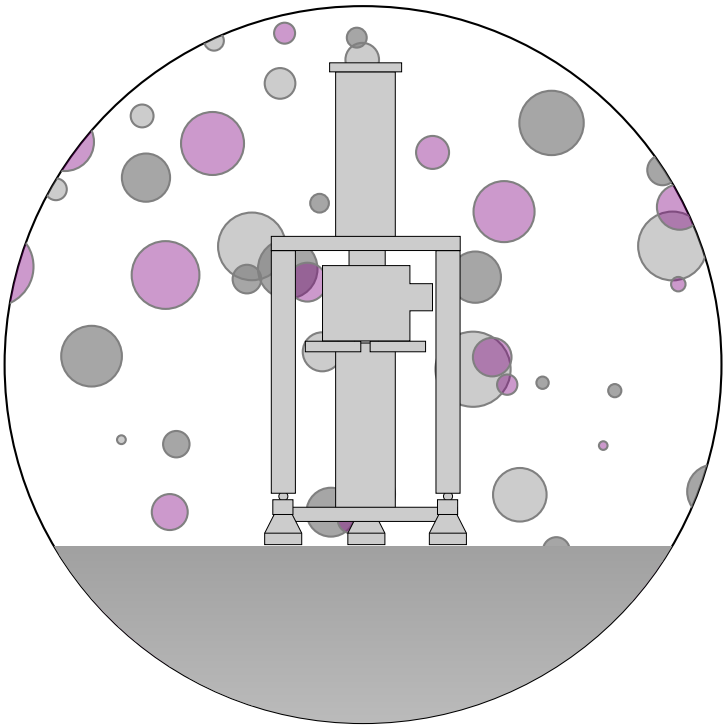


grat Manual

Marcin Rajner
Politechnika Warszawska | Warsaw University of Technology

Contents

Chapter 1

grat overview

1.1 Purpose

This program was created to make computation of atmospheric gravity correction easier. Still developing. Consider visiting later...

Version

TESTING!

Date

2013-01-12

Author

Marcin Rajner
Politechnika Warszawska | Warsaw University of Technology

Warning

This program is written in Fortran90 standard but uses some featerus of 2003 specification (e.g., `'newunit='`). It was also written for `Intel Fortran Compiler` hence some commands can be unavailable for other compilers (e.g., <integer_parameter> for `IO` statements. This should be easily modifiable according to your output needs. Also you need to have `iso_fortran_env` module available to guess the number of output_unit for your compiler. When you don't want a `log_file` and you don't switch `verbose` all unneceserry information whitch are normally collected goes to `/dev/null` file. This is *nix system default trash. For other system or file system organization, please change this value in `get_cmd_line` module.

Attention

`grat` and value_check needs a `netCDF` library net

1.2 Usage

After sucsesfull compiling make sure the executables are in your search path

There is main program `grat` and some utilities program. For the options see the appropriate help:

- grat

- value_check
- polygon_check

Chapter 2

grat

Chapter 3

ilustration

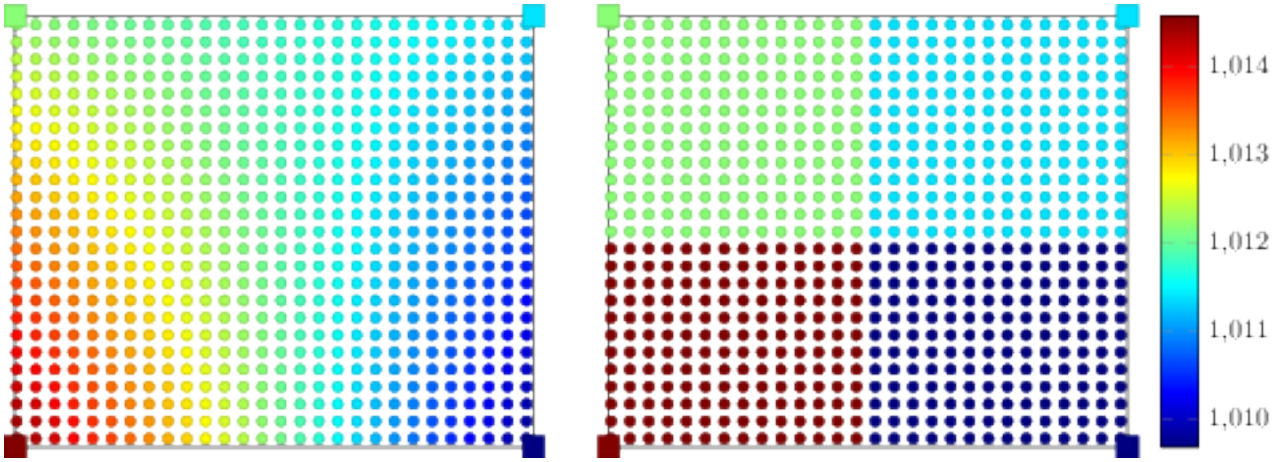


Figure 3.1: example

Chapter 4

External resources

- `project page` (git repository)
- html version of this manual give source for grant presentation
- `[pdf]` command line options (in Polish)

Chapter 5

polygon_check

This program can be used to check the default behaviour of point selection used by module grat_polygon

Chapter 6

value_check

Chapter 7

Todo List

Subprogram constants::ispline (u, x, y, b, c, d, n)

give source

Subprogram constants::jd (year, month, day, hh, mm, ss)

mjd!

Subprogram constants::spline (x, y, b, c, d, n)

give source

Subprogram get_cmd_line::is_numeric (string)

Add source name

Subprogram get_cmd_line::parse_green (cmd_line_entry)

add maximum minimum distances for integration

make it mulitichoice: -Lfile:s,file2:b ...

when no given take defaults

rozbudować

Subprogram mod_green::convolve_moreverbose (latin, lonin, azimuth, azstep, distance, distancestep)

site height from model

Chapter 8

Data Type Index

8.1 Data Types List

Here are the data types with brief descriptions:

Chapter 9

File Index

9.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 10

Data Type Documentation

10.1 get_cmd_line::additional_info Type Reference

Public Attributes

- character(len=55), dimension(:),
  allocatable **names**

10.1.1 Detailed Description

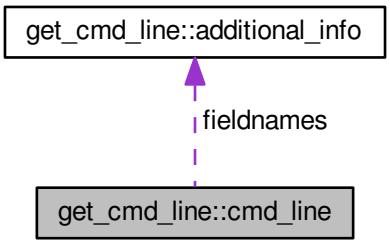
Definition at line 58 of file get_cmd_line.f90.

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

- grat/src/get_cmd_line.f90

10.2 get_cmd_line::cmd_line Type Reference

Collaboration diagram for get_cmd_line::cmd_line:



Public Attributes

- character(2) **switch**

- integer **fields**
- character(len=255), dimension(:), allocatable **field**
- type(additional_info), dimension(:), allocatable **fieldnames**

10.2.1 Detailed Description

Definition at line 61 of file get_cmd_line.f90.

