip][0] - 1, tbl->u[id][ig][ip][0][0],
            tbl->u[id][ig][ip][0][tbl->nu[id][ig][ip][0] - 1],
            tbl->nu[id][ig][ip][0] - 1, tbl->eps[id][ig][ip][0][0],
            tbl->eps[id][ig][ip][0][tbl->nu[id][ig][ip][0] - 1]);
    }
}
```

scan_ctl()

```
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 5114 of file jurassic.c.

```
                 {

  FILE *in = NULL;

  char dummy[LEN], fullname1[LEN], fullname2[LEN], line[LEN],
    rvarname[LEN], rval[LEN];

  int contain = 0;

  /* Open file... */
  if (argv[1][0] != '-')
    if (!(in = fopen(argv[1], "r")))
      ERRMSG("Cannot open file!");

  /* Set full variable name... */
  if (arridx >= 0) {
    sprintf(fullname1, "%s[%d]", varname, arridx);
    sprintf(fullname2, "%s[*]", varname);
  } else {
    sprintf(fullname1, "%s", varname);
    sprintf(fullname2, "%s", varname);
  }

  /* Read data... */
  if (in != NULL)
    while (fgets(line, LEN, in))
      if (sscanf(line, "%s %s %s", rvarname, dummy, rval) == 3)
        if (strcasecmp(rvarname, fullname1) == 0 ||
            strcasecmp(rvarname, fullname2) == 0) {
          contain = 1;
          break;
```

```
        }
  for (int i = 1; i < argc - 1; i++)
    if (strcasecmp(argv[i], fullname1) == 0 ||
        strcasecmp(argv[i], fullname2) == 0) {
      sprintf(rval, "%s", argv[i + 1]);
      contain = 1;
      break;
    }

  /* Close file... */
  if (in != NULL)
    fclose(in);

  /* Check for missing variables... */
  if (!contain) {
    if (strlen(defvalue) > 0)
      sprintf(rval, "%s", defvalue);
    else
      ERRMSG("Missing variable %s!\n", fullname1);
  }

  /* Write info... */
  LOG(1, "%s = %s", fullname1, rval);

  /* Return values... */
  if (value != NULL)
    sprintf(value, "%s", rval);
  return atof(rval);
}
```

sza()

```
double sza (
        double sec,
        double lon,
        double lat )
```

Calculate solar zenith angle.

Definition at line 5183 of file jurassic.c.

```
                      {

  /* Number of days and fraction with respect to 2000-01-01T12:00Z... */
  const double D = sec / 86400 - 0.5;

  /* Geocentric apparent ecliptic longitude [rad]... */
  const double g = DEG2RAD(357.529 + 0.98560028 * D);
  const double q = 280.459 + 0.98564736 * D;
  const double L = DEG2RAD(q + 1.915 * sin(g) + 0.020 * sin(2 * g));

  /* Mean obliquity of the ecliptic [rad]... */
  const double e = DEG2RAD(23.439 - 0.00000036 * D);

  /* Declination [rad]... */
  const double dec = asin(sin(e) * sin(L));

  /* Right ascension [rad]... */
  const double ra = atan2(cos(e) * sin(L), cos(L));

  /* Greenwich Mean Sidereal Time [h]... */
  const double GMST = 18.697374558 + 24.06570982441908 * D;

  /* Local Sidereal Time [h]... */
  const double LST = GMST + lon / 15;

  /* Hour angle [rad]... */
  const double h = LST / 12 * M_PI - ra;

  /* Convert latitude... */
  const double latr = DEG2RAD(lat);

  /* Return solar zenith angle [deg]... */
  return RAD2DEG(acos(sin(latr) * sin(dec) + cos(latr) * cos(dec) * cos(h)));
}
```

tangent_point()

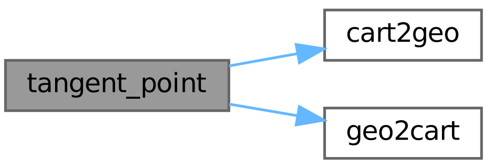
```
void tangent_point (
            const los_t * los,
            double * tpz,
            double * tplon,
            double * tplat )
```

Find tangent point of a given LOS.

Definition at line 5223 of file jurassic.c.

```
                      {

  double dummy, v[3], v0[3], v2[3];

  /* Find minimum altitude... */
  const size_t ip = gsl_stats_min_index(los->z, 1, (size_t) los->np);

  /* Nadir or zenith... */
  if (ip <= 0 || ip >= (size_t) los->np - 1) {
    *tpz = los->z[los->np - 1];
    *tplon = los->lon[los->np - 1];
    *tplat = los->lat[los->np - 1];
  }

  /* Limb... */
  else {

    /* Determine interpolating polynomial y=a*x^2+b*x+c... */
    const double yy0 = los->z[ip - 1];
    const double yy1 = los->z[ip];
    const double yy2 = los->z[ip + 1];
    const double x1 = sqrt(POW2(los->ds[ip]) - POW2(yy1 - yy0));
    const double x2 = x1 + sqrt(POW2(los->ds[ip + 1]) - POW2(yy2 - yy1));
    const double a = 1 / (x1 - x2) * (-(yy0 - yy1) / x1 + (yy0 - yy2) / x2);
    const double b = -(yy0 - yy1) / x1 - a * x1;
    const double c = yy0;

    /* Get tangent point location... */
    const double x = -b / (2 * a);
    *tpz = a * x * x + b * x + c;
    geo2cart(los->z[ip - 1], los->lon[ip - 1], los->lat[ip - 1], v0);
    geo2cart(los->z[ip + 1], los->lon[ip + 1], los->lat[ip + 1], v2);
    for (int i = 0; i < 3; i++)
      v[i] = LIN(0.0, v0[i], x2, v2[i], x);
    cart2geo(v, &dummy, tplon, tplat);
  }
}
```

Here is the call graph for this function:



time2jsec()

```
void time2jsec (
            const int year,
```

```
            const int mon,
            const int day,
            const int hour,
            const int min,
            const int sec,
            const double remain,
            double * jsec )
```

Convert date to seconds.

Definition at line 5267 of file jurassic.c.

```
                    {

  struct tm t0, t1;

  t0.tm_year = 100;
  t0.tm_mon = 0;
  t0.tm_mday = 1;
  t0.tm_hour = 0;
  t0.tm_min = 0;
  t0.tm_sec = 0;

  t1.tm_year = year - 1900;
  t1.tm_mon = mon - 1;
  t1.tm_mday = day;
  t1.tm_hour = hour;
  t1.tm_min = min;
  t1.tm_sec = sec;

  *jsec = (double) timegm(&t1) - (double) timegm(&t0) + remain;
}
```

timer()

```
void timer (
            const char * name,
            const char * file,
            const char * func,
            int line,
            int mode )
```

Measure wall-clock time.

Definition at line 5298 of file jurassic.c.

```
                {

  static double w0[10];

  static int l0[10], nt;

  /* Start new timer... */
  if (mode == 1) {
    w0[nt] = omp_get_wtime();
    l0[nt] = line;
    if ((++nt) >= 10)
      ERRMSG("Too many timers!");
  }

  /* Write elapsed time... */
  else {

    /* Check timer index... */
    if (nt - 1 < 0)
      ERRMSG("Coding error!");

    /* Write elapsed time... */
    LOG(1, "Timer '%s' (%s, %s, l%d-%d): %.3f sec",
        name, file, func, l0[nt - 1], line, omp_get_wtime() - w0[nt - 1]);
  }

  /* Stop timer... */
  if (mode == 3)
    nt--;
}
```

write_atm()

```
void write_atm (
            const char * dirname,
            const char * filename,
            const ctl_t * ctl,
            const atm_t * atm )
```

Write atmospheric data.

Definition at line 5336 of file jurassic.c.

```
                          {

  FILE *out;

  char file[LEN];

  int n = 6;

  /* Set filename... */
  if (dirname != NULL)
    sprintf(file, "%s/%s", dirname, filename);
  else
    sprintf(file, "%s", filename);

  /* Write info... */
  LOG(1, "Write atmospheric data: %s", file);

  /* Create file... */
  if (!(out = fopen(file, "w")))
    ERRMSG("Cannot create file!");

  /* Write header... */
  fprintf(out,
          "# $1 = time (seconds since 2000-01-01T00:00Z)\n"
          "# $2 = altitude [km]\n"
          "# $3 = longitude [deg]\n"
          "# $4 = latitude [deg]\n"
          "# $5 = pressure [hPa]\n" "# $6 = temperature [K]\n");
  for (int ig = 0; ig < ctl->ng; ig++)
    fprintf(out, "# $%d = %s volume mixing ratio [ppv]\n",
            ++n, ctl->emitter[ig]);
  for (int iw = 0; iw < ctl->nw; iw++)
    fprintf(out, "# $%d = extinction (window %d) [km^-1]\n", ++n, iw);
  if (ctl->ncl > 0) {
    fprintf(out, "# $%d = cloud layer height [km]\n", ++n);
    fprintf(out, "# $%d = cloud layer depth [km]\n", ++n);
    for (int icl = 0; icl < ctl->ncl; icl++)
      fprintf(out, "# $%d = cloud layer extinction (%.4f cm^-1) [km^-1]\n",
              ++n, ctl->clnu[icl]);
  }
  if (ctl->nsf > 0) {
    fprintf(out, "# $%d = surface layer height [km]\n", ++n);
    fprintf(out, "# $%d = surface layer pressure [hPa]\n", ++n);
    fprintf(out, "# $%d = surface layer temperature [K]\n", ++n);
    for (int isf = 0; isf < ctl->nsf; isf++)
      fprintf(out, "# $%d = surface layer emissivity (%.4f cm^-1)\n",
              ++n, ctl->sfnu[isf]);
  }

  /* Write data... */
  for (int ip = 0; ip < atm->np; ip++) {
    if (ip == 0 || atm->time[ip] != atm->time[ip - 1])
      fprintf(out, "\n");
    fprintf(out, "%.2f %g %g %g %g %g", atm->time[ip], atm->z[ip],
            atm->lon[ip], atm->lat[ip], atm->p[ip], atm->t[ip]);
    for (int ig = 0; ig < ctl->ng; ig++)
      fprintf(out, " %g", atm->q[ig][ip]);
    for (int iw = 0; iw < ctl->nw; iw++)
      fprintf(out, " %g", atm->k[iw][ip]);
    if (ctl->ncl > 0) {
      fprintf(out, " %g %g", atm->clz, atm->cldz);
      for (int icl = 0; icl < ctl->ncl; icl++)
        fprintf(out, " %g", atm->clk[icl]);
    }
    if (ctl->nsf > 0) {
      fprintf(out, " %g %g %g", atm->sfz, atm->sfp, atm->sft);
      for (int isf = 0; isf < ctl->nsf; isf++)
        fprintf(out, " %g", atm->sfeps[isf]);
    }
    fprintf(out, "\n");
```

```
  }

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

  /* Write info... */
  double mini, maxi;
  LOG(2, "Number of data points: %d", atm->np);
  gsl_stats_minmax(&mini, &maxi, atm->time, 1, (size_t) atm->np);
  LOG(2, "Time range: %.2f ... %.2f s", mini, maxi);
  gsl_stats_minmax(&mini, &maxi, atm->z, 1, (size_t) atm->np);
  LOG(2, "Altitude range: %g ... %g km", mini, maxi);
  gsl_stats_minmax(&mini, &maxi, atm->lon, 1, (size_t) atm->np);
  LOG(2, "Longitude range: %g ... %g deg", mini, maxi);
  gsl_stats_minmax(&mini, &maxi, atm->lat, 1, (size_t) atm->np);
  LOG(2, "Latitude range: %g ... %g deg", mini, maxi);
  gsl_stats_minmax(&mini, &maxi, atm->p, 1, (size_t) atm->np);
  LOG(2, "Pressure range: %g ... %g hPa", maxi, mini);
  gsl_stats_minmax(&mini, &maxi, atm->t, 1, (size_t) atm->np);
  LOG(2, "Temperature range: %g ... %g K", mini, maxi);
  for (int ig = 0; ig < ctl->ng; ig++) {
    gsl_stats_minmax(&mini, &maxi, atm->q[ig], 1, (size_t) atm->np);
    LOG(2, "Emitter %s range: %g ... %g ppv", ctl->emitter[ig], mini, maxi);
  }
  for (int iw = 0; iw < ctl->nw; iw++) {
    gsl_stats_minmax(&mini, &maxi, atm->k[iw], 1, (size_t) atm->np);
    LOG(2, "Extinction range (window %d): %g ... %g km^-1", iw, mini, maxi);
  }
  if (ctl->ncl > 0 && atm->np == 0) {
    LOG(2, "Cloud layer: z= %g km | dz= %g km | k= %g ... %g km^-1",
        atm->clz, atm->cldz, atm->clk[0], atm->clk[ctl->ncl - 1]);
  } else
    LOG(2, "Cloud layer: none");
  if (ctl->nsf > 0 && atm->np == 0) {
    LOG(2,
        "Surface layer: z_s= %g km | p_s= %g hPa | T_s = %g K | eps= %g ... %g",
        atm->sfz, atm->sfp, atm->sft, atm->sfeps[0],
        atm->sfeps[ctl->nsf - 1]);
  } else
    LOG(2, "Surface layer: none");
}
```

write_atm_rfm()

```
void write_atm_rfm (
        const char * filename,
        const ctl_t * ctl,
        const atm_t * atm )
```

Write atmospheric data in RFM format.

Definition at line 5454 of file jurassic.c.

```
                      {

  FILE *out;

  /* Write info... */
  LOG(1, "Write RFM data: %s", filename);

  /* Create file... */
  if (!(out = fopen(filename, "w")))
    ERRMSG("Cannot create file!");

  /* Write data... */
  fprintf(out, "%d\n", atm->np);
  fprintf(out, "*HGT [km]\n");
  for (int ip = 0; ip < atm->np; ip++)
    fprintf(out, "%g\n", atm->z[ip]);
  fprintf(out, "*PRE [mb]\n");
  for (int ip = 0; ip < atm->np; ip++)
    fprintf(out, "%g\n", atm->p[ip]);
  fprintf(out, "*TEM [K]\n");
  for (int ip = 0; ip < atm->np; ip++)
    fprintf(out, "%g\n", atm->t[ip]);
  for (int ig = 0; ig < ctl->ng; ig++) {
    fprintf(out, "*%s [ppmv]\n", ctl->emitter[ig]);
    for (int ip = 0; ip < atm->np; ip++)
      fprintf(out, "%g\n", atm->q[ig][ip] * 1e6);
  }
```

```
  fprintf(out, "*END\n");

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

write_matrix()

```
void write_matrix (
            const char * dirname,
            const char * filename,
            const ctl_t * ctl,
            const gsl_matrix * matrix,
            const atm_t * atm,
            const obs_t * obs,
            const char * rowspace,
            const char * colspace,
            const char * sort )
```

Write matrix.

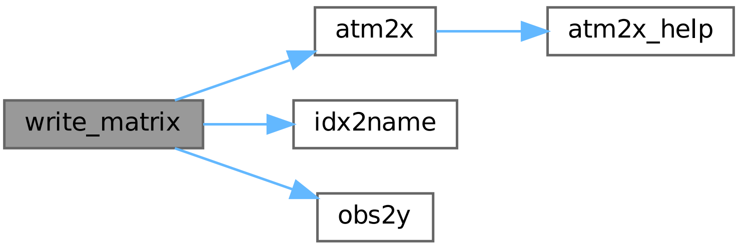
Definition at line 5492 of file jurassic.c.

```
                    {

  FILE *out;

  char file[LEN], quantity[LEN];

  int *cida, *ciqa, *cipa, *cira, *rida, *riqa, *ripa, *rira;

  size_t i, j, nc, nr;

  /* Check output flag... */
  if (!ctl->write_matrix)
    return;

  /* Allocate... */
  ALLOC(cida, int,
        M);
  ALLOC(ciqa, int,
        N);
  ALLOC(cipa, int,
        N);
  ALLOC(cira, int,
        M);
  ALLOC(rida, int,
        M);
  ALLOC(riqa, int,
        N);
  ALLOC(ripa, int,
        N);
  ALLOC(rira, int,
        M);

  /* Set filename... */
  if (dirname != NULL)
    sprintf(file, "%s/%s", dirname, filename);
  else
    sprintf(file, "%s", filename);

  /* Write info... */
  LOG(1, "Write matrix: %s", file);

  /* Create file... */
  if (!(out = fopen(file, "w")))
    ERRMSG("Cannot create file!");

  /* Write header (row space)... */
  if (rowspace[0] == 'y') {

    fprintf(out,
            "# $1 = Row: index (measurement space)\n"
            "# $2 = Row: channel wavenumber [cm^-1]\n"
            "# $3 = Row: time (seconds since 2000-01-01T00:00Z)\n"
            "# $4 = Row: view point altitude [km]\n"
            "# $5 = Row: view point longitude [deg]\n"
```

```
	    "# $6 = Row: view point latitude [deg]\n");

    /* Get number of rows... */
    nr = obs2y(ctl, obs, NULL, rida, rira);

  } else {

    fprintf(out,
	    "# $1 = Row: index (state space)\n"
	    "# $2 = Row: name of quantity\n"
	    "# $3 = Row: time (seconds since 2000-01-01T00:00Z)\n"
	    "# $4 = Row: altitude [km]\n"
	    "# $5 = Row: longitude [deg]\n" "# $6 = Row: latitude [deg]\n");

    /* Get number of rows... */
    nr = atm2x(ctl, atm, NULL, riqa, ripa);
  }

  /* Write header (column space)... */
  if (colspace[0] == 'y') {

    fprintf(out,
	    "# $7 = Col: index (measurement space)\n"
	    "# $8 = Col: channel wavenumber [cm^-1]\n"
	    "# $9 = Col: time (seconds since 2000-01-01T00:00Z)\n"
	    "# $10 = Col: view point altitude [km]\n"
	    "# $11 = Col: view point longitude [deg]\n"
	    "# $12 = Col: view point latitude [deg]\n");

    /* Get number of columns... */
    nc = obs2y(ctl, obs, NULL, cida, cira);

  } else {

    fprintf(out,
	    "# $7 = Col: index (state space)\n"
	    "# $8 = Col: name of quantity\n"
	    "# $9 = Col: time (seconds since 2000-01-01T00:00Z)\n"
	    "# $10 = Col: altitude [km]\n"
	    "# $11 = Col: longitude [deg]\n" "# $12 = Col: latitude [deg]\n");

    /* Get number of columns... */
    nc = atm2x(ctl, atm, NULL, ciqa, cipa);
  }

  /* Write header entry... */
  fprintf(out, "# $13 = Matrix element\n\n");

  /* Write matrix data... */
  i = j = 0;
  while (i < nr && j < nc) {

    /* Write info about the row... */
    if (rowspace[0] == 'y')
      fprintf(out, "%d %.4f %.2f %g %g %g",
	      (int) i, ctl->nu[rida[i]],
	      obs->time[rira[i]], obs->vpz[rira[i]],
	      obs->vplon[rira[i]], obs->vplat[rira[i]]);
    else {
      idx2name(ctl, riqa[i], quantity);
      fprintf(out, "%d %s %.2f %g %g %g", (int) i, quantity,
	      atm->time[ripa[i]], atm->z[ripa[i]],
	      atm->lon[ripa[i]], atm->lat[ripa[i]]);
    }

    /* Write info about the column... */
    if (colspace[0] == 'y')
      fprintf(out, " %d %.4f %.2f %g %g %g",
	      (int) j, ctl->nu[cida[j]],
	      obs->time[cira[j]], obs->vpz[cira[j]],
	      obs->vplon[cira[j]], obs->vplat[cira[j]]);
    else {
      idx2name(ctl, ciqa[j], quantity);
      fprintf(out, " %d %s %.2f %g %g %g", (int) j, quantity,
	      atm->time[cipa[j]], atm->z[cipa[j]],
	      atm->lon[cipa[j]], atm->lat[cipa[j]]);
    }

    /* Write matrix entry... */
    fprintf(out, " %g\n", gsl_matrix_get(matrix, i, j));

    /* Set matrix indices... */
    if (sort[0] == 'r') {
      j++;
      if (j >= nc) {
	j = 0;
	i++;
```

```
        fprintf(out, "\n");
      }
    } else {
      i++;
      if (i >= nr) {
        i = 0;
        j++;
        fprintf(out, "\n");
      }
    }
  }

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

  /* Free... */
  free(cida);
  free(ciqa);
  free(cipa);
  free(cira);
  free(rida);
  free(riqa);
  free(ripa);
  free(rira);
}
```

Here is the call graph for this function:



write_obs()

```
void write_obs (
        const char * dirname,
        const char * filename,
        const ctl_t * ctl,
        const obs_t * obs )
```

Write observation data.

Definition at line 5670 of file jurassic.c.

```
                          {

  FILE *out;

  char file[LEN];

  int n = 10;

  /* Set filename... */
  if (dirname != NULL)
    sprintf(file, "%s/%s", dirname, filename);
  else
    sprintf(file, "%s", filename);

```

```
/* Write info... */
LOG(1, "Write observation data: %s", file);

/* Create file... */
if (!(out = fopen(file, "w")))
  ERRMSG("Cannot create file!");

/* Write header... */
fprintf(out,
        "# $1 = time (seconds since 2000-01-01T00:00Z)\n"
        "# $2 = observer altitude [km]\n"
        "# $3 = observer longitude [deg]\n"
        "# $4 = observer latitude [deg]\n"
        "# $5 = view point altitude [km]\n"
        "# $6 = view point longitude [deg]\n"
        "# $7 = view point latitude [deg]\n"
        "# $8 = tangent point altitude [km]\n"
        "# $9 = tangent point longitude [deg]\n"
        "# $10 = tangent point latitude [deg]\n");
for (int id = 0; id < ctl->nd; id++)
  if (ctl->write_bbt)
    fprintf(out, "# $%d = brightness temperature (%.4f cm^-1) [K]\n",
            ++n, ctl->nu[id]);
  else
    fprintf(out, "# $%d = radiance (%.4f cm^-1) [W/(m^2 sr cm^-1)]\n",
            ++n, ctl->nu[id]);
for (int id = 0; id < ctl->nd; id++)
  fprintf(out, "# $%d = transmittance (%.4f cm^-1) [-]\n", ++n,
          ctl->nu[id]);

/* Write data... */
for (int ir = 0; ir < obs->nr; ir++) {
  if (ir == 0 || obs->time[ir] != obs->time[ir - 1])
    fprintf(out, "\n");
  fprintf(out, "%.2f %g %g %g %g %g %g %g %g %g", obs->time[ir],
          obs->obsz[ir], obs->obslon[ir], obs->obslat[ir],
          obs->vpz[ir], obs->vplon[ir], obs->vplat[ir],
          obs->tpz[ir], obs->tplon[ir], obs->tplat[ir]);
  for (int id = 0; id < ctl->nd; id++)
    fprintf(out, " %g", obs->rad[id][ir]);
  for (int id = 0; id < ctl->nd; id++)
    fprintf(out, " %g", obs->tau[id][ir]);
  fprintf(out, "\n");
}

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

/* Write info... */
double mini, maxi;
LOG(2, "Number of ray paths: %d", obs->nr);
gsl_stats_minmax(&mini, &maxi, obs->time, 1, (size_t) obs->nr);
LOG(2, "Time range: %.2f ... %.2f s", mini, maxi);
gsl_stats_minmax(&mini, &maxi, obs->obsz, 1, (size_t) obs->nr);
LOG(2, "Observer altitude range: %g ... %g km", mini, maxi);
gsl_stats_minmax(&mini, &maxi, obs->obslon, 1, (size_t) obs->nr);
LOG(2, "Observer longitude range: %g ... %g deg", mini, maxi);
gsl_stats_minmax(&mini, &maxi, obs->obslat, 1, (size_t) obs->nr);
LOG(2, "Observer latitude range: %g ... %g deg", mini, maxi);
gsl_stats_minmax(&mini, &maxi, obs->vpz, 1, (size_t) obs->nr);
LOG(2, "View point altitude range: %g ... %g km", mini, maxi);
gsl_stats_minmax(&mini, &maxi, obs->vplon, 1, (size_t) obs->nr);
LOG(2, "View point longitude range: %g ... %g deg", mini, maxi);
gsl_stats_minmax(&mini, &maxi, obs->vplat, 1, (size_t) obs->nr);
LOG(2, "View point latitude range: %g ... %g deg", mini, maxi);
gsl_stats_minmax(&mini, &maxi, obs->tpz, 1, (size_t) obs->nr);
LOG(2, "Tangent point altitude range: %g ... %g km", mini, maxi);
gsl_stats_minmax(&mini, &maxi, obs->tplon, 1, (size_t) obs->nr);
LOG(2, "Tangent point longitude range: %g ... %g deg", mini, maxi);
gsl_stats_minmax(&mini, &maxi, obs->tplat, 1, (size_t) obs->nr);
LOG(2, "Tangent point latitude range: %g ... %g deg", mini, maxi);
for (int id = 0; id < ctl->nd; id++) {
  gsl_stats_minmax(&mini, &maxi, obs->rad[id], 1, (size_t) obs->nr);
  if (ctl->write_bbt) {
    LOG(2, "Brightness temperature (%.4f cm^-1) range: %g ... %g K",
        ctl->nu[id], mini, maxi);
  } else {
    LOG(2, "Radiance (%.4f cm^-1) range: %g ... %g W/(m^2 sr cm^-1)",
        ctl->nu[id], mini, maxi);
  }
}
for (int id = 0; id < ctl->nd; id++) {
  gsl_stats_minmax(&mini, &maxi, obs->tau[id], 1, (size_t) obs->nr);
  if (ctl->write_bbt) {
    LOG(2, "Transmittance (%.4f cm^-1) range: %g ... %g",
        ctl->nu[id], mini, maxi);
  }
```

```
  }
}
```

write_shape()

```
void write_shape (
            const char * filename,
            const double * x,
            const double * y,
            const int n )
```

Write shape function.

Definition at line 5780 of file jurassic.c.

```
                     {

  FILE *out;

  /* Write info... */
  LOG(1, "Write shape function: %s", filename);

  /* Create file... */
  if (!(out = fopen(filename, "w")))
    ERRMSG("Cannot create file!");

  /* Write header... */
  fprintf(out,
          "# $1 = shape function x-value [-]\n"
          "# $2 = shape function y-value [-]\n\n");

  /* Write data... */
  for (int i = 0; i < n; i++)
    fprintf(out, "%.10g %.10g\n", x[i], y[i]);

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

write_tbl()

```
void write_tbl (
            const ctl_t * ctl,
            const tbl_t * tbl )
```

Write look-up table data.

Definition at line 5810 of file jurassic.c.

```
                          {

  FILE *out;

  char filename[2 * LEN];

  /* Loop over emitters and detectors... */
  for (int ig = 0; ig < ctl->ng; ig++)
    for (int id = 0; id < ctl->nd; id++) {

      /* Set filename... */
      sprintf(filename, "%s_%.4f_%s.%s", ctl->tblbase,
              ctl->nu[id], ctl->emitter[ig],
              ctl->tblfmt == 1 ? "tab" : "bin");

      /* Write info... */
      LOG(1, "Write emissivity table: %s", filename);

      /* Create file... */
      if (!(out = fopen(filename, "w")))
        ERRMSG("Cannot create file!");

      /* Write ASCII data... */
      if (ctl->tblfmt == 1) {
```

```

      /* Write header... */
      fprintf(out,
              "# $1 = pressure [hPa]\n"
              "# $2 = temperature [K]\n"
              "# $3 = column density [molecules/cm^2]\n"
              "# $4 = emissivity [-]\n");

      /* Save table file... */
      for (int ip = 0; ip < tbl->np[id][ig]; ip++)
        for (int it = 0; it < tbl->nt[id][ig][ip]; it++) {
          fprintf(out, "\n");
          for (int iu = 0; iu < tbl->nu[id][ig][ip][it]; iu++)
            fprintf(out, "%g %g %e %e\n",
                    tbl->p[id][ig][ip], tbl->t[id][ig][ip][it],
                    tbl->u[id][ig][ip][it][iu],
                    tbl->eps[id][ig][ip][it][iu]);
        }
    }

    /* Write binary data... */
    else if (ctl->tblfmt == 2) {
      FWRITE(&tbl->np[id][ig], int,
             1,
             out);
      FWRITE(tbl->p[id][ig], double,
               (size_t) tbl->np[id][ig],
             out);
      for (int ip = 0; ip < tbl->np[id][ig]; ip++) {
        FWRITE(&tbl->nt[id][ig][ip], int,
               1,
               out);
        FWRITE(tbl->t[id][ig][ip], double,
                 (size_t) tbl->nt[id][ig][ip],
               out);
        for (int it = 0; it < tbl->nt[id][ig][ip]; it++) {
          FWRITE(&tbl->nu[id][ig][ip][it], int,
                 1,
                 out);
          FWRITE(tbl->u[id][ig][ip][it], float,
                   (size_t) tbl->nu[id][ig][ip][it],
                 out);
          FWRITE(tbl->eps[id][ig][ip][it], float,
                   (size_t) tbl->nu[id][ig][ip][it],
                 out);
        }
      }
    }

    /* Error message... */
    else
      ERRMSG("Unknown look-up table format!");

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

x2atm()

```
void x2atm (
            const ctl_t * ctl,
            const gsl_vector * x,
            atm_t * atm )
```

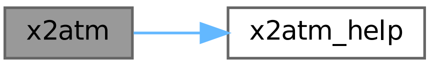
Decompose parameter vector or state vector.

Definition at line 5896 of file jurassic.c.

```
                    {

  size_t n = 0;

  /* Get pressure... */
  for (int ip = 0; ip < atm->np; ip++)
    if (atm->z[ip] >= ctl->retp_zmin && atm->z[ip] <= ctl->retp_zmax)
      x2atm_help(&atm->p[ip], x, &n);

  /* Get temperature... */
  for (int ip = 0; ip < atm->np; ip++)
```

```
    if (atm->z[ip] >= ctl->rett_zmin && atm->z[ip] <= ctl->rett_zmax)
      x2atm_help(&atm->t[ip], x, &n);

  /* Get volume mixing ratio... */
  for (int ig = 0; ig < ctl->ng; ig++)
    for (int ip = 0; ip < atm->np; ip++)
      if (atm->z[ip] >= ctl->retq_zmin[ig]
          && atm->z[ip] <= ctl->retq_zmax[ig])
        x2atm_help(&atm->q[ig][ip], x, &n);

  /* Get extinction... */
  for (int iw = 0; iw < ctl->nw; iw++)
    for (int ip = 0; ip < atm->np; ip++)
      if (atm->z[ip] >= ctl->retk_zmin[iw]
          && atm->z[ip] <= ctl->retk_zmax[iw])
        x2atm_help(&atm->k[iw][ip], x, &n);

  /* Get cloud data... */
  if (ctl->ret_clz)
    x2atm_help(&atm->clz, x, &n);
  if (ctl->ret_cldz)
    x2atm_help(&atm->cldz, x, &n);
  if (ctl->ret_clk)
    for (int icl = 0; icl < ctl->ncl; icl++)
      x2atm_help(&atm->clk[icl], x, &n);

  /* Get surface data... */
  if (ctl->ret_sfz)
    x2atm_help(&atm->sfz, x, &n);
  if (ctl->ret_sfp)
    x2atm_help(&atm->sfp, x, &n);
  if (ctl->ret_sft)
    x2atm_help(&atm->sft, x, &n);
  if (ctl->ret_sfeps)
    for (int isf = 0; isf < ctl->nsf; isf++)
      x2atm_help(&atm->sfeps[isf], x, &n);
}
```

Here is the call graph for this function:



x2atm_help()

```
void x2atm_help (
            double * value,
            const gsl_vector * x,
            size_t * n )
```

Get element from state vector.

Definition at line 5950 of file jurassic.c.

```
               {

  /* Get state vector element... */
  *value = gsl_vector_get(x, *n);
  (*n)++;
}
```

y2obs()

```
void y2obs (
            const ctl_t * ctl,
            const gsl_vector * y,
            obs_t * obs )
```

Decompose measurement vector.

Definition at line 5962 of file jurassic.c.

```
                 {

  size_t m = 0;

  /* Decompose measurement vector... */
  for (int ir = 0; ir < obs->nr; ir++)
    for (int id = 0; id < ctl->nd; id++)
      if (isfinite(obs->rad[id][ir])) {
        obs->rad[id][ir] = gsl_vector_get(y, m);
        m++;
      }
}
```

5.8 jurassic.h

Go to the documentation of this file.

```
/*
  This file is part of JURASSIC.

  JURASSIC is free software: you can redistribute it and/or modify
  it under the terms of the GNU General Public License as published by
  the Free Software Foundation, either version 3 of the License, or
  (at your option) any later version.

  JURASSIC is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
  GNU General Public License for more details.

  You should have received a copy of the GNU General Public License
  along with JURASSIC. If not, see <http://www.gnu.org/licenses/>.