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

- grat/src/get_cmd_line.f90

10.3 constants Module Reference

Public Member Functions

- subroutine spline_interpolation (x, y, x_interpolated, y_interpolated)

  For given vectors x1, y1 and x2, y2 it gives x2interpolated for x1.
- subroutine spline (x, y, b, c, d, n)

  This subroutine was taken from.
- real function ispline (u, x, y, b, c, d, n)

  This subroutine was taken from.
- integer function ntokens (line)

  taken from ArkM http://www.tek-tips.com/viewthread.cfm?qid=1688013
- subroutine skip_header (unit, comment_char_optional)

  This routine skips the lines with comment chars (default '#') from opened files (unit) to read.
- real function jd (year, month, day, hh, mm, ss)

  downloaded from http://aa.usno.navy.mil/faq/docs/jd_formula.php
- real(dp) function **mjd** (date)
- subroutine **invmjd** (mjd, date)

Public Attributes

- integer, parameter dp = 8

  real (kind_real) => real (kind = 8)
- integer, parameter sp = 4

  real (kind_real) => real (kind = 4)
- real(dp), parameter t0 = 288.15

  surface temperature for standard atmosphere [K] (15 degC)
- real(dp), parameter g0 = 9.80665

  mean gravity on the Earth [m/s2]
- real(dp), parameter r0 = 6356.766

  Earth radius (US Std. atm. 1976) [km].
- real(dp), parameter p0 = 1013.25

  surface pressure for standard Earth [hPa]
- real(dp), parameter g = 6.672e-11

  Cavendish constant $[m^3/kg/s^2]$.
- real(dp), parameter r_air = 287.05

  dry air constant [J/kg/K]

- real(dp), parameter pi = 4∗atan(1.)

  pi = 3.141592... [ ]
- real(dp), parameter rho_crust = 2670.

  mean density of crust [kg/m3]
- real(dp), parameter rho_earth = 5500.

  mean density of Earth [kg/m3]

10.3.1 Detailed Description

Definition at line 5 of file constants.f90.

10.3.2 Member Function/Subroutine Documentation

10.3.2.1 real function constants::ispline (real(dp) *u,* real(dp), dimension(n) *x,* real(dp), dimension(n) *y,* real(dp), dimension(n) *b,* real(dp), dimension(n) *c,* real(dp), dimension(n) *d,* integer *n*)

This subroutine was taken from.

Todo give source

Definition at line 158 of file constants.f90.

10.3.2.2 real function constants::jd (integer, intent(in) *year,* integer, intent(in) *month,* integer, intent(in) *day,* integer, intent(in) *hh,* integer, intent(in) *mm,* integer, intent(in) *ss*)

downloaded from `http://aa.usno.navy.mil/faq/docs/jd_formula.php`

Todo mjd!

Definition at line 253 of file constants.f90.

10.3.2.3 subroutine constants::spline (real(dp), dimension(n) *x,* real(dp), dimension(n) *y,* real(dp), dimension(n) *b,* real(dp), dimension(n) *c,* real(dp), dimension(n) *d,* integer *n*)

This subroutine was taken from.

Todo give source

Definition at line 68 of file constants.f90.

10.3.2.4 subroutine constants::spline_interpolation (real(dp), dimension (:), intent(in), allocatable *x,* real(dp), dimension (:), intent(in), allocatable *y,* real(dp), dimension (:), intent(in), allocatable *x_interpolated,* real(dp), dimension (:), intent(out), allocatable *y_interpolated*)

For given vectors x1, y1 and x2, y2 it gives x2interpolated for x1.

uses `ispline` and `spline` subroutines

Definition at line 28 of file constants.f90.

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

- grat/src/constants.f90

10.4 get_cmd_line::dateandmjd Type Reference

Public Attributes

- real(dp) **mjd**
- integer, dimension(6) **date**

10.4.1 Detailed Description

Definition at line 46 of file get_cmd_line.f90.

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

- grat/src/get_cmd_line.f90

10.5 get_cmd_line::file Type Reference

Public Attributes

- character(:), allocatable **name**
- character(len=50), dimension(5) **names** = ["z"
- integer **unit** = output_unit
- logical **if** = .false.
- logical **first_call** = .true.
- real(sp), dimension(4) **limits**
- real(sp), dimension(:), allocatable **lat**
- real(sp), dimension(:), allocatable **lon**
- real(sp), dimension(:), allocatable **time**
- real(sp), dimension(:), allocatable **level**
- integer, dimension(:,:), allocatable **date**
- real(sp), dimension(2) **latrange**
- real(sp), dimension(2) **lonrange**
- logical **if_constant_value**
- real(sp) **constant_value**
- real(sp), dimension(:,:,:), allocatable data

  4 dimension - lat , lon , level , mjd
- integer **ncid**
- integer **interpolation** = 1

10.5.1 Detailed Description

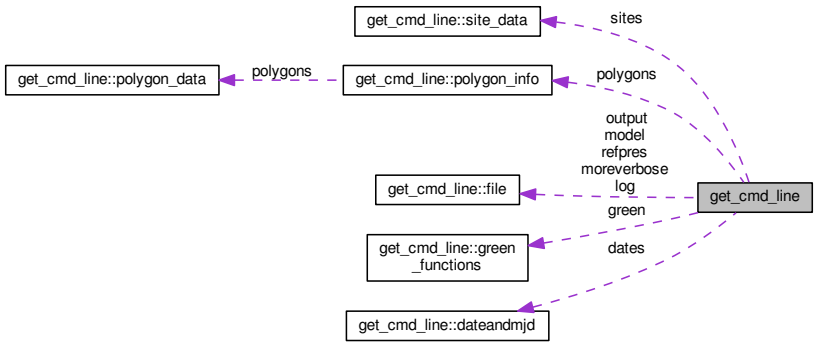
Definition at line 93 of file get_cmd_line.f90.

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

- grat/src/get_cmd_line.f90

10.6 get_cmd_line Module Reference

Collaboration diagram for get_cmd_line:



Data Types

- type additional_info
- type cmd_line
- type dateandmjd
- type file
- type green_functions
- type polygon_data
- type polygon_info
- type site_data

Public Member Functions

- subroutine intro (program_calling)

  This subroutine counts the command line arguments.
- subroutine if_minimum_args (program_calling)

  Check if at least all obligatory command line arguments were given if not print warning.
- logical function if_switch_program (program_calling, switch)

  This function is true if switch is used by calling program or false if it is not.
- subroutine parse_option (cmd_line_entry, program_calling)

  This subroutine counts the command line arguments and parse appropriately.
- subroutine parse_green (cmd_line_entry)

  This subroutine parse -G option i.e. reads Greens function.
- integer function count_separator (dummy, separator)

  change the paths accordingly
- subroutine get_cmd_line_entry (dummy, cmd_line_entry, program_calling)

  This subroutine fills the fields of command line entry for every input arg.
- subroutine get_model_info (model, cmd_line_entry, field)

  This subroutine fills the model info.
- subroutine parse_gmt_like_boundaries (cmd_line_entry)

  This subroutine checks if given limits for model are proper.
- subroutine read_site_file (file_name)

Read site list from file.

- subroutine parse_dates (cmd_line_entry)

  Parse date given as 20110503020103 to yy mm dd hh mm ss and mjd.
- subroutine **string2date** (string, date)
- logical function is_numeric (string)

  Auxiliary function.
- logical function file_exists (string)

  Check if file exists , return logical.
- real(dp) function d2r (degree)

  degree -> radian
- real(dp) function r2d (radian)

  radian -> degree
- subroutine print_version (program_calling)

  Print version of program depending on program calling.
- subroutine print_settings (program_calling)

  Print settings.
- subroutine **print_help** (program_calling)
- subroutine **print_warning** (warn, unit)
- integer function nmodels (model)

  Counts number of properly specified models.

Public Attributes

- type(green_functions), dimension(:), allocatable **green**
- integer, dimension(2) **denser** = [1
- type(polygon_info), dimension(2) **polygons**
- real(kind=4) **cpu_start**
- real(kind=4) cpu_finish

  for time execution of program
- type(dateandmjd), dimension(:), allocatable **dates**
- type(site_data), dimension(:), allocatable **sites**
- integer fileunit_tmp

  unit of scratch file
- integer, dimension(8) execution_date

  To give time stamp of execution.
- character(len=2) method = "2D"

  computation method
- character(:), allocatable **filename_site**
- integer **fileunit_site**
- type(file) **log**
- type(file) **output**
- type(file) **refpres**
- type(file), dimension(:), allocatable **model**
- type(file) **moreverbose**
- character(len=40), dimension(5) **model_names** = ["pressure_surface"
- character(len=5), dimension(5) **green_names** = ["GN "
- logical if_verbose = .false.

  whether print all information

- logical **inverted_barometer** = .true.
- character(50), dimension(2) **interpolation_names** = ["nearest"
- character(len=255), parameter **form_header** = '(60("#"))'
- character(len=255), parameter **form_separator** = '(60("-"))'
- character(len=255), parameter **form_inheader** = '(("#"),1x,a56,1x,("#"))'
- character(len=255), parameter **form_60** = "(a,100(1x,g0))"
- character(len=255), parameter **form_61** = "(2x,a,100(1x,g0))"
- character(len=255), parameter **form_62** = "(4x,a,100(1x,g0))"
- character(len=255), parameter **form_63** = "(6x,100(x,g0))"
- character(len=255), parameter **form_64** = "(4x,4x,a,4x,a)"

10.6.1 Detailed Description

Definition at line 8 of file get_cmd_line.f90.

10.6.2 Member Function/Subroutine Documentation

10.6.2.1 integer function get_cmd_line::count_separator (character(∗), intent(in) *dummy,* character(1), intent(in), optional *separator*)

change the paths accordingly

Counts occurence of character (separator, default comma) in string

Definition at line 508 of file get_cmd_line.f90.

10.6.2.2 subroutine get_cmd_line::intro (character(len=∗) *program_calling*)

This subroutine counts the command line arguments.

Depending on command line options set all initial parameters and reports it

Definition at line 171 of file get_cmd_line.f90.

10.6.2.3 logical function get_cmd_line::is_numeric (character(len=∗), intent(in) *string*)

Auxiliary function.

check if argument given as string is valid number Taken from www

Todo Add source name

Definition at line 861 of file get_cmd_line.f90.

10.6.2.4 subroutine get_cmd_line::parse_dates (type(cmd_line) *cmd_line_entry*)

Parse date given as 20110503020103 to yy mm dd hh mm ss and mjd.

Warning

decimal seconds are not allowed

Definition at line 785 of file get_cmd_line.f90.

10.6.2.5 subroutine get_cmd_line::parse_green (type (cmd_line) *cmd_line_entry*)

This subroutine parse -G option i.e. reads Greens function.

Todo add maximum minimum distances for integration

Todo make it mulitichoice: -Lfile:s,file2:b ...

Todo when no given take defaults

Todo rozbudować

Definition at line 409 of file get_cmd_line.f90.

10.6.2.6 subroutine get_cmd_line::read_site_file (character(len=*), intent(in) *file_name*)

Read site list from file.

checks for arguments and put it into array `sites`

Definition at line 699 of file get_cmd_line.f90.

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

- grat/src/get_cmd_line.f90

10.7 get_cmd_line::green_functions Type Reference

Public Attributes

- real(dp), dimension(:), allocatable **distance**
- real(dp), dimension(:), allocatable **data**
- logical **if**

10.7.1 Detailed Description

Definition at line 18 of file get_cmd_line.f90.

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

- grat/src/get_cmd_line.f90

10.8 mod_aggf Module Reference

Public Member Functions

- subroutine compute_aggfdt (psi, aggfdt, delta_, aggf)

  Compute first derivative of AGGF with respect to temperature for specific angular distance (psi)
- subroutine read_tabulated_green (table, author)

  Wczytuje tablice danych AGGF.
- subroutine compute_aggf (psi, aggf_val, hmin, hmax, dh, if_normalization, t_zero, h, first_derivative_h, first_derivative_z, fels_type)

This subroutine computes the value of atmospheric gravity green functions (AGGF) on the basis of spherical distance (psi)

- subroutine standard_density (height, rho, t_zero, fels_type)

  first derivative (respective to station height) micro Gal height / km
- subroutine standard_pressure (height, pressure, p_zero, t_zero, h_zero, if_simplificated, fels_type, inverted)

  Computes pressure [hPa] for specific height.
- subroutine transfer_pressure (height1, height2, pressure1, pressure2, temperature, polish_meteo)
- subroutine standard_gravity (height, g)

  Compute gravity acceleration of the Earth for the specific height using formula.
- real(sp) function geop2geom (geopotential_height)

  Compute geometric height from geopotential heights.
- subroutine surface_temperature (height, temperature1, temperature2, fels_type, tolerance)

  Iterative computation of surface temp. from given height using bisection method.
- subroutine standard_temperature (height, temperature, t_zero, fels_type)

  Compute standard temperature [K] for specific height [km].
- real function gn_thin_layer (psi)

  Compute AGGF GN for thin layer.
- integer function size_ntimes_denser (size_original, ndenser)

  returns numbers of arguments for n times denser size
- real(dp) function bouger (R_opt)

  Bouger plate computation.
- real(dp) function simple_def (R)

  Bouger plate computation see eq. page 288.

10.8.1 Detailed Description

Definition at line 9 of file mod_aggf.f90.

10.8.2 Member Function/Subroutine Documentation

10.8.2.1 real(dp) function mod_aggf::bouger (real(dp), optional *R_opt*)

Bouger plate computation.

Parameters

r_opt	height of point above the cylinder

Definition at line 479 of file mod_aggf.f90.

10.8.2.2 subroutine mod_aggf::compute_aggf (real(dp), intent(in) *psi,* real(dp), intent(out) *aggf_val,* real(dp), intent(in), optional *hmin,* real(dp), intent(in), optional *hmax,* real(dp), intent(in), optional *dh,* logical, intent(in), optional *if_normalization,* real(dp), intent(in), optional *t_zero,* real(dp), intent(in), optional *h,* logical, intent(in), optional *first_derivative_h,* logical, intent(in), optional *first_derivative_z,* character (len=*), intent(in), optional *fels_type*)

This subroutine computes the value of atmospheric gravity green functions (AGGF) on the basis of spherical distance (psi)

Parameters

in	*psi*	spherical distance from site [degree]
in	*h*	station height [km] (default=0)

Parameters

hmin	minimum height, starting point [km] (default=0)
hmax	maximum height. eding point [km] (default=60)
dh	integration step [km] (default=0.0001 -> 10 cm)
t_zero	temperature at the surface [K] (default=288.15=t0)

Definition at line 110 of file mod_aggf.f90.

10.8.2.3 subroutine mod_aggf::compute_aggfdt (real(dp), intent(in) *psi,* real(dp), intent(out) *aggfdt,* real(dp), intent(in), optional *delta_,* logical, intent(in), optional *aggf*)

Compute first derivative of AGGF with respect to temperature for specific angular distance (psi)

optional argument define (-dt;-dt) range See equation 19 in Huang et al. [2005] Same simple method is applied for aggf(gn) if `aggf` optional parameter is set to .true.

Warning

Please do not use `aggf`=.true. this option was added only for testing some numerical routines

Definition at line 27 of file mod_aggf.f90.

10.8.2.4 real function mod_aggf::gn_thin_layer (real(dp), intent(in) *psi*)

Compute AGGF GN for thin layer.