  Copyright (C) 2003-2025 Forschungszentrum Juelich GmbH
*/

#ifndef JURASSIC_H
#define JURASSIC_H

#include <gsl/gsl_math.h>
#include <gsl/gsl_blas.h>
#include <gsl/gsl_linalg.h>
#include <gsl/gsl_randist.h>
#include <gsl/gsl_rng.h>
#include <gsl/gsl_statistics.h>
#include <math.h>
#include <omp.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>

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

#define ALLOC(ptr, type, n)                          \
  if((ptr=malloc((size_t)(n)*sizeof(type)))==NULL)   \
    ERRMSG("Out of memory!");

#define BRIGHT(rad, nu)                              \
  (C2 * (nu) / gsl_log1p(C1 * POW3(nu) / (rad)))

#define DEG2RAD(deg)                         \
  ((deg) * (M_PI / 180.0))

#define DIST(a, b) sqrt(DIST2(a, b))

```

```
#define DIST2(a, b)                                                     \
  ((a[0]-b[0])*(a[0]-b[0])+(a[1]-b[1])*(a[1]-b[1])+(a[2]-b[2])*(a[2]-b[2]))

#define DOTP(a, b) (a[0]*b[0]+a[1]*b[1]+a[2]*b[2])

#define FREAD(ptr, type, size, out) {                                   \
    if(fread(ptr, sizeof(type), size, out)!=size)                       \
      ERRMSG("Error while reading!");                                   \
  }

#define FWRITE(ptr, type, size, out) {                                  \
    if(fwrite(ptr, sizeof(type), size, out)!=size)                      \
      ERRMSG("Error while writing!");                                   \
  }

#define MAX(a,b)                                \
  (((a)>(b))?(a):(b))

#define MIN(a,b)                                \
  (((a)<(b))?(a):(b))

#define LIN(x0, y0, x1, y1, x)                  \
  ((y0)+((y1)-(y0))/((x1)-(x0))*((x)-(x0)))

#define LOGX(x0, y0, x1, y1, x)                         \
  (((x)/(x0)>0 && (x1)/(x0)>0)                          \
   ? ((y0)+((y1)-(y0))*log((x)/(x0))/log((x1)/(x0)))    \
   : LIN(x0, y0, x1, y1, x))

#define LOGY(x0, y0, x1, y1, x)                                 \
  (((y1)/(y0)>0)                                                \
   ? ((y0)*exp(log((y1)/(y0))/((x1)-(x0))*((x)-(x0))))          \
   : LIN(x0, y0, x1, y1, x))

#define NORM(a) sqrt(DOTP(a, a))

#define PLANCK(T, nu)                           \
  (C1 * POW3(nu) / gsl_expm1(C2 * (nu) / (T)))

#define POW2(x) ((x)*(x))

#define POW3(x) ((x)*(x)*(x))

#define RAD2DEG(rad)                            \
  ((rad) * (180.0 / M_PI))

#define REFRAC(p, T)                            \
  (7.753e-05 * (p) / (T))

#define TIMER(name, mode)                               \
  {timer(name, __FILE__, __func__, __LINE__, mode);}

#define TOK(line, tok, format, var) {                   \
    if(((tok)=strtok((line), " \t"))) {                 \
      if(sscanf(tok, format, &(var))!=1) continue;      \
    } else ERRMSG("Error while reading!");              \
  }

/* ------------------------------------------------------------
   Log messages...
   ------------------------------------------------------------ */

#ifndef LOGLEV
#define LOGLEV 2
#endif

#define LOG(level, ...) {                                               \
    if(level >= 2)                                                      \
      printf("  ");                                                     \
    if(level <= LOGLEV) {                                               \
      printf(__VA_ARGS__);                                              \
      printf("\n");                                                     \
    }                                                                   \
  }

#define WARN(...) {                                                     \
    printf("\nWarning (%s, %s, l%d): ", __FILE__, __func__, __LINE__);  \
    LOG(0, __VA_ARGS__);                                                \
  }

#define ERRMSG(...) {                                                   \
    printf("\nError (%s, %s, l%d): ", __FILE__, __func__, __LINE__);    \
    LOG(0, __VA_ARGS__);                                                \
    exit(EXIT_FAILURE);                                                 \
  }

#define PRINT(format, var)                                              \
```

```
  printf("Print (%s, %s, l%d): %s= "format"\n",                    \
         __FILE__, __func__, __LINE__, #var, var);

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

#ifndef C1
#define C1 1.19104259e-8
#endif

#ifndef C2
#define C2 1.43877506
#endif

#ifndef EPSMIN
#define EPSMIN 0
#endif

#ifndef EPSMAX
#define EPSMAX 1
#endif

#ifndef G0
#define G0 9.80665
#endif

#ifndef H0
#define H0 7.0
#endif

#ifndef KB
#define KB 1.3806504e-23
#endif

#ifndef ME
#define ME 5.976e24
#endif

#ifndef NA
#define NA 6.02214199e23
#endif

#ifndef RE
#define RE 6367.421
#endif

#ifndef RI
#define RI 8.3144598
#endif

#ifndef P0
#define P0 1013.25
#endif

#ifndef T0
#define T0 273.15
#endif

#ifndef TMIN
#define TMIN 100.
#endif

#ifndef TMAX
#define TMAX 400.
#endif

#ifndef TSUN
#define TSUN 5780.
#endif

#ifndef UMIN
#define UMIN 0
#endif

#ifndef UMAX
#define UMAX 1e30
#endif

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

#ifndef NCL
#define NCL 8
#endif

```

```
#ifndef ND
#define ND 128
#endif

#ifndef NG
#define NG 8
#endif

#ifndef NP
#define NP 256
#endif

#ifndef NR
#define NR 256
#endif

#ifndef NSF
#define NSF 8
#endif

#ifndef NW
#define NW 4
#endif

#ifndef LEN
#define LEN 10000
#endif

#ifndef M
#define M (NR*ND)
#endif

#ifndef N
#define N ((2+NG+NW)*NP+NCL+NSF+5)
#endif

#ifndef NQ
#define NQ (7+NG+NW+NCL+NSF)
#endif

#ifndef NLOS
#define NLOS 4096
#endif

#ifndef NSHAPE
#define NSHAPE 20000
#endif

#ifndef NFOV
#define NFOV 5
#endif

#ifndef TBLNP
#define TBLNP 41
#endif

#ifndef TBLNT
#define TBLNT 30
#endif

#ifndef TBLNU
#define TBLNU 320
#endif

#ifndef TBLNS
#define TBLNS 1200
#endif

#ifndef RFMNPTS
#define RFMNPTS 10000000
#endif

#ifndef RFMLINE
#define RFMLINE 100000
#endif

/* ------------------------------------------------------------
   Quantity indices...
   ------------------------------------------------------------ */

#define IDXP 0

#define IDXT 1

#define IDXQ(ig) (2+ig)

#define IDXK(iw) (2+ctl->ng+iw)
```

```
#define IDXCLZ (2+ctl->ng+ctl->nw)
#define IDXCLDZ (3+ctl->ng+ctl->nw)
#define IDXCLK(icl) (4+ctl->ng+ctl->nw+icl)
#define IDXSFZ (4+ctl->ng+ctl->nw+ctl->ncl)
#define IDXSFP (5+ctl->ng+ctl->nw+ctl->ncl)
#define IDXSFT (6+ctl->ng+ctl->nw+ctl->ncl)
#define IDXSFEPS(isf) (7+ctl->ng+ctl->nw+ctl->ncl+isf)
/* ------------------------------------------------------------
   Structs...
   ------------------------------------------------------------ */
typedef struct {
  int np;
  double time[NP];
  double z[NP];
  double lon[NP];
  double lat[NP];
  double p[NP];
  double t[NP];
  double q[NG][NP];
  double k[NW][NP];
  double clz;
  double cldz;
  double clk[NCL];
  double sfz;
  double sfp;
  double sft;
  double sfeps[NSF];
} atm_t;
typedef struct {
  int ng;
  char emitter[NG][LEN];
  int nd;
  double nu[ND];
  int nw;
  int window[ND];
  int ncl;
  double clnu[NCL];
  int nsf;
  double sfnu[NSF];
  int sftype;
  double sfsza;
  char tblbase[LEN];
  int tblfmt;
  double hydz;
```

```
  int ctm_co2;
  int ctm_h2o;
  int ctm_n2;
  int ctm_o2;
  int refrac;
  double rayds;
  double raydz;
  char fov[LEN];
  double retp_zmin;
  double retp_zmax;
  double rett_zmin;
  double rett_zmax;
  double retq_zmin[NG];
  double retq_zmax[NG];
  double retk_zmin[NW];
  double retk_zmax[NW];
  int ret_clz;
  int ret_cldz;
  int ret_clk;
  int ret_sfz;
  int ret_sfp;
  int ret_sft;
  int ret_sfeps;
  int write_bbt;
  int write_matrix;
  int formod;
  char rfmbin[LEN];
  char rfmhit[LEN];
  char rfmxsc[NG][LEN];
} ctl_t;
typedef struct {
  int np;
  double z[NLOS];
  double lon[NLOS];
  double lat[NLOS];
  double p[NLOS];
  double t[NLOS];
  double q[NLOS][NG];
  double k[NLOS][ND];
  double sft;
  double sfeps[ND];
  double ds[NLOS];
  double u[NLOS][NG];
  double cgp[NLOS][NG];
```

```
  double cgt[NLOS][NG];
  double cgu[NLOS][NG];
  double eps[NLOS][ND];
  double src[NLOS][ND];
} los_t;
typedef struct {
  int nr;
  double time[NR];
  double obsz[NR];
  double obslon[NR];
  double obslat[NR];
  double vpz[NR];
  double vplon[NR];
  double vplat[NR];
  double tpz[NR];
  double tplon[NR];
  double tplat[NR];
  double tau[ND][NR];
  double rad[ND][NR];
} obs_t;
typedef struct {
  int np[ND][NG];
  int nt[ND][NG][TBLNP];
  int nu[ND][NG][TBLNP][TBLNT];
  double p[ND][NG][TBLNP];
  double t[ND][NG][TBLNP][TBLNT];
  float u[ND][NG][TBLNP][TBLNT][TBLNU];
  float eps[ND][NG][TBLNP][TBLNT][TBLNU];
  double st[TBLNS];
  double sr[TBLNS][ND];
} tbl_t;
/* ------------------------------------------------------------
   Functions...
   ------------------------------------------------------------ */
size_t atm2x(
  const ctl_t * ctl,
  const atm_t * atm,
  gsl_vector * x,
  int *iqa,
  int *ipa);
void atm2x_help(
  const double value,
  const int value_iqa,
  const int value_ip,
  gsl_vector * x,
  int *iqa,
  int *ipa,
  size_t *n);
void cart2geo(
  const double *x,
  double *z,
  double *lon,
```

```
  double *lat);

void climatology(
  const ctl_t * ctl,
  atm_t * atm_mean);

double ctmco2(
  const double nu,
  const double p,
  const double t,
  const double u);

double ctmh2o(
  const double nu,
  const double p,
  const double t,
  const double q,
  const double u);

double ctmn2(
  const double nu,
  const double p,
  const double t);

double ctmo2(
  const double nu,
  const double p,
  const double t);

void copy_atm(
  const ctl_t * ctl,
  atm_t * atm_dest,
  const atm_t * atm_src,
  const int init);

void copy_obs(
  const ctl_t * ctl,
  obs_t * obs_dest,
  const obs_t * obs_src,
  const int init);

int find_emitter(
  const ctl_t * ctl,
  const char *emitter);

void formod(
  const ctl_t * ctl,
  atm_t * atm,
  obs_t * obs);

void formod_continua(
  const ctl_t * ctl,
  const los_t * los,
  const int ip,
  double *beta);

void formod_fov(
  const ctl_t * ctl,
  obs_t * obs);

void formod_pencil(
  const ctl_t * ctl,
  const atm_t * atm,
  obs_t * obs,
  const int ir);

void formod_rfm(
  const ctl_t * ctl,
  const atm_t * atm,
  obs_t * obs);

void formod_srcfunc(
  const ctl_t * ctl,
  const tbl_t * tbl,
  const double t,
  double *src);

void geo2cart(
  const double z,
  const double lon,
  const double lat,
  double *x);

void hydrostatic(
  const ctl_t * ctl,
  atm_t * atm);

```

```
void idx2name(
  const ctl_t * ctl,
  const int idx,
  char *quantity);

void init_srcfunc(
  const ctl_t * ctl,
  tbl_t * tbl);

void intpol_atm(
  const ctl_t * ctl,
  const atm_t * atm,
  const double z,
  double *p,
  double *t,
  double *q,
  double *k);

void intpol_tbl_cga(
  const ctl_t * ctl,
  const tbl_t * tbl,
  const los_t * los,
  const int ip,
  double tau_path[ND][NG],
  double tau_seg[ND]);

void intpol_tbl_ega(
  const ctl_t * ctl,
  const tbl_t * tbl,
  const los_t * los,
  const int ip,
  double tau_path[ND][NG],
  double tau_seg[ND]);

double intpol_tbl_eps(
  const tbl_t * tbl,
  const int ig,
  const int id,
  const int ip,
  const int it,
  const double u);

double intpol_tbl_u(
  const tbl_t * tbl,
  const int ig,
  const int id,
  const int ip,
  const int it,
  const double eps);

void jsec2time(
  const double jsec,
  int *year,
  int *mon,
  int *day,
  int *hour,
  int *min,
  int *sec,
  double *remain);

void kernel(
  ctl_t * ctl,
  atm_t * atm,
  obs_t * obs,
  gsl_matrix * k);

int locate_irr(
  const double *xx,
  const int n,
  const double x);

int locate_reg(
  const double *xx,
  const int n,
  const double x);

int locate_tbl(
  const float *xx,
  const int n,
  const double x);

size_t obs2y(
  const ctl_t * ctl,
  const obs_t * obs,
  gsl_vector * y,
  int *ida,
  int *ira);
```

```
void raytrace(
  const ctl_t * ctl,
  const atm_t * atm,
  obs_t * obs,
  los_t * los,
  const int ir);

void read_atm(
  const char *dirname,
  const char *filename,
  const ctl_t * ctl,
  atm_t * atm);

void read_ctl(
  int argc,
  char *argv[],
  ctl_t * ctl);

void read_matrix(
  const char *dirname,
  const char *filename,
  gsl_matrix * matrix);

void read_obs(
  const char *dirname,
  const char *filename,
  const ctl_t * ctl,
  obs_t * obs);

double read_obs_rfm(
  const char *basename,
  const double z,
  double *nu,
  double *f,
  int n);

void read_rfm_spec(
  const char *filename,
  double *nu,
  double *rad,
  int *npts);

void read_shape(
  const char *filename,
  double *x,
  double *y,
  int *n);

void read_tbl(
  const ctl_t * ctl,
  tbl_t * tbl);

double scan_ctl(
  int argc,
  char *argv[],
  const char *varname,
  int arridx,
  const char *defvalue,
  char *value);

double sza(
  double sec,
  double lon,
  double lat);

void tangent_point(
  const los_t * los,
  double *tpz,
  double *tplon,
  double *tplat);

void time2jsec(
  const int year,
  const int mon,
  const int day,
  const int hour,
  const int min,
  const int sec,
  const double remain,
  double *jsec);

void timer(
  const char *name,
  const char *file,
  const char *func,
  int line,
```

```
  int mode);

void write_atm(
  const char *dirname,
  const char *filename,
  const ctl_t * ctl,
  const atm_t * atm);

void write_atm_rfm(
  const char *filename,
  const ctl_t * ctl,
  const atm_t * atm);

void write_matrix(
  const char *dirname,
  const char *filename,
  const ctl_t * ctl,
  const gsl_matrix * matrix,
  const atm_t * atm,
  const obs_t * obs,
  const char *rowspace,
  const char *colspace,
  const char *sort);

void write_obs(
  const char *dirname,
  const char *filename,
  const ctl_t * ctl,
  const obs_t * obs);

void write_shape(
  const char *filename,
  const double *x,
  const double *y,
  const int n);

void write_tbl(
  const ctl_t * ctl,
  const tbl_t * tbl);

void x2atm(
  const ctl_t * ctl,
  const gsl_vector * x,
  atm_t * atm);

void x2atm_help(
  double *value,
  const gsl_vector * x,
  size_t *n);

void y2obs(
  const ctl_t * ctl,
  const gsl_vector * y,
  obs_t * obs);

#endif
```

5.9 libiasi.c File Reference

Functions

- void add_var (int ncid, const char ∗varname, const char ∗unit, const char ∗longname, int type, int dimid[ ], int ∗varid, int ndims)

  Add variable to netCDF file.
- void background_poly_help (double ∗xx, double ∗yy, int n, int dim)

  Get background based on polynomial fits.
- void background_poly (wave_t ∗wave, int dim_x, int dim_y)

  Get background based on polynomial fits.
- void iasi_read (char ∗filename, iasi_rad_t ∗iasi_rad)

  Read IASI Level-1 data and convert to radiation type.
- void noise (wave_t ∗wave, double ∗mu, double ∗sig)

  Estimate noise.
- 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 variance (wave_t *wave, double dh)

  Compute local variance.

- double wgs84 (double lat)

  Calculate Earth radius according to WGS-84 reference ellipsoid.

- void write_l1 (char *filename, iasi_l1_t *l1)

  Write IASI Level-1 data.

- void write_l2 (char *filename, iasi_l2_t *l2)

  Write IASI Level-2 data.

5.9.1 Function Documentation

add_var()

```
void add_var (
            int ncid,
            const char * varname,
            const char * unit,
            const char * longname,
            int type,
            int dimid[ ],
            int * varid,
            int ndims )
```

Add variable to netCDF file.

Definition at line 5 of file libiasi.c.

```
               {

  /* Check if variable exists... */
  if (nc_inq_varid(ncid, varname, varid) != NC_NOERR) {

    /* Define variable... */
    NC(nc_def_var(ncid, varname, type, ndims, dimid, varid));

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

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

background_poly_help()

```
void background_poly_help (
            double * xx,
            double * yy,
            int n,
            int dim )
```

Get background based on polynomial fits.

Definition at line 32 of file libiasi.c.

```
             {

  gsl_multifit_linear_workspace *work;
  gsl_matrix *cov, *X;
  gsl_vector *c, *x, *y;
```

```
  double chisq, xx2[WX > WY ? WX : WY], yy2[WX > WY ? WX : WY];

  size_t i, i2, n2 = 0;

  /* Check for nan... */
  for (i = 0; i < (size_t) n; i++)
    if (gsl_finite(yy[i])) {
      xx2[n2] = xx[i];
      yy2[n2] = yy[i];
      n2++;
    }
  if ((int) n2 < dim || n2 < 0.9 * n) {
    for (i = 0; i < (size_t) n; i++)
      yy[i] = GSL_NAN;
    return;
  }

  /* Allocate... */
  work = gsl_multifit_linear_alloc((size_t) n2, (size_t) dim);
  cov = gsl_matrix_alloc((size_t) dim, (size_t) dim);
  X = gsl_matrix_alloc((size_t) n2, (size_t) dim);
  c = gsl_vector_alloc((size_t) dim);
  x = gsl_vector_alloc((size_t) n2);
  y = gsl_vector_alloc((size_t) n2);

  /* Compute polynomial fit... */
  for (i = 0; i < (size_t) n2; i++) {
    gsl_vector_set(x, i, xx2[i]);
    gsl_vector_set(y, i, yy2[i]);
    for (i2 = 0; i2 < (size_t) dim; i2++)
      gsl_matrix_set(X, i, i2, pow(gsl_vector_get(x, i), (double) i2));
  }
  gsl_multifit_linear(X, y, c, cov, &chisq, work);
  for (i = 0; i < (size_t) n; i++)
    yy[i] = gsl_poly_eval(c->data, (int) dim, xx[i]);

  /* Free... */
  gsl_multifit_linear_free(work);
  gsl_matrix_free(cov);
  gsl_matrix_free(X);
  gsl_vector_free(c);
  gsl_vector_free(x);
  gsl_vector_free(y);
}
```

background_poly()

```
void background_poly (
        wave_t * wave,
        int dim_x,
        int dim_y )
```

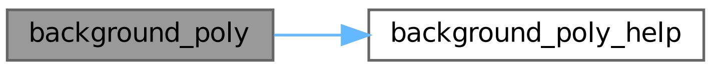
Get background based on polynomial fits.

Definition at line 89 of file libiasi.c.

```
                {

  double help[WX], x[WX], x2[WY], y[WX], y2[WY];

  int ix, iy;

  /* Copy temperatures to background... */
  for (ix = 0; ix < wave->nx; ix++)
    for (iy = 0; iy < wave->ny; iy++) {
      wave->bg[ix][iy] = wave->temp[ix][iy];
      wave->pt[ix][iy] = 0;
    }

  /* Check parameters... */
  if (dim_x <= 0 && dim_y <= 0)
    return;

  /* Compute fit in x-direction... */
  if (dim_x > 0)
    for (iy = 0; iy < wave->ny; iy++) {
      for (ix = 0; ix <= 53; ix++) {
        x[ix] = (double) ix;
        y[ix] = wave->bg[ix][iy];
```

```
      }
      background_poly_help(x, y, 54, dim_x);
      for (ix = 0; ix <= 29; ix++)
        help[ix] = y[ix];

      for (ix = 6; ix <= 59; ix++) {
        x[ix - 6] = (double) ix;
        y[ix - 6] = wave->bg[ix][iy];
      }
      background_poly_help(x, y, 54, dim_x);
      for (ix = 30; ix <= 59; ix++)
        help[ix] = y[ix - 6];

      for (ix = 0; ix < wave->nx; ix++)
        wave->bg[ix][iy] = help[ix];
    }

  /* Compute fit in y-direction... */
  if (dim_y > 0)
    for (ix = 0; ix < wave->nx; ix++) {
      for (iy = 0; iy < wave->ny; iy++) {
        x2[iy] = (int) iy;
        y2[iy] = wave->bg[ix][iy];
      }
      background_poly_help(x2, y2, wave->ny, dim_y);
      for (iy = 0; iy < wave->ny; iy++)
        wave->bg[ix][iy] = y2[iy];
    }

  /* Recompute perturbations... */
  for (ix = 0; ix < wave->nx; ix++)
    for (iy = 0; iy < wave->ny; iy++)
      wave->pt[ix][iy] = wave->temp[ix][iy] - wave->bg[ix][iy];
}
```

Here is the call graph for this function:



iasi_read()

```
void iasi_read (
            char * filename,
            iasi_rad_t * iasi_rad )
```

Read IASI Level-1 data and convert to radiation type.

Definition at line 152 of file libiasi.c.

```
                            {

  const char *product_class;

  coda_product *pf;

  coda_cursor cursor;

  iasi_raw_t *iasi_raw;

  int i, j, w, tr1, tr2, tr1_lpm, tr1_rpm, tr2_lpm, tr2_rpm,
    ichan, mdr_i, num_dims = 1;

  long dim[] = { 1, 2, 3, 4, 5, 6, 7, 8, 9 };

  short int IDefScaleSondNbScale, IDefScaleSondNsfirst[10],
```

```
    IDefScaleSondNslast[10], IDefScaleSondScaleFactor[10];

  float sc, scaling[IASI_L1_NCHAN];

  /* Initialize CODA... */
  coda_init();

  /* Allocate... */
  ALLOC(iasi_raw, iasi_raw_t, 1);

  /* Open IASI file... */
  CODA(coda_open(filename, &pf));
  CODA(coda_get_product_class(pf, &product_class));
  CODA(coda_cursor_set_product(&cursor, pf));

  /* Get scaling parameters... */
  CODA(coda_cursor_goto_record_field_by_name(&cursor, "GIADR_ScaleFactors"));

  CODA(coda_cursor_goto_record_field_by_name
       (&cursor, "IDefScaleSondNbScale"));
  CODA(coda_cursor_read_int16(&cursor, &IDefScaleSondNbScale));
  CODA(coda_cursor_goto_parent(&cursor));

  CODA(coda_cursor_goto_record_field_by_name
       (&cursor, "IDefScaleSondNsfirst"));
  CODA(coda_cursor_read_int16_array
       (&cursor, IDefScaleSondNsfirst, coda_array_ordering_c));
  CODA(coda_cursor_goto_parent(&cursor));

  CODA(coda_cursor_goto_record_field_by_name(&cursor, "IDefScaleSondNslast"));
  CODA(coda_cursor_read_int16_array
       (&cursor, IDefScaleSondNslast, coda_array_ordering_c));
  CODA(coda_cursor_goto_parent(&cursor));

  CODA(coda_cursor_goto_record_field_by_name
       (&cursor, "IDefScaleSondScaleFactor"));
  CODA(coda_cursor_read_int16_array
       (&cursor, IDefScaleSondScaleFactor, coda_array_ordering_c));

  /* Compute scaling factors... */
  for (ichan = 0; ichan < IASI_L1_NCHAN; ichan++)
    scaling[ichan] = GSL_NAN;
  for (i = 0; i < IDefScaleSondNbScale; i++) {
    sc = (float) pow(10.0, -IDefScaleSondScaleFactor[i]);
    for (ichan = IDefScaleSondNsfirst[i] - 1;
         ichan < IDefScaleSondNslast[i]; ichan++) {
      w = ichan - IASI_IDefNsfirst1b + 1;
      if (w >= 0 && w < IASI_L1_NCHAN)
        scaling[w] = sc;
    }
  }

  /* Get number of tracks in record... */
  CODA(coda_cursor_goto_root(&cursor));
  CODA(coda_cursor_goto_record_field_by_name(&cursor, "MDR"));
  CODA(coda_cursor_get_array_dim(&cursor, &num_dims, dim));
  iasi_raw->ntrack = dim[0];

  /* Read tracks one by one... */
  for (mdr_i = 0; mdr_i < iasi_raw->ntrack; mdr_i++) {

    /* Reset cursor position... */
    CODA(coda_cursor_goto_root(&cursor));

    /* Move cursor to radiation data... */
    CODA(coda_cursor_goto_record_field_by_name(&cursor, "MDR"));
    CODA(coda_cursor_goto_array_element_by_index(&cursor, mdr_i));
    CODA(coda_cursor_goto_record_field_by_name(&cursor, "MDR"));
    CODA(coda_cursor_goto_record_field_by_name(&cursor, "GS1cSpect"));
    CODA(coda_cursor_read_int16_array
         (&cursor, &iasi_raw->Radiation[mdr_i][0][0][0],
          coda_array_ordering_c));

    /* Read time... */
    CODA(coda_cursor_goto_parent(&cursor));
    CODA(coda_cursor_goto_record_field_by_name(&cursor, "OnboardUTC"));
    CODA(coda_cursor_read_double_array
         (&cursor, &iasi_raw->Time[mdr_i][0], coda_array_ordering_c));

    /* Read coordinates... */
    CODA(coda_cursor_goto_parent(&cursor));
    CODA(coda_cursor_goto_record_field_by_name(&cursor, "GGeoSondLoc"));
    CODA(coda_cursor_read_double_array
         (&cursor, &iasi_raw->Loc[mdr_i][0][0][0], coda_array_ordering_c));

    /* Read satellite altitude... */
    CODA(coda_cursor_goto_parent(&cursor));
```

```
    CODA(coda_cursor_goto_record_field_by_name(&cursor,
                                               "EARTH_SATELLITE_DISTANCE"));
    CODA(coda_cursor_read_uint32(&cursor, &iasi_raw->Sat_z[mdr_i]));

    /* Read spectral range... */
    iasi_raw->IDefSpectDWn1b[mdr_i] = IASI_IDefSpectDWn1b / 100.0;

    CODA(coda_cursor_goto_parent(&cursor));
    CODA(coda_cursor_goto_record_field_by_name(&cursor, "IDefNsfirst1b"));
    CODA(coda_cursor_read_int32(&cursor, &iasi_raw->IDefNsfirst1b[mdr_i]));
    if (iasi_raw->IDefNsfirst1b[mdr_i] != IASI_IDefNsfirst1b)
      ERRMSG("Unexpected value for IDefNsfirst1b!");

    CODA(coda_cursor_goto_parent(&cursor));
    CODA(coda_cursor_goto_record_field_by_name(&cursor, "IDefNslast1b"));
    CODA(coda_cursor_read_int32(&cursor, &iasi_raw->IDefNslast1b[mdr_i]));
    if (iasi_raw->IDefNslast1b[mdr_i] != IASI_IDefNslast1b)
      ERRMSG("Unexpected value for IDefNslast1b!");

    /* Compute wavenumber... */
    if (mdr_i == 0)
      for (i = 0; i < IASI_L1_NCHAN; i++)
        iasi_raw->Wavenumber[i] =
          iasi_raw->IDefSpectDWn1b[mdr_i] *
          (float) (iasi_raw->IDefNsfirst1b[mdr_i] + i - 1);
  }

  /* Close file... */
  CODA(coda_close(pf));

  /* Finalize CODA... */
  coda_done();

  /* Set number of tracks... */
  iasi_rad->ntrack = (int) (iasi_raw->ntrack * 2);

  /* Copy wavenumbers... */
  for (ichan = 0; ichan < IASI_L1_NCHAN; ichan++)
    iasi_rad->freq[ichan] = iasi_raw->Wavenumber[ichan];

  /* Copy footprint data... */
  for (mdr_i = 0; mdr_i < iasi_raw->ntrack; mdr_i++) {
    tr1 = mdr_i * 2;
    tr2 = mdr_i * 2 + 1;
    tr1_lpm = 3;
    tr1_rpm = 0;
    tr2_lpm = 2;
    tr2_rpm = 1;

    /* Copy time (2x2 matrix has same measurement time)...  */
    for (i = 0; i < IASI_NXTRACK; i++) {
      iasi_rad->Time[tr1][i * 2] = iasi_raw->Time[mdr_i][i];
      iasi_rad->Time[tr1][i * 2 + 1] = iasi_raw->Time[mdr_i][i];
      iasi_rad->Time[tr2][i * 2] = iasi_raw->Time[mdr_i][i];
      iasi_rad->Time[tr2][i * 2 + 1] = iasi_raw->Time[mdr_i][i];
    }

    /* Copy location... */
    for (i = 0; i < IASI_NXTRACK; i++) {
      iasi_rad->Longitude[tr1][i * 2] = iasi_raw->Loc[mdr_i][i][tr1_lpm][0];
      iasi_rad->Longitude[tr1][i * 2 + 1] =
        iasi_raw->Loc[mdr_i][i][tr1_rpm][0];
      iasi_rad->Latitude[tr1][i * 2] = iasi_raw->Loc[mdr_i][i][tr1_lpm][1];
      iasi_rad->Latitude[tr1][i * 2 + 1] =
        iasi_raw->Loc[mdr_i][i][tr1_rpm][1];

      iasi_rad->Longitude[tr2][i * 2] = iasi_raw->Loc[mdr_i][i][tr2_lpm][0];
      iasi_rad->Longitude[tr2][i * 2 + 1] =
        iasi_raw->Loc[mdr_i][i][tr2_rpm][0];
      iasi_rad->Latitude[tr2][i * 2] = iasi_raw->Loc[mdr_i][i][tr2_lpm][1];
      iasi_rad->Latitude[tr2][i * 2 + 1] =
        iasi_raw->Loc[mdr_i][i][tr2_rpm][1];
    }

    /* Copy satellite location (we only have one height value)... */
    iasi_rad->Sat_lon[tr1] = iasi_rad->Longitude[tr1][28];
    iasi_rad->Sat_lat[tr1] = iasi_rad->Latitude[tr1][28];
    iasi_rad->Sat_lon[tr2] = iasi_rad->Longitude[tr2][28];
    iasi_rad->Sat_lat[tr2] = iasi_rad->Latitude[tr2][28];
    iasi_rad->Sat_z[tr1] =
      iasi_raw->Sat_z[mdr_i] / 1000.0 - wgs84(iasi_rad->Sat_lat[tr1]);
    iasi_rad->Sat_z[tr2] =
      iasi_raw->Sat_z[mdr_i] / 1000.0 - wgs84(iasi_rad->Sat_lat[tr2]);

    /* Copy radiation data... */
    for (i = 0; i < IASI_NXTRACK; i++) {
      for (ichan = 0; ichan < IASI_L1_NCHAN; ichan++) {
```

```
        sc = scaling[ichan] * 100.0f;
        iasi_rad->Rad[tr1][i * 2][ichan] =
          iasi_raw->Radiation[mdr_i][i][tr1_lpm][ichan] * sc;
        iasi_rad->Rad[tr1][i * 2 + 1][ichan] =
          iasi_raw->Radiation[mdr_i][i][tr1_rpm][ichan] * sc;
        iasi_rad->Rad[tr2][i * 2][ichan] =
          iasi_raw->Radiation[mdr_i][i][tr2_lpm][ichan] * sc;
        iasi_rad->Rad[tr2][i * 2 + 1][ichan] =
          iasi_raw->Radiation[mdr_i][i][tr2_rpm][ichan] * sc;
      }
    }
  }

  /* Check radiance data... */
  for (i = 0; i < iasi_rad->ntrack; i++)
    for (j = 0; j < L1_NXTRACK; j++)
      if (iasi_rad->Rad[i][j][6753] > iasi_rad->Rad[i][j][6757]
          || iasi_rad->Rad[i][j][6753] < 0)
        for (ichan = 0; ichan < IASI_L1_NCHAN; ichan++)
          iasi_rad->Rad[i][j][ichan] = GSL_NAN;

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

Here is the call graph for this function:

noise()

```
void noise (
            wave_t * wave,
            double * mu,
            double * sig )
```

Estimate noise.

Definition at line 371 of file libiasi.c.