Simple function added to provide complete module but this should not be used for atmosphere layer See eq p. 491 in Merriam [1992]

Definition at line 455 of file mod_aggf.f90.

10.8.2.5 subroutine mod_aggf::read_tabulated_green (real(dp), dimension(:,:), intent(inout), allocatable *table,* character (len = *), intent(in), optional *author*)

Wczytuje tablice danych AGGF.

- merriam Merriam [1992]
- huang Huang et al. [2005]
- rajner **?**

This is just quick solution for `example_aggf` program in `grat` see the more general routine `parse_green()`

Definition at line 66 of file mod_aggf.f90.

10.8.2.6 real(dp) function mod_aggf::simple_def (real(dp) *R*)

Bouger plate computation see eq. page 288.

Warburton and Goodkind [1977]

Definition at line 501 of file mod_aggf.f90.

10.8.2.7 integer function mod_aggf::size_ntimes_denser (integer, intent(in) *size_original,* integer, intent(in) *ndenser*)

returns numbers of arguments for n times denser size

i.e. * * * * –> * . . * . . * . . * (3 times denser)

Definition at line 470 of file mod_aggf.f90.

10.8.2.8 subroutine mod_aggf::standard_density (real(dp), intent(in) *height,* real(dp), intent(out) *rho,* real(dp), intent(in), optional *t_zero,* character(len = 22), optional *fels_type*)

first derivative (respective to station height) micro Gal height / km

direct derivative of equation 20 Huang et al. [2005] first derivative (respective to column height) according to equation 26 in Huang et al. [2005] micro Gal / hPa / km aggf GN micro Gal / hPa if you put the optional parameter `if_normalization=`.false. this block will be skipped by default the normalization is applied according to Merriam [1992] Compute air density for given altitude for standard atmosphere

using formulae 12 in Huang et al. [2005]

Parameters

in	*height*	height [km]
in	*t_zero*	if this parameter is given

Definition at line 194 of file mod_aggf.f90.

10.8.2.9 subroutine mod_aggf::standard_gravity (real(dp), intent(in) *height,* real(dp), intent(out) *g*)

Compute gravity acceleration of the Earth for the specific height using formula.

see Comitee on extension of the Standard Atmosphere [1976]

Definition at line 301 of file mod_aggf.f90.

10.8.2.10 subroutine mod_aggf::standard_pressure (real(dp), intent(in) *height,* real(dp), intent(out) *pressure,* real(dp), intent(in), optional *p_zero,* real(dp), intent(in), optional *t_zero,* real(dp), intent(in), optional *h_zero,* logical, intent(in), optional *if_simplificated,* character(len = 22), optional *fels_type,* logical, intent(in), optional *inverted*)

Computes pressure [hPa] for specific height.

See Comitee on extension of the Standard Atmosphere [1976] or Huang et al. [2005] for details. Uses formulae 5 from Huang et al. [2005]. Simplified method if optional argument if_simplificated = .true.

Definition at line 219 of file mod_aggf.f90.

10.8.2.11 subroutine mod_aggf::standard_temperature (real(dp), intent(in) *height,* real(dp), intent(out) *temperature,* real(dp), intent(in), optional *t_zero,* character (len=*), intent(in), optional *fels_type*)

Compute standard temperature [K] for specific height [km].

if t_zero is specified use this as surface temperature otherwise use T0. A set of predifined temperature profiles ca be set using optional argument fels_type Fels [1986]

Parameters

in	*fels_type*	• US standard atmosphere (default) • tropical • subtropical_summer • subtropical_winter • subarctic_summer • subarctic_winter

Definition at line 369 of file mod_aggf.f90.

10.8.2.12 subroutine mod_aggf::transfer_pressure (real (dp), intent(in) *height1,* real (dp), intent(in) *height2,* real (dp), intent(in) *pressure1,* real(dp), intent(out) *pressure2,* real (dp), intent(in), optional *temperature,* logical, intent(in), optional *polish_meteo*)

Warning

OBSOLETE ROUTINE – use `standard_pressure()` instead with optional args

Definition at line 267 of file mod_aggf.f90.

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

- grat/src/mod_aggf.f90

10.9 mod_data Module Reference

This modele gives routines to read, and write data.

Public Member Functions

- subroutine put_grd (model, time, level, filename_opt)

  Put netCDF COARDS compliant.
- subroutine read_netcdf (model)

  Read netCDF file into memory.
- subroutine get_variable (model, date)

  Get values from netCDF file for specified variables.
- subroutine nctime2date (model)

  Change time in netcdf to dates.
- subroutine get_dimension (model, i)

  Get dimension, allocate memory and fill with values.
- subroutine unpack_netcdf (model)

  Unpack variable.
- subroutine check (status)

  Check the return code from netCDF manipulation.
- subroutine get_value (model, lat, lon, val, level, method)

  Returns the value from model file.
- real function **bilinear** (x, y, aux)
- subroutine **invspt** (alp, del, b, rlong)

10.9.1 Detailed Description

This modele gives routines to read, and write data.

The netCDF format is widely used in geoscienses. Moreover it is self-describing and machine independent. It also allows for reading and writing small subset of data therefore very efficient for large datafiles (this case) net

Definition at line 10 of file mod_data.f90.

10.9.2 Member Function/Subroutine Documentation

10.9.2.1 subroutine mod_data::check (integer, intent(in) *status*)

Check the return code from netCDF manipulation.

from net

Definition at line 216 of file mod_data.f90.

10.9.2.2 subroutine mod_data::put_grd (type (file) *model,* integer *time,* integer *level,* character (∗), intent(in), optional *filename_opt*)

Put netCDF COARDS compliant.

for GMT drawing

Definition at line 25 of file mod_data.f90.

10.9.2.3 subroutine mod_data::unpack_netcdf (type(file) *model*)

Unpack variable.

from net

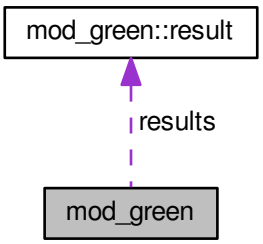
Definition at line 198 of file mod_data.f90.

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

- grat/src/mod_data.f90

10.10 mod_green Module Reference

Collaboration diagram for mod_green:



Data Types

- type result

Public Member Functions

- subroutine **green_unification** (green, green_common, denser)
- subroutine **spher_area** (distance, ddistance, azstp, area)
- subroutine **spher_trig** (latin, lonin, distance, azimuth, latout, lonout)
- subroutine **convolve** (site, green, results, denserdist, denseraz)
- subroutine convolve_moreverbose (latin, lonin, azimuth, azstep, distance, distancestep)

Public Attributes

- real(dp), dimension(:,:),
  allocatable **green_common**
- type(result), dimension(:),
  allocatable **results**

10.10.1 Detailed Description

Definition at line 1 of file mod_green.f90.

10.10.2 Member Function/Subroutine Documentation

10.10.2.1 subroutine mod_green::convolve_moreverbose (real(sp), intent(in) *latin,* real(sp), intent(in) *lonin,* real(sp), intent(in) *azimuth,* real(sp), intent(in) *azstep,* real(dp) *distance,* real(dp) *distancestep*)

Todo site height from model

Definition at line 183 of file mod_green.f90.