```
                 {

  int ix, ix2, iy, iy2, n = 0, okay;

  /* Init... */
  *mu = 0;
  *sig = 0;

  /* Estimate noise (Immerkaer, 1996)... */
  for (ix = 1; ix < wave->nx - 1; ix++)
    for (iy = 1; iy < wave->ny - 1; iy++) {

      /* Check data... */
      okay = 1;
      for (ix2 = ix - 1; ix2 <= ix + 1; ix2++)
        for (iy2 = iy - 1; iy2 <= iy + 1; iy2++)
          if (!gsl_finite(wave->temp[ix2][iy2]))
            okay = 0;
      if (!okay)
        continue;

      /* Get mean noise... */
      n++;
      *mu += wave->temp[ix][iy];
```

```
    *sig += gsl_pow_2(+4. / 6. * wave->temp[ix][iy]
                      - 2. / 6. * (wave->temp[ix - 1][iy]
                                   + wave->temp[ix + 1][iy]
                                   + wave->temp[ix][iy - 1]
                                   + wave->temp[ix][iy + 1])
                      + 1. / 6. * (wave->temp[ix - 1][iy - 1]
                                   + wave->temp[ix + 1][iy - 1]
                                   + wave->temp[ix - 1][iy + 1]
                                   + wave->temp[ix + 1][iy + 1]));
  }

  /* Normalize... */
  *mu /= (double) n;
  *sig = sqrt(*sig / (double) n);
}
```

pert2wave()

```
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 416 of file libiasi.c.

```
                 {

  double x0[3], x1[3];

  int itrack, ixtrack;

  /* Check ranges... */
  track0 = GSL_MIN(GSL_MAX(track0, 0), pert->ntrack - 1);
  track1 = GSL_MIN(GSL_MAX(track1, 0), pert->ntrack - 1);
  xtrack0 = GSL_MIN(GSL_MAX(xtrack0, 0), pert->nxtrack - 1);
  xtrack1 = GSL_MIN(GSL_MAX(xtrack1, 0), pert->nxtrack - 1);

  /* Set size... */
  wave->nx = xtrack1 - xtrack0 + 1;
  if (wave->nx > WX)
    ERRMSG("Too many across-track values!");
  wave->ny = track1 - track0 + 1;
  if (wave->ny > WY)
    ERRMSG("Too many along-track values!");

  /* Loop over footprints... */
  for (itrack = track0; itrack <= track1; itrack++)
    for (ixtrack = xtrack0; ixtrack <= xtrack1; ixtrack++) {

      /* Get distances... */
      if (itrack == track0) {
        wave->x[0] = 0;
        if (ixtrack > xtrack0) {
          geo2cart(0, pert->lon[itrack][ixtrack - 1],
                   pert->lat[itrack][ixtrack - 1], x0);
          geo2cart(0, pert->lon[itrack][ixtrack],
                   pert->lat[itrack][ixtrack], x1);
          wave->x[ixtrack - xtrack0] =
            wave->x[ixtrack - xtrack0 - 1] + DIST(x0, x1);
        }
      }
      if (ixtrack == xtrack0) {
        wave->y[0] = 0;
        if (itrack > track0) {
          geo2cart(0, pert->lon[itrack - 1][ixtrack],
                   pert->lat[itrack - 1][ixtrack], x0);
          geo2cart(0, pert->lon[itrack][ixtrack],
                   pert->lat[itrack][ixtrack], x1);
          wave->y[itrack - track0] =
            wave->y[itrack - track0 - 1] + DIST(x0, x1);
        }
      }

      /* Save geolocation... */
```

```
      wave->time = pert->time[(track0 + track1) / 2][(xtrack0 + xtrack1) / 2];
      wave->z = 0;
      wave->lon[ixtrack - xtrack0][itrack - track0] =
        pert->lon[itrack][ixtrack];
      wave->lat[ixtrack - xtrack0][itrack - track0] =
        pert->lat[itrack][ixtrack];

      /* Save temperature data... */
      wave->temp[ixtrack - xtrack0][itrack - track0]
        = pert->bt[itrack][ixtrack];
      wave->bg[ixtrack - xtrack0][itrack - track0]
        = pert->bt[itrack][ixtrack] - pert->pt[itrack][ixtrack];
      wave->pt[ixtrack - xtrack0][itrack - track0]
        = pert->pt[itrack][ixtrack];
      wave->var[ixtrack - xtrack0][itrack - track0]
        = pert->var[itrack][ixtrack];
    }
}
```

Here is the call graph for this function:

variance()

```
void variance (
            wave_t * wave,
            double dh )
```

Compute local variance.

Definition at line 492 of file libiasi.c.

```
               {

  double dh2, mu, help;

  int dx, dy, ix, ix2, iy, iy2, n;

  /* Check parameters... */
  if (dh <= 0)
    return;

  /* Compute squared radius... */
  dh2 = gsl_pow_2(dh);

  /* Get sampling distances... */
  dx =
    (int) (dh / fabs(wave->x[wave->nx - 1] - wave->x[0]) * (wave->nx - 1.0) +
           1);
  dy =
    (int) (dh / fabs(wave->y[wave->ny - 1] - wave->y[0]) * (wave->ny - 1.0) +
           1);

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

      /* Init... */
      mu = help = 0;
      n = 0;

      /* Get data... */
      for (ix2 = GSL_MAX(ix - dx, 0); ix2 <= GSL_MIN(ix + dx, wave->nx - 1);
           ix2++)
```

```
      for (iy2 = GSL_MAX(iy - dy, 0); iy2 <= GSL_MIN(iy + dy, wave->ny - 1);
           iy2++)
        if ((gsl_pow_2(wave->x[ix] - wave->x[ix2])
             + gsl_pow_2(wave->y[iy] - wave->y[iy2])) <= dh2)
          if (gsl_finite(wave->pt[ix2][iy2])) {
            mu += wave->pt[ix2][iy2];
            help += gsl_pow_2(wave->pt[ix2][iy2]);
            n++;
          }

    /* Compute local variance... */
    if (n > 1)
      wave->var[ix][iy] = help / n - gsl_pow_2(mu / n);
    else
      wave->var[ix][iy] = GSL_NAN;
  }
}
```

wgs84()

```
double wgs84 (
        double lat )
```

Calculate Earth radius according to WGS-84 reference ellipsoid.

Definition at line 546 of file libiasi.c.

```
                {

  const double a = 6378.1370, b = 6356.7523;

  double cphi, sphi;

  cphi = cos(lat * M_PI / 180.);
  sphi = sin(lat * M_PI / 180.);

  return sqrt((gsl_pow_2(a * a * cphi) + gsl_pow_2(b * b * sphi))
              / (gsl_pow_2(a * cphi) + gsl_pow_2(b * sphi)));
}
```

write_l1()

```
void write_l1 (
        char * filename,
        iasi_l1_t * l1 )
```

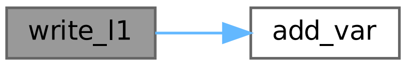
Write IASI Level-1 data.

Definition at line 562 of file libiasi.c.

```
                    {

  int dimid[10], ncid, time_id, lon_id, lat_id,
    sat_z_id, sat_lon_id, sat_lat_id, nu_id, rad_id;

  /* Open or create netCDF file... */
  printf("Write IASI Level-1 file: %s\n", filename);
  if (nc_open(filename, NC_WRITE, &ncid) != NC_NOERR) {
    NC(nc_create(filename, NC_CLOBBER, &ncid));
  } else {
    NC(nc_redef(ncid));
  }

  /* Set dimensions... */
  if (nc_inq_dimid(ncid, "L1_NTRACK", &dimid[0]) != NC_NOERR)
    NC(nc_def_dim(ncid, "L1_NTRACK", l1->ntrack, &dimid[0]));
  if (nc_inq_dimid(ncid, "L1_NXTRACK", &dimid[1]) != NC_NOERR)
    NC(nc_def_dim(ncid, "L1_NXTRACK", L1_NXTRACK, &dimid[1]));
  if (nc_inq_dimid(ncid, "L1_NCHAN", &dimid[2]) != NC_NOERR)
    NC(nc_def_dim(ncid, "L1_NCHAN", L1_NCHAN, &dimid[2]));

  /* Add variables... */
  add_var(ncid, "l1_time", "s", "time (seconds since 2000-01-01T00:00Z)",
          NC_DOUBLE, dimid, &time_id, 2);
  add_var(ncid, "l1_lon", "deg", "longitude", NC_DOUBLE, dimid, &lon_id, 2);
```

```
  add_var(ncid, "l1_lat", "deg", "latitude", NC_DOUBLE, dimid, &lat_id, 2);
  add_var(ncid, "l1_sat_z", "km", "satellite altitude",
          NC_DOUBLE, dimid, &sat_z_id, 1);
  add_var(ncid, "l1_sat_lon", "deg", "(estimated) satellite longitude",
          NC_DOUBLE, dimid, &sat_lon_id, 1);
  add_var(ncid, "l1_sat_lat", "deg", "(estimated) satellite latitude",
          NC_DOUBLE, dimid, &sat_lat_id, 1);
  add_var(ncid, "l1_nu", "cm^-1", "channel wavenumber",
          NC_DOUBLE, &dimid[2], &nu_id, 1);
  add_var(ncid, "l1_rad", "W/(m^2 sr cm^-1)", "channel radiance",
          NC_FLOAT, dimid, &rad_id, 3);

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

  /* Write data... */
  NC(nc_put_var_double(ncid, time_id, l1->time[0]));
  NC(nc_put_var_double(ncid, lon_id, l1->lon[0]));
  NC(nc_put_var_double(ncid, lat_id, l1->lat[0]));
  NC(nc_put_var_double(ncid, sat_z_id, l1->sat_z));
  NC(nc_put_var_double(ncid, sat_lon_id, l1->sat_lon));
  NC(nc_put_var_double(ncid, sat_lat_id, l1->sat_lat));
  NC(nc_put_var_double(ncid, nu_id, l1->nu));
  NC(nc_put_var_float(ncid, rad_id, &l1->rad[0][0][0]));

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

Here is the call graph for this function:



write_l2()

```
void write_l2 (
            char * filename,
            iasi_l2_t * l2 )
```

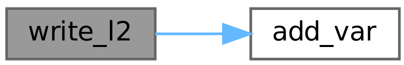
Write IASI Level-2 data.

Definition at line 620 of file libiasi.c.

```
                     {

  int dimid[10], ncid, time_id, z_id, lon_id, lat_id, p_id, t_id;

  /* Create netCDF file... */
  printf("Write IASI Level-2 file: %s\n", filename);
  if (nc_open(filename, NC_WRITE, &ncid) != NC_NOERR) {
    NC(nc_create(filename, NC_CLOBBER, &ncid));
  } else {
    NC(nc_redef(ncid));
  }

  /* Set dimensions... */
  if (nc_inq_dimid(ncid, "L2_NTRACK", &dimid[0]) != NC_NOERR)
    NC(nc_def_dim(ncid, "L2_NTRACK", l2->ntrack, &dimid[0]));
  if (nc_inq_dimid(ncid, "L2_NXTRACK", &dimid[1]) != NC_NOERR)
    NC(nc_def_dim(ncid, "L2_NXTRACK", L2_NXTRACK, &dimid[1]));
  if (nc_inq_dimid(ncid, "L2_NLAY", &dimid[2]) != NC_NOERR)
    NC(nc_def_dim(ncid, "L2_NLAY", L2_NLAY, &dimid[2]));

  /* Add variables... */
  add_var(ncid, "l2_time", "s", "time (seconds since 2000-01-01T00:00Z)",
```

```
           NC_DOUBLE, dimid, &time_id, 2);
  add_var(ncid, "l2_z", "km", "altitude", NC_DOUBLE, dimid, &z_id, 3);
  add_var(ncid, "l2_lon", "deg", "longitude", NC_DOUBLE, dimid, &lon_id, 2);
  add_var(ncid, "l2_lat", "deg", "latitude", NC_DOUBLE, dimid, &lat_id, 2);
  add_var(ncid, "l2_press", "hPa", "pressure",
           NC_DOUBLE, &dimid[2], &p_id, 1);
  add_var(ncid, "l2_temp", "K", "temperature", NC_DOUBLE, dimid, &t_id, 3);

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

  /* Write data... */
  NC(nc_put_var_double(ncid, time_id, l2->time[0]));
  NC(nc_put_var_double(ncid, z_id, l2->z[0][0]));
  NC(nc_put_var_double(ncid, lon_id, l2->lon[0]));
  NC(nc_put_var_double(ncid, lat_id, l2->lat[0]));
  NC(nc_put_var_double(ncid, p_id, l2->p));
  NC(nc_put_var_double(ncid, t_id, l2->t[0][0]));

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

Here is the call graph for this function:



5.10 libiasi.c

Go to the documentation of this file.

```
#include "libiasi.h"

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

void add_var(
  int ncid,
  const char *varname,
  const char *unit,
  const char *longname,
  int type,
  int dimid[],
  int *varid,
  int ndims) {

  /* Check if variable exists... */
  if (nc_inq_varid(ncid, varname, varid) != NC_NOERR) {

    /* Define variable... */
    NC(nc_def_var(ncid, varname, type, ndims, dimid, varid));

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

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

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

void background_poly_help(
  double *xx,
  double *yy,
  int n,
  int dim) {

  gsl_multifit_linear_workspace *work;
```

```
  gsl_matrix *cov, *X;
  gsl_vector *c, *x, *y;

  double chisq, xx2[WX > WY ? WX : WY], yy2[WX > WY ? WX : WY];

  size_t i, i2, n2 = 0;

  /* Check for nan... */
  for (i = 0; i < (size_t) n; i++)
    if (gsl_finite(yy[i])) {
      xx2[n2] = xx[i];
      yy2[n2] = yy[i];
      n2++;
    }
  if ((int) n2 < dim || n2 < 0.9 * n) {
    for (i = 0; i < (size_t) n; i++)
      yy[i] = GSL_NAN;
    return;
  }

  /* Allocate... */
  work = gsl_multifit_linear_alloc((size_t) n2, (size_t) dim);
  cov = gsl_matrix_alloc((size_t) dim, (size_t) dim);
  X = gsl_matrix_alloc((size_t) n2, (size_t) dim);
  c = gsl_vector_alloc((size_t) dim);
  x = gsl_vector_alloc((size_t) n2);
  y = gsl_vector_alloc((size_t) n2);

  /* Compute polynomial fit... */
  for (i = 0; i < (size_t) n2; i++) {
    gsl_vector_set(x, i, xx2[i]);
    gsl_vector_set(y, i, yy2[i]);
    for (i2 = 0; i2 < (size_t) dim; i2++)
      gsl_matrix_set(X, i, i2, pow(gsl_vector_get(x, i), (double) i2));
  }
  gsl_multifit_linear(X, y, c, cov, &chisq, work);
  for (i = 0; i < (size_t) n; i++)
    yy[i] = gsl_poly_eval(c->data, (int) dim, xx[i]);

  /* Free... */
  gsl_multifit_linear_free(work);
  gsl_matrix_free(cov);
  gsl_matrix_free(X);
  gsl_vector_free(c);
  gsl_vector_free(x);
  gsl_vector_free(y);
}

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

void background_poly(
  wave_t * wave,
  int dim_x,
  int dim_y) {

  double help[WX], x[WX], x2[WY], y[WX], y2[WY];

  int ix, iy;

  /* Copy temperatures to background... */
  for (ix = 0; ix < wave->nx; ix++)
    for (iy = 0; iy < wave->ny; iy++) {
      wave->bg[ix][iy] = wave->temp[ix][iy];
      wave->pt[ix][iy] = 0;
    }

  /* Check parameters... */
  if (dim_x <= 0 && dim_y <= 0)
    return;

  /* Compute fit in x-direction... */
  if (dim_x > 0)
    for (iy = 0; iy < wave->ny; iy++) {
      for (ix = 0; ix <= 53; ix++) {
        x[ix] = (double) ix;
        y[ix] = wave->bg[ix][iy];
      }
      background_poly_help(x, y, 54, dim_x);
      for (ix = 0; ix <= 29; ix++)
        help[ix] = y[ix];

      for (ix = 6; ix <= 59; ix++) {
        x[ix - 6] = (double) ix;
        y[ix - 6] = wave->bg[ix][iy];
      }
      background_poly_help(x, y, 54, dim_x);
      for (ix = 30; ix <= 59; ix++)
```

```
        help[ix] = y[ix - 6];

      for (ix = 0; ix < wave->nx; ix++)
        wave->bg[ix][iy] = help[ix];
    }

  /* Compute fit in y-direction... */
  if (dim_y > 0)
    for (ix = 0; ix < wave->nx; ix++) {
      for (iy = 0; iy < wave->ny; iy++) {
        x2[iy] = (int) iy;
        y2[iy] = wave->bg[ix][iy];
      }
      background_poly_help(x2, y2, wave->ny, dim_y);
      for (iy = 0; iy < wave->ny; iy++)
        wave->bg[ix][iy] = y2[iy];
    }

  /* Recompute perturbations... */
  for (ix = 0; ix < wave->nx; ix++)
    for (iy = 0; iy < wave->ny; iy++)
      wave->pt[ix][iy] = wave->temp[ix][iy] - wave->bg[ix][iy];
}

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

void iasi_read(
  char *filename,
  iasi_rad_t * iasi_rad) {

  const char *product_class;

  coda_product *pf;

  coda_cursor cursor;

  iasi_raw_t *iasi_raw;

  int i, j, w, tr1, tr2, tr1_lpm, tr1_rpm, tr2_lpm, tr2_rpm,
    ichan, mdr_i, num_dims = 1;

  long dim[] = { 1, 2, 3, 4, 5, 6, 7, 8, 9 };

  short int IDefScaleSondNbScale, IDefScaleSondNsfirst[10],
    IDefScaleSondNslast[10], IDefScaleSondScaleFactor[10];

  float sc, scaling[IASI_L1_NCHAN];

  /* Initialize CODA... */
  coda_init();

  /* Allocate... */
  ALLOC(iasi_raw, iasi_raw_t, 1);

  /* Open IASI file... */
  CODA(coda_open(filename, &pf));
  CODA(coda_get_product_class(pf, &product_class));
  CODA(coda_cursor_set_product(&cursor, pf));

  /* Get scaling parameters... */
  CODA(coda_cursor_goto_record_field_by_name(&cursor, "GIADR_ScaleFactors"));

  CODA(coda_cursor_goto_record_field_by_name
       (&cursor, "IDefScaleSondNbScale"));
  CODA(coda_cursor_read_int16(&cursor, &IDefScaleSondNbScale));
  CODA(coda_cursor_goto_parent(&cursor));

  CODA(coda_cursor_goto_record_field_by_name
       (&cursor, "IDefScaleSondNsfirst"));
  CODA(coda_cursor_read_int16_array
       (&cursor, IDefScaleSondNsfirst, coda_array_ordering_c));
  CODA(coda_cursor_goto_parent(&cursor));

  CODA(coda_cursor_goto_record_field_by_name(&cursor, "IDefScaleSondNslast"));
  CODA(coda_cursor_read_int16_array
       (&cursor, IDefScaleSondNslast, coda_array_ordering_c));
  CODA(coda_cursor_goto_parent(&cursor));

  CODA(coda_cursor_goto_record_field_by_name
       (&cursor, "IDefScaleSondScaleFactor"));
  CODA(coda_cursor_read_int16_array
       (&cursor, IDefScaleSondScaleFactor, coda_array_ordering_c));

  /* Compute scaling factors... */
  for (ichan = 0; ichan < IASI_L1_NCHAN; ichan++)
    scaling[ichan] = GSL_NAN;
  for (i = 0; i < IDefScaleSondNbScale; i++) {
```

```
    sc = (float) pow(10.0, -IDefScaleSondScaleFactor[i]);
    for (ichan = IDefScaleSondNsfirst[i] - 1;
         ichan < IDefScaleSondNslast[i]; ichan++) {
      w = ichan - IASI_IDefNsfirst1b + 1;
      if (w >= 0 && w < IASI_L1_NCHAN)
        scaling[w] = sc;
    }
  }

  /* Get number of tracks in record... */
  CODA(coda_cursor_goto_root(&cursor));
  CODA(coda_cursor_goto_record_field_by_name(&cursor, "MDR"));
  CODA(coda_cursor_get_array_dim(&cursor, &num_dims, dim));
  iasi_raw->ntrack = dim[0];

  /* Read tracks one by one... */
  for (mdr_i = 0; mdr_i < iasi_raw->ntrack; mdr_i++) {

    /* Reset cursor position... */
    CODA(coda_cursor_goto_root(&cursor));

    /* Move cursor to radiation data... */
    CODA(coda_cursor_goto_record_field_by_name(&cursor, "MDR"));
    CODA(coda_cursor_goto_array_element_by_index(&cursor, mdr_i));
    CODA(coda_cursor_goto_record_field_by_name(&cursor, "MDR"));
    CODA(coda_cursor_goto_record_field_by_name(&cursor, "GS1cSpect"));
    CODA(coda_cursor_read_int16_array
         (&cursor, &iasi_raw->Radiation[mdr_i][0][0][0],
          coda_array_ordering_c));

    /* Read time... */
    CODA(coda_cursor_goto_parent(&cursor));
    CODA(coda_cursor_goto_record_field_by_name(&cursor, "OnboardUTC"));
    CODA(coda_cursor_read_double_array
         (&cursor, &iasi_raw->Time[mdr_i][0], coda_array_ordering_c));

    /* Read coordinates... */
    CODA(coda_cursor_goto_parent(&cursor));
    CODA(coda_cursor_goto_record_field_by_name(&cursor, "GGeoSondLoc"));
    CODA(coda_cursor_read_double_array
         (&cursor, &iasi_raw->Loc[mdr_i][0][0][0], coda_array_ordering_c));

    /* Read satellite altitude... */
    CODA(coda_cursor_goto_parent(&cursor));
    CODA(coda_cursor_goto_record_field_by_name(&cursor,
                                               "EARTH_SATELLITE_DISTANCE"));
    CODA(coda_cursor_read_uint32(&cursor, &iasi_raw->Sat_z[mdr_i]));

    /* Read spectral range... */
    iasi_raw->IDefSpectDWn1b[mdr_i] = IASI_IDefSpectDWn1b / 100.0;

    CODA(coda_cursor_goto_parent(&cursor));
    CODA(coda_cursor_goto_record_field_by_name(&cursor, "IDefNsfirst1b"));
    CODA(coda_cursor_read_int32(&cursor, &iasi_raw->IDefNsfirst1b[mdr_i]));
    if (iasi_raw->IDefNsfirst1b[mdr_i] != IASI_IDefNsfirst1b)
      ERRMSG("Unexpected value for IDefNsfirst1b!");

    CODA(coda_cursor_goto_parent(&cursor));
    CODA(coda_cursor_goto_record_field_by_name(&cursor, "IDefNslast1b"));
    CODA(coda_cursor_read_int32(&cursor, &iasi_raw->IDefNslast1b[mdr_i]));
    if (iasi_raw->IDefNslast1b[mdr_i] != IASI_IDefNslast1b)
      ERRMSG("Unexpected value for IDefNslast1b!");

    /* Compute wavenumber... */
    if (mdr_i == 0)
      for (i = 0; i < IASI_L1_NCHAN; i++)
        iasi_raw->Wavenumber[i] =
          iasi_raw->IDefSpectDWn1b[mdr_i] *
          (float) (iasi_raw->IDefNsfirst1b[mdr_i] + i - 1);
  }

  /* Close file... */
  CODA(coda_close(pf));

  /* Finalize CODA... */
  coda_done();

  /* Set number of tracks... */
  iasi_rad->ntrack = (int) (iasi_raw->ntrack * 2);

  /* Copy wavenumbers... */
  for (ichan = 0; ichan < IASI_L1_NCHAN; ichan++)
    iasi_rad->freq[ichan] = iasi_raw->Wavenumber[ichan];

  /* Copy footprint data... */
  for (mdr_i = 0; mdr_i < iasi_raw->ntrack; mdr_i++) {
    tr1 = mdr_i * 2;
```

```
    tr2 = mdr_i * 2 + 1;
    tr1_lpm = 3;
    tr1_rpm = 0;
    tr2_lpm = 2;
    tr2_rpm = 1;

    /* Copy time (2x2 matrix has same measurement time)...  */
    for (i = 0; i < IASI_NXTRACK; i++) {
      iasi_rad->Time[tr1][i * 2] = iasi_raw->Time[mdr_i][i];
      iasi_rad->Time[tr1][i * 2 + 1] = iasi_raw->Time[mdr_i][i];
      iasi_rad->Time[tr2][i * 2] = iasi_raw->Time[mdr_i][i];
      iasi_rad->Time[tr2][i * 2 + 1] = iasi_raw->Time[mdr_i][i];
    }

    /* Copy location... */
    for (i = 0; i < IASI_NXTRACK; i++) {
      iasi_rad->Longitude[tr1][i * 2] = iasi_raw->Loc[mdr_i][i][tr1_lpm][0];
      iasi_rad->Longitude[tr1][i * 2 + 1] =
        iasi_raw->Loc[mdr_i][i][tr1_rpm][0];
      iasi_rad->Latitude[tr1][i * 2] = iasi_raw->Loc[mdr_i][i][tr1_lpm][1];
      iasi_rad->Latitude[tr1][i * 2 + 1] =
        iasi_raw->Loc[mdr_i][i][tr1_rpm][1];

      iasi_rad->Longitude[tr2][i * 2] = iasi_raw->Loc[mdr_i][i][tr2_lpm][0];
      iasi_rad->Longitude[tr2][i * 2 + 1] =
        iasi_raw->Loc[mdr_i][i][tr2_rpm][0];
      iasi_rad->Latitude[tr2][i * 2] = iasi_raw->Loc[mdr_i][i][tr2_lpm][1];
      iasi_rad->Latitude[tr2][i * 2 + 1] =
        iasi_raw->Loc[mdr_i][i][tr2_rpm][1];
    }

    /* Copy satellite location (we only have one height value)... */
    iasi_rad->Sat_lon[tr1] = iasi_rad->Longitude[tr1][28];
    iasi_rad->Sat_lat[tr1] = iasi_rad->Latitude[tr1][28];
    iasi_rad->Sat_lon[tr2] = iasi_rad->Longitude[tr2][28];
    iasi_rad->Sat_lat[tr2] = iasi_rad->Latitude[tr2][28];
    iasi_rad->Sat_z[tr1] =
      iasi_raw->Sat_z[mdr_i] / 1000.0 - wgs84(iasi_rad->Sat_lat[tr1]);
    iasi_rad->Sat_z[tr2] =
      iasi_raw->Sat_z[mdr_i] / 1000.0 - wgs84(iasi_rad->Sat_lat[tr2]);

    /* Copy radiation data... */
    for (i = 0; i < IASI_NXTRACK; i++) {
      for (ichan = 0; ichan < IASI_L1_NCHAN; ichan++) {
        sc = scaling[ichan] * 100.0f;
        iasi_rad->Rad[tr1][i * 2][ichan] =
          iasi_raw->Radiation[mdr_i][i][tr1_lpm][ichan] * sc;
        iasi_rad->Rad[tr1][i * 2 + 1][ichan] =
          iasi_raw->Radiation[mdr_i][i][tr1_rpm][ichan] * sc;
        iasi_rad->Rad[tr2][i * 2][ichan] =
          iasi_raw->Radiation[mdr_i][i][tr2_lpm][ichan] * sc;
        iasi_rad->Rad[tr2][i * 2 + 1][ichan] =
          iasi_raw->Radiation[mdr_i][i][tr2_rpm][ichan] * sc;
      }
    }
  }

  /* Check radiance data... */
  for (i = 0; i < iasi_rad->ntrack; i++)
    for (j = 0; j < L1_NXTRACK; j++)
      if (iasi_rad->Rad[i][j][6753] > iasi_rad->Rad[i][j][6757]
          || iasi_rad->Rad[i][j][6753] < 0)
        for (ichan = 0; ichan < IASI_L1_NCHAN; ichan++)
          iasi_rad->Rad[i][j][ichan] = GSL_NAN;

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

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

void noise(
  wave_t * wave,
  double *mu,
  double *sig) {

  int ix, ix2, iy, iy2, n = 0, okay;

  /* Init... */
  *mu = 0;
  *sig = 0;

  /* Estimate noise (Immerkaer, 1996)... */
  for (ix = 1; ix < wave->nx - 1; ix++)
    for (iy = 1; iy < wave->ny - 1; iy++) {

      /* Check data... */
```

```
      okay = 1;
      for (ix2 = ix - 1; ix2 <= ix + 1; ix2++)
        for (iy2 = iy - 1; iy2 <= iy + 1; iy2++)
          if (!gsl_finite(wave->temp[ix2][iy2]))
            okay = 0;
      if (!okay)
        continue;

      /* Get mean noise... */
      n++;
      *mu += wave->temp[ix][iy];
      *sig += gsl_pow_2(+4. / 6. * wave->temp[ix][iy]
                        - 2. / 6. * (wave->temp[ix - 1][iy]
                                     + wave->temp[ix + 1][iy]
                                     + wave->temp[ix][iy - 1]
                                     + wave->temp[ix][iy + 1])
                        + 1. / 6. * (wave->temp[ix - 1][iy - 1]
                                     + wave->temp[ix + 1][iy - 1]
                                     + wave->temp[ix - 1][iy + 1]
                                     + wave->temp[ix + 1][iy + 1]));
    }

  /* Normalize... */
  *mu /= (double) n;
  *sig = sqrt(*sig / (double) n);
}

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

void pert2wave(
  pert_t * pert,
  wave_t * wave,
  int track0,
  int track1,
  int xtrack0,
  int xtrack1) {

  double x0[3], x1[3];

  int itrack, ixtrack;

  /* Check ranges... */
  track0 = GSL_MIN(GSL_MAX(track0, 0), pert->ntrack - 1);
  track1 = GSL_MIN(GSL_MAX(track1, 0), pert->ntrack - 1);
  xtrack0 = GSL_MIN(GSL_MAX(xtrack0, 0), pert->nxtrack - 1);
  xtrack1 = GSL_MIN(GSL_MAX(xtrack1, 0), pert->nxtrack - 1);

  /* Set size... */
  wave->nx = xtrack1 - xtrack0 + 1;
  if (wave->nx > WX)
    ERRMSG("Too many across-track values!");
  wave->ny = track1 - track0 + 1;
  if (wave->ny > WY)
    ERRMSG("Too many along-track values!");

  /* Loop over footprints... */
  for (itrack = track0; itrack <= track1; itrack++)
    for (ixtrack = xtrack0; ixtrack <= xtrack1; ixtrack++) {

      /* Get distances... */
      if (itrack == track0) {
        wave->x[0] = 0;
        if (ixtrack > xtrack0) {
          geo2cart(0, pert->lon[itrack][ixtrack - 1],
                   pert->lat[itrack][ixtrack - 1], x0);
          geo2cart(0, pert->lon[itrack][ixtrack],
                   pert->lat[itrack][ixtrack], x1);
          wave->x[ixtrack - xtrack0] =
            wave->x[ixtrack - xtrack0 - 1] + DIST(x0, x1);
        }
      }
      if (ixtrack == xtrack0) {
        wave->y[0] = 0;
        if (itrack > track0) {
          geo2cart(0, pert->lon[itrack - 1][ixtrack],
                   pert->lat[itrack - 1][ixtrack], x0);
          geo2cart(0, pert->lon[itrack][ixtrack],
                   pert->lat[itrack][ixtrack], x1);
          wave->y[itrack - track0] =
            wave->y[itrack - track0 - 1] + DIST(x0, x1);
        }
      }

      /* Save geolocation... */
      wave->time = pert->time[(track0 + track1) / 2][(xtrack0 + xtrack1) / 2];
      wave->z = 0;
      wave->lon[ixtrack - xtrack0][itrack - track0] =
```

```
      pert->lon[itrack][ixtrack];
    wave->lat[ixtrack - xtrack0][itrack - track0] =
      pert->lat[itrack][ixtrack];

    /* Save temperature data... */
    wave->temp[ixtrack - xtrack0][itrack - track0]
      = pert->bt[itrack][ixtrack];
    wave->bg[ixtrack - xtrack0][itrack - track0]
      = pert->bt[itrack][ixtrack] - pert->pt[itrack][ixtrack];
    wave->pt[ixtrack - xtrack0][itrack - track0]
      = pert->pt[itrack][ixtrack];
    wave->var[ixtrack - xtrack0][itrack - track0]
      = pert->var[itrack][ixtrack];
  }
}

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

void variance(
  wave_t * wave,
  double dh) {

  double dh2, mu, help;

  int dx, dy, ix, ix2, iy, iy2, n;

  /* Check parameters... */
  if (dh <= 0)
    return;

  /* Compute squared radius... */
  dh2 = gsl_pow_2(dh);

  /* Get sampling distances... */
  dx =
    (int) (dh / fabs(wave->x[wave->nx - 1] - wave->x[0]) * (wave->nx - 1.0) +
           1);
  dy =
    (int) (dh / fabs(wave->y[wave->ny - 1] - wave->y[0]) * (wave->ny - 1.0) +
           1);

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

      /* Init... */
      mu = help = 0;
      n = 0;

      /* Get data... */
      for (ix2 = GSL_MAX(ix - dx, 0); ix2 <= GSL_MIN(ix + dx, wave->nx - 1);
           ix2++)
        for (iy2 = GSL_MAX(iy - dy, 0); iy2 <= GSL_MIN(iy + dy, wave->ny - 1);
             iy2++)
          if ((gsl_pow_2(wave->x[ix] - wave->x[ix2])
               + gsl_pow_2(wave->y[iy] - wave->y[iy2])) <= dh2)
            if (gsl_finite(wave->pt[ix2][iy2])) {
              mu += wave->pt[ix2][iy2];
              help += gsl_pow_2(wave->pt[ix2][iy2]);
              n++;
            }

      /* Compute local variance... */
      if (n > 1)
        wave->var[ix][iy] = help / n - gsl_pow_2(mu / n);
      else
        wave->var[ix][iy] = GSL_NAN;
    }
}

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

double wgs84(
  double lat) {

  const double a = 6378.1370, b = 6356.7523;

  double cphi, sphi;

  cphi = cos(lat * M_PI / 180.);
  sphi = sin(lat * M_PI / 180.);

  return sqrt((gsl_pow_2(a * a * cphi) + gsl_pow_2(b * b * sphi))
              / (gsl_pow_2(a * cphi) + gsl_pow_2(b * sphi)));
}

/*****************************************************************************/
```

```c

void write_l1(
  char *filename,
  iasi_l1_t * l1) {

  int dimid[10], ncid, time_id, lon_id, lat_id,
    sat_z_id, sat_lon_id, sat_lat_id, nu_id, rad_id;

  /* Open or create netCDF file... */
  printf("Write IASI Level-1 file: %s\n", filename);
  if (nc_open(filename, NC_WRITE, &ncid) != NC_NOERR) {
    NC(nc_create(filename, NC_CLOBBER, &ncid));
  } else {
    NC(nc_redef(ncid));
  }

  /* Set dimensions... */
  if (nc_inq_dimid(ncid, "L1_NTRACK", &dimid[0]) != NC_NOERR)
    NC(nc_def_dim(ncid, "L1_NTRACK", l1->ntrack, &dimid[0]));
  if (nc_inq_dimid(ncid, "L1_NXTRACK", &dimid[1]) != NC_NOERR)
    NC(nc_def_dim(ncid, "L1_NXTRACK", L1_NXTRACK, &dimid[1]));
  if (nc_inq_dimid(ncid, "L1_NCHAN", &dimid[2]) != NC_NOERR)
    NC(nc_def_dim(ncid, "L1_NCHAN", L1_NCHAN, &dimid[2]));

  /* Add variables... */
  add_var(ncid, "l1_time", "s", "time (seconds since 2000-01-01T00:00Z)",
          NC_DOUBLE, dimid, &time_id, 2);
  add_var(ncid, "l1_lon", "deg", "longitude", NC_DOUBLE, dimid, &lon_id, 2);
  add_var(ncid, "l1_lat", "deg", "latitude", NC_DOUBLE, dimid, &lat_id, 2);
  add_var(ncid, "l1_sat_z", "km", "satellite altitude",
          NC_DOUBLE, dimid, &sat_z_id, 1);
  add_var(ncid, "l1_sat_lon", "deg", "(estimated) satellite longitude",
          NC_DOUBLE, dimid, &sat_lon_id, 1);
  add_var(ncid, "l1_sat_lat", "deg", "(estimated) satellite latitude",
          NC_DOUBLE, dimid, &sat_lat_id, 1);
  add_var(ncid, "l1_nu", "cm^-1", "channel wavenumber",
          NC_DOUBLE, &dimid[2], &nu_id, 1);
  add_var(ncid, "l1_rad", "W/(m^2 sr cm^-1)", "channel radiance",
          NC_FLOAT, dimid, &rad_id, 3);

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

  /* Write data... */
  NC(nc_put_var_double(ncid, time_id, l1->time[0]));
  NC(nc_put_var_double(ncid, lon_id, l1->lon[0]));
  NC(nc_put_var_double(ncid, lat_id, l1->lat[0]));
  NC(nc_put_var_double(ncid, sat_z_id, l1->sat_z));
  NC(nc_put_var_double(ncid, sat_lon_id, l1->sat_lon));
  NC(nc_put_var_double(ncid, sat_lat_id, l1->sat_lat));
  NC(nc_put_var_double(ncid, nu_id, l1->nu));
  NC(nc_put_var_float(ncid, rad_id, &l1->rad[0][0][0]));

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

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

void write_l2(
  char *filename,
  iasi_l2_t * l2) {

  int dimid[10], ncid, time_id, z_id, lon_id, lat_id, p_id, t_id;

  /* Create netCDF file... */
  printf("Write IASI Level-2 file: %s\n", filename);
  if (nc_open(filename, NC_WRITE, &ncid) != NC_NOERR) {
    NC(nc_create(filename, NC_CLOBBER, &ncid));
  } else {
    NC(nc_redef(ncid));
  }

  /* Set dimensions... */
  if (nc_inq_dimid(ncid, "L2_NTRACK", &dimid[0]) != NC_NOERR)
    NC(nc_def_dim(ncid, "L2_NTRACK", l2->ntrack, &dimid[0]));
  if (nc_inq_dimid(ncid, "L2_NXTRACK", &dimid[1]) != NC_NOERR)
    NC(nc_def_dim(ncid, "L2_NXTRACK", L2_NXTRACK, &dimid[1]));
  if (nc_inq_dimid(ncid, "L2_NLAY", &dimid[2]) != NC_NOERR)
    NC(nc_def_dim(ncid, "L2_NLAY", L2_NLAY, &dimid[2]));

  /* Add variables... */
  add_var(ncid, "l2_time", "s", "time (seconds since 2000-01-01T00:00Z)",
          NC_DOUBLE, dimid, &time_id, 2);
  add_var(ncid, "l2_z", "km", "altitude", NC_DOUBLE, dimid, &z_id, 3);
  add_var(ncid, "l2_lon", "deg", "longitude", NC_DOUBLE, dimid, &lon_id, 2);
  add_var(ncid, "l2_lat", "deg", "latitude", NC_DOUBLE, dimid, &lat_id, 2);
```

```
  add_var(ncid, "l2_press", "hPa", "pressure",
          NC_DOUBLE, &dimid[2], &p_id, 1);
  add_var(ncid, "l2_temp", "K", "temperature", NC_DOUBLE, dimid, &t_id, 3);

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

  /* Write data... */
  NC(nc_put_var_double(ncid, time_id, l2->time[0]));
  NC(nc_put_var_double(ncid, z_id, l2->z[0][0]));
  NC(nc_put_var_double(ncid, lon_id, l2->lon[0]));
  NC(nc_put_var_double(ncid, lat_id, l2->lat[0]));
  NC(nc_put_var_double(ncid, p_id, l2->p));
  NC(nc_put_var_double(ncid, t_id, l2->t[0][0]));

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

5.11 libiasi.h File Reference

Data Structures

- struct iasi_l1_t

  IASI Level-1 data.
- struct iasi_l2_t

  IASI Level-2 data.
- struct pert_t

  Perturbation data.
- struct iasi_raw_t

  IASI raw Level-1 data.
- struct iasi_rad_t

  IASI converted Level-1 radiation data.
- struct wave_t

  Wave analysis data.

Functions

- void add_var (int ncid, const char ∗varname, const char ∗unit, const char ∗longname, int type, int dimid[ ], int ∗varid, int ndims)

  Add variable to netCDF file.
- void background_poly (wave_t ∗wave, int dim_x, int dim_y)

  Get background based on polynomial fits.
- void background_poly_help (double ∗xx, double ∗yy, int n, int dim)

  Get background based on polynomial fits.
- int get_chan_for_wavenumber (float wavenumber)

  Get closest channel for a wavenumber [cm] (uses expected min wavenumber).
- void iasi_read (char ∗filename, iasi_rad_t ∗iasi_rad)

  Read IASI Level-1 data and convert to radiation type.
- void noise (wave_t ∗wave, double ∗mu, double ∗sig)

  Estimate noise.
- 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 variance (wave_t ∗wave, double dh)

  Compute local variance.
- double wgs84 (double lat)

  Calculate Earth radius according to WGS-84 reference ellipsoid.
- void write_l1 (char ∗filename, iasi_l1_t ∗l1)

  Write IASI Level-1 data.
- void write_l2 (char ∗filename, iasi_l2_t ∗l2)

  Write IASI Level-2 data.

5.11.1 Function Documentation

add_var()

```
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 5 of file libiasi.c.

```
                {

  /* Check if variable exists... */
  if (nc_inq_varid(ncid, varname, varid) != NC_NOERR) {

    /* Define variable... */
    NC(nc_def_var(ncid, varname, type, ndims, dimid, varid));

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

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

background_poly()

```
void background_poly (
            wave_t * wave,
            int dim_x,
            int dim_y )
```

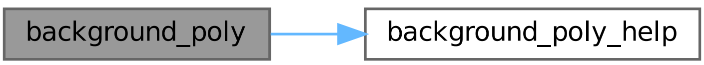
Get background based on polynomial fits.

Definition at line 89 of file libiasi.c.

```
                {

  double help[WX], x[WX], x2[WY], y[WX], y2[WY];

  int ix, iy;

  /* Copy temperatures to background... */
  for (ix = 0; ix < wave->nx; ix++)
    for (iy = 0; iy < wave->ny; iy++) {
      wave->bg[ix][iy] = wave->temp[ix][iy];
      wave->pt[ix][iy] = 0;
    }

  /* Check parameters... */
  if (dim_x <= 0 && dim_y <= 0)
    return;

  /* Compute fit in x-direction... */
  if (dim_x > 0)
    for (iy = 0; iy < wave->ny; iy++) {
      for (ix = 0; ix <= 53; ix++) {
        x[ix] = (double) ix;
        y[ix] = wave->bg[ix][iy];
```

```
      }
      background_poly_help(x, y, 54, dim_x);
      for (ix = 0; ix <= 29; ix++)
        help[ix] = y[ix];

      for (ix = 6; ix <= 59; ix++) {
        x[ix - 6] = (double) ix;
        y[ix - 6] = wave->bg[ix][iy];
      }
      background_poly_help(x, y, 54, dim_x);
      for (ix = 30; ix <= 59; ix++)
        help[ix] = y[ix - 6];

      for (ix = 0; ix < wave->nx; ix++)
        wave->bg[ix][iy] = help[ix];
    }

  /* Compute fit in y-direction... */
  if (dim_y > 0)
    for (ix = 0; ix < wave->nx; ix++) {
      for (iy = 0; iy < wave->ny; iy++) {
        x2[iy] = (int) iy;
        y2[iy] = wave->bg[ix][iy];
      }
      background_poly_help(x2, y2, wave->ny, dim_y);
      for (iy = 0; iy < wave->ny; iy++)
        wave->bg[ix][iy] = y2[iy];
    }

  /* Recompute perturbations... */
  for (ix = 0; ix < wave->nx; ix++)
    for (iy = 0; iy < wave->ny; iy++)
      wave->pt[ix][iy] = wave->temp[ix][iy] - wave->bg[ix][iy];
}
```

Here is the call graph for this function:



background_poly_help()

```
void background_poly_help (
              double * xx,
              double * yy,
              int n,
              int dim )
```

Get background based on polynomial fits.

Definition at line 32 of file libiasi.c.

```
          {

  gsl_multifit_linear_workspace *work;
  gsl_matrix *cov, *X;
  gsl_vector *c, *x, *y;

  double chisq, xx2[WX > WY ? WX : WY], yy2[WX > WY ? WX : WY];

  size_t i, i2, n2 = 0;

  /* Check for nan... */
  for (i = 0; i < (size_t) n; i++)
    if (gsl_finite(yy[i])) {
```

```
      xx2[n2] = xx[i];
      yy2[n2] = yy[i];
      n2++;
    }
  if ((int) n2 < dim || n2 < 0.9 * n) {
    for (i = 0; i < (size_t) n; i++)
      yy[i] = GSL_NAN;
    return;
  }

  /* Allocate... */
  work = gsl_multifit_linear_alloc((size_t) n2, (size_t) dim);
  cov = gsl_matrix_alloc((size_t) dim, (size_t) dim);
  X = gsl_matrix_alloc((size_t) n2, (size_t) dim);
  c = gsl_vector_alloc((size_t) dim);
  x = gsl_vector_alloc((size_t) n2);
  y = gsl_vector_alloc((size_t) n2);

  /* Compute polynomial fit... */
  for (i = 0; i < (size_t) n2; i++) {
    gsl_vector_set(x, i, xx2[i]);
    gsl_vector_set(y, i, yy2[i]);
    for (i2 = 0; i2 < (size_t) dim; i2++)
      gsl_matrix_set(X, i, i2, pow(gsl_vector_get(x, i), (double) i2));
  }
  gsl_multifit_linear(X, y, c, cov, &chisq, work);
  for (i = 0; i < (size_t) n; i++)
    yy[i] = gsl_poly_eval(c->data, (int) dim, xx[i]);

  /* Free... */
  gsl_multifit_linear_free(work);
  gsl_matrix_free(cov);
  gsl_matrix_free(X);
  gsl_vector_free(c);
  gsl_vector_free(x);
  gsl_vector_free(y);
}
```

get_chan_for_wavenumber()

```
int get_chan_for_wavenumber (
               float wavenumber )
```

Get closest channel for a wavenumber [cm] (uses expected min wavenumber).

iasi_read()

```
void iasi_read (
               char * filename,
               iasi_rad_t * iasi_rad )
```

Read IASI Level-1 data and convert to radiation type.

Definition at line 152 of file libiasi.c.

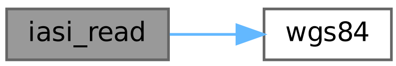
```
                                {

  const char *product_class;

  coda_product *pf;

  coda_cursor cursor;

  iasi_raw_t *iasi_raw;

  int i, j, w, tr1, tr2, tr1_lpm, tr1_rpm, tr2_lpm, tr2_rpm,
    ichan, mdr_i, num_dims = 1;

  long dim[] = { 1, 2, 3, 4, 5, 6, 7, 8, 9 };

```

```
  short int IDefScaleSondNbScale, IDefScaleSondNsfirst[10],
    IDefScaleSondNslast[10], IDefScaleSondScaleFactor[10];

  float sc, scaling[IASI_L1_NCHAN];

  /* Initialize CODA... */
  coda_init();

  /* Allocate... */
  ALLOC(iasi_raw, iasi_raw_t, 1);

  /* Open IASI file... */
  CODA(coda_open(filename, &pf));
  CODA(coda_get_product_class(pf, &product_class));
  CODA(coda_cursor_set_product(&cursor, pf));

  /* Get scaling parameters... */
  CODA(coda_cursor_goto_record_field_by_name(&cursor, "GIADR_ScaleFactors"));

  CODA(coda_cursor_goto_record_field_by_name
       (&cursor, "IDefScaleSondNbScale"));
  CODA(coda_cursor_read_int16(&cursor, &IDefScaleSondNbScale));
  CODA(coda_cursor_goto_parent(&cursor));

  CODA(coda_cursor_goto_record_field_by_name
       (&cursor, "IDefScaleSondNsfirst"));
  CODA(coda_cursor_read_int16_array
       (&cursor, IDefScaleSondNsfirst, coda_array_ordering_c));
  CODA(coda_cursor_goto_parent(&cursor));

  CODA(coda_cursor_goto_record_field_by_name(&cursor, "IDefScaleSondNslast"));
  CODA(coda_cursor_read_int16_array
       (&cursor, IDefScaleSondNslast, coda_array_ordering_c));
  CODA(coda_cursor_goto_parent(&cursor));

  CODA(coda_cursor_goto_record_field_by_name
       (&cursor, "IDefScaleSondScaleFactor"));
  CODA(coda_cursor_read_int16_array
       (&cursor, IDefScaleSondScaleFactor, coda_array_ordering_c));

  /* Compute scaling factors... */
  for (ichan = 0; ichan < IASI_L1_NCHAN; ichan++)
    scaling[ichan] = GSL_NAN;
  for (i = 0; i < IDefScaleSondNbScale; i++) {
    sc = (float) pow(10.0, -IDefScaleSondScaleFactor[i]);
    for (ichan = IDefScaleSondNsfirst[i] - 1;
         ichan < IDefScaleSondNslast[i]; ichan++) {
      w = ichan - IASI_IDefNsfirst1b + 1;
      if (w >= 0 && w < IASI_L1_NCHAN)
        scaling[w] = sc;
    }
  }

  /* Get number of tracks in record... */
  CODA(coda_cursor_goto_root(&cursor));
  CODA(coda_cursor_goto_record_field_by_name(&cursor, "MDR"));
  CODA(coda_cursor_get_array_dim(&cursor, &num_dims, dim));
  iasi_raw->ntrack = dim[0];

  /* Read tracks one by one... */
  for (mdr_i = 0; mdr_i < iasi_raw->ntrack; mdr_i++) {

    /* Reset cursor position... */
    CODA(coda_cursor_goto_root(&cursor));

    /* Move cursor to radiation data... */
    CODA(coda_cursor_goto_record_field_by_name(&cursor, "MDR"));
    CODA(coda_cursor_goto_array_element_by_index(&cursor, mdr_i));
    CODA(coda_cursor_goto_record_field_by_name(&cursor, "MDR"));
    CODA(coda_cursor_goto_record_field_by_name(&cursor, "GS1cSpect"));
    CODA(coda_cursor_read_int16_array
         (&cursor, &iasi_raw->Radiation[mdr_i][0][0][0],
          coda_array_ordering_c));

    /* Read time... */
    CODA(coda_cursor_goto_parent(&cursor));
    CODA(coda_cursor_goto_record_field_by_name(&cursor, "OnboardUTC"));
    CODA(coda_cursor_read_double_array
         (&cursor, &iasi_raw->Time[mdr_i][0], coda_array_ordering_c));

    /* Read coordinates... */
    CODA(coda_cursor_goto_parent(&cursor));
    CODA(coda_cursor_goto_record_field_by_name(&cursor, "GGeoSondLoc"));
    CODA(coda_cursor_read_double_array
         (&cursor, &iasi_raw->Loc[mdr_i][0][0][0], coda_array_ordering_c));

    /* Read satellite altitude... */
```

```
    CODA(coda_cursor_goto_parent(&cursor));
    CODA(coda_cursor_goto_record_field_by_name(&cursor,
                                               "EARTH_SATELLITE_DISTANCE"));
    CODA(coda_cursor_read_uint32(&cursor, &iasi_raw->Sat_z[mdr_i]));

    /* Read spectral range... */
    iasi_raw->IDefSpectDWn1b[mdr_i] = IASI_IDefSpectDWn1b / 100.0;

    CODA(coda_cursor_goto_parent(&cursor));
    CODA(coda_cursor_goto_record_field_by_name(&cursor, "IDefNsfirst1b"));
    CODA(coda_cursor_read_int32(&cursor, &iasi_raw->IDefNsfirst1b[mdr_i]));
    if (iasi_raw->IDefNsfirst1b[mdr_i] != IASI_IDefNsfirst1b)
      ERRMSG("Unexpected value for IDefNsfirst1b!");

    CODA(coda_cursor_goto_parent(&cursor));
    CODA(coda_cursor_goto_record_field_by_name(&cursor, "IDefNslast1b"));
    CODA(coda_cursor_read_int32(&cursor, &iasi_raw->IDefNslast1b[mdr_i]));
    if (iasi_raw->IDefNslast1b[mdr_i] != IASI_IDefNslast1b)
      ERRMSG("Unexpected value for IDefNslast1b!");

    /* Compute wavenumber... */
    if (mdr_i == 0)
      for (i = 0; i < IASI_L1_NCHAN; i++)
        iasi_raw->Wavenumber[i] =
          iasi_raw->IDefSpectDWn1b[mdr_i] *
          (float) (iasi_raw->IDefNsfirst1b[mdr_i] + i - 1);
  }

  /* Close file... */
  CODA(coda_close(pf));

  /* Finalize CODA... */
  coda_done();

  /* Set number of tracks... */
  iasi_rad->ntrack = (int) (iasi_raw->ntrack * 2);

  /* Copy wavenumbers... */
  for (ichan = 0; ichan < IASI_L1_NCHAN; ichan++)
    iasi_rad->freq[ichan] = iasi_raw->Wavenumber[ichan];

  /* Copy footprint data... */
  for (mdr_i = 0; mdr_i < iasi_raw->ntrack; mdr_i++) {
    tr1 = mdr_i * 2;
    tr2 = mdr_i * 2 + 1;
    tr1_lpm = 3;
    tr1_rpm = 0;
    tr2_lpm = 2;
    tr2_rpm = 1;

    /* Copy time (2x2 matrix has same measurement time)...  */
    for (i = 0; i < IASI_NXTRACK; i++) {
      iasi_rad->Time[tr1][i * 2] = iasi_raw->Time[mdr_i][i];
      iasi_rad->Time[tr1][i * 2 + 1] = iasi_raw->Time[mdr_i][i];
      iasi_rad->Time[tr2][i * 2] = iasi_raw->Time[mdr_i][i];
      iasi_rad->Time[tr2][i * 2 + 1] = iasi_raw->Time[mdr_i][i];
    }

    /* Copy location... */
    for (i = 0; i < IASI_NXTRACK; i++) {
      iasi_rad->Longitude[tr1][i * 2] = iasi_raw->Loc[mdr_i][i][tr1_lpm][0];
      iasi_rad->Longitude[tr1][i * 2 + 1] =
        iasi_raw->Loc[mdr_i][i][tr1_rpm][0];
      iasi_rad->Latitude[tr1][i * 2] = iasi_raw->Loc[mdr_i][i][tr1_lpm][1];
      iasi_rad->Latitude[tr1][i * 2 + 1] =
        iasi_raw->Loc[mdr_i][i][tr1_rpm][1];

      iasi_rad->Longitude[tr2][i * 2] = iasi_raw->Loc[mdr_i][i][tr2_lpm][0];
      iasi_rad->Longitude[tr2][i * 2 + 1] =
        iasi_raw->Loc[mdr_i][i][tr2_rpm][0];
      iasi_rad->Latitude[tr2][i * 2] = iasi_raw->Loc[mdr_i][i][tr2_lpm][1];
      iasi_rad->Latitude[tr2][i * 2 + 1] =
        iasi_raw->Loc[mdr_i][i][tr2_rpm][1];
    }

    /* Copy satellite location (we only have one height value)... */
    iasi_rad->Sat_lon[tr1] = iasi_rad->Longitude[tr1][28];
    iasi_rad->Sat_lat[tr1] = iasi_rad->Latitude[tr1][28];
    iasi_rad->Sat_lon[tr2] = iasi_rad->Longitude[tr2][28];
    iasi_rad->Sat_lat[tr2] = iasi_rad->Latitude[tr2][28];
    iasi_rad->Sat_z[tr1] =
      iasi_raw->Sat_z[mdr_i] / 1000.0 - wgs84(iasi_rad->Sat_lat[tr1]);
    iasi_rad->Sat_z[tr2] =
      iasi_raw->Sat_z[mdr_i] / 1000.0 - wgs84(iasi_rad->Sat_lat[tr2]);

    /* Copy radiation data... */
    for (i = 0; i < IASI_NXTRACK; i++) {
```

```
      for (ichan = 0; ichan < IASI_L1_NCHAN; ichan++) {
        sc = scaling[ichan] * 100.0f;
        iasi_rad->Rad[tr1][i * 2][ichan] =
          iasi_raw->Radiation[mdr_i][i][tr1_lpm][ichan] * sc;
        iasi_rad->Rad[tr1][i * 2 + 1][ichan] =
          iasi_raw->Radiation[mdr_i][i][tr1_rpm][ichan] * sc;
        iasi_rad->Rad[tr2][i * 2][ichan] =
          iasi_raw->Radiation[mdr_i][i][tr2_lpm][ichan] * sc;
        iasi_rad->Rad[tr2][i * 2 + 1][ichan] =
          iasi_raw->Radiation[mdr_i][i][tr2_rpm][ichan] * sc;
      }
    }
  }

  /* Check radiance data... */
  for (i = 0; i < iasi_rad->ntrack; i++)
    for (j = 0; j < L1_NXTRACK; j++)
      if (iasi_rad->Rad[i][j][6753] > iasi_rad->Rad[i][j][6757]
          || iasi_rad->Rad[i][j][6753] < 0)
        for (ichan = 0; ichan < IASI_L1_NCHAN; ichan++)
          iasi_rad->Rad[i][j][ichan] = GSL_NAN;

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

Here is the call graph for this function:



noise()

```
void noise (
        wave_t * wave,
        double * mu,
        double * sig )
```

Estimate noise.

Definition at line 371 of file libiasi.c.

```
               {

  int ix, ix2, iy, iy2, n = 0, okay;

  /* Init... */
  *mu = 0;
  *sig = 0;

  /* Estimate noise (Immerkaer, 1996)... */
  for (ix = 1; ix < wave->nx - 1; ix++)
    for (iy = 1; iy < wave->ny - 1; iy++) {

      /* Check data... */
      okay = 1;
      for (ix2 = ix - 1; ix2 <= ix + 1; ix2++)
        for (iy2 = iy - 1; iy2 <= iy + 1; iy2++)
          if (!gsl_finite(wave->temp[ix2][iy2]))
            okay = 0;
      if (!okay)
        continue;

      /* Get mean noise... */
      n++;
```

```
      *mu += wave->temp[ix][iy];
      *sig += gsl_pow_2(+4. / 6. * wave->temp[ix][iy]
                         - 2. / 6. * (wave->temp[ix - 1][iy]
                                      + wave->temp[ix + 1][iy]
                                      + wave->temp[ix][iy - 1]
                                      + wave->temp[ix][iy + 1])
                         + 1. / 6. * (wave->temp[ix - 1][iy - 1]
                                      + wave->temp[ix + 1][iy - 1]
                                      + wave->temp[ix - 1][iy + 1]
                                      + wave->temp[ix + 1][iy + 1]));
    }

  /* Normalize... */
  *mu /= (double) n;
  *sig = sqrt(*sig / (double) n);
}
```

pert2wave()

```
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 416 of file libiasi.c.

```
                  {

  double x0[3], x1[3];

  int itrack, ixtrack;

  /* Check ranges... */
  track0 = GSL_MIN(GSL_MAX(track0, 0), pert->ntrack - 1);
  track1 = GSL_MIN(GSL_MAX(track1, 0), pert->ntrack - 1);
  xtrack0 = GSL_MIN(GSL_MAX(xtrack0, 0), pert->nxtrack - 1);
  xtrack1 = GSL_MIN(GSL_MAX(xtrack1, 0), pert->nxtrack - 1);

  /* Set size... */
  wave->nx = xtrack1 - xtrack0 + 1;
  if (wave->nx > WX)
    ERRMSG("Too many across-track values!");
  wave->ny = track1 - track0 + 1;
  if (wave->ny > WY)
    ERRMSG("Too many along-track values!");

  /* Loop over footprints... */
  for (itrack = track0; itrack <= track1; itrack++)
    for (ixtrack = xtrack0; ixtrack <= xtrack1; ixtrack++) {

      /* Get distances... */
      if (itrack == track0) {
        wave->x[0] = 0;
        if (ixtrack > xtrack0) {
          geo2cart(0, pert->lon[itrack][ixtrack - 1],
                   pert->lat[itrack][ixtrack - 1], x0);
          geo2cart(0, pert->lon[itrack][ixtrack],
                   pert->lat[itrack][ixtrack], x1);
          wave->x[ixtrack - xtrack0] =
            wave->x[ixtrack - xtrack0 - 1] + DIST(x0, x1);
        }
      }
      if (ixtrack == xtrack0) {
        wave->y[0] = 0;
        if (itrack > track0) {
          geo2cart(0, pert->lon[itrack - 1][ixtrack],
                   pert->lat[itrack - 1][ixtrack], x0);
          geo2cart(0, pert->lon[itrack][ixtrack],
                   pert->lat[itrack][ixtrack], x1);
          wave->y[itrack - track0] =
            wave->y[itrack - track0 - 1] + DIST(x0, x1);
        }
      }

```

```
      /* Save geolocation... */
      wave->time = pert->time[(track0 + track1) / 2][(xtrack0 + xtrack1) / 2];
      wave->z = 0;
      wave->lon[ixtrack - xtrack0][itrack - track0] =
        pert->lon[itrack][ixtrack];
      wave->lat[ixtrack - xtrack0][itrack - track0] =
        pert->lat[itrack][ixtrack];

      /* Save temperature data... */
      wave->temp[ixtrack - xtrack0][itrack - track0]
        = pert->bt[itrack][ixtrack];
      wave->bg[ixtrack - xtrack0][itrack - track0]
        = pert->bt[itrack][ixtrack] - pert->pt[itrack][ixtrack];
      wave->pt[ixtrack - xtrack0][itrack - track0]
        = pert->pt[itrack][ixtrack];
      wave->var[ixtrack - xtrack0][itrack - track0]
        = pert->var[itrack][ixtrack];
    }
}
```

Here is the call graph for this function:

variance()

```
void variance (
            wave_t * wave,
            double dh )
```

Compute local variance.

Definition at line 492 of file libiasi.c.

```
             {

  double dh2, mu, help;

  int dx, dy, ix, ix2, iy, iy2, n;

  /* Check parameters... */
  if (dh <= 0)
    return;

  /* Compute squared radius... */
  dh2 = gsl_pow_2(dh);

  /* Get sampling distances... */
  dx =
    (int) (dh / fabs(wave->x[wave->nx - 1] - wave->x[0]) * (wave->nx - 1.0) +
           1);
  dy =
    (int) (dh / fabs(wave->y[wave->ny - 1] - wave->y[0]) * (wave->ny - 1.0) +
           1);

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

      /* Init... */
      mu = help = 0;
      n = 0;

      /* Get data... */
      for (ix2 = GSL_MAX(ix - dx, 0); ix2 <= GSL_MIN(ix + dx, wave->nx - 1);
```

```
            ix2++)
         for (iy2 = GSL_MAX(iy - dy, 0); iy2 <= GSL_MIN(iy + dy, wave->ny - 1);
              iy2++)
           if ((gsl_pow_2(wave->x[ix] - wave->x[ix2])
                + gsl_pow_2(wave->y[iy] - wave->y[iy2])) <= dh2)
             if (gsl_finite(wave->pt[ix2][iy2])) {
               mu += wave->pt[ix2][iy2];
               help += gsl_pow_2(wave->pt[ix2][iy2]);
               n++;
             }

       /* Compute local variance... */
       if (n > 1)
         wave->var[ix][iy] = help / n - gsl_pow_2(mu / n);
       else
         wave->var[ix][iy] = GSL_NAN;
     }
}
```

wgs84()

```
double wgs84 (
               double lat )
```

Calculate Earth radius according to WGS-84 reference ellipsoid.

Definition at line 546 of file libiasi.c.

```
                  {

  const double a = 6378.1370, b = 6356.7523;

  double cphi, sphi;

  cphi = cos(lat * M_PI / 180.);
  sphi = sin(lat * M_PI / 180.);

  return sqrt((gsl_pow_2(a * a * cphi) + gsl_pow_2(b * b * sphi))
              / (gsl_pow_2(a * cphi) + gsl_pow_2(b * sphi)));
}
```

write_l1()

```
void write_l1 (
               char * filename,
               iasi_l1_t * l1 )
```

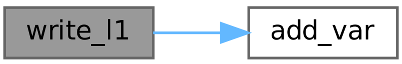
Write IASI Level-1 data.

Definition at line 562 of file libiasi.c.