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

- grat/src/mod_green.f90

10.11 mod_polygon Module Reference

Public Member Functions

- subroutine read_polygon (polygon)

  Reads polygon data.
- subroutine chkgon (rlong, rlat, polygon, iok)

  check if point is in closed polygon
- integer function **if_inpoly** (x, y, coords)
- integer function ncross (x1, y1, x2, y2)

  finds whether the segment from point 1 to point 2 crosses the negative x-axis or goes through the origin (this is the signed crossing number)

10.11.1 Detailed Description

Definition at line 1 of file mod_polygon.f90.

10.11.2 Member Function/Subroutine Documentation

10.11.2.1 subroutine mod_polygon::chkgon (real(sp), intent(in) *rlong,* real(sp), intent(in) *rlat,* type(polygon_info), intent(in) *polygon,* integer, intent(out) *iok*)

check if point is in closed polygon

if it is first call it loads the model into memory inspired by spotl Agnew [1997] adopted to grat and Fortran90 syntax From original description

Definition at line 82 of file mod_polygon.f90.

10.11.2.2 integer function mod_polygon::ncross (real(sp), intent(in) *x1,* real(sp), intent(in) *y1,* real(sp), intent(in) *x2,* real(sp), intent(in) *y2*)

finds whether the segment from point 1 to point 2 crosses the negative x-axis or goes through the origin (this is the signed crossing number)

```
return value       nature of crossing
    4               segment goes through the origin
    2               segment crosses from below
    1               segment ends on -x axis from below
                     or starts on it and goes up
    0               no crossing
   -1               segment ends on -x axis from above
                     or starts on it and goes down
   -2               segment crosses from above
```

taken from spotl Agnew [1997] slightly modified

Definition at line 196 of file mod_polygon.f90.

10.11.2.3 subroutine mod_polygon::read_polygon (type(polygon_info) *polygon*)

Reads polygon data.

inspired by spotl Agnew [1997]

Definition at line 12 of file mod_polygon.f90.

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

- grat/src/mod_polygon.f90

10.12 get_cmd_line::polygon_data Type Reference

Public Attributes

- logical **use**
- real(sp), dimension(:,:), allocatable **coords**

10.12.1 Detailed Description

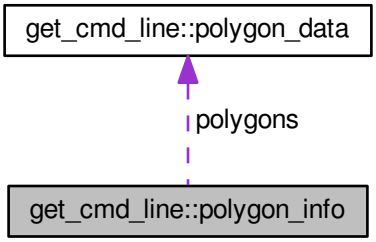
Definition at line 29 of file get_cmd_line.f90.

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

- grat/src/get_cmd_line.f90

10.13 get_cmd_line::polygon_info Type Reference

Collaboration diagram for get_cmd_line::polygon_info:



Public Attributes

- integer **unit**
- character(:), allocatable **name**
- type(polygon_data), dimension(:), allocatable **polygons**
- logical **if**

10.13.1 Detailed Description

Definition at line 34 of file get_cmd_line.f90.

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

- grat/src/get_cmd_line.f90

10.14 mod_green::result Type Reference

Public Attributes

- real(sp) **n** = 0.
- real(sp) **dt** = 0.
- real(sp) **e** = 0.
- real(sp) **dh** = 0.
- real(sp) **dz** = 0.

10.14.1 Detailed Description

Definition at line 9 of file mod_green.f90.

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

- grat/src/mod_green.f90

10.15 get_cmd_line::site_data Type Reference

Public Attributes

- character(:), allocatable **name**
- real(sp) **lat**
- real(sp) **lon**
- real(sp) **height**

10.15.1 Detailed Description

Definition at line 71 of file get_cmd_line.f90.

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

- grat/src/get_cmd_line.f90

Chapter 11

File Documentation

11.1 grat/doc/interpolation_ilustration.sh File Reference

Variables

- set o nounset for co in n b do **if** [${co}="b"]
- then **interp**

11.1.1 Detailed Description

Definition in file interpolation_ilustration.sh.

11.1.2 Variable Documentation

11.1.2.1 then interp

Initial value:

```
2
   else
     interp=1
   fi
  ../bin/value_check -F ../data/ncep_reanalysis/pres.sfc.2011.nc:pres
    -S 2.51/4.99/0.05/2.45
```

Definition at line 17 of file interpolation_ilustration.sh.

11.2 interpolation_ilustration.sh

```
00001 #!/bin/bash -
00002 #
      ================================================================================
00003 #          FILE: interpolation_ilustration.sh
00004 #         USAGE: ./interpolation_ilustration.sh
00005 #   DESCRIPTION:
00006 #       OPTIONS: ---
00007 #        AUTHOR: mrajner
00008 #       CREATED: 05.12.2012 10:38:30 CET
00009 #      REVISION:  ---
00010 #
      ================================================================================
00011
00012 ## \file
00013 set -o nounset                              # Treat unset variables as an error
00014 for co in n b
00015 do
```

```
00016    if [ ${co} = "b" ] ; then
00017      interp=2
00018    else
00019      interp=1
00020    fi
00021   ../bin/value_check -F ../data/ncep_reanalysis/pres.sfc.2011.nc:pres \
00022     -S 2.51/4.99/0.05/2.45,0.091,0.1 -I ${interp} \
00023     -o interp${co}1.dat   -L interpl1.dat :b
00024 done
00025   perl -n -i -e 'print if $. <= 4' interpl1.dat
00026
```

11.3 grat/src/constants.f90 File Reference

This module define some constant values used.

Data Types

- module constants

11.3.1 Detailed Description

This module define some constant values used.

Definition in file constants.f90.