```
                      {

  int dimid[10], ncid, time_id, lon_id, lat_id,
    sat_z_id, sat_lon_id, sat_lat_id, nu_id, rad_id;

  /* Open or create netCDF file... */
  printf("Write IASI Level-1 file: %s\n", filename);
  if (nc_open(filename, NC_WRITE, &ncid) != NC_NOERR) {
    NC(nc_create(filename, NC_CLOBBER, &ncid));
  } else {
    NC(nc_redef(ncid));
  }

  /* Set dimensions... */
  if (nc_inq_dimid(ncid, "L1_NTRACK", &dimid[0]) != NC_NOERR)
    NC(nc_def_dim(ncid, "L1_NTRACK", l1->ntrack, &dimid[0]));
  if (nc_inq_dimid(ncid, "L1_NXTRACK", &dimid[1]) != NC_NOERR)
    NC(nc_def_dim(ncid, "L1_NXTRACK", L1_NXTRACK, &dimid[1]));
  if (nc_inq_dimid(ncid, "L1_NCHAN", &dimid[2]) != NC_NOERR)
    NC(nc_def_dim(ncid, "L1_NCHAN", L1_NCHAN, &dimid[2]));

  /* Add variables... */
  add_var(ncid, "l1_time", "s", "time (seconds since 2000-01-01T00:00Z)",
          NC_DOUBLE, dimid, &time_id, 2);
```

```
  add_var(ncid, "l1_lon", "deg", "longitude", NC_DOUBLE, dimid, &lon_id, 2);
  add_var(ncid, "l1_lat", "deg", "latitude", NC_DOUBLE, dimid, &lat_id, 2);
  add_var(ncid, "l1_sat_z", "km", "satellite altitude",
          NC_DOUBLE, dimid, &sat_z_id, 1);
  add_var(ncid, "l1_sat_lon", "deg", "(estimated) satellite longitude",
          NC_DOUBLE, dimid, &sat_lon_id, 1);
  add_var(ncid, "l1_sat_lat", "deg", "(estimated) satellite latitude",
          NC_DOUBLE, dimid, &sat_lat_id, 1);
  add_var(ncid, "l1_nu", "cm^-1", "channel wavenumber",
          NC_DOUBLE, &dimid[2], &nu_id, 1);
  add_var(ncid, "l1_rad", "W/(m^2 sr cm^-1)", "channel radiance",
          NC_FLOAT, dimid, &rad_id, 3);

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

  /* Write data... */
  NC(nc_put_var_double(ncid, time_id, l1->time[0]));
  NC(nc_put_var_double(ncid, lon_id, l1->lon[0]));
  NC(nc_put_var_double(ncid, lat_id, l1->lat[0]));
  NC(nc_put_var_double(ncid, sat_z_id, l1->sat_z));
  NC(nc_put_var_double(ncid, sat_lon_id, l1->sat_lon));
  NC(nc_put_var_double(ncid, sat_lat_id, l1->sat_lat));
  NC(nc_put_var_double(ncid, nu_id, l1->nu));
  NC(nc_put_var_float(ncid, rad_id, &l1->rad[0][0][0]));

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

Here is the call graph for this function:



write_l2()

```
void write_l2 (
          char * filename,
          iasi_l2_t * l2 )
```

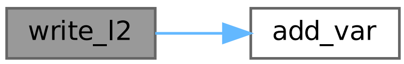
Write IASI Level-2 data.

Definition at line 620 of file libiasi.c.

```
                  {

  int dimid[10], ncid, time_id, z_id, lon_id, lat_id, p_id, t_id;

  /* Create netCDF file... */
  printf("Write IASI Level-2 file: %s\n", filename);
  if (nc_open(filename, NC_WRITE, &ncid) != NC_NOERR) {
    NC(nc_create(filename, NC_CLOBBER, &ncid));
  } else {
    NC(nc_redef(ncid));
  }

  /* Set dimensions... */
  if (nc_inq_dimid(ncid, "L2_NTRACK", &dimid[0]) != NC_NOERR)
    NC(nc_def_dim(ncid, "L2_NTRACK", l2->ntrack, &dimid[0]));
  if (nc_inq_dimid(ncid, "L2_NXTRACK", &dimid[1]) != NC_NOERR)
    NC(nc_def_dim(ncid, "L2_NXTRACK", L2_NXTRACK, &dimid[1]));
  if (nc_inq_dimid(ncid, "L2_NLAY", &dimid[2]) != NC_NOERR)
    NC(nc_def_dim(ncid, "L2_NLAY", L2_NLAY, &dimid[2]));

  /* Add variables... */
```

```
  add_var(ncid, "l2_time", "s", "time (seconds since 2000-01-01T00:00Z)",
          NC_DOUBLE, dimid, &time_id, 2);
  add_var(ncid, "l2_z", "km", "altitude", NC_DOUBLE, dimid, &z_id, 3);
  add_var(ncid, "l2_lon", "deg", "longitude", NC_DOUBLE, dimid, &lon_id, 2);
  add_var(ncid, "l2_lat", "deg", "latitude", NC_DOUBLE, dimid, &lat_id, 2);
  add_var(ncid, "l2_press", "hPa", "pressure",
          NC_DOUBLE, &dimid[2], &p_id, 1);
  add_var(ncid, "l2_temp", "K", "temperature", NC_DOUBLE, dimid, &t_id, 3);

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

  /* Write data... */
  NC(nc_put_var_double(ncid, time_id, l2->time[0]));
  NC(nc_put_var_double(ncid, z_id, l2->z[0][0]));
  NC(nc_put_var_double(ncid, lon_id, l2->lon[0]));
  NC(nc_put_var_double(ncid, lat_id, l2->lat[0]));
  NC(nc_put_var_double(ncid, p_id, l2->p));
  NC(nc_put_var_double(ncid, t_id, l2->t[0][0]));

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

Here is the call graph for this function:



5.12 libiasi.h

Go to the documentation of this file.

```
#include <netcdf.h>
#include <gsl/gsl_randist.h>
#include <gsl/gsl_fft_complex.h>
#include <gsl/gsl_multifit.h>
#include <gsl/gsl_poly.h>
#include <gsl/gsl_sort.h>
#include <gsl/gsl_spline.h>
#include "coda.h"
#include "jurassic.h"

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

#define L1_NCHAN 33

#define L1_NTRACK 1800

#define L1_NXTRACK 60

#define L2_NLAY 27

#define L2_NTRACK 1800

#define L2_NXTRACK 60

#define IASI_L1_NCHAN 8700

#define IASI_NXTRACK 30

#define IASI_PM 4

#define IASI_IDefNsfirst1b 2581

#define IASI_IDefNslast1b 11041

#define IASI_IDefSpectDWn1b 25
```

```

#define PERT_NTRACK 132000

#define PERT_NXTRACK 360

#define WX 300

#define WY 33000

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

#define CODA(cmd) {                                         \
    int coda_result=(cmd);                                  \
    if(coda_result!=0)                                      \
      ERRMSG("%s", coda_errno_to_string(coda_errno));      \
  }

#define NC(cmd) {                                       \
  int nc_result=(cmd);                                  \
  if(nc_result!=NC_NOERR)                               \
    ERRMSG("%s", nc_strerror(nc_result));               \
}

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

typedef struct {

  size_t ntrack;

  double time[L1_NTRACK][L1_NXTRACK];

  double lon[L1_NTRACK][L1_NXTRACK];

  double lat[L1_NTRACK][L1_NXTRACK];

  double sat_z[L1_NTRACK];

  double sat_lon[L1_NTRACK];

  double sat_lat[L1_NTRACK];

  double nu[L1_NCHAN];

  float rad[L1_NTRACK][L1_NXTRACK][L1_NCHAN];

} iasi_l1_t;

typedef struct {

  size_t ntrack;

  double time[L2_NTRACK][L2_NXTRACK];

  double z[L2_NTRACK][L2_NXTRACK][L2_NLAY];

  double lon[L2_NTRACK][L2_NXTRACK];

  double lat[L2_NTRACK][L2_NXTRACK];

  double p[L2_NLAY];

  double t[L2_NTRACK][L2_NXTRACK][L2_NLAY];

} iasi_l2_t;

typedef struct {

  int ntrack;

  int nxtrack;

  double time[PERT_NTRACK][PERT_NXTRACK];

  double lon[PERT_NTRACK][PERT_NXTRACK];

  double lat[PERT_NTRACK][PERT_NXTRACK];

  double dc[PERT_NTRACK][PERT_NXTRACK];

  double bt[PERT_NTRACK][PERT_NXTRACK];

  double pt[PERT_NTRACK][PERT_NXTRACK];

```

```
  double var[PERT_NTRACK][PERT_NXTRACK];

} pert_t;

typedef struct {

  long ntrack;

  float IDefSpectDWn1b[L1_NTRACK];

  int32_t IDefNsfirst1b[L1_NTRACK];

  int32_t IDefNslast1b[L1_NTRACK];

  double Time[L1_NTRACK][IASI_NXTRACK];

  double Loc[L1_NTRACK][IASI_NXTRACK][IASI_PM][2];

  float Wavenumber[IASI_L1_NCHAN];

  short int Radiation[L1_NTRACK][IASI_NXTRACK][IASI_PM][IASI_L1_NCHAN];

  unsigned int Sat_z[L1_NTRACK];

} iasi_raw_t;

typedef struct {

  int ntrack;

  double freq[IASI_L1_NCHAN];

  double Time[L1_NTRACK][L1_NXTRACK];

  double Longitude[L1_NTRACK][L1_NXTRACK];

  double Latitude[L1_NTRACK][L1_NXTRACK];

  float Rad[L1_NTRACK][L1_NXTRACK][IASI_L1_NCHAN];

  double Sat_z[L1_NTRACK];

  double Sat_lon[L1_NTRACK];

  double Sat_lat[L1_NTRACK];

} iasi_rad_t;

typedef struct {

  int nx;

  int ny;

  double time;

  double z;

  double lon[WX][WY];

  double lat[WX][WY];

  double x[WX];

  double y[WY];

  double temp[WX][WY];

  double bg[WX][WY];

  double pt[WX][WY];

  double var[WX][WY];

} wave_t;

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

void add_var(
  int ncid,
  const char *varname,
  const char *unit,
  const char *longname,
  int type,
  int dimid[],
```

```
00292   int *varid,
00293   int ndims);
00294
00296 void background_poly(
00297   wave_t * wave,
00298   int dim_x,
00299   int dim_y);
00300
00302 void background_poly_help(
00303   double *xx,
00304   double *yy,
00305   int n,
00306   int dim);
00307
00309 int get_chan_for_wavenumber(
00310   float wavenumber);
00311
00313 void iasi_read(
00314   char *filename,
00315   iasi_rad_t * iasi_rad);
00316
00318 void noise(
00319   wave_t * wave,
00320   double *mu,
00321   double *sig);
00322
00324 void pert2wave(
00325   pert_t * pert,
00326   wave_t * wave,
00327   int track0,
00328   int track1,
00329   int xtrack0,
00330   int xtrack1);
00331
00333 void variance(
00334   wave_t * wave,
00335   double dh);
00336
00338 double wgs84(
00339   double lat);
00340
00342 void write_l1(
00343   char *filename,
00344   iasi_l1_t * l1);
00345
00347 void write_l2(
00348   char *filename,
00349   iasi_l2_t * l2);
```

5.13 noise.c File Reference

Functions

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

5.13.1 Function Documentation

main()

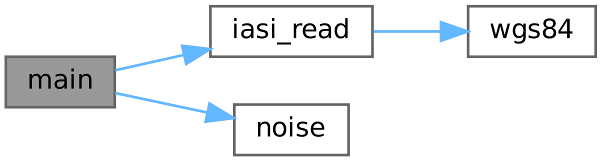
```
int main (
    int argc,
    char * argv[ ] )
```

Definition at line 3 of file noise.c.

```
00005                  {
00006
00007   static iasi_rad_t *iasi_rad;
00008
00009   static wave_t wave;
00010
00011   static FILE *out;
00012
00013   static double mu, nesr, sigma;
```

```
  static int ichan, itrack, ix, iy;

  /* Check arguments... */
  if (argc < 4)
    ERRMSG("Give parameters: <ctl> <iasi_l1_file> <noise.tab>");

  /* Allocate... */
  ALLOC(iasi_rad, iasi_rad_t, 1);

  /* Read IASI data... */
  printf("Read IASI data: %s\n", argv[2]);
  iasi_read(argv[2], iasi_rad);

  /* Create file... */
  printf("Write noise data: %s\n", argv[3]);
  if (!(out = fopen(argv[3], "w")))
    ERRMSG("Cannot create file!");

  /* Write header... */
  fprintf(out,
          "# $1 = track index\n"
          "# $2 = channel index\n"
          "# $3 = wavenumber [1/cm]\n"
          "# $4 = mean BT [K]\n"
          "# $5 = NEDT [K]\n"
          "# $6 = NESR [W/(m^2 sr cm^-1)]\n");

  /* Analyze blocks of data... */
  for (itrack = 0; itrack < iasi_rad->ntrack; itrack += 60) {

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

    /* Loop over channels... */
    for (ichan = 0; ichan < IASI_L1_NCHAN; ichan++) {

      /* Set wave struct... */
      wave.nx = L1_NXTRACK;
      wave.ny = 0;
      for (iy = itrack; iy < GSL_MIN(itrack + 60, iasi_rad->ntrack); iy++) {
        for (ix = 0; ix < wave.nx; ix++)
          wave.temp[ix][wave.ny] = BRIGHT(iasi_rad->Rad[iy][ix][ichan],
                                          iasi_rad->freq[ichan]);
        wave.ny++;
      }

      /* Check number of data points... */
      if (wave.ny >= 55) {

        /* Get noise... */
        noise(&wave, &mu, &sigma);

        /* Get NESR... */
        nesr=PLANCK(mu+sigma, iasi_rad->freq[ichan])
          -PLANCK(mu, iasi_rad->freq[ichan]);

        /* Write output... */
        if (gsl_finite(sigma))
          fprintf(out, "%d %d %.4f %g %g %g\n", itrack, ichan,
                  iasi_rad->freq[ichan], mu, sigma, nesr);
      }
    }
  }

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

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

  return EXIT_SUCCESS;
}
```

Here is the call graph for this function:



5.14 noise.c

Go to the documentation of this file.

```
#include "libiasi.h"

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

  static iasi_rad_t *iasi_rad;

  static wave_t wave;

  static FILE *out;

  static double mu, nesr, sigma;

  static int ichan, itrack, ix, iy;

  /* Check arguments... */
  if (argc < 4)
    ERRMSG("Give parameters: <ctl> <iasi_l1_file> <noise.tab>");

  /* Allocate... */
  ALLOC(iasi_rad, iasi_rad_t, 1);

  /* Read IASI data... */
  printf("Read IASI data: %s\n", argv[2]);
  iasi_read(argv[2], iasi_rad);

  /* Create file... */
  printf("Write noise data: %s\n", argv[3]);
  if (!(out = fopen(argv[3], "w")))
    ERRMSG("Cannot create file!");

  /* Write header... */
  fprintf(out,
          "# $1 = track index\n"
          "# $2 = channel index\n"
          "# $3 = wavenumber [1/cm]\n"
          "# $4 = mean BT [K]\n"
          "# $5 = NEDT [K]\n"
          "# $6 = NESR [W/(m^2 sr cm^-1)]\n");

  /* Analyze blocks of data... */
  for (itrack = 0; itrack < iasi_rad->ntrack; itrack += 60) {

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

    /* Loop over channels... */
    for (ichan = 0; ichan < IASI_L1_NCHAN; ichan++) {

      /* Set wave struct... */
      wave.nx = L1_NXTRACK;
      wave.ny = 0;
      for (iy = itrack; iy < GSL_MIN(itrack + 60, iasi_rad->ntrack); iy++) {
        for (ix = 0; ix < wave.nx; ix++)
          wave.temp[ix][wave.ny] = BRIGHT(iasi_rad->Rad[iy][ix][ichan],
                                          iasi_rad->freq[ichan]);
```

```
        wave.ny++;
      }

      /* Check number of data points... */
      if (wave.ny >= 55) {

        /* Get noise... */
        noise(&wave, &mu, &sigma);

        /* Get NESR... */
        nesr=PLANCK(mu+sigma, iasi_rad->freq[ichan])
          -PLANCK(mu, iasi_rad->freq[ichan]);

        /* Write output... */
        if (gsl_finite(sigma))
          fprintf(out, "%d %d %.4f %g %g %g\n", itrack, ichan,
                  iasi_rad->freq[ichan], mu, sigma, nesr);
      }
    }
  }

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

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

  return EXIT_SUCCESS;
}
```

5.15 perturbation.c File Reference

Functions

- void addatt (int ncid, int varid, const char *unit, const char *long_name)
- int main (int argc, char *argv[ ])

5.15.1 Function Documentation

addatt()

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

Definition at line 384 of file perturbation.c.

```
                          {

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

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

main()

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

Definition at line 31 of file perturbation.c.

```
                 {
```

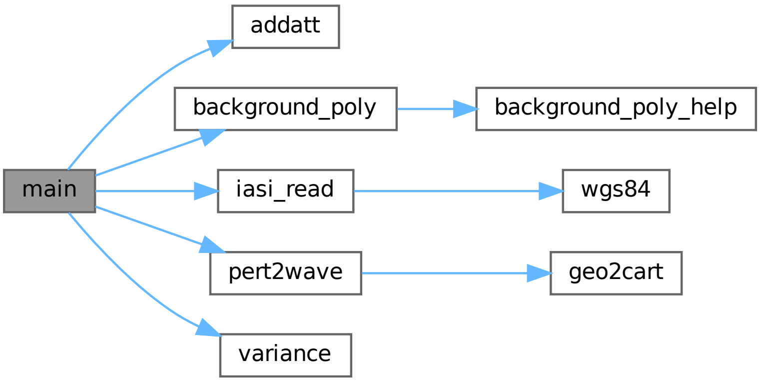
```

static iasi_rad_t *iasi_rad;

static pert_t *pert_4mu, *pert_15mu_low, *pert_15mu_high;

static wave_t wave;

static double numean, radmean, var_dh = 100.;

static int list_4mu[N4]
  = { 6711, 6712, 6713, 6714, 6715, 6716, 6717, 6718, 6719, 6720,
  6721, 6722, 6723, 6724, 6725, 6726, 6727, 6728, 6729, 6730, 6731,
  6732, 6733, 6734, 6735, 6736, 6737, 6738, 6739, 6740, 6741, 6742,
  6743, 6744, 6745, 6746, 6747, 6748, 6749, 6750, 6751, 6752, 6753,
  6754, 6755, 6756, 6757, 6758, 6759, 6760, 6761, 6762, 6763, 6764,
  6765, 6766, 6767, 6768, 6769, 6770, 6771, 6772, 6773, 6774, 6775,
  6776, 6777, 6778, 6779, 6780, 6781, 6782, 6783, 6784, 6785, 6786,
  6787, 6788, 6789, 6790, 6791, 6792, 6793, 6794, 6795, 6796, 6797,
  6798, 6799, 6800, 6801, 6802, 6803, 6804, 6830, 6831, 6832, 6833,
  6834, 6835, 6836, 6837, 6838, 6839, 6840, 6841, 6842, 6843, 6844,
  6845, 6846, 6847, 6848, 6849, 6850, 6851, 6852, 6853, 6854, 6855,
  6856, 6857, 6858, 6859, 6860, 6861, 6862, 6863, 6864, 6865, 6866,
  6867, 6868, 6869, 6870, 6871, 6872, 6873, 6874, 6875, 6876, 6877,
  6878, 6879, 6880, 6881, 6882, 6883, 6884, 6885, 6886, 6887
};

static int list_15mu_low[N15_LOW]
  = { 22, 28, 34, 40, 46, 52, 58, 72, 100, 105, 112, 118, 119,
  124, 125, 130, 131, 136, 137, 143, 144
};

static int list_15mu_high[N15_HIGH]
= { 91, 92 };

static int ix, iy, dimid[2], i, n, ncid, track, track0, xtrack,
  time_varid, lon_varid, lat_varid, bt_4mu_varid, bt_4mu_pt_varid,
  bt_4mu_var_varid, bt_8mu_varid, bt_15mu_low_varid, bt_15mu_low_pt_varid,
  bt_15mu_low_var_varid, bt_15mu_high_varid, bt_15mu_high_pt_varid,
  bt_15mu_high_var_varid, iarg;

static size_t start[2], count[2];

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

/* Allocate... */
ALLOC(iasi_rad, iasi_rad_t, 1);
ALLOC(pert_4mu, pert_t, 1);
ALLOC(pert_15mu_low, pert_t, 1);
ALLOC(pert_15mu_high, pert_t, 1);

/* ------------------------------------------------------------
   Read HDF files...
   ------------------------------------------------------------ */

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

  /* Read IASI data... */
  printf("Read IASI Level-1C data file: %s\n", argv[iarg]);
  iasi_read(argv[iarg], iasi_rad);

  /* Save geolocation... */
  pert_4mu->ntrack += iasi_rad->ntrack;
  if (pert_4mu->ntrack > PERT_NTRACK)
    ERRMSG("Too many granules!");
  pert_4mu->nxtrack = L1_NXTRACK;
  if (pert_4mu->nxtrack > PERT_NXTRACK)
    ERRMSG("Too many tracks!");
  for (track = 0; track < iasi_rad->ntrack; track++)
    for (xtrack = 0; xtrack < L1_NXTRACK; xtrack++) {
      pert_4mu->time[track0 + track][xtrack]
        = iasi_rad->Time[track][xtrack];
      pert_4mu->lon[track0 + track][xtrack]
        = iasi_rad->Longitude[track][xtrack];
      pert_4mu->lat[track0 + track][xtrack]
        = iasi_rad->Latitude[track][xtrack];
    }

  pert_15mu_low->ntrack += iasi_rad->ntrack;
  if (pert_15mu_low->ntrack > PERT_NTRACK)
    ERRMSG("Too many granules!");
  pert_15mu_low->nxtrack = L1_NXTRACK;
  if (pert_15mu_low->nxtrack > PERT_NXTRACK)
    ERRMSG("Too many tracks!");
  for (track = 0; track < iasi_rad->ntrack; track++)
```

```
      for (xtrack = 0; xtrack < L1_NXTRACK; xtrack++) {
        pert_15mu_low->time[track0 + track][xtrack]
          = iasi_rad->Time[track][xtrack];
        pert_15mu_low->lon[track0 + track][xtrack]
          = iasi_rad->Longitude[track][xtrack];
        pert_15mu_low->lat[track0 + track][xtrack]
          = iasi_rad->Latitude[track][xtrack];
      }

    pert_15mu_high->ntrack += iasi_rad->ntrack;
    if (pert_15mu_high->ntrack > PERT_NTRACK)
      ERRMSG("Too many granules!");
    pert_15mu_high->nxtrack = L1_NXTRACK;
    if (pert_15mu_high->nxtrack > PERT_NXTRACK)
      ERRMSG("Too many tracks!");
    for (track = 0; track < iasi_rad->ntrack; track++)
      for (xtrack = 0; xtrack < L1_NXTRACK; xtrack++) {
        pert_15mu_high->time[track0 + track][xtrack]
          = iasi_rad->Time[track][xtrack];
        pert_15mu_high->lon[track0 + track][xtrack]
          = iasi_rad->Longitude[track][xtrack];
        pert_15mu_high->lat[track0 + track][xtrack]
          = iasi_rad->Latitude[track][xtrack];
      }

    /* Get 8.1 micron brightness temperature... */
    for (track = 0; track < iasi_rad->ntrack; track++)
      for (xtrack = 0; xtrack < L1_NXTRACK; xtrack++)
        pert_4mu->dc[track0 + track][xtrack]
          = BRIGHT(iasi_rad->Rad[track][xtrack][2345],
                   iasi_rad->freq[2345]);

    /* Get 4.3 micron brightness temperature... */
    for (track = 0; track < iasi_rad->ntrack; track++)
      for (xtrack = 0; xtrack < L1_NXTRACK; xtrack++) {
        n = 0;
        numean = radmean = 0;
        for (i = 0; i < N4; i++)
          if (gsl_finite(iasi_rad->Rad[track][xtrack][list_4mu[i]])) {
            radmean += iasi_rad->Rad[track][xtrack][list_4mu[i]];
            numean += iasi_rad->freq[list_4mu[i]];
            n++;
          }
        if (n > 0.9 * N4)
          pert_4mu->bt[track0 + track][xtrack]
            = BRIGHT(radmean / n, numean / n);
        else
          pert_4mu->bt[track0 + track][xtrack] = GSL_NAN;
      }

    /* Get 15 micron brightness temperature (low altitudes)... */
    for (track = 0; track < iasi_rad->ntrack; track++)
      for (xtrack = 0; xtrack < L1_NXTRACK; xtrack++) {
        n = 0;
        numean = radmean = 0;
        for (i = 0; i < N15_LOW; i++)
          if (gsl_finite(iasi_rad->Rad[track][xtrack][list_15mu_low[i]])) {
            radmean += iasi_rad->Rad[track][xtrack][list_15mu_low[i]];
            numean += iasi_rad->freq[list_15mu_low[i]];
            n++;
          }
        if (n > 0.9 * N15_LOW)
          pert_15mu_low->bt[track0 + track][xtrack]
            = BRIGHT(radmean / n, numean / n);
        else
          pert_15mu_low->bt[track0 + track][xtrack] = GSL_NAN;
      }

    /* Get 15 micron brightness temperature (high altitudes)... */
    for (track = 0; track < iasi_rad->ntrack; track++)
      for (xtrack = 0; xtrack < L1_NXTRACK; xtrack++) {
        n = 0;
        numean = radmean = 0;
        for (i = 0; i < N15_HIGH; i++)
          if (gsl_finite(iasi_rad->Rad[track][xtrack][list_15mu_high[i]])) {
            radmean += iasi_rad->Rad[track][xtrack][list_15mu_high[i]];
            numean += iasi_rad->freq[list_15mu_high[i]];
            n++;
          }
        if (n > 0.9 * N15_HIGH)
          pert_15mu_high->bt[track0 + track][xtrack]
            = BRIGHT(radmean / n, numean / n);
        else
          pert_15mu_high->bt[track0 + track][xtrack] = GSL_NAN;
      }

    /* Increment track counter... */
```

```
    track0 += iasi_rad->ntrack;
  }

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

  /* Convert to wave analysis struct... */
  pert2wave(pert_4mu, &wave,
            0, pert_4mu->ntrack - 1, 0, pert_4mu->nxtrack - 1);

  /* Estimate background... */
  background_poly(&wave, 5, 0);

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

  /* Copy data... */
  for (ix = 0; ix < wave.nx; ix++)
    for (iy = 0; iy < wave.ny; iy++) {
      pert_4mu->pt[iy][ix] = wave.pt[ix][iy];
      pert_4mu->var[iy][ix] = wave.var[ix][iy];
    }

  /* Convert to wave analysis struct... */
  pert2wave(pert_15mu_low, &wave,
            0, pert_15mu_low->ntrack - 1, 0, pert_15mu_low->nxtrack - 1);

  /* Estimate background... */
  background_poly(&wave, 5, 0);

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

  /* Copy data... */
  for (ix = 0; ix < wave.nx; ix++)
    for (iy = 0; iy < wave.ny; iy++) {
      pert_15mu_low->pt[iy][ix] = wave.pt[ix][iy];
      pert_15mu_low->var[iy][ix] = wave.var[ix][iy];
    }

  /* Convert to wave analysis struct... */
  pert2wave(pert_15mu_high, &wave,
            0, pert_15mu_high->ntrack - 1, 0, pert_15mu_high->nxtrack - 1);

  /* Estimate background... */
  background_poly(&wave, 5, 0);

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

  /* Copy data... */
  for (ix = 0; ix < wave.nx; ix++)
    for (iy = 0; iy < wave.ny; iy++) {
      pert_15mu_high->pt[iy][ix] = wave.pt[ix][iy];
      pert_15mu_high->var[iy][ix] = wave.var[ix][iy];
    }

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

  /* Create netCDF file... */
  printf("Write perturbation data file: %s\n", argv[1]);
  NC(nc_create(argv[1], NC_CLOBBER, &ncid));

  /* Set dimensions... */
  NC(nc_def_dim(ncid, "NTRACK", NC_UNLIMITED, &dimid[0]));
  NC(nc_def_dim(ncid, "NXTRACK", L1_NXTRACK, &dimid[1]));

  /* Add variables... */
  NC(nc_def_var(ncid, "time", NC_DOUBLE, 2, dimid, &time_varid));
  addatt(ncid, time_varid, "s", "time (seconds since 2000-01-01T00:00Z)");
  NC(nc_def_var(ncid, "lon", NC_DOUBLE, 2, dimid, &lon_varid));
  addatt(ncid, lon_varid, "deg", "footprint longitude");
  NC(nc_def_var(ncid, "lat", NC_DOUBLE, 2, dimid, &lat_varid));
  addatt(ncid, lat_varid, "deg", "footprint latitude");

  NC(nc_def_var(ncid, "bt_8mu", NC_FLOAT, 2, dimid, &bt_8mu_varid));
  addatt(ncid, bt_8mu_varid, "K", "brightness temperature at 8.1 micron");

  NC(nc_def_var(ncid, "bt_4mu", NC_FLOAT, 2, dimid, &bt_4mu_varid));
  addatt(ncid, bt_4mu_varid, "K", "brightness temperature" " at 4.3 micron");
  NC(nc_def_var(ncid, "bt_4mu_pt", NC_FLOAT, 2, dimid, &bt_4mu_pt_varid));
  addatt(ncid, bt_4mu_pt_varid, "K", "brightness temperature perturbation"
         " at 4.3 micron");
  NC(nc_def_var(ncid, "bt_4mu_var", NC_FLOAT, 2, dimid, &bt_4mu_var_varid));
```

```
  addatt(ncid, bt_4mu_var_varid, "K^2", "brightness temperature variance"
         " at 4.3 micron");

  NC(nc_def_var(ncid, "bt_15mu_low", NC_FLOAT, 2, dimid, &bt_15mu_low_varid));
  addatt(ncid, bt_15mu_low_varid, "K", "brightness temperature"
         " at 15 micron (low altitudes)");
  NC(nc_def_var(ncid, "bt_15mu_low_pt", NC_FLOAT, 2, dimid,
                &bt_15mu_low_pt_varid));
  addatt(ncid, bt_15mu_low_pt_varid, "K",
         "brightness temperature perturbation"
         " at 15 micron (low altitudes)");
  NC(nc_def_var
     (ncid, "bt_15mu_low_var", NC_FLOAT, 2, dimid, &bt_15mu_low_var_varid));
  addatt(ncid, bt_15mu_low_var_varid, "K^2",
         "brightness temperature variance" " at 15 micron (low altitudes)");

  NC(nc_def_var(ncid, "bt_15mu_high", NC_FLOAT, 2, dimid,
                &bt_15mu_high_varid));
  addatt(ncid, bt_15mu_high_varid, "K", "brightness temperature"
         " at 15 micron (high altitudes)");
  NC(nc_def_var(ncid, "bt_15mu_high_pt", NC_FLOAT, 2, dimid,
                &bt_15mu_high_pt_varid));
  addatt(ncid, bt_15mu_high_pt_varid, "K",
         "brightness temperature perturbation"
         " at 15 micron (high altitudes)");
  NC(nc_def_var
     (ncid, "bt_15mu_high_var", NC_FLOAT, 2, dimid, &bt_15mu_high_var_varid));
  addatt(ncid, bt_15mu_high_var_varid, "K^2",
         "brightness temperature variance" " at 15 micron (high altitudes)");

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

  /* Loop over tracks... */
  for (track = 0; track < pert_4mu->ntrack; track++) {

    /* Set array sizes... */
    start[0] = (size_t) track;
    start[1] = 0;
    count[0] = 1;
    count[1] = (size_t) pert_4mu->nxtrack;

    /* Write data... */
    NC(nc_put_vara_double(ncid, time_varid, start, count,
                          pert_4mu->time[track]));
    NC(nc_put_vara_double(ncid, lon_varid, start, count,
                          pert_4mu->lon[track]));
    NC(nc_put_vara_double(ncid, lat_varid, start, count,
                          pert_4mu->lat[track]));

    NC(nc_put_vara_double(ncid, bt_8mu_varid, start, count,
                          pert_4mu->dc[track]));

    NC(nc_put_vara_double(ncid, bt_4mu_varid, start, count,
                          pert_4mu->bt[track]));
    NC(nc_put_vara_double(ncid, bt_4mu_pt_varid, start, count,
                          pert_4mu->pt[track]));
    NC(nc_put_vara_double(ncid, bt_4mu_var_varid, start, count,
                          pert_4mu->var[track]));

    NC(nc_put_vara_double(ncid, bt_15mu_low_varid, start, count,
                          pert_15mu_low->bt[track]));
    NC(nc_put_vara_double(ncid, bt_15mu_low_pt_varid, start, count,
                          pert_15mu_low->pt[track]));
    NC(nc_put_vara_double(ncid, bt_15mu_low_var_varid, start, count,
                          pert_15mu_low->var[track]));

    NC(nc_put_vara_double(ncid, bt_15mu_high_varid, start, count,
                          pert_15mu_high->bt[track]));
    NC(nc_put_vara_double(ncid, bt_15mu_high_pt_varid, start, count,
                          pert_15mu_high->pt[track]));
    NC(nc_put_vara_double(ncid, bt_15mu_high_var_varid, start, count,
                          pert_15mu_high->var[track]));
  }

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

  /* Free... */
  free(iasi_rad);
  free(pert_4mu);
  free(pert_15mu_low);
  free(pert_15mu_high);

  return EXIT_SUCCESS;
}
```

Here is the call graph for this function:



5.16 perturbation.c

Go to the documentation of this file.

```
#include "libiasi.h"

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

/* Number of 4 micron channels: */
#define N4 152

/* Number of 15 micron channels (low altitudes): */
#define N15_LOW 21

/* Number of 15 micron channels (high altitudes): */
#define N15_HIGH 2

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

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

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

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

  static iasi_rad_t *iasi_rad;

  static pert_t *pert_4mu, *pert_15mu_low, *pert_15mu_high;

  static wave_t wave;

  static double numean, radmean, var_dh = 100.;

  static int list_4mu[N4]
    = { 6711, 6712, 6713, 6714, 6715, 6716, 6717, 6718, 6719, 6720,
    6721, 6722, 6723, 6724, 6725, 6726, 6727, 6728, 6729, 6730, 6731,
    6732, 6733, 6734, 6735, 6736, 6737, 6738, 6739, 6740, 6741, 6742,
    6743, 6744, 6745, 6746, 6747, 6748, 6749, 6750, 6751, 6752, 6753,
    6754, 6755, 6756, 6757, 6758, 6759, 6760, 6761, 6762, 6763, 6764,
```

```
    6765, 6766, 6767, 6768, 6769, 6770, 6771, 6772, 6773, 6774, 6775,
    6776, 6777, 6778, 6779, 6780, 6781, 6782, 6783, 6784, 6785, 6786,
    6787, 6788, 6789, 6790, 6791, 6792, 6793, 6794, 6795, 6796, 6797,
    6798, 6799, 6800, 6801, 6802, 6803, 6804, 6830, 6831, 6832, 6833,
    6834, 6835, 6836, 6837, 6838, 6839, 6840, 6841, 6842, 6843, 6844,
    6845, 6846, 6847, 6848, 6849, 6850, 6851, 6852, 6853, 6854, 6855,
    6856, 6857, 6858, 6859, 6860, 6861, 6862, 6863, 6864, 6865, 6866,
    6867, 6868, 6869, 6870, 6871, 6872, 6873, 6874, 6875, 6876, 6877,
    6878, 6879, 6880, 6881, 6882, 6883, 6884, 6885, 6886, 6887
  };

  static int list_15mu_low[N15_LOW]
    = { 22, 28, 34, 40, 46, 52, 58, 72, 100, 105, 112, 118, 119,
    124, 125, 130, 131, 136, 137, 143, 144
  };

  static int list_15mu_high[N15_HIGH]
  = { 91, 92 };

  static int ix, iy, dimid[2], i, n, ncid, track, track0, xtrack,
    time_varid, lon_varid, lat_varid, bt_4mu_varid, bt_4mu_pt_varid,
    bt_4mu_var_varid, bt_8mu_varid, bt_15mu_low_varid, bt_15mu_low_pt_varid,
    bt_15mu_low_var_varid, bt_15mu_high_varid, bt_15mu_high_pt_varid,
    bt_15mu_high_var_varid, iarg;

  static size_t start[2], count[2];

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

  /* Allocate... */
  ALLOC(iasi_rad, iasi_rad_t, 1);
  ALLOC(pert_4mu, pert_t, 1);
  ALLOC(pert_15mu_low, pert_t, 1);
  ALLOC(pert_15mu_high, pert_t, 1);

  /* ------------------------------------------------------------
     Read HDF files...
     ------------------------------------------------------------ */

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

    /* Read IASI data... */
    printf("Read IASI Level-1C data file: %s\n", argv[iarg]);
    iasi_read(argv[iarg], iasi_rad);

    /* Save geolocation... */
    pert_4mu->ntrack += iasi_rad->ntrack;
    if (pert_4mu->ntrack > PERT_NTRACK)
      ERRMSG("Too many granules!");
    pert_4mu->nxtrack = L1_NXTRACK;
    if (pert_4mu->nxtrack > PERT_NXTRACK)
      ERRMSG("Too many tracks!");
    for (track = 0; track < iasi_rad->ntrack; track++)
      for (xtrack = 0; xtrack < L1_NXTRACK; xtrack++) {
        pert_4mu->time[track0 + track][xtrack]
          = iasi_rad->Time[track][xtrack];
        pert_4mu->lon[track0 + track][xtrack]
          = iasi_rad->Longitude[track][xtrack];
        pert_4mu->lat[track0 + track][xtrack]
          = iasi_rad->Latitude[track][xtrack];
      }

    pert_15mu_low->ntrack += iasi_rad->ntrack;
    if (pert_15mu_low->ntrack > PERT_NTRACK)
      ERRMSG("Too many granules!");
    pert_15mu_low->nxtrack = L1_NXTRACK;
    if (pert_15mu_low->nxtrack > PERT_NXTRACK)
      ERRMSG("Too many tracks!");
    for (track = 0; track < iasi_rad->ntrack; track++)
      for (xtrack = 0; xtrack < L1_NXTRACK; xtrack++) {
        pert_15mu_low->time[track0 + track][xtrack]
          = iasi_rad->Time[track][xtrack];
        pert_15mu_low->lon[track0 + track][xtrack]
          = iasi_rad->Longitude[track][xtrack];
        pert_15mu_low->lat[track0 + track][xtrack]
          = iasi_rad->Latitude[track][xtrack];
      }

    pert_15mu_high->ntrack += iasi_rad->ntrack;
    if (pert_15mu_high->ntrack > PERT_NTRACK)
      ERRMSG("Too many granules!");
    pert_15mu_high->nxtrack = L1_NXTRACK;
    if (pert_15mu_high->nxtrack > PERT_NXTRACK)
      ERRMSG("Too many tracks!");
```

```
  for (track = 0; track < iasi_rad->ntrack; track++)
    for (xtrack = 0; xtrack < L1_NXTRACK; xtrack++) {
      pert_15mu_high->time[track0 + track][xtrack]
        = iasi_rad->Time[track][xtrack];
      pert_15mu_high->lon[track0 + track][xtrack]
        = iasi_rad->Longitude[track][xtrack];
      pert_15mu_high->lat[track0 + track][xtrack]
        = iasi_rad->Latitude[track][xtrack];
    }

  /* Get 8.1 micron brightness temperature... */
  for (track = 0; track < iasi_rad->ntrack; track++)
    for (xtrack = 0; xtrack < L1_NXTRACK; xtrack++)
      pert_4mu->dc[track0 + track][xtrack]
        = BRIGHT(iasi_rad->Rad[track][xtrack][2345],
                 iasi_rad->freq[2345]);

  /* Get 4.3 micron brightness temperature... */
  for (track = 0; track < iasi_rad->ntrack; track++)
    for (xtrack = 0; xtrack < L1_NXTRACK; xtrack++) {
      n = 0;
      numean = radmean = 0;
      for (i = 0; i < N4; i++)
        if (gsl_finite(iasi_rad->Rad[track][xtrack][list_4mu[i]])) {
          radmean += iasi_rad->Rad[track][xtrack][list_4mu[i]];
          numean += iasi_rad->freq[list_4mu[i]];
          n++;
        }
      if (n > 0.9 * N4)
        pert_4mu->bt[track0 + track][xtrack]
          = BRIGHT(radmean / n, numean / n);
      else
        pert_4mu->bt[track0 + track][xtrack] = GSL_NAN;
    }

  /* Get 15 micron brightness temperature (low altitudes)... */
  for (track = 0; track < iasi_rad->ntrack; track++)
    for (xtrack = 0; xtrack < L1_NXTRACK; xtrack++) {
      n = 0;
      numean = radmean = 0;
      for (i = 0; i < N15_LOW; i++)
        if (gsl_finite(iasi_rad->Rad[track][xtrack][list_15mu_low[i]])) {
          radmean += iasi_rad->Rad[track][xtrack][list_15mu_low[i]];
          numean += iasi_rad->freq[list_15mu_low[i]];
          n++;
        }
      if (n > 0.9 * N15_LOW)
        pert_15mu_low->bt[track0 + track][xtrack]
          = BRIGHT(radmean / n, numean / n);
      else
        pert_15mu_low->bt[track0 + track][xtrack] = GSL_NAN;
    }

  /* Get 15 micron brightness temperature (high altitudes)... */
  for (track = 0; track < iasi_rad->ntrack; track++)
    for (xtrack = 0; xtrack < L1_NXTRACK; xtrack++) {
      n = 0;
      numean = radmean = 0;
      for (i = 0; i < N15_HIGH; i++)
        if (gsl_finite(iasi_rad->Rad[track][xtrack][list_15mu_high[i]])) {
          radmean += iasi_rad->Rad[track][xtrack][list_15mu_high[i]];
          numean += iasi_rad->freq[list_15mu_high[i]];
          n++;
        }
      if (n > 0.9 * N15_HIGH)
        pert_15mu_high->bt[track0 + track][xtrack]
          = BRIGHT(radmean / n, numean / n);
      else
        pert_15mu_high->bt[track0 + track][xtrack] = GSL_NAN;
    }

  /* Increment track counter... */
  track0 += iasi_rad->ntrack;
}

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

/* Convert to wave analysis struct... */
pert2wave(pert_4mu, &wave,
          0, pert_4mu->ntrack - 1, 0, pert_4mu->nxtrack - 1);

/* Estimate background... */
background_poly(&wave, 5, 0);

/* Compute variance... */
```

```
  variance(&wave, var_dh);

  /* Copy data... */
  for (ix = 0; ix < wave.nx; ix++)
    for (iy = 0; iy < wave.ny; iy++) {
      pert_4mu->pt[iy][ix] = wave.pt[ix][iy];
      pert_4mu->var[iy][ix] = wave.var[ix][iy];
    }

  /* Convert to wave analysis struct... */
  pert2wave(pert_15mu_low, &wave,
            0, pert_15mu_low->ntrack - 1, 0, pert_15mu_low->nxtrack - 1);

  /* Estimate background... */
  background_poly(&wave, 5, 0);

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

  /* Copy data... */
  for (ix = 0; ix < wave.nx; ix++)
    for (iy = 0; iy < wave.ny; iy++) {
      pert_15mu_low->pt[iy][ix] = wave.pt[ix][iy];
      pert_15mu_low->var[iy][ix] = wave.var[ix][iy];
    }

  /* Convert to wave analysis struct... */
  pert2wave(pert_15mu_high, &wave,
            0, pert_15mu_high->ntrack - 1, 0, pert_15mu_high->nxtrack - 1);

  /* Estimate background... */
  background_poly(&wave, 5, 0);

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

  /* Copy data... */
  for (ix = 0; ix < wave.nx; ix++)
    for (iy = 0; iy < wave.ny; iy++) {
      pert_15mu_high->pt[iy][ix] = wave.pt[ix][iy];
      pert_15mu_high->var[iy][ix] = wave.var[ix][iy];
    }

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

  /* Create netCDF file... */
  printf("Write perturbation data file: %s\n", argv[1]);
  NC(nc_create(argv[1], NC_CLOBBER, &ncid));

  /* Set dimensions... */
  NC(nc_def_dim(ncid, "NTRACK", NC_UNLIMITED, &dimid[0]));
  NC(nc_def_dim(ncid, "NXTRACK", L1_NXTRACK, &dimid[1]));

  /* Add variables... */
  NC(nc_def_var(ncid, "time", NC_DOUBLE, 2, dimid, &time_varid));
  addatt(ncid, time_varid, "s", "time (seconds since 2000-01-01T00:00Z)");
  NC(nc_def_var(ncid, "lon", NC_DOUBLE, 2, dimid, &lon_varid));
  addatt(ncid, lon_varid, "deg", "footprint longitude");
  NC(nc_def_var(ncid, "lat", NC_DOUBLE, 2, dimid, &lat_varid));
  addatt(ncid, lat_varid, "deg", "footprint latitude");

  NC(nc_def_var(ncid, "bt_8mu", NC_FLOAT, 2, dimid, &bt_8mu_varid));
  addatt(ncid, bt_8mu_varid, "K", "brightness temperature at 8.1 micron");

  NC(nc_def_var(ncid, "bt_4mu", NC_FLOAT, 2, dimid, &bt_4mu_varid));
  addatt(ncid, bt_4mu_varid, "K", "brightness temperature" " at 4.3 micron");
  NC(nc_def_var(ncid, "bt_4mu_pt", NC_FLOAT, 2, dimid, &bt_4mu_pt_varid));
  addatt(ncid, bt_4mu_pt_varid, "K", "brightness temperature perturbation"
         " at 4.3 micron");
  NC(nc_def_var(ncid, "bt_4mu_var", NC_FLOAT, 2, dimid, &bt_4mu_var_varid));
  addatt(ncid, bt_4mu_var_varid, "K^2", "brightness temperature variance"
         " at 4.3 micron");

  NC(nc_def_var(ncid, "bt_15mu_low", NC_FLOAT, 2, dimid, &bt_15mu_low_varid));
  addatt(ncid, bt_15mu_low_varid, "K", "brightness temperature"
         " at 15 micron (low altitudes)");
  NC(nc_def_var(ncid, "bt_15mu_low_pt", NC_FLOAT, 2, dimid,
                &bt_15mu_low_pt_varid));
  addatt(ncid, bt_15mu_low_pt_varid, "K",
         "brightness temperature perturbation"
         " at 15 micron (low altitudes)");
  NC(nc_def_var
     (ncid, "bt_15mu_low_var", NC_FLOAT, 2, dimid, &bt_15mu_low_var_varid));
  addatt(ncid, bt_15mu_low_var_varid, "K^2",
         "brightness temperature variance" " at 15 micron (low altitudes)");
```

```
  NC(nc_def_var(ncid, "bt_15mu_high", NC_FLOAT, 2, dimid,
                &bt_15mu_high_varid));
  addatt(ncid, bt_15mu_high_varid, "K", "brightness temperature"
         " at 15 micron (high altitudes)");
  NC(nc_def_var(ncid, "bt_15mu_high_pt", NC_FLOAT, 2, dimid,
                &bt_15mu_high_pt_varid));
  addatt(ncid, bt_15mu_high_pt_varid, "K",
         "brightness temperature perturbation"
         " at 15 micron (high altitudes)");
  NC(nc_def_var
     (ncid, "bt_15mu_high_var", NC_FLOAT, 2, dimid, &bt_15mu_high_var_varid));
  addatt(ncid, bt_15mu_high_var_varid, "K^2",
         "brightness temperature variance" " at 15 micron (high altitudes)");

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

  /* Loop over tracks... */
  for (track = 0; track < pert_4mu->ntrack; track++) {

    /* Set array sizes... */
    start[0] = (size_t) track;
    start[1] = 0;
    count[0] = 1;
    count[1] = (size_t) pert_4mu->nxtrack;

    /* Write data... */
    NC(nc_put_vara_double(ncid, time_varid, start, count,
                          pert_4mu->time[track]));
    NC(nc_put_vara_double(ncid, lon_varid, start, count,
                          pert_4mu->lon[track]));
    NC(nc_put_vara_double(ncid, lat_varid, start, count,
                          pert_4mu->lat[track]));

    NC(nc_put_vara_double(ncid, bt_8mu_varid, start, count,
                          pert_4mu->dc[track]));

    NC(nc_put_vara_double(ncid, bt_4mu_varid, start, count,
                          pert_4mu->bt[track]));
    NC(nc_put_vara_double(ncid, bt_4mu_pt_varid, start, count,
                          pert_4mu->pt[track]));
    NC(nc_put_vara_double(ncid, bt_4mu_var_varid, start, count,
                          pert_4mu->var[track]));

    NC(nc_put_vara_double(ncid, bt_15mu_low_varid, start, count,
                          pert_15mu_low->bt[track]));
    NC(nc_put_vara_double(ncid, bt_15mu_low_pt_varid, start, count,
                          pert_15mu_low->pt[track]));
    NC(nc_put_vara_double(ncid, bt_15mu_low_var_varid, start, count,
                          pert_15mu_low->var[track]));

    NC(nc_put_vara_double(ncid, bt_15mu_high_varid, start, count,
                          pert_15mu_high->bt[track]));
    NC(nc_put_vara_double(ncid, bt_15mu_high_pt_varid, start, count,
                          pert_15mu_high->pt[track]));
    NC(nc_put_vara_double(ncid, bt_15mu_high_var_varid, start, count,
                          pert_15mu_high->var[track]));
  }

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

  /* Free... */
  free(iasi_rad);
  free(pert_4mu);
  free(pert_15mu_low);
  free(pert_15mu_high);

  return EXIT_SUCCESS;
}

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

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

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

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

5.17 retrieval.c File Reference

Data Structures

- struct ncd_t

  Buffer for netCDF data.
- struct ret_t

  Retrieval control parameters.

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 init_l2 (ncd_t *ncd, int track, int xtrack, ctl_t *ctl, atm_t *atm)

  Initialize with IASI 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^TBA 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.
- void write_nc (char *filename, ncd_t *ncd)

  Write to netCDF file...
- int main (int argc, char *argv[ ])

5.17.1 Function Documentation

add_var()

```
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 481 of file retrieval.c.

```
                {

  /* Check if variable exists... */
  if (nc_inq_varid(ncid, varname, varid) != NC_NOERR) {

    /* Define variable... */
    NC(nc_def_var(ncid, varname, type, ndims, dimid, varid));

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

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

buffer_nc()

```
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 508 of file retrieval.c.

```
              {

  int ip;

  /* Set number of data points... */
  ncd->np = np1 - np0 + 1;

  /* Save retrieval data... */
  ncd->ret_chisq[track * L1_NXTRACK + xtrack] = (float) chisq;
  ncd->ret_p[track * L1_NXTRACK + xtrack] = (float) atm->p[np0];
  for (ip = np0; ip <= np1; ip++) {
    ncd->ret_z[ip - np0] = (float) atm->z[ip];
    ncd->ret_t[(track * L1_NXTRACK + xtrack) * ncd->np + ip - np0] =
      (gsl_finite(chisq) ? (float) atm->t[ip] : GSL_NAN);
  }
}
```

cost_function()

```
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 534 of file retrieval.c.

```
                                  {

  gsl_vector *x_aux, *y_aux;

  double chisq_a, chisq_m = 0;

  size_t i, m, n;

  /* Get sizes... */
  m = dy->size;
  n = dx->size;

  /* Allocate... */
  x_aux = gsl_vector_alloc(n);
  y_aux = gsl_vector_alloc(m);

  /* Determine normalized cost function...
     (chi^2 = 1/m * [dy^T * S_eps^{-1} * dy + dx^T * S_a^{-1} * dx]) */
  for (i = 0; i < m; i++)
    chisq_m +=
      gsl_pow_2(gsl_vector_get(dy, i) * gsl_vector_get(sig_eps_inv, i));
  gsl_blas_dgemv(CblasNoTrans, 1.0, s_a_inv, dx, 0.0, x_aux);
  gsl_blas_ddot(dx, x_aux, &chisq_a);

  /* Free... */
  gsl_vector_free(x_aux);
  gsl_vector_free(y_aux);

  /* Return cost function value... */
  return (chisq_m + chisq_a) / (double) m;
}
```

init_l2()

```
void init_l2 (
        ncd_t * ncd,
        int track,
        int xtrack,
        ctl_t * ctl,
        atm_t * atm )
```

Initialize with IASI Level-2 data.

Definition at line 572 of file retrieval.c.

```
                 {

  static atm_t atm_iasi;

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

  int ip, lay;

  /* Store IASI data in atmospheric data struct... */
  atm_iasi.np = 0;
  for (lay = 0; lay < L2_NLAY; lay++)
    if (gsl_finite(ncd->l2_z[track][xtrack][lay])
        && ncd->l2_z[track][xtrack][lay] <= 60.) {
      atm_iasi.z[atm_iasi.np] = ncd->l2_z[track][xtrack][lay];
      atm_iasi.p[atm_iasi.np] = ncd->l2_p[lay];
      atm_iasi.t[atm_iasi.np] = ncd->l2_t[track][xtrack][lay];
      if ((++atm_iasi.np) > NP)
        ERRMSG("Too many layers!");
    }

  /* Check number of levels... */
  if (atm_iasi.np < 2)
    return;

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

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

```

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

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

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

Here is the call graph for this function:

matrix_invert()

```
void matrix_invert (
            gsl_matrix * a )
```

Invert symmetric matrix.

Definition at line 628 of file retrieval.c.

```
                  {

  size_t diag = 1, i, j, n;

  /* Get size... */
  n = a->size1;

  /* Check if matrix is diagonal... */
  for (i = 0; i < n && diag; i++)
    for (j = i + 1; j < n; j++)
      if (gsl_matrix_get(a, i, j) != 0) {
        diag = 0;
        break;
      }

  /* Quick inversion of diagonal matrix... */
  if (diag)
    for (i = 0; i < n; i++)
      gsl_matrix_set(a, i, i, 1 / gsl_matrix_get(a, i, i));

  /* Matrix inversion by means of Cholesky decomposition... */
  else {
    gsl_linalg_cholesky_decomp(a);
    gsl_linalg_cholesky_invert(a);
  }
}
```

matrix_product()

```
void matrix_product (
            gsl_matrix * a,
```

```
gsl_vector * b,
int transpose,
gsl_matrix * c )
```

Compute matrix product A^TBA or ABA^T for diagonal matrix B.

Definition at line 658 of file retrieval.c.

```
                    {

  gsl_matrix *aux;

  size_t i, j, m, n;

  /* Set sizes... */
  m = a->size1;
  n = a->size2;

  /* Allocate... */
  aux = gsl_matrix_alloc(m, n);

  /* Compute A^T B A... */
  if (transpose == 1) {

    /* Compute B^1/2 A... */
    for (i = 0; i < m; i++)
      for (j = 0; j < n; j++)
        gsl_matrix_set(aux, i, j,
                       gsl_vector_get(b, i) * gsl_matrix_get(a, i, j));

    /* Compute A^T B A = (B^1/2 A)^T (B^1/2 A)... */
    gsl_blas_dgemm(CblasTrans, CblasNoTrans, 1.0, aux, aux, 0.0, c);
  }

  /* Compute A B A^T... */
  else if (transpose == 2) {

    /* Compute A B^1/2... */
    for (i = 0; i < m; i++)
      for (j = 0; j < n; j++)
        gsl_matrix_set(aux, i, j,
                       gsl_matrix_get(a, i, j) * gsl_vector_get(b, j));

    /* Compute A B A^T = (A B^1/2) (A B^1/2)^T... */
    gsl_blas_dgemm(CblasNoTrans, CblasTrans, 1.0, aux, aux, 0.0, c);
  }

  /* Free... */
  gsl_matrix_free(aux);
}
```

optimal_estimation()

```
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.

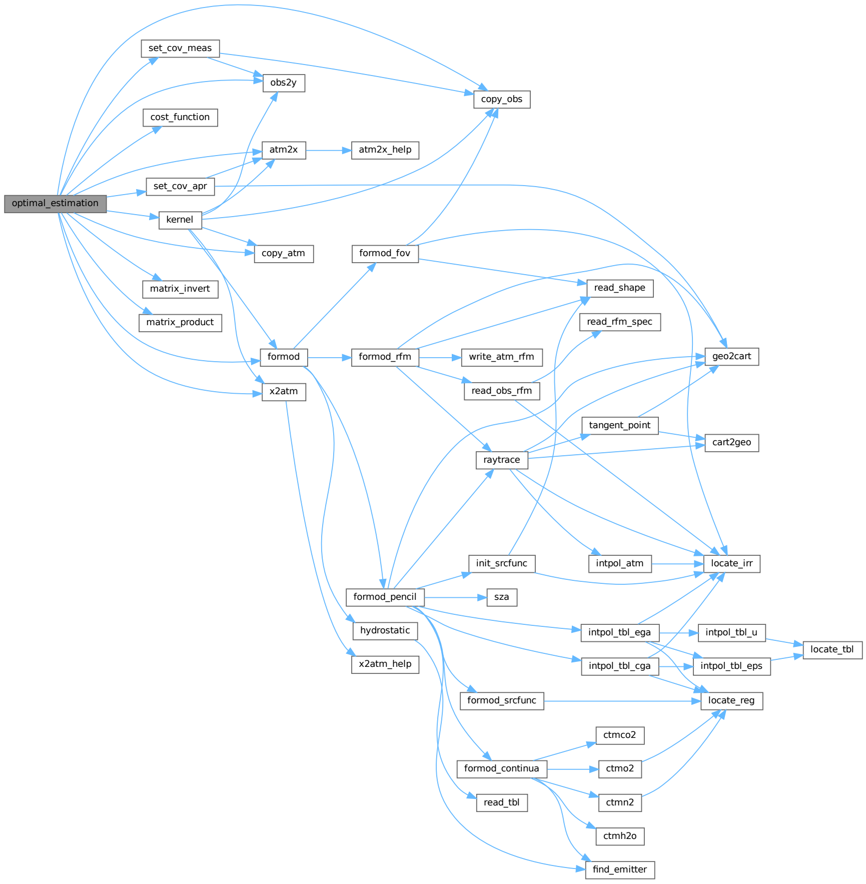
Definition at line 707 of file retrieval.c.

```
                   {

  static int ipa[N], iqa[N];

  gsl_matrix *a, *cov, *k_i, *s_a_inv;

  gsl_vector *b, *dx, *dy, *sig_eps_inv, *sig_formod, *sig_noise,
    *x_a, *x_i, *x_step, *y_aux, *y_i, *y_m;

  double chisq_old, disq = 0, lmpar = 0.001;
```

```
int ig, ip, it = 0, it2, iw;

size_t i, m, n;

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

/* Get sizes... */
m = obs2y(ctl, obs_meas, NULL, NULL, NULL);
n = atm2x(ctl, atm_apr, NULL, iqa, ipa);
if (m <= 0 || n <= 0) {
  *chisq = GSL_NAN;
  return;
}

/* Allocate... */
a = gsl_matrix_alloc(n, n);
cov = gsl_matrix_alloc(n, n);
k_i = gsl_matrix_alloc(m, n);
s_a_inv = gsl_matrix_alloc(n, n);

b = gsl_vector_alloc(n);
dx = gsl_vector_alloc(n);
dy = gsl_vector_alloc(m);
sig_eps_inv = gsl_vector_alloc(m);
sig_formod = gsl_vector_alloc(m);
sig_noise = gsl_vector_alloc(m);
x_a = gsl_vector_alloc(n);
x_i = gsl_vector_alloc(n);
x_step = gsl_vector_alloc(n);
y_aux = gsl_vector_alloc(m);
y_i = gsl_vector_alloc(m);
y_m = gsl_vector_alloc(m);

/* Set initial state... */
copy_atm(ctl, atm_i, atm_apr, 0);
copy_obs(ctl, obs_i, obs_meas, 0);
formod(ctl, atm_i, obs_i);

/* Set state vectors and observation vectors... */
atm2x(ctl, atm_apr, x_a, NULL, NULL);
atm2x(ctl, atm_i, x_i, NULL, NULL);
obs2y(ctl, obs_meas, y_m, NULL, NULL);
obs2y(ctl, obs_i, y_i, NULL, NULL);

/* Set inverse a priori covariance S_a^-1... */
set_cov_apr(ret, ctl, atm_apr, iqa, ipa, s_a_inv);
matrix_invert(s_a_inv);

/* Get measurement errors... */
set_cov_meas(ret, ctl, obs_meas, sig_noise, sig_formod, sig_eps_inv);

/* Determine dx = x_i - x_a and dy = y - F(x_i) ... */
gsl_vector_memcpy(dx, x_i);
gsl_vector_sub(dx, x_a);
gsl_vector_memcpy(dy, y_m);
gsl_vector_sub(dy, y_i);

/* Compute cost function... */
*chisq = cost_function(dx, dy, s_a_inv, sig_eps_inv);

/* Compute initial kernel... */
kernel(ctl, atm_i, obs_i, k_i);

/* ------------------------------------------------------------
   Levenberg-Marquardt minimization...
   ------------------------------------------------------------ */

/* Outer loop... */
for (it = 1; it <= ret->conv_itmax; it++) {

  /* Store current cost function value... */
  chisq_old = *chisq;

  /* Compute kernel matrix K_i... */
  if (it > 1 && it % ret->kernel_recomp == 0)
    kernel(ctl, atm_i, obs_i, k_i);

  /* Compute K_i^T * S_eps^{-1} * K_i ... */
  if (it == 1 || it % ret->kernel_recomp == 0)
    matrix_product(k_i, sig_eps_inv, 1, cov);

  /* Determine b = K_i^T * S_eps^{-1} * dy - S_a^{-1} * dx ... */
  for (i = 0; i < m; i++)
    gsl_vector_set(y_aux, i, gsl_vector_get(dy, i)
```

```
                     * gsl_pow_2(gsl_vector_get(sig_eps_inv, i)));
    gsl_blas_dgemv(CblasTrans, 1.0, k_i, y_aux, 0.0, b);
    gsl_blas_dgemv(CblasNoTrans, -1.0, s_a_inv, dx, 1.0, b);

    /* Inner loop... */
    for (it2 = 0; it2 < 20; it2++) {

      /* Compute A = (1 + lmpar) * S_a^{-1} + K_i^T * S_eps^{-1} * K_i ... */
      gsl_matrix_memcpy(a, s_a_inv);
      gsl_matrix_scale(a, 1 + lmpar);
      gsl_matrix_add(a, cov);

      /* Solve A * x_step = b by means of Cholesky decomposition... */
      gsl_linalg_cholesky_decomp(a);
      gsl_linalg_cholesky_solve(a, b, x_step);

      /* Update atmospheric state... */
      gsl_vector_add(x_i, x_step);
      copy_atm(ctl, atm_i, atm_apr, 0);
      copy_obs(ctl, obs_i, obs_meas, 0);
      x2atm(ctl, x_i, atm_i);

      /* Check atmospheric state... */
      for (ip = 0; ip < atm_i->np; ip++) {
        atm_i->p[ip] = GSL_MIN(GSL_MAX(atm_i->p[ip], 5e-7), 5e4);
        atm_i->t[ip] = GSL_MIN(GSL_MAX(atm_i->t[ip], 100), 400);
        for (ig = 0; ig < ctl->ng; ig++)
          atm_i->q[ig][ip] = GSL_MIN(GSL_MAX(atm_i->q[ig][ip], 0), 1);
        for (iw = 0; iw < ctl->nw; iw++)
          atm_i->k[iw][ip] = GSL_MAX(atm_i->k[iw][ip], 0);
      }

      /* Forward calculation... */
      formod(ctl, atm_i, obs_i);
      obs2y(ctl, obs_i, y_i, NULL, NULL);

      /* Determine dx = x_i - x_a and dy = y - F(x_i) ... */
      gsl_vector_memcpy(dx, x_i);
      gsl_vector_sub(dx, x_a);
      gsl_vector_memcpy(dy, y_m);
      gsl_vector_sub(dy, y_i);

      /* Compute cost function... */
      *chisq = cost_function(dx, dy, s_a_inv, sig_eps_inv);

      /* Modify Levenberg-Marquardt parameter... */
      if (*chisq > chisq_old) {
        lmpar *= 10;
        gsl_vector_sub(x_i, x_step);
      } else {
        lmpar /= 10;
        break;
      }
    }

    /* Get normalized step size in state space... */
    gsl_blas_ddot(x_step, b, &disq);
    disq /= (double) n;

    /* Convergence test... */
    if ((it == 1 || it % ret->kernel_recomp == 0) && disq < ret->conv_dmin)
      break;
  }

  /* ------------------------------------------------------------
     Finalize...
     ------------------------------------------------------------ */

  gsl_matrix_free(a);
  gsl_matrix_free(cov);
  gsl_matrix_free(k_i);
  gsl_matrix_free(s_a_inv);

  gsl_vector_free(b);
  gsl_vector_free(dx);
  gsl_vector_free(dy);
  gsl_vector_free(sig_eps_inv);
  gsl_vector_free(sig_formod);
  gsl_vector_free(sig_noise);
  gsl_vector_free(x_a);
  gsl_vector_free(x_i);
  gsl_vector_free(x_step);
  gsl_vector_free(y_aux);
  gsl_vector_free(y_i);
  gsl_vector_free(y_m);
}
```

Here is the call graph for this function:



read_nc()

```
void read_nc (
            char * filename,
            ncd_t * ncd )
```

Read netCDF file.