11.4 constants.f90

```
00001 !
      ================================================================================
00002 !> \file
00003 !! This module define some constant values used
00004 !
      ================================================================================
00005 module constants
00006
00007   implicit none
00008   integer , parameter :: dp = 8 !< real (kind_real) => real (kind = 8 )
00009   integer , parameter :: sp = 4 !< real (kind_real) => real (kind = 4 )
00010   real(dp) , parameter :: &
00011     T0          = 288.15,     &  !< surface temperature for standard atmosphere
      [K] (15 degC)
00012     g0          = 9.80665,    &  !< mean gravity on the Earth [m/s2]
00013     r0          = 6356.766,   &  !< Earth radius (US Std. atm. 1976)  [km]
00014     p0          = 1013.25,    &  !< surface pressure for standard Earth [hPa]
00015     G           = 6.672e-11,  &  !< Cavendish constant \f$[m^3/kg/s^2]\f$
00016     R_air       = 287.05,     &  !< dry air constant  [J/kg/K]
00017     pi          = 4*atan(1.), &  !< pi = 3.141592... [ ]
00018     rho_crust   = 2670.  ,    &  !< mean density of crust [kg/m3]
00019     rho_earth   = 5500.          !< mean density of Earth [kg/m3]
00020
00021 contains
00022
00023 !
      ================================================================================
00024 !> For given vectors x1, y1 and x2, y2 it gives x2interpolated for x1
00025 !!
00026 !! uses \c ispline and \c spline subroutines
00027 !
      ================================================================================
00028 subroutine spline_interpolation(x,y, x_interpolated,
      y_interpolated)
00029   implicit none
00030   real(dp) , allocatable , dimension (:) ,intent(in) :: x, y, x_interpolated
00031   real(dp) , allocatable , dimension (:) , intent(out) :: y_interpolated
00032   real(dp) , dimension (:) , allocatable :: b, c, d
00033   integer :: i
00034
00035   allocate (b(size(x)))
00036   allocate (c(size(x)))
00037   allocate (d(size(x)))
00038   allocate (y_interpolated(size(x_interpolated)))
```

```
00039
00040   call spline( x , y, b , c, d, size(x))
00041
00042   do i=1, size(x_interpolated)
00043      y_interpolated(i) = ispline(x_interpolated(i) , x , y , b , c , d ,
       size (x) )
00044   enddo
00045
00046 end subroutine
00047
00048 !
       ================================================================================
00049 !> This subroutine was taken from
00050 !! \todo give source
00051 !
       ================================================================================
00052 !  Calculate the coefficients b(i), c(i), and d(i), i=1,2,...,n
00053 !  for cubic spline interpolation
00054 !  s(x) = y(i) + b(i)*(x-x(i)) + c(i)*(x-x(i))**2 + d(i)*(x-x(i))**3
00055 !  for  x(i) <= x <= x(i+1)
00056 !  Alex G: January 2010
00057 !------------------------------------------------------------------------
00058 !  input..
00059 !  x = the arrays of data abscissas (in strictly increasing order)
00060 !  y = the arrays of data ordinates
00061 !  n = size of the arrays xi() and yi() (n>=2)
00062 !  output..
00063 !  b, c, d  = arrays of spline coefficients
00064 !  comments ...
00065 !  spline.f90 program is based on fortran version of program spline.f
00066 !  the accompanying function fspline can be used for interpolation
00067 !
       ================================================================================
00068 subroutine spline (x, y, b, c, d, n)
00069   implicit none
00070   integer n
00071   real(dp) :: x(n), y(n), b(n), c(n), d(n)
00072   integer i, j, gap
00073   real ::  h
00074
00075   gap = n-1
00076   ! check input
00077   if ( n < 2 ) return
00078   if ( n < 3 ) then
00079     b(1) = (y(2)-y(1))/(x(2)-x(1))   ! linear interpolation
00080     c(1) = 0.
00081     d(1) = 0.
00082     b(2) = b(1)
00083     c(2) = 0.
00084     d(2) = 0.
00085     return
00086   end if
00087   !
00088   ! step 1: preparation
00089   !
00090   d(1) = x(2) - x(1)
00091   c(2) = (y(2) - y(1))/d(1)
00092   do i = 2, gap
00093     d(i) = x(i+1) - x(i)
00094     b(i) = 2.0*(d(i-1) + d(i))
00095     c(i+1) = (y(i+1) - y(i))/d(i)
00096     c(i) = c(i+1) - c(i)
00097   end do
00098   !
00099   ! step 2: end conditions
00100   !
00101   b(1) = -d(1)
00102   b(n) = -d(n-1)
00103   c(1) = 0.0
00104   c(n) = 0.0
00105   if(n /= 3) then
00106     c(1) = c(3)/(x(4)-x(2)) - c(2)/(x(3)-x(1))
00107     c(n) = c(n-1)/(x(n)-x(n-2)) - c(n-2)/(x(n-1)-x(n-3))
00108     c(1) = c(1)*d(1)**2/(x(4)-x(1))
00109     c(n) = -c(n)*d(n-1)**2/(x(n)-x(n-3))
00110   end if
00111   !
00112   ! step 3: forward elimination
00113   !
00114   do i = 2, n
00115     h = d(i-1)/b(i-1)
00116     b(i) = b(i) - h*d(i-1)
00117     c(i) = c(i) - h*c(i-1)
00118   end do
00119   !
00120   ! step 4: back substitution
00121   !
```

```
00122   c(n) = c(n)/b(n)
00123   do j = 1, gap
00124     i = n-j
00125     c(i) = (c(i) - d(i)*c(i+1))/b(i)
00126   end do
00127   !
00128   ! step 5: compute spline coefficients
00129   !
00130   b(n) = (y(n) - y(gap))/d(gap) + d(gap)*(c(gap) + 2.0*c(n))
00131   do i = 1, gap
00132     b(i) = (y(i+1) - y(i))/d(i) - d(i)*(c(i+1) + 2.0*c(i))
00133     d(i) = (c(i+1) - c(i))/d(i)
00134     c(i) = 3.*c(i)
00135   end do
00136   c(n) = 3.0*c(n)
00137   d(n) = d(n-1)
00138 end subroutine spline
00139
00140
00141 !
      ================================================================================
00142 !> This subroutine was taken from
00143 !! \todo give source
00144 !
      ================================================================================
00145 !=====================================================================
00146 ! function ispline evaluates the cubic spline interpolation at point z
00147 ! ispline = y(i)+b(i)*(u-x(i))+c(i)*(u-x(i))**2+d(i)*(u-x(i))**3
00148 ! where  x(i) <= u <= x(i+1)
00149 !---------------------------------------------------------------------
00150 ! input..
00151 ! u       = the abscissa at which the spline is to be evaluated
00152 ! x, y    = the arrays of given data points
00153 ! b, c, d = arrays of spline coefficients computed by spline