Definition at line 900 of file retrieval.c.

```
                {

  int dimid, varid;

  size_t len;

  /* Open netCDF file... */
  printf("Read netCDF file: %s\n", filename);
  NC(nc_open(filename, NC_WRITE, &ncd->ncid));

  /* Read number of tracks... */
  NC(nc_inq_dimid(ncd->ncid, "L1_NTRACK", &dimid));
  NC(nc_inq_dimlen(ncd->ncid, dimid, &len));
  ncd->ntrack = (int) len;

  /* Read Level-1 data... */
  NC(nc_inq_varid(ncd->ncid, "l1_time", &varid));
  NC(nc_get_var_double(ncd->ncid, varid, ncd->l1_time[0]));
  NC(nc_inq_varid(ncd->ncid, "l1_lon", &varid));
  NC(nc_get_var_double(ncd->ncid, varid, ncd->l1_lon[0]));
```

```
    NC(nc_inq_varid(ncd->ncid, "l1_lat", &varid));
    NC(nc_get_var_double(ncd->ncid, varid, ncd->l1_lat[0]));
    NC(nc_inq_varid(ncd->ncid, "l1_sat_z", &varid));
    NC(nc_get_var_double(ncd->ncid, varid, ncd->l1_sat_z));
    NC(nc_inq_varid(ncd->ncid, "l1_sat_lon", &varid));
    NC(nc_get_var_double(ncd->ncid, varid, ncd->l1_sat_lon));
    NC(nc_inq_varid(ncd->ncid, "l1_sat_lat", &varid));
    NC(nc_get_var_double(ncd->ncid, varid, ncd->l1_sat_lat));
    NC(nc_inq_varid(ncd->ncid, "l1_nu", &varid));
    NC(nc_get_var_double(ncd->ncid, varid, ncd->l1_nu));
    NC(nc_inq_varid(ncd->ncid, "l1_rad", &varid));
    NC(nc_get_var_float(ncd->ncid, varid, ncd->l1_rad[0][0]));

    /* Read Level-2 data... */
    NC(nc_inq_varid(ncd->ncid, "l2_z", &varid));
    NC(nc_get_var_double(ncd->ncid, varid, ncd->l2_z[0][0]));
    NC(nc_inq_varid(ncd->ncid, "l2_press", &varid));
    NC(nc_get_var_double(ncd->ncid, varid, ncd->l2_p));
    NC(nc_inq_varid(ncd->ncid, "l2_temp", &varid));
    NC(nc_get_var_double(ncd->ncid, varid, ncd->l2_t[0][0]));
}
```

read_ret_ctl()

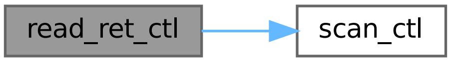
```
void read_ret_ctl (
            int argc,
            char * argv[],
            ctl_t * ctl,
            ret_t * ret )
```

Read retrieval control parameters.

Definition at line 946 of file retrieval.c.

```
                 {

  int id, ig, iw;

  /* Iteration control... */
  ret->kernel_recomp =
    (int) scan_ctl(argc, argv, "KERNEL_RECOMP", -1, "3", NULL);
  ret->conv_itmax = (int) scan_ctl(argc, argv, "CONV_ITMAX", -1, "30", NULL);
  ret->conv_dmin = scan_ctl(argc, argv, "CONV_DMIN", -1, "0.1", NULL);

  for (id = 0; id < ctl->nd; id++)
    ret->err_formod[id] = scan_ctl(argc, argv, "ERR_FORMOD", id, "0", NULL);

  for (id = 0; id < ctl->nd; id++)
    ret->err_noise[id] = scan_ctl(argc, argv, "ERR_NOISE", id, "0", NULL);

  ret->err_press = scan_ctl(argc, argv, "ERR_PRESS", -1, "0", NULL);
  ret->err_press_cz = scan_ctl(argc, argv, "ERR_PRESS_CZ", -1, "-999", NULL);
  ret->err_press_ch = scan_ctl(argc, argv, "ERR_PRESS_CH", -1, "-999", NULL);

  ret->err_temp = scan_ctl(argc, argv, "ERR_TEMP", -1, "0", NULL);
  ret->err_temp_cz = scan_ctl(argc, argv, "ERR_TEMP_CZ", -1, "-999", NULL);
  ret->err_temp_ch = scan_ctl(argc, argv, "ERR_TEMP_CH", -1, "-999", NULL);

  for (ig = 0; ig < ctl->ng; ig++) {
    ret->err_q[ig] = scan_ctl(argc, argv, "ERR_Q", ig, "0", NULL);
    ret->err_q_cz[ig] = scan_ctl(argc, argv, "ERR_Q_CZ", ig, "-999", NULL);
    ret->err_q_ch[ig] = scan_ctl(argc, argv, "ERR_Q_CH", ig, "-999", NULL);
  }

  for (iw = 0; iw < ctl->nw; iw++) {
    ret->err_k[iw] = scan_ctl(argc, argv, "ERR_K", iw, "0", NULL);
    ret->err_k_cz[iw] = scan_ctl(argc, argv, "ERR_K_CZ", iw, "-999", NULL);
    ret->err_k_ch[iw] = scan_ctl(argc, argv, "ERR_K_CH", iw, "-999", NULL);
  }
}
```

Here is the call graph for this function:



set_cov_apr()

```
void set_cov_apr (
            ret_t * ret,
            ctl_t * ctl,
            atm_t * atm,
            int * iqa,
            int * ipa,
            gsl_matrix * s_a )
```

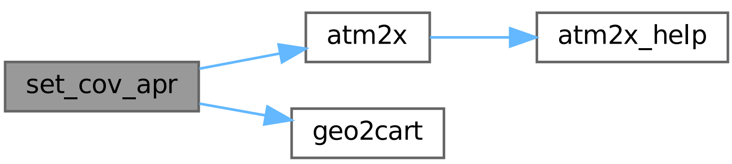
Set a priori covariance.

Definition at line 989 of file retrieval.c.

```
                      {

  gsl_vector *x_a;

  double ch, cz, rho, x0[3], x1[3];

  int ig, iw;

  size_t i, j, n;

  /* Get sizes... */
  n = s_a->size1;

  /* Allocate... */
  x_a = gsl_vector_alloc(n);

  /* Get sigma vector... */
  atm2x(ctl, atm, x_a, NULL, NULL);
  for (i = 0; i < n; i++) {
    if (iqa[i] == IDXP)
      gsl_vector_set(x_a, i, ret->err_press / 100 * gsl_vector_get(x_a, i));
    if (iqa[i] == IDXT)
      gsl_vector_set(x_a, i, ret->err_temp);
    for (ig = 0; ig < ctl->ng; ig++)
      if (iqa[i] == IDXQ(ig))
        gsl_vector_set(x_a, i, ret->err_q[ig] / 100 * gsl_vector_get(x_a, i));
    for (iw = 0; iw < ctl->nw; iw++)
      if (iqa[i] == IDXK(iw))
        gsl_vector_set(x_a, i, ret->err_k[iw]);
  }

  /* Check standard deviations... */
  for (i = 0; i < n; i++)
    if (gsl_pow_2(gsl_vector_get(x_a, i)) <= 0)
      ERRMSG("Check a priori data (zero standard deviation)!");

  /* Initialize diagonal covariance... */
  gsl_matrix_set_zero(s_a);
  for (i = 0; i < n; i++)
    gsl_matrix_set(s_a, i, i, gsl_pow_2(gsl_vector_get(x_a, i)));

  /* Loop over matrix elements... */
  for (i = 0; i < n; i++)
    for (j = 0; j < n; j++)
      if (i != j && iqa[i] == iqa[j]) {

```

```
        /* Initialize... */
        cz = ch = 0;

        /* Set correlation lengths for pressure... */
        if (iqa[i] == IDXP) {
          cz = ret->err_press_cz;
          ch = ret->err_press_ch;
        }

        /* Set correlation lengths for temperature... */
        if (iqa[i] == IDXT) {
          cz = ret->err_temp_cz;
          ch = ret->err_temp_ch;
        }

        /* Set correlation lengths for volume mixing ratios... */
        for (ig = 0; ig < ctl->ng; ig++)
          if (iqa[i] == IDXQ(ig)) {
            cz = ret->err_q_cz[ig];
            ch = ret->err_q_ch[ig];
          }

        /* Set correlation lengths for extinction... */
        for (iw = 0; iw < ctl->nw; iw++)
          if (iqa[i] == IDXK(iw)) {
            cz = ret->err_k_cz[iw];
            ch = ret->err_k_ch[iw];
          }

        /* Compute correlations... */
        if (cz > 0 && ch > 0) {

          /* Get Cartesian coordinates... */
          geo2cart(0, atm->lon[ipa[i]], atm->lat[ipa[i]], x0);
          geo2cart(0, atm->lon[ipa[j]], atm->lat[ipa[j]], x1);

          /* Compute correlations... */
          rho =
            exp(-DIST(x0, x1) / ch -
                fabs(atm->z[ipa[i]] - atm->z[ipa[j]]) / cz);

          /* Set covariance... */
          gsl_matrix_set(s_a, i, j, gsl_vector_get(x_a, i)
                         * gsl_vector_get(x_a, j) * rho);
        }
      }

  /* Free... */
  gsl_vector_free(x_a);
}
```

Here is the call graph for this function:



set_cov_meas()

```
void set_cov_meas (
            ret_t * ret,
            ctl_t * ctl,
            obs_t * obs,
```

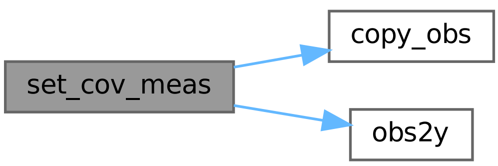
```
gsl_vector * sig_noise,
gsl_vector * sig_formod,
gsl_vector * sig_eps_inv )
```

Set measurement errors.

Definition at line 1094 of file retrieval.c.

```
                          {

  static obs_t obs_err;

  int id, ir;

  size_t i, m;

  /* Get size... */
  m = sig_eps_inv->size;

  /* Noise error (always considered in retrieval fit)... */
  copy_obs(ctl, &obs_err, obs, 1);
  for (ir = 0; ir < obs_err.nr; ir++)
    for (id = 0; id < ctl->nd; id++)
      obs_err.rad[id][ir]
        = (gsl_finite(obs->rad[id][ir]) ? ret->err_noise[id] : GSL_NAN);
  obs2y(ctl, &obs_err, sig_noise, NULL, NULL);

  /* Forward model error (always considered in retrieval fit)... */
  copy_obs(ctl, &obs_err, obs, 1);
  for (ir = 0; ir < obs_err.nr; ir++)
    for (id = 0; id < ctl->nd; id++)
      obs_err.rad[id][ir]
        = fabs(ret->err_formod[id] / 100 * obs->rad[id][ir]);
  obs2y(ctl, &obs_err, sig_formod, NULL, NULL);

  /* Total error... */
  for (i = 0; i < m; i++)
    gsl_vector_set(sig_eps_inv, i,
                   1 / sqrt(gsl_pow_2(gsl_vector_get(sig_noise, i))
                            + gsl_pow_2(gsl_vector_get(sig_formod, i))));

  /* Check standard deviations... */
  for (i = 0; i < m; i++)
    if (gsl_vector_get(sig_eps_inv, i) <= 0)
      ERRMSG("Check measurement errors (zero standard deviation)!");
}
```

Here is the call graph for this function:



write_nc()

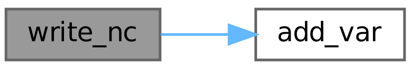
```
void write_nc (
            char * filename,
            ncd_t * ncd )
```

Write to netCDF file...

Definition at line 1141 of file retrieval.c.

```
                  {

  int dimid[10], c_id, p_id, t_id, z_id;

  /* Create netCDF file... */
  printf("Write netCDF file: %s\n", filename);

  /* Read existing dimensions... */
  NC(nc_inq_dimid(ncd->ncid, "L1_NTRACK", &dimid[0]));
  NC(nc_inq_dimid(ncd->ncid, "L1_NXTRACK", &dimid[1]));

  /* Set define mode... */
  NC(nc_redef(ncd->ncid));

  /* Set new dimensions... */
  if (nc_inq_dimid(ncd->ncid, "RET_NP", &dimid[2]) != NC_NOERR)
    NC(nc_def_dim(ncd->ncid, "RET_NP", (size_t) ncd->np, &dimid[2]));

  /* Set new variables... */
  add_var(ncd->ncid, "ret_z", "km", "altitude", NC_FLOAT, &dimid[2], &z_id,
          1);
  add_var(ncd->ncid, "ret_press", "hPa", "pressure", NC_FLOAT, dimid, &p_id,
          2);
  add_var(ncd->ncid, "ret_temp", "K", "temperature", NC_FLOAT, dimid, &t_id,
          3);
  add_var(ncd->ncid, "ret_chisq", "1", "chi^2 value of fit", NC_FLOAT, dimid,
          &c_id, 2);

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

  /* Write data... */
  NC(nc_put_var_float(ncd->ncid, z_id, ncd->ret_z));
  NC(nc_put_var_float(ncd->ncid, p_id, ncd->ret_p));
  NC(nc_put_var_float(ncd->ncid, t_id, ncd->ret_t));
  NC(nc_put_var_float(ncd->ncid, c_id, ncd->ret_chisq));

  /* Close netCDF file... */
  NC(nc_close(ncd->ncid));
}
```

Here is the call graph for this function:



main()

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

Definition at line 258 of file retrieval.c.

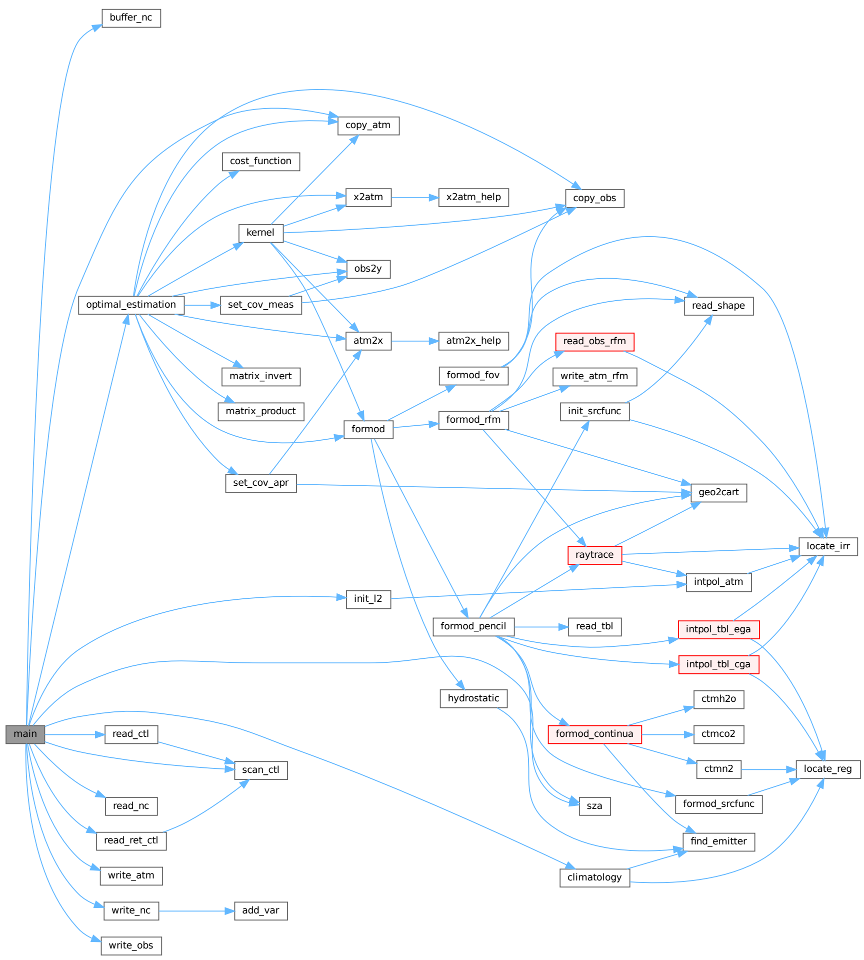
```
                   {

  static ctl_t ctl;
  static atm_t atm_apr, atm_clim, atm_i;
  static obs_t obs_i, obs_meas;
  static ncd_t ncd;
  static ret_t ret;

  FILE *in;

  char filename[LEN], filename2[LEN];

```

```c
  double chisq, sza_thresh, z[NP], t0;

  int channel[ND], i, id, ip, iz, nz, ntask = -1, rank, size,
    np0, np1, track, track0, track1, xtrack, xtrack0, xtrack1,
    task0, task1, debug;

  /* ------------------------------------------------------------
     Init...
     ------------------------------------------------------------ */

  /* MPI... */
  MPI_Init(&argc, &argv);
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  MPI_Comm_size(MPI_COMM_WORLD, &size);

  /* Measure CPU time... */
  TIMER("total", 1);

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

  /* Read control parameters... */
  read_ctl(argc, argv, &ctl);
  read_ret_ctl(argc, argv, &ctl, &ret);
  debug = (int) scan_ctl(argc, argv, "DEBUG", -1, "1", NULL);

  /* Read retrieval grid... */
  nz = (int) scan_ctl(argc, argv, "NZ", -1, "", NULL);
  if (nz > NP)
    ERRMSG("Too many altitudes!");
  for (iz = 0; iz < nz; iz++)
    z[iz] = scan_ctl(argc, argv, "Z", iz, "", NULL);

  /* Read task range... */
  task0 = (int) scan_ctl(argc, argv, "TASK_MIN", -1, "0", NULL);
  task1 = (int) scan_ctl(argc, argv, "TASK_MAX", -1, "99999", NULL);

  /* Read track range... */
  track0 = (int) scan_ctl(argc, argv, "TRACK_MIN", -1, "0", NULL);
  track1 = (int) scan_ctl(argc, argv, "TRACK_MAX", -1, "99999", NULL);

  /* Read xtrack range... */
  xtrack0 = (int) scan_ctl(argc, argv, "XTRACK_MIN", -1, "0", NULL);
  xtrack1 = (int) scan_ctl(argc, argv, "XTRACK_MAX", -1, "59", NULL);

  /* Read height range... */
  np0 = (int) scan_ctl(argc, argv, "NP_MIN", -1, "0", NULL);
  np1 = (int) scan_ctl(argc, argv, "NP_MAX", -1, "100", NULL);
  np1 = GSL_MIN(np1, nz - 1);

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

  /* ------------------------------------------------------------
     Distribute granules...
     ------------------------------------------------------------ */

  /* Open filelist... */
  printf("Read filelist: %s\n", argv[2]);
  if (!(in = fopen(argv[2], "r")))
    ERRMSG("Cannot open filelist!");

  /* Loop over netCDF files... */
  while (fscanf(in, "%s", filename) != EOF) {

    /* Distribute files with MPI... */
    if ((++ntask) % size != rank)
      continue;

    /* Check task range... */
    if (ntask < task0 || ntask > task1)
      continue;

    /* Write info... */
    printf("Retrieve file %s on rank %d of %d (with %d threads)...\n",
           filename, rank + 1, size, omp_get_max_threads());

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

    /* Read netCDF file... */
    read_nc(filename, &ncd);

    /* Adjust number of tracks... */
    if (track1 >= ncd.ntrack)
```

```
    track1 = ncd.ntrack - 1;

  /* Identify radiance channels... */
  for (id = 0; id < ctl.nd; id++) {
    channel[id] = -999;
    for (i = 0; i < L1_NCHAN; i++)
      if (fabs(ctl.nu[id] - ncd.l1_nu[i]) < 0.1)
        channel[id] = i;
    if (channel[id] < 0)
      ERRMSG("Cannot identify radiance channel!");
  }

  /* Set climatological data for center of granule... */
  atm_clim.np = nz;
  for (iz = 0; iz < nz; iz++)
    atm_clim.z[iz] = z[iz];
  climatology(&ctl, &atm_clim);

  /* ------------------------------------------------------------
     Retrieval...
     ------------------------------------------------------------ */

  /* Loop over swaths... */
  for (track = track0; track <= track1; track++) {

    /* Loop over scan... */
    for (xtrack = xtrack0; xtrack <= xtrack1; xtrack++) {

      /* Init timer... */
      t0 = omp_get_wtime();

      /* Store observation data... */
      obs_meas.nr = 1;
      obs_meas.time[0] = ncd.l1_time[track][xtrack];
      obs_meas.obsz[0] = ncd.l1_sat_z[track];
      obs_meas.obslon[0] = ncd.l1_sat_lon[track];
      obs_meas.obslat[0] = ncd.l1_sat_lat[track];
      obs_meas.vplon[0] = ncd.l1_lon[track][xtrack];
      obs_meas.vplat[0] = ncd.l1_lat[track][xtrack];
      for (id = 0; id < ctl.nd; id++)
        obs_meas.rad[id][0] = ncd.l1_rad[track][xtrack][channel[id]];

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

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

      /* Merge Level-2 data... */
      init_l2(&ncd, track, xtrack, &ctl, &atm_apr);

      /* Retrieval... */
      optimal_estimation(&ret, &ctl, &obs_meas, &obs_i,
                         &atm_apr, &atm_i, &chisq);

      /* Buffer results... */
      buffer_nc(&atm_i, chisq, &ncd, track, xtrack, np0, np1);

      /* Write debug information... */
      if (debug >= 1)
        printf
          ("  task= %4d | track= %5d | xtrack= %3d | chi^2= %8.3f | time= %8.3f s\n",
           ntask, track, xtrack, chisq, omp_get_wtime() - t0);
      if (debug >= 2) {
        sprintf(filename2, "atm_apr_%d_%d_%d.tab", ntask, track, xtrack);
        write_atm(NULL, filename2, &ctl, &atm_apr);
        sprintf(filename2, "atm_i_%d_%d_%d.tab", ntask, track, xtrack);
        write_atm(NULL, filename2, &ctl, &atm_i);
        sprintf(filename2, "obs_meas_%d_%d_%d.tab", ntask, track, xtrack);
        write_obs(NULL, filename2, &ctl, &obs_meas);
        sprintf(filename2, "obs_i_%d_%d_%d.tab", ntask, track, xtrack);
        write_obs(NULL, filename2, &ctl, &obs_i);
      }
    }
  }

  /* ------------------------------------------------------------
     Finalize...
```

```
       ------------------------------------------------------------ */

    /* Write netCDF file... */
    write_nc(filename, &ncd);

    /* Write info... */
    printf("Retrieval finished on rank %d of %d!\n", rank, size);
  }

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

  /* Measure CPU time... */
  TIMER("total", 3);

  /* Report memory usage... */
  printf("MEMORY_ATM = %g MByte\n", 4. * sizeof(atm_t) / 1024. / 1024.);
  printf("MEMORY_CTL = %g MByte\n", 1. * sizeof(ctl_t) / 1024. / 1024.);
  printf("MEMORY_NCD = %g MByte\n", 1. * sizeof(ncd_t) / 1024. / 1024.);
  printf("MEMORY_OBS = %g MByte\n", 3. * sizeof(atm_t) / 1024. / 1024.);
  printf("MEMORY_RET = %g MByte\n", 1. * sizeof(ret_t) / 1024. / 1024.);
  printf("MEMORY_TBL = %g MByte\n", 1. * sizeof(tbl_t) / 1024. / 1024.);

  /* Report problem size... */
  printf("SIZE_TASKS = %d\n", size);
  printf("SIZE_THREADS = %d\n", omp_get_max_threads());

  /* MPI... */
  MPI_Finalize();

  return EXIT_SUCCESS;
}
```

Here is the call graph for this function:



5.18 retrieval.c

Go to the documentation of this file.

```
#include <mpi.h>
#include <omp.h>
#include <netcdf.h>
#include "jurassic.h"

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

#define NC(cmd) {                                     \
  int nc_result=(cmd);                                \
  if(nc_result!=NC_NOERR)                             \
    ERRMSG("%s", nc_strerror(nc_result));             \
}

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

#define L1_NCHAN 33

#define L1_NTRACK 1800

#define L1_NXTRACK 60

#define L2_NLAY 27

#define L2_NTRACK 1800

#define L2_NXTRACK 60

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

typedef struct {

  int ncid;

  int np;

  int ntrack;

  double l1_time[L1_NTRACK][L1_NXTRACK];

  double l1_lon[L1_NTRACK][L1_NXTRACK];

  double l1_lat[L1_NTRACK][L1_NXTRACK];

  double l1_sat_z[L1_NTRACK];

  double l1_sat_lon[L1_NTRACK];

  double l1_sat_lat[L1_NTRACK];

  double l1_nu[L1_NCHAN];

  float l1_rad[L1_NTRACK][L1_NXTRACK][L1_NCHAN];

  double l2_z[L2_NTRACK][L2_NXTRACK][L2_NLAY];

  double l2_p[L2_NLAY];

  double l2_t[L2_NTRACK][L2_NXTRACK][L2_NLAY];

  float ret_z[NP];

  float ret_p[L1_NTRACK * L1_NXTRACK];

  float ret_t[L1_NTRACK * L1_NXTRACK * NP];

  float ret_chisq[L1_NTRACK * L1_NXTRACK];

} ncd_t;

typedef struct {

  int kernel_recomp;

  int conv_itmax;

  double conv_dmin;

```

```
  double err_formod[ND];

  double err_noise[ND];

  double err_press;

  double err_press_cz;

  double err_press_ch;

  double err_temp;

  double err_temp_cz;

  double err_temp_ch;

  double err_q[NG];

  double err_q_cz[NG];

  double err_q_ch[NG];

  double err_k[NW];

  double err_k_cz[NW];

  double err_k_ch[NW];

} ret_t;

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

void add_var(
  int ncid,
  const char *varname,
  const char *unit,
  const char *longname,
  int type,
  int dimid[],
  int *varid,
  int ndims);

void buffer_nc(
  atm_t * atm,
  double chisq,
  ncd_t * ncd,
  int track,
  int xtrack,
  int np0,
  int np1);

double cost_function(
  gsl_vector * dx,
  gsl_vector * dy,
  gsl_matrix * s_a_inv,
  gsl_vector * sig_eps_inv);

void init_l2(
  ncd_t * ncd,
  int track,
  int xtrack,
  ctl_t * ctl,
  atm_t * atm);

void matrix_invert(
  gsl_matrix * a);

void matrix_product(
  gsl_matrix * a,
  gsl_vector * b,
  int transpose,
  gsl_matrix * c);

void optimal_estimation(
  ret_t * ret,
  ctl_t * ctl,
  obs_t * obs_meas,
  obs_t * obs_i,
  atm_t * atm_apr,
  atm_t * atm_i,
  double *chisq);

void read_nc(
  char *filename,
  ncd_t * ncd);
```

```
void read_ret_ctl(
  int argc,
  char *argv[],
  ctl_t * ctl,
  ret_t * ret);

void set_cov_apr(
  ret_t * ret,
  ctl_t * ctl,
  atm_t * atm,
  int *iqa,
  int *ipa,
  gsl_matrix * s_a);

void set_cov_meas(
  ret_t * ret,
  ctl_t * ctl,
  obs_t * obs,
  gsl_vector * sig_noise,
  gsl_vector * sig_formod,
  gsl_vector * sig_eps_inv);

void write_nc(
  char *filename,
  ncd_t * ncd);

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

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

  static ctl_t ctl;
  static atm_t atm_apr, atm_clim, atm_i;
  static obs_t obs_i, obs_meas;
  static ncd_t ncd;
  static ret_t ret;

  FILE *in;

  char filename[LEN], filename2[LEN];

  double chisq, sza_thresh, z[NP], t0;

  int channel[ND], i, id, ip, iz, nz, ntask = -1, rank, size,
    np0, np1, track, track0, track1, xtrack, xtrack0, xtrack1,
    task0, task1, debug;

  /* ------------------------------------------------------------
     Init...
     ------------------------------------------------------------ */

  /* MPI... */
  MPI_Init(&argc, &argv);
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  MPI_Comm_size(MPI_COMM_WORLD, &size);

  /* Measure CPU time... */
  TIMER("total", 1);

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

  /* Read control parameters... */
  read_ctl(argc, argv, &ctl);
  read_ret_ctl(argc, argv, &ctl, &ret);
  debug = (int) scan_ctl(argc, argv, "DEBUG", -1, "1", NULL);

  /* Read retrieval grid... */
  nz = (int) scan_ctl(argc, argv, "NZ", -1, "", NULL);
  if (nz > NP)
    ERRMSG("Too many altitudes!");
  for (iz = 0; iz < nz; iz++)
    z[iz] = scan_ctl(argc, argv, "Z", iz, "", NULL);

  /* Read task range... */
  task0 = (int) scan_ctl(argc, argv, "TASK_MIN", -1, "0", NULL);
  task1 = (int) scan_ctl(argc, argv, "TASK_MAX", -1, "99999", NULL);

  /* Read track range... */
  track0 = (int) scan_ctl(argc, argv, "TRACK_MIN", -1, "0", NULL);
  track1 = (int) scan_ctl(argc, argv, "TRACK_MAX", -1, "99999", NULL);

```

```
/* Read xtrack range... */
xtrack0 = (int) scan_ctl(argc, argv, "XTRACK_MIN", -1, "0", NULL);
xtrack1 = (int) scan_ctl(argc, argv, "XTRACK_MAX", -1, "59", NULL);

/* Read height range... */
np0 = (int) scan_ctl(argc, argv, "NP_MIN", -1, "0", NULL);
np1 = (int) scan_ctl(argc, argv, "NP_MAX", -1, "100", NULL);
np1 = GSL_MIN(np1, nz - 1);

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

/* ------------------------------------------------------------
   Distribute granules...
   ------------------------------------------------------------ */

/* Open filelist... */
printf("Read filelist: %s\n", argv[2]);
if (!(in = fopen(argv[2], "r")))
  ERRMSG("Cannot open filelist!");

/* Loop over netCDF files... */
while (fscanf(in, "%s", filename) != EOF) {

  /* Distribute files with MPI... */
  if ((++ntask) % size != rank)
    continue;

  /* Check task range... */
  if (ntask < task0 || ntask > task1)
    continue;

  /* Write info... */
  printf("Retrieve file %s on rank %d of %d (with %d threads)...\n",
         filename, rank + 1, size, omp_get_max_threads());

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

  /* Read netCDF file... */
  read_nc(filename, &ncd);

  /* Adjust number of tracks... */
  if (track1 >= ncd.ntrack)
    track1 = ncd.ntrack - 1;

  /* Identify radiance channels... */
  for (id = 0; id < ctl.nd; id++) {
    channel[id] = -999;
    for (i = 0; i < L1_NCHAN; i++)
      if (fabs(ctl.nu[id] - ncd.l1_nu[i]) < 0.1)
        channel[id] = i;
    if (channel[id] < 0)
      ERRMSG("Cannot identify radiance channel!");
  }

  /* Set climatological data for center of granule... */
  atm_clim.np = nz;
  for (iz = 0; iz < nz; iz++)
    atm_clim.z[iz] = z[iz];
  climatology(&ctl, &atm_clim);

  /* ------------------------------------------------------------
     Retrieval...
     ------------------------------------------------------------ */

  /* Loop over swaths... */
  for (track = track0; track <= track1; track++) {

    /* Loop over scan... */
    for (xtrack = xtrack0; xtrack <= xtrack1; xtrack++) {

      /* Init timer... */
      t0 = omp_get_wtime();

      /* Store observation data... */
      obs_meas.nr = 1;
      obs_meas.time[0] = ncd.l1_time[track][xtrack];
      obs_meas.obsz[0] = ncd.l1_sat_z[track];
      obs_meas.obslon[0] = ncd.l1_sat_lon[track];
      obs_meas.obslat[0] = ncd.l1_sat_lat[track];
      obs_meas.vplon[0] = ncd.l1_lon[track][xtrack];
      obs_meas.vplat[0] = ncd.l1_lat[track][xtrack];
      for (id = 0; id < ctl.nd; id++)
        obs_meas.rad[id][0] = ncd.l1_rad[track][xtrack][channel[id]];

```

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

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

        /* Merge Level-2 data... */
        init_l2(&ncd, track, xtrack, &ctl, &atm_apr);

        /* Retrieval... */
        optimal_estimation(&ret, &ctl, &obs_meas, &obs_i,
                           &atm_apr, &atm_i, &chisq);

        /* Buffer results... */
        buffer_nc(&atm_i, chisq, &ncd, track, xtrack, np0, np1);

        /* Write debug information... */
        if (debug >= 1)
          printf
            ("  task= %4d | track= %5d | xtrack= %3d | chi^2= %8.3f | time= %8.3f s\n",
             ntask, track, xtrack, chisq, omp_get_wtime() - t0);
        if (debug >= 2) {
          sprintf(filename2, "atm_apr_%d_%d_%d.tab", ntask, track, xtrack);
          write_atm(NULL, filename2, &ctl, &atm_apr);
          sprintf(filename2, "atm_i_%d_%d_%d.tab", ntask, track, xtrack);
          write_atm(NULL, filename2, &ctl, &atm_i);
          sprintf(filename2, "obs_meas_%d_%d_%d.tab", ntask, track, xtrack);
          write_obs(NULL, filename2, &ctl, &obs_meas);
          sprintf(filename2, "obs_i_%d_%d_%d.tab", ntask, track, xtrack);
          write_obs(NULL, filename2, &ctl, &obs_i);
        }
      }
    }

    /* ------------------------------------------------------------
       Finalize...
       ------------------------------------------------------------ */

    /* Write netCDF file... */
    write_nc(filename, &ncd);

    /* Write info... */
    printf("Retrieval finished on rank %d of %d!\n", rank, size);
  }

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

  /* Measure CPU time... */
  TIMER("total", 3);

  /* Report memory usage... */
  printf("MEMORY_ATM = %g MByte\n", 4. * sizeof(atm_t) / 1024. / 1024.);
  printf("MEMORY_CTL = %g MByte\n", 1. * sizeof(ctl_t) / 1024. / 1024.);
  printf("MEMORY_NCD = %g MByte\n", 1. * sizeof(ncd_t) / 1024. / 1024.);
  printf("MEMORY_OBS = %g MByte\n", 3. * sizeof(atm_t) / 1024. / 1024.);
  printf("MEMORY_RET = %g MByte\n", 1. * sizeof(ret_t) / 1024. / 1024.);
  printf("MEMORY_TBL = %g MByte\n", 1. * sizeof(tbl_t) / 1024. / 1024.);

  /* Report problem size... */
  printf("SIZE_TASKS = %d\n", size);
  printf("SIZE_THREADS = %d\n", omp_get_max_threads());

  /* MPI... */
  MPI_Finalize();

  return EXIT_SUCCESS;
}

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

void add_var(
  int ncid,
  const char *varname,
  const char *unit,
  const char *longname,
  int type,
  int dimid[],
```

```
  int *varid,
  int ndims) {

  /* Check if variable exists... */
  if (nc_inq_varid(ncid, varname, varid) != NC_NOERR) {

    /* Define variable... */
    NC(nc_def_var(ncid, varname, type, ndims, dimid, varid));

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

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

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

void buffer_nc(
  atm_t * atm,
  double chisq,
  ncd_t * ncd,
  int track,
  int xtrack,
  int np0,
  int np1) {

  int ip;

  /* Set number of data points... */
  ncd->np = np1 - np0 + 1;

  /* Save retrieval data... */
  ncd->ret_chisq[track * L1_NXTRACK + xtrack] = (float) chisq;
  ncd->ret_p[track * L1_NXTRACK + xtrack] = (float) atm->p[np0];
  for (ip = np0; ip <= np1; ip++) {
    ncd->ret_z[ip - np0] = (float) atm->z[ip];
    ncd->ret_t[(track * L1_NXTRACK + xtrack) * ncd->np + ip - np0] =
      (gsl_finite(chisq) ? (float) atm->t[ip] : GSL_NAN);
  }
}

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

double cost_function(
  gsl_vector * dx,
  gsl_vector * dy,
  gsl_matrix * s_a_inv,
  gsl_vector * sig_eps_inv) {

  gsl_vector *x_aux, *y_aux;

  double chisq_a, chisq_m = 0;

  size_t i, m, n;

  /* Get sizes... */
  m = dy->size;
  n = dx->size;

  /* Allocate... */
  x_aux = gsl_vector_alloc(n);
  y_aux = gsl_vector_alloc(m);

  /* Determine normalized cost function...
     (chi^2 = 1/m * [dy^T * S_eps^{-1} * dy + dx^T * S_a^{-1} * dx]) */
  for (i = 0; i < m; i++)
    chisq_m +=
      gsl_pow_2(gsl_vector_get(dy, i) * gsl_vector_get(sig_eps_inv, i));
  gsl_blas_dgemv(CblasNoTrans, 1.0, s_a_inv, dx, 0.0, x_aux);
  gsl_blas_ddot(dx, x_aux, &chisq_a);

  /* Free... */
  gsl_vector_free(x_aux);
  gsl_vector_free(y_aux);

  /* Return cost function value... */
  return (chisq_m + chisq_a) / (double) m;
}

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

void init_l2(
  ncd_t * ncd,
  int track,
```

```
  int xtrack,
  ctl_t * ctl,
  atm_t * atm) {

  static atm_t atm_iasi;

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

  int ip, lay;

  /* Store IASI data in atmospheric data struct... */
  atm_iasi.np = 0;
  for (lay = 0; lay < L2_NLAY; lay++)
    if (gsl_finite(ncd->l2_z[track][xtrack][lay])
        && ncd->l2_z[track][xtrack][lay] <= 60.) {
      atm_iasi.z[atm_iasi.np] = ncd->l2_z[track][xtrack][lay];
      atm_iasi.p[atm_iasi.np] = ncd->l2_p[lay];
      atm_iasi.t[atm_iasi.np] = ncd->l2_t[track][xtrack][lay];
      if ((++atm_iasi.np) > NP)
        ERRMSG("Too many layers!");
    }

  /* Check number of levels... */
  if (atm_iasi.np < 2)
    return;

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

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

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

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

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

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

void matrix_invert(
  gsl_matrix * a) {

  size_t diag = 1, i, j, n;

  /* Get size... */
  n = a->size1;

  /* Check if matrix is diagonal... */
  for (i = 0; i < n && diag; i++)
    for (j = i + 1; j < n; j++)
      if (gsl_matrix_get(a, i, j) != 0) {
        diag = 0;
        break;
      }

  /* Quick inversion of diagonal matrix... */
  if (diag)
    for (i = 0; i < n; i++)
      gsl_matrix_set(a, i, i, 1 / gsl_matrix_get(a, i, i));

  /* Matrix inversion by means of Cholesky decomposition... */
  else {
    gsl_linalg_cholesky_decomp(a);
    gsl_linalg_cholesky_invert(a);
  }
}

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

void matrix_product(
  gsl_matrix * a,
  gsl_vector * b,
  int transpose,
```

```
  gsl_matrix * c) {

  gsl_matrix *aux;

  size_t i, j, m, n;

  /* Set sizes... */
  m = a->size1;
  n = a->size2;

  /* Allocate... */
  aux = gsl_matrix_alloc(m, n);

  /* Compute A^T B A... */
  if (transpose == 1) {

    /* Compute B^1/2 A... */
    for (i = 0; i < m; i++)
      for (j = 0; j < n; j++)
        gsl_matrix_set(aux, i, j,
                       gsl_vector_get(b, i) * gsl_matrix_get(a, i, j));

    /* Compute A^T B A = (B^1/2 A)^T (B^1/2 A)... */
    gsl_blas_dgemm(CblasTrans, CblasNoTrans, 1.0, aux, aux, 0.0, c);
  }

  /* Compute A B A^T... */
  else if (transpose == 2) {

    /* Compute A B^1/2... */
    for (i = 0; i < m; i++)
      for (j = 0; j < n; j++)
        gsl_matrix_set(aux, i, j,
                       gsl_matrix_get(a, i, j) * gsl_vector_get(b, j));

    /* Compute A B A^T = (A B^1/2) (A B^1/2)^T... */
    gsl_blas_dgemm(CblasNoTrans, CblasTrans, 1.0, aux, aux, 0.0, c);
  }

  /* Free... */
  gsl_matrix_free(aux);
}

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

void optimal_estimation(
  ret_t * ret,
  ctl_t * ctl,
  obs_t * obs_meas,
  obs_t * obs_i,
  atm_t * atm_apr,
  atm_t * atm_i,
  double *chisq) {

  static int ipa[N], iqa[N];

  gsl_matrix *a, *cov, *k_i, *s_a_inv;

  gsl_vector *b, *dx, *dy, *sig_eps_inv, *sig_formod, *sig_noise,
    *x_a, *x_i, *x_step, *y_aux, *y_i, *y_m;

  double chisq_old, disq = 0, lmpar = 0.001;

  int ig, ip, it = 0, it2, iw;

  size_t i, m, n;

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

  /* Get sizes... */
  m = obs2y(ctl, obs_meas, NULL, NULL, NULL);
  n = atm2x(ctl, atm_apr, NULL, iqa, ipa);
  if (m <= 0 || n <= 0) {
    *chisq = GSL_NAN;
    return;
  }

  /* Allocate... */
  a = gsl_matrix_alloc(n, n);
  cov = gsl_matrix_alloc(n, n);
  k_i = gsl_matrix_alloc(m, n);
  s_a_inv = gsl_matrix_alloc(n, n);

  b = gsl_vector_alloc(n);
  dx = gsl_vector_alloc(n);
```

```
  dy = gsl_vector_alloc(m);
  sig_eps_inv = gsl_vector_alloc(m);
  sig_formod = gsl_vector_alloc(m);
  sig_noise = gsl_vector_alloc(m);
  x_a = gsl_vector_alloc(n);
  x_i = gsl_vector_alloc(n);
  x_step = gsl_vector_alloc(n);
  y_aux = gsl_vector_alloc(m);
  y_i = gsl_vector_alloc(m);
  y_m = gsl_vector_alloc(m);

  /* Set initial state... */
  copy_atm(ctl, atm_i, atm_apr, 0);
  copy_obs(ctl, obs_i, obs_meas, 0);
  formod(ctl, atm_i, obs_i);

  /* Set state vectors and observation vectors... */
  atm2x(ctl, atm_apr, x_a, NULL, NULL);
  atm2x(ctl, atm_i, x_i, NULL, NULL);
  obs2y(ctl, obs_meas, y_m, NULL, NULL);
  obs2y(ctl, obs_i, y_i, NULL, NULL);

  /* Set inverse a priori covariance S_a^-1... */
  set_cov_apr(ret, ctl, atm_apr, iqa, ipa, s_a_inv);
  matrix_invert(s_a_inv);

  /* Get measurement errors... */
  set_cov_meas(ret, ctl, obs_meas, sig_noise, sig_formod, sig_eps_inv);

  /* Determine dx = x_i - x_a and dy = y - F(x_i) ... */
  gsl_vector_memcpy(dx, x_i);
  gsl_vector_sub(dx, x_a);
  gsl_vector_memcpy(dy, y_m);
  gsl_vector_sub(dy, y_i);

  /* Compute cost function... */
  *chisq = cost_function(dx, dy, s_a_inv, sig_eps_inv);

  /* Compute initial kernel... */
  kernel(ctl, atm_i, obs_i, k_i);

  /* ------------------------------------------------------------
     Levenberg-Marquardt minimization...
     ------------------------------------------------------------ */

  /* Outer loop... */
  for (it = 1; it <= ret->conv_itmax; it++) {

    /* Store current cost function value... */
    chisq_old = *chisq;

    /* Compute kernel matrix K_i... */
    if (it > 1 && it % ret->kernel_recomp == 0)
      kernel(ctl, atm_i, obs_i, k_i);

    /* Compute K_i^T * S_eps^{-1} * K_i ... */
    if (it == 1 || it % ret->kernel_recomp == 0)
      matrix_product(k_i, sig_eps_inv, 1, cov);

    /* Determine b = K_i^T * S_eps^{-1} * dy - S_a^{-1} * dx ... */
    for (i = 0; i < m; i++)
      gsl_vector_set(y_aux, i, gsl_vector_get(dy, i)
                     * gsl_pow_2(gsl_vector_get(sig_eps_inv, i)));
    gsl_blas_dgemv(CblasTrans, 1.0, k_i, y_aux, 0.0, b);
    gsl_blas_dgemv(CblasNoTrans, -1.0, s_a_inv, dx, 1.0, b);

    /* Inner loop... */
    for (it2 = 0; it2 < 20; it2++) {

      /* Compute A = (1 + lmpar) * S_a^{-1} + K_i^T * S_eps^{-1} * K_i ... */
      gsl_matrix_memcpy(a, s_a_inv);
      gsl_matrix_scale(a, 1 + lmpar);
      gsl_matrix_add(a, cov);

      /* Solve A * x_step = b by means of Cholesky decomposition... */
      gsl_linalg_cholesky_decomp(a);
      gsl_linalg_cholesky_solve(a, b, x_step);

      /* Update atmospheric state... */
      gsl_vector_add(x_i, x_step);
      copy_atm(ctl, atm_i, atm_apr, 0);
      copy_obs(ctl, obs_i, obs_meas, 0);
      x2atm(ctl, x_i, atm_i);

      /* Check atmospheric state... */
      for (ip = 0; ip < atm_i->np; ip++) {
        atm_i->p[ip] = GSL_MIN(GSL_MAX(atm_i->p[ip], 5e-7), 5e4);
```

```
        atm_i->t[ip] = GSL_MIN(GSL_MAX(atm_i->t[ip], 100), 400);
        for (ig = 0; ig < ctl->ng; ig++)
          atm_i->q[ig][ip] = GSL_MIN(GSL_MAX(atm_i->q[ig][ip], 0), 1);
        for (iw = 0; iw < ctl->nw; iw++)
          atm_i->k[iw][ip] = GSL_MAX(atm_i->k[iw][ip], 0);
      }

      /* Forward calculation... */
      formod(ctl, atm_i, obs_i);
      obs2y(ctl, obs_i, y_i, NULL, NULL);

      /* Determine dx = x_i - x_a and dy = y - F(x_i) ... */
      gsl_vector_memcpy(dx, x_i);
      gsl_vector_sub(dx, x_a);
      gsl_vector_memcpy(dy, y_m);
      gsl_vector_sub(dy, y_i);

      /* Compute cost function... */
      *chisq = cost_function(dx, dy, s_a_inv, sig_eps_inv);

      /* Modify Levenberg-Marquardt parameter... */
      if (*chisq > chisq_old) {
        lmpar *= 10;
        gsl_vector_sub(x_i, x_step);
      } else {
        lmpar /= 10;
        break;
      }
    }

    /* Get normalized step size in state space... */
    gsl_blas_ddot(x_step, b, &disq);
    disq /= (double) n;

    /* Convergence test... */
    if ((it == 1 || it % ret->kernel_recomp == 0) && disq < ret->conv_dmin)
      break;
  }

  /* ------------------------------------------------------------
     Finalize...
     ------------------------------------------------------------ */

  gsl_matrix_free(a);
  gsl_matrix_free(cov);
  gsl_matrix_free(k_i);
  gsl_matrix_free(s_a_inv);

  gsl_vector_free(b);
  gsl_vector_free(dx);
  gsl_vector_free(dy);
  gsl_vector_free(sig_eps_inv);
  gsl_vector_free(sig_formod);
  gsl_vector_free(sig_noise);
  gsl_vector_free(x_a);
  gsl_vector_free(x_i);
  gsl_vector_free(x_step);
  gsl_vector_free(y_aux);
  gsl_vector_free(y_i);
  gsl_vector_free(y_m);
}

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

void read_nc(
  char *filename,
  ncd_t * ncd) {

  int dimid, varid;

  size_t len;

  /* Open netCDF file... */
  printf("Read netCDF file: %s\n", filename);
  NC(nc_open(filename, NC_WRITE, &ncd->ncid));

  /* Read number of tracks... */
  NC(nc_inq_dimid(ncd->ncid, "L1_NTRACK", &dimid));
  NC(nc_inq_dimlen(ncd->ncid, dimid, &len));
  ncd->ntrack = (int) len;

  /* Read Level-1 data... */
  NC(nc_inq_varid(ncd->ncid, "l1_time", &varid));
  NC(nc_get_var_double(ncd->ncid, varid, ncd->l1_time[0]));
  NC(nc_inq_varid(ncd->ncid, "l1_lon", &varid));
  NC(nc_get_var_double(ncd->ncid, varid, ncd->l1_lon[0]));
  NC(nc_inq_varid(ncd->ncid, "l1_lat", &varid));
```

```
  NC(nc_get_var_double(ncd->ncid, varid, ncd->l1_lat[0]));
  NC(nc_inq_varid(ncd->ncid, "l1_sat_z", &varid));
  NC(nc_get_var_double(ncd->ncid, varid, ncd->l1_sat_z));
  NC(nc_inq_varid(ncd->ncid, "l1_sat_lon", &varid));
  NC(nc_get_var_double(ncd->ncid, varid, ncd->l1_sat_lon));
  NC(nc_inq_varid(ncd->ncid, "l1_sat_lat", &varid));
  NC(nc_get_var_double(ncd->ncid, varid, ncd->l1_sat_lat));
  NC(nc_inq_varid(ncd->ncid, "l1_nu", &varid));
  NC(nc_get_var_double(ncd->ncid, varid, ncd->l1_nu));
  NC(nc_inq_varid(ncd->ncid, "l1_rad", &varid));
  NC(nc_get_var_float(ncd->ncid, varid, ncd->l1_rad[0][0]));

  /* Read Level-2 data... */
  NC(nc_inq_varid(ncd->ncid, "l2_z", &varid));
  NC(nc_get_var_double(ncd->ncid, varid, ncd->l2_z[0][0]));
  NC(nc_inq_varid(ncd->ncid, "l2_press", &varid));
  NC(nc_get_var_double(ncd->ncid, varid, ncd->l2_p));
  NC(nc_inq_varid(ncd->ncid, "l2_temp", &varid));
  NC(nc_get_var_double(ncd->ncid, varid, ncd->l2_t[0][0]));
}

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

void read_ret_ctl(
  int argc,
  char *argv[],
  ctl_t * ctl,
  ret_t * ret) {

  int id, ig, iw;

  /* Iteration control... */
  ret->kernel_recomp =
    (int) scan_ctl(argc, argv, "KERNEL_RECOMP", -1, "3", NULL);
  ret->conv_itmax = (int) scan_ctl(argc, argv, "CONV_ITMAX", -1, "30", NULL);
  ret->conv_dmin = scan_ctl(argc, argv, "CONV_DMIN", -1, "0.1", NULL);

  for (id = 0; id < ctl->nd; id++)
    ret->err_formod[id] = scan_ctl(argc, argv, "ERR_FORMOD", id, "0", NULL);

  for (id = 0; id < ctl->nd; id++)
    ret->err_noise[id] = scan_ctl(argc, argv, "ERR_NOISE", id, "0", NULL);

  ret->err_press = scan_ctl(argc, argv, "ERR_PRESS", -1, "0", NULL);
  ret->err_press_cz = scan_ctl(argc, argv, "ERR_PRESS_CZ", -1, "-999", NULL);
  ret->err_press_ch = scan_ctl(argc, argv, "ERR_PRESS_CH", -1, "-999", NULL);

  ret->err_temp = scan_ctl(argc, argv, "ERR_TEMP", -1, "0", NULL);
  ret->err_temp_cz = scan_ctl(argc, argv, "ERR_TEMP_CZ", -1, "-999", NULL);
  ret->err_temp_ch = scan_ctl(argc, argv, "ERR_TEMP_CH", -1, "-999", NULL);

  for (ig = 0; ig < ctl->ng; ig++) {
    ret->err_q[ig] = scan_ctl(argc, argv, "ERR_Q", ig, "0", NULL);
    ret->err_q_cz[ig] = scan_ctl(argc, argv, "ERR_Q_CZ", ig, "-999", NULL);
    ret->err_q_ch[ig] = scan_ctl(argc, argv, "ERR_Q_CH", ig, "-999", NULL);
  }

  for (iw = 0; iw < ctl->nw; iw++) {
    ret->err_k[iw] = scan_ctl(argc, argv, "ERR_K", iw, "0", NULL);
    ret->err_k_cz[iw] = scan_ctl(argc, argv, "ERR_K_CZ", iw, "-999", NULL);
    ret->err_k_ch[iw] = scan_ctl(argc, argv, "ERR_K_CH", iw, "-999", NULL);
  }
}

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

void set_cov_apr(
  ret_t * ret,
  ctl_t * ctl,
  atm_t * atm,
  int *iqa,
  int *ipa,
  gsl_matrix * s_a) {

  gsl_vector *x_a;

  double ch, cz, rho, x0[3], x1[3];

  int ig, iw;

  size_t i, j, n;

  /* Get sizes... */
  n = s_a->size1;

  /* Allocate... */
  x_a = gsl_vector_alloc(n);
```

```
  /* Get sigma vector... */
  atm2x(ctl, atm, x_a, NULL, NULL);
  for (i = 0; i < n; i++) {
    if (iqa[i] == IDXP)
      gsl_vector_set(x_a, i, ret->err_press / 100 * gsl_vector_get(x_a, i));
    if (iqa[i] == IDXT)
      gsl_vector_set(x_a, i, ret->err_temp);
    for (ig = 0; ig < ctl->ng; ig++)
      if (iqa[i] == IDXQ(ig))
        gsl_vector_set(x_a, i, ret->err_q[ig] / 100 * gsl_vector_get(x_a, i));
    for (iw = 0; iw < ctl->nw; iw++)
      if (iqa[i] == IDXK(iw))
        gsl_vector_set(x_a, i, ret->err_k[iw]);
  }

  /* Check standard deviations... */
  for (i = 0; i < n; i++)
    if (gsl_pow_2(gsl_vector_get(x_a, i)) <= 0)
      ERRMSG("Check a priori data (zero standard deviation)!");

  /* Initialize diagonal covariance... */
  gsl_matrix_set_zero(s_a);
  for (i = 0; i < n; i++)
    gsl_matrix_set(s_a, i, i, gsl_pow_2(gsl_vector_get(x_a, i)));

  /* Loop over matrix elements... */
  for (i = 0; i < n; i++)
    for (j = 0; j < n; j++)
      if (i != j && iqa[i] == iqa[j]) {

        /* Initialize... */
        cz = ch = 0;

        /* Set correlation lengths for pressure... */
        if (iqa[i] == IDXP) {
          cz = ret->err_press_cz;
          ch = ret->err_press_ch;
        }

        /* Set correlation lengths for temperature... */
        if (iqa[i] == IDXT) {
          cz = ret->err_temp_cz;
          ch = ret->err_temp_ch;
        }

        /* Set correlation lengths for volume mixing ratios... */
        for (ig = 0; ig < ctl->ng; ig++)
          if (iqa[i] == IDXQ(ig)) {
            cz = ret->err_q_cz[ig];
            ch = ret->err_q_ch[ig];
          }

        /* Set correlation lengths for extinction... */
        for (iw = 0; iw < ctl->nw; iw++)
          if (iqa[i] == IDXK(iw)) {
            cz = ret->err_k_cz[iw];
            ch = ret->err_k_ch[iw];
          }

        /* Compute correlations... */
        if (cz > 0 && ch > 0) {

          /* Get Cartesian coordinates... */
          geo2cart(0, atm->lon[ipa[i]], atm->lat[ipa[i]], x0);
          geo2cart(0, atm->lon[ipa[j]], atm->lat[ipa[j]], x1);

          /* Compute correlations... */
          rho =
            exp(-DIST(x0, x1) / ch -
                fabs(atm->z[ipa[i]] - atm->z[ipa[j]]) / cz);

          /* Set covariance... */
          gsl_matrix_set(s_a, i, j, gsl_vector_get(x_a, i)
                         * gsl_vector_get(x_a, j) * rho);
        }
      }

  /* Free... */
  gsl_vector_free(x_a);
}

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

void set_cov_meas(
  ret_t * ret,
  ctl_t * ctl,
```

```
  obs_t * obs,
  gsl_vector * sig_noise,
  gsl_vector * sig_formod,
  gsl_vector * sig_eps_inv) {

  static obs_t obs_err;

  int id, ir;

  size_t i, m;

  /* Get size... */
  m = sig_eps_inv->size;

  /* Noise error (always considered in retrieval fit)... */
  copy_obs(ctl, &obs_err, obs, 1);
  for (ir = 0; ir < obs_err.nr; ir++)
    for (id = 0; id < ctl->nd; id++)
      obs_err.rad[id][ir]
        = (gsl_finite(obs->rad[id][ir]) ? ret->err_noise[id] : GSL_NAN);
  obs2y(ctl, &obs_err, sig_noise, NULL, NULL);

  /* Forward model error (always considered in retrieval fit)... */
  copy_obs(ctl, &obs_err, obs, 1);
  for (ir = 0; ir < obs_err.nr; ir++)
    for (id = 0; id < ctl->nd; id++)
      obs_err.rad[id][ir]
        = fabs(ret->err_formod[id] / 100 * obs->rad[id][ir]);
  obs2y(ctl, &obs_err, sig_formod, NULL, NULL);

  /* Total error... */
  for (i = 0; i < m; i++)
    gsl_vector_set(sig_eps_inv, i,
                   1 / sqrt(gsl_pow_2(gsl_vector_get(sig_noise, i))
                            + gsl_pow_2(gsl_vector_get(sig_formod, i))));

  /* Check standard deviations... */
  for (i = 0; i < m; i++)
    if (gsl_vector_get(sig_eps_inv, i) <= 0)
      ERRMSG("Check measurement errors (zero standard deviation)!");
}

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

void write_nc(
  char *filename,
  ncd_t * ncd) {

  int dimid[10], c_id, p_id, t_id, z_id;

  /* Create netCDF file... */
  printf("Write netCDF file: %s\n", filename);

  /* Read existing dimensions... */
  NC(nc_inq_dimid(ncd->ncid, "L1_NTRACK", &dimid[0]));
  NC(nc_inq_dimid(ncd->ncid, "L1_NXTRACK", &dimid[1]));

  /* Set define mode... */
  NC(nc_redef(ncd->ncid));

  /* Set new dimensions... */
  if (nc_inq_dimid(ncd->ncid, "RET_NP", &dimid[2]) != NC_NOERR)
    NC(nc_def_dim(ncd->ncid, "RET_NP", (size_t) ncd->np, &dimid[2]));

  /* Set new variables... */
  add_var(ncd->ncid, "ret_z", "km", "altitude", NC_FLOAT, &dimid[2], &z_id,
          1);
  add_var(ncd->ncid, "ret_press", "hPa", "pressure", NC_FLOAT, dimid, &p_id,
          2);
  add_var(ncd->ncid, "ret_temp", "K", "temperature", NC_FLOAT, dimid, &t_id,
          3);
  add_var(ncd->ncid, "ret_chisq", "1", "chi^2 value of fit", NC_FLOAT, dimid,
          &c_id, 2);

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

  /* Write data... */
  NC(nc_put_var_float(ncd->ncid, z_id, ncd->ret_z));
  NC(nc_put_var_float(ncd->ncid, p_id, ncd->ret_p));
  NC(nc_put_var_float(ncd->ncid, t_id, ncd->ret_t));
  NC(nc_put_var_float(ncd->ncid, c_id, ncd->ret_chisq));

  /* Close netCDF file... */
  NC(nc_close(ncd->ncid));
}
```

5.19 spec2tab.c File Reference

Functions

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

5.19.1 Function Documentation

main()

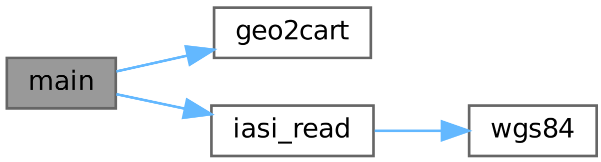
```
int main (
            int argc,
            char * argv[ ] )
```

Definition at line 3 of file spec2tab.c.

```
                  {

  static iasi_rad_t *iasi_rad;

  FILE *out;

  double dmin = 1e100, x0[3], x1[3];

  int ichan, track = -1, track2, xtrack = -1, xtrack2;

  /* Check arguments... */
  if (argc != 6)
    ERRMSG("Give parameters: <iasi_l1b_file> "
           "[index <track> <xtrack> | geo <lon> <lat>] <spec.tab>");

  /* Allocate... */
  ALLOC(iasi_rad, iasi_rad_t, 1);

  /* Read IASI data... */
  printf("Read IASI Level-1C data file: %s\n", argv[1]);
  iasi_read(argv[1], iasi_rad);

  /* Get indices... */
  if (argv[2][0] == 'i') {
    track = atoi(argv[3]);
    xtrack = atoi(argv[4]);
  }

  /* Find nearest footprint... */
  else {
    geo2cart(0, atof(argv[3]), atof(argv[4]), x0);
    for (track2 = 0; track2 < iasi_rad->ntrack; track2++)
      for (xtrack2 = 0; xtrack2 < L1_NXTRACK; xtrack2++) {
        geo2cart(0, iasi_rad->Longitude[track2][xtrack2],
                 iasi_rad->Latitude[track2][xtrack2], x1);
        if (DIST2(x0, x1) < dmin) {
          dmin = DIST2(x0, x1);
          track = track2;
          xtrack = xtrack2;
        }
      }
    if (dmin > 2500)
      ERRMSG("Geolocation not covered by granule!");
  }

  /* Check indices... */
  if (track < 0 || track >= iasi_rad->ntrack)
    ERRMSG("Along-track index out of range!");
  if (xtrack < 0 || xtrack >= L1_NXTRACK)
    ERRMSG("Across-track index out of range!");

  /* Create file... */
  printf("Write spectrum: %s\n", argv[5]);
  if (!(out = fopen(argv[5], "w")))
    ERRMSG("Cannot create file!");

  /* Write header... */
  fprintf(out,
          "# $1 = time (seconds since 01-JAN-2000, 00:00 UTC)\n"
          "# $2 = satellite longitude [deg]\n"
          "# $3 = satellite latitude [deg]\n"
```

```
           "# $4 = footprint longitude [deg]\n"
           "# $5 = footprint latitude [deg]\n"
           "# $6 = wavenumber [cm^-1]\n"
           "# $7 = brightness temperature [K]\n"
           "# $8 = radiance [W/(m^2 sr cm^-1)]\n\n");

  /* Write data... */
  for (ichan = 0; ichan < IASI_L1_NCHAN; ichan++)
    fprintf(out, "%.2f %g %g %g %g %g %g %g\n",
            iasi_rad->Time[track][xtrack],
            iasi_rad->Sat_lon[track],
            iasi_rad->Sat_lat[track],
            iasi_rad->Longitude[track][xtrack],
            iasi_rad->Latitude[track][xtrack],
            iasi_rad->freq[ichan],
            BRIGHT(iasi_rad->Rad[track][xtrack][ichan],
                   iasi_rad->freq[ichan]),
            iasi_rad->Rad[track][xtrack][ichan]);

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

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

  return EXIT_SUCCESS;
}
```

Here is the call graph for this function:



5.20 spec2tab.c

Go to the documentation of this file.

```
#include "libiasi.h"

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

  static iasi_rad_t *iasi_rad;

  FILE *out;

  double dmin = 1e100, x0[3], x1[3];

  int ichan, track = -1, track2, xtrack = -1, xtrack2;

  /* Check arguments... */
  if (argc != 6)
    ERRMSG("Give parameters: <iasi_l1b_file> "
           "[index <track> <xtrack> | geo <lon> <lat>] <spec.tab>");

  /* Allocate... */
  ALLOC(iasi_rad, iasi_rad_t, 1);

  /* Read IASI data... */
  printf("Read IASI Level-1C data file: %s\n", argv[1]);
  iasi_read(argv[1], iasi_rad);

  /* Get indices... */
  if (argv[2][0] == 'i') {
```

```
    track = atoi(argv[3]);
    xtrack = atoi(argv[4]);
  }

  /* Find nearest footprint... */
  else {
    geo2cart(0, atof(argv[3]), atof(argv[4]), x0);
    for (track2 = 0; track2 < iasi_rad->ntrack; track2++)
      for (xtrack2 = 0; xtrack2 < L1_NXTRACK; xtrack2++) {
        geo2cart(0, iasi_rad->Longitude[track2][xtrack2],
                 iasi_rad->Latitude[track2][xtrack2], x1);
        if (DIST2(x0, x1) < dmin) {
          dmin = DIST2(x0, x1);
          track = track2;
          xtrack = xtrack2;
        }
      }
    if (dmin > 2500)
      ERRMSG("Geolocation not covered by granule!");
  }

  /* Check indices... */
  if (track < 0 || track >= iasi_rad->ntrack)
    ERRMSG("Along-track index out of range!");
  if (xtrack < 0 || xtrack >= L1_NXTRACK)
    ERRMSG("Across-track index out of range!");

  /* Create file... */
  printf("Write spectrum: %s\n", argv[5]);
  if (!(out = fopen(argv[5], "w")))
    ERRMSG("Cannot create file!");

  /* Write header... */
  fprintf(out,
          "# $1 = time (seconds since 01-JAN-2000, 00:00 UTC)\n"
          "# $2 = satellite longitude [deg]\n"
          "# $3 = satellite latitude [deg]\n"
          "# $4 = footprint longitude [deg]\n"
          "# $5 = footprint latitude [deg]\n"
          "# $6 = wavenumber [cm^-1]\n"
          "# $7 = brightness temperature [K]\n"
          "# $8 = radiance [W/(m^2 sr cm^-1)]\n\n");

  /* Write data... */
  for (ichan = 0; ichan < IASI_L1_NCHAN; ichan++)
    fprintf(out, "%.2f %g %g %g %g %g %g %g\n",
            iasi_rad->Time[track][xtrack],
            iasi_rad->Sat_lon[track],
            iasi_rad->Sat_lat[track],
            iasi_rad->Longitude[track][xtrack],
            iasi_rad->Latitude[track][xtrack],
            iasi_rad->freq[ichan],
            BRIGHT(iasi_rad->Rad[track][xtrack][ichan],
                   iasi_rad->freq[ichan]),
            iasi_rad->Rad[track][xtrack][ichan]);

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

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

  return EXIT_SUCCESS;
}
```

Index