00154 ! n       = the number of data points
00155 ! output:
00156 ! ispline = interpolated value at point u
00157 !=====================================================================
00158 function ispline(u, x, y, b, c, d, n)
00159 implicit none
00160 real ispline
00161 integer n
00162 real(dp)::  u, x(n), y(n), b(n), c(n), d(n)
00163 integer ::  i, j, k
00164 real :: dx
00165
00166 ! if u is ouside the x() interval take a boundary value (left or right)
00167 if(u <= x(1)) then
00168   ispline = y(1)
00169   return
00170 end if
00171 if(u >= x(n)) then
00172   ispline = y(n)
00173   return
00174 end if
00175
00176 !*
00177 !  binary search for for i, such that x(i) <= u <= x(i+1)
00178 !*
00179 i = 1
00180 j = n+1
00181 do while (j > i+1)
00182   k = (i+j)/2
00183   if(u < x(k)) then
00184     j=k
00185     else
00186     i=k
00187    end if
00188 end do
00189 !*
00190 !  evaluate spline interpolation
00191 !*
00192 dx = u - x(i)
00193 ispline = y(i) + dx*(b(i) + dx*(c(i) + dx*d(i)))
00194 end function ispline
00195
00196 !
      ================================================================================
00197 !> taken from ArkM http://www.tek-tips.com/viewthread.cfm?qid=1688013
00198 !
      ================================================================================
00199 integer function ntokens(line)
00200 character,intent(in):: line*(*)
00201 integer i, n, toks
00202
00203 i = 1;
00204 n = len_trim(line)
```

```
00205 toks = 0
00206 ntokens = 0
00207 do while(i <= n)
00208    do while(line(i:i) == ' ')
00209      i = i + 1
00210      if (n < i) return
00211    enddo
00212    toks = toks + 1
00213    ntokens = toks
00214    do
00215      i = i + 1
00216      if (n < i) return
00217      if (line(i:i) == ' ') exit
00218    enddo
00219 enddo
00220 end function ntokens
00221
00222 !
      ================================================================================
00223 !> This routine skips the lines with comment chars (default '#')
00224 !! from opened files (unit) to read
00225 !
      ================================================================================
00226 subroutine skip_header ( unit , comment_char_optional )
00227   use iso_fortran_env
00228   implicit none
00229   integer , intent (in) :: unit
00230   character (len = 1) , optional :: comment_char_optional
00231   character (len = 60 ) :: dummy
00232   character (len = 1)  :: comment_char
00233   integer :: io_stat
00234
00235   if (present( comment_char_optional ) ) then
00236     comment_char = comment_char_optional
00237   else
00238     comment_char = '#'
00239   endif
00240
00241   read ( unit, * , iostat = io_stat) dummy
00242   if(io_stat == iostat_end) return
00243
00244   do while ( dummy(1:1) .eq. comment_char )
00245     read ( unit, * , iostat = io_stat ) dummy
00246     if(io_stat == iostat_end) return
00247   enddo
00248   backspace(unit)
00249 end subroutine
00250
00251 !> downloaded from  http://aa.usno.navy.mil/faq/docs/jd_formula.php
00252 !! \todo mjd!
00253 real function jd (year,month,day, hh,mm,ss)
00254   implicit none
00255   integer, intent(in) ::  year,month,day
00256   integer, intent(in) :: hh,mm, ss
00257   integer :: i , j , k
00258   i= year
00259   j= month
00260   k= day
00261   jd= k-32075+1461*(i+4800+(j-14)/12)/4+367*(j-2-(j-14)/12*12)/12-3*((i+4900+
      (j-14)/12)/100)/4  + (hh/24.) &
00262   + mm/(24.*60.) +ss/(24.*60.*60.) ! - 2400000.5
00263   return
00264 end function
00265
00266 !subroutine gdate (jd, year,month,day,hh,mm,ss)
00267 !  !! modyfikacja mrajner 20120922
00268 !  !! pobrane  http://aa.usno.navy.mil/faq/docs/jd_formula.php
00269 !  implicit none
00270 !  real, intent(in):: jd
00271 !  real :: aux
00272 !  integer,intent(out) :: year,month,day,hh,mm,ss
00273 !  integer :: i,j,k,l,n
00274
00275 !  l= int((jd+68569))
00276 !  n= 4*l/146097
00277 !  l= l-(146097*n+3)/4
00278 !  i= 4000*(l+1)/1461001
00279 !  l= l-1461*i/4+31
00280 !  j= 80*l/2447
00281 !  k= l-2447*j/80
00282 !  l= j/11
00283 !  j= j+2-12*l
00284 !  i= 100*(n-49)+i+l
00285
00286 !  year= i
00287 !  month= j
00288 !  day= k
```

```
00289
00290 !  aux= jd - int(jd) + 0.0001/86400 ! ostatni argument zapewnia poprawe
00291 !                                   ! jeżeli ss jest integer
00292 !  hh= aux*24
00293 !  mm= aux*24*60    - hh*60
00294 !  ss= aux*24*60*60 - hh*60*60 - mm*60
00295 !end subroutine
00296 real(dp) function mjd  (date)
00297   implicit none
00298   integer ,intent(in) :: date (6)
00299   integer :: aux (6)
00300   integer :: i , k
00301   real(dp) :: dayfrac
00302
00303   aux=date
00304   if ( aux(2) .le.  2) then
00305       aux(1)  = date(1) -  1
00306       aux(2) = date(2) +  12
00307   endif
00308   i = aux(1)/100
00309   k = 2 - i + int(i/4);
00310   mjd = int(365.25 * aux(1) ) - 679006
00311   dayfrac =  aux(4) / 24. + date(5)/(24. * 60. ) + date(6)/(24. * 3600. )
00312   mjd = mjd + int(30.6001*( aux(2) + 1)) + date(3) + k + dayfrac
00313 end function
00314
00315 subroutine invmjd (mjd , date)
00316   implicit none
00317   real(dp), intent (in) :: mjd
00318   integer , intent (out):: date (6)
00319   integer :: t1 ,t4 , h , t2 , t3 , ih1 , ih2
00320   real(dp) :: dayfrac
00321
00322   date =0
00323
00324   t1 = 1+ int(mjd) + 2400000
00325   t4 = mjd - int(mjd);
00326    h = int((t1 - 1867216.25)/36524.25);
00327   t2 = t1 + 1 + h - int(h/4)
00328   t3 = t2 - 1720995
00329   ih1 = int((t3 -122.1)/365.25)
00330   t1 = int(365.25 * ih1)
00331   ih2 = int((t3 - t1)/30.6001);
00332   date(3) = (t3 - t1 - int(30.6001 * ih2)) + t4;
00333   date(2) = ih2 - 1;
00334   if (ih2 .gt. 13) date(2) = ih2 - 13
00335    date(1) = ih1
00336   if (date(2).le. 2) date(1) = date(1) + 1
00337
00338   dayfrac = mjd - int(mjd) + 1./ (60*60*1000)
00339   date(4) = int(dayfrac * 24. )
00340   date(5) = ( dayfrac - date(4) / 24. ) * 60 * 24
00341   date(6) = ( dayfrac - date(4) / 24. - date(5)/(24.*60.)  ) * 60 * 24 *60
00342   if (date(6) .eq. 60 ) then
00343     date(6)=0
00344     date(5)=date(5) + 1
00345   endif
00346 end subroutine
00347
00348 end module constants
```

11.5 grat/src/get_cmd_line.f90 File Reference

This module sets the initial values for parameters reads from command line and gives help it allows to specify commands with or without spaces therefore it is convienient to use with auto completion of names.

Data Types

- module get_cmd_line
- type get_cmd_line::green_functions
- type get_cmd_line::polygon_data
- type get_cmd_line::polygon_info
- type get_cmd_line::dateandmjd
- type get_cmd_line::additional_info
- type get_cmd_line::cmd_line

- type get_cmd_line::site_data
- type get_cmd_line::file

11.5.1 Detailed Description

This module sets the initial values for parameters reads from command line and gives help it allows to specify commands with or without spaces therefore it is convienient to use with auto completion of names.

Definition in file get_cmd_line.f90.

11.6 get_cmd_line.f90

```
00001 ! ================================================================================
00002 !> \file
00003 !! \brief This module sets the initial values for parameters
00004 !! reads from command line and gives help
00005 !! it allows to specify commands with or without spaces therefore it is
00006 !! convienient to use with auto completion of names
00007 ! ================================================================================
00008 module get_cmd_line
00009   use iso_fortran_env
00010   use constants
00011
00012   implicit none
00013
00014   !-----------------------------------------------------
00015   ! Greens function
00016   !-----------------------------------------------------
00017
00018   type green_functions
00019     real(dp),allocatable,dimension(:) :: distance
00020     real(dp),allocatable,dimension(:) :: data
00021     logical  :: if
00022   end type
00023   type(green_functions), allocatable , dimension(:) :: green
00024   integer :: denser(2) = [1,1]
00025
00026   !-----------------------------------------------------
00027   ! polygons
00028   !-----------------------------------------------------
00029   type polygon_data
00030     logical :: use
00031     real(sp), allocatable , dimension (:,:) :: coords
00032   end type
00033
00034   type polygon_info
00035     integer :: unit
00036     character(:), allocatable  :: name
00037     type(polygon_data) , dimension (:) , allocatable :: polygons
00038     logical :: if
00039   end type
00040
00041   type(polygon_info) , dimension (2) :: polygons
00042
00043   !-----------------------------------------------------
00044   ! dates
00045   !-----------------------------------------------------
00046   type dateandmjd
00047     real(dp) :: mjd
00048     integer,dimension (6) :: date
00049   end type
00050
00051   real(kind=4) :: cpu_start , cpu_finish  !< for time execution of program
00052   type(dateandmjd) , allocatable,dimension (:) :: dates
00053
00054
00055   !-----------------------------------------------------
00056   ! command line entry
00057   !-----------------------------------------------------
00058   type additional_info
00059     character (len=55) ,allocatable ,dimension(:) :: names
00060   end type
00061   type cmd_line
00062     character(2) :: switch
00063     integer ::      fields
00064     character (len=255) ,allocatable ,dimension(:) :: field
00065     type (additional_info), allocatable , dimension(:) ::
      fieldnames
```

```
00066   end type
00067
00068   !-----------------------------------------------
00069   ! site information
00070   !-----------------------------------------------
00071   type site_data
00072    character(:), allocatable :: name
00073    real(sp)                  :: lat,lon,height
00074   end type
00075
00076   type(site_data) , allocatable , dimension(:) :: sites
00077
00078   !-----------------------------------------------
00079   ! various
00080   !-----------------------------------------------
00081   integer :: fileunit_tmp               !< unit of scratch file
00082   integer,dimension(8):: execution_date !< To give time stamp of execution
00083   character (len = 2) :: method = "2D"  !< computation method
00084
00085   !-----------------------------------------------
00086   ! Site names file
00087   !-----------------------------------------------
00088   character(:), allocatable &
00089           :: filename_site
00090   integer :: fileunit_site
00091
00092
00093   type file
00094     character(:), allocatable &
00095           :: name
00096     ! varname , lonname,latname,levelname , timename
00097     character(len=50) :: names(5) = [ "z", "lon", "lat","level","time"]
00098
00099     integer :: unit = output_unit
00100
00101     ! if file was determined
00102     logical :: if =.false.
00103
00104     ! to read into only once
00105     logical :: first_call =.true.
00106
00107     ! boundary of model e , w ,s ,n
00108     real(sp):: limits(4)
00109
00110 !     resolution of model in lon lat
00111 !    real(sp):: resolution(2)
00112
00113     real(sp) , allocatable ,dimension(:) :: lat , lon , time ,level
00114     integer , allocatable , dimension(:,: ) :: date
00115
00116     real (sp), dimension(2) :: latrange , lonrange
00117
00118     ! todo
00119     logical :: if_constant_value
00120     real(sp):: constant_value
00121
00122     ! data
00123     !> 4 dimension - lat , lon , level , mjd
00124     ! todo
00125     real(sp) , allocatable , dimension (:,:,:) :: data
00126
00127     ! netcdf identifiers
00128     integer :: ncid
00129     integer :: interpolation = 1
00130   end type
00131
00132   ! External files
00133   type(file) ::  log  , output , refpres
00134   type(file) , allocatable, dimension (:) :: model
00135   type(file) :: moreverbose
00136
00137   character (len =40) :: model_names (5) = ["pressure_surface" , &
00138     "temperature_surface" , "topography" , "landsea" , "pressure levels" ]
00139
00140
00141   character(len=5) :: green_names(5) = [ "GN   ", "GN/dt", "GN/dh","GN/dz","GE
      "]
00142
00143
00144   ! Verbose information and the output for log_file
00145   logical :: if_verbose  = .false.  !< whether print all information
00146   logical :: inverted_barometer  = .true.
00147
00148   character (50) :: interpolation_names (2) &
00149     = [ "nearest" , "bilinear" ]
00150
00151   !-----------------------------------------------
```

```
00152   ! For preety printing
00153   !-----------------------------------------------------
00154   character(len=255), parameter ::  &
00155     form_header    = '(60("#"))' , &
00156     form_separator = '(60("-"))' , &
00157     form_inheader  = '(("#"),1x,a56,1x,("#"))' , &
00158     form_60        = "(a,100(1x,g0))",          &
00159     form_61        = "(2x,a,100(1x,g0))",     &
00160     form_62        = "(4x,a,100(1x,g0))",       &
00161     form_63        = "(6x,100(x,g0))",        &
00162     form_64        = "(4x,4x,a,4x,a)"
00163
00164
00165 contains
00166 ! ================================================================================
00167 !> This subroutine counts the command line arguments
00168 !!
00169 !! Depending on command line options set all initial parameters and reports it
00170 ! ================================================================================
00171 subroutine intro (program_calling)
00172   implicit none
00173   integer :: i, j
00174   character(len=255) :: dummy, dummy2,arg
00175   character(len=*) :: program_calling
00176   type(cmd_line) :: cmd_line_entry
00177
00178   if(iargc().eq.0) then
00179     write(output_unit , '(a)' ) , 'Short description: ./'//program_calling//'
       -h'
00180     call exit
00181   else
00182     open(newunit=fileunit_tmp,status='scratch')
00183     write (fileunit_tmp,form_61) "command invoked"
00184     call get_command(dummy)
00185     write (fileunit_tmp,form_62) trim(dummy)
00186     do i = 1 , iargc()
00187       call get_command_argument(i,dummy)
00188       ! allow specification like '-F file' and '-Ffile'
00189       call get_command_argument(i+1,dummy2)
00190       if (dummy(1:1).eq."-") then
00191         arg = trim(dummy)
00192       else
00193         arg=trim(arg)//trim(dummy)
00194       endif
00195       if(dummy2(1:1).eq."-".or.i.eq.iargc()) then
00196          call get_cmd_line_entry(arg, cmd_line_entry ,
      program_calling = program_calling)
00197       endif
00198     enddo
00199
00200     call if_minimum_args( program_calling = program_calling )
00201
00202     ! Where and if to log the additional information
00203     if (log%if) then
00204       ! if file name was given then automaticall switch verbose mode
00205       if_verbose = .true.
00206       open (newunit = log%unit, file = log%name , action = "write" )
00207     else
00208       ! if you don't specify log file, or not switch on verbose mode
00209       ! all additional information will go to trash
00210       ! Change  /dev/null accordingly if your file system does not
00211       ! support this name
00212       if (.not.if_verbose) then
00213         open (newunit=log%unit, file = "/dev/null", action = "write" )
00214       endif
00215     endif
00216   endif
00217 end subroutine
00218
00219 ! ================================================================================
00220 !> Check if at least all obligatory command line arguments were given
00221 !! if not print warning
00222 ! ================================================================================
00223 subroutine if_minimum_args ( program_calling )
00224   implicit none
00225   character (*) , intent(in) :: program_calling
00226
00227   if (program_calling.eq."grat" ) then
00228
00229     if (size(sites) .eq. 0) then
00230       write(error_unit, * ) "ERROR:", program_calling
00231       write(error_unit, * ) "ERROR:", "no sites!"
00232       call exit
00233     endif
00234   elseif(program_calling.eq."polygon_check" ) then
00235   endif
00236 end subroutine
```

```
00237
00238 ! ========================================================================
00239 !> This function is true if switch is used by calling program or false if it
00240 !! is not
00241 ! ========================================================================
00242 logical function if_switch_program (program_calling , switch )
00243   implicit none
00244   character(len=*), intent (in) :: program_calling
00245   character(len=*),  intent (in) :: switch
00246   character, dimension(:) , allocatable :: accepted_switch
00247   integer :: i
00248
00249   ! default
00250   if_switch_program=.false.
00251
00252   ! depending on program calling decide if switch is permitted
00253   if (program_calling.eq."grat") then
00254     allocate( accepted_switch(15) )
00255     accepted_switch = [ "V" , "f" , "S", "B" , "L" , "G" , "P" , "p", &
00256       "o" , "F" , "I" , "D" , "L" , "v" , "h"   ]
00257   elseif(program_calling.eq."polygon_check") then
00258     allocate( accepted_switch(12) )
00259     accepted_switch = [ "V" , "f" , "A", "B" , "L" , "P" , "o", "S" , &
00260       "h" , "v" , "I" , "i"]
00261   elseif(program_calling.eq."value_check") then
00262     allocate( accepted_switch(9) )
00263     accepted_switch = [ "V" , "F" , "o", "S" , "h" , "v" , "I" , "D" , "L"]
00264   else
00265     if_switch_program=.true.
00266     return
00267   endif
00268
00269   ! loop trough accepted switches
00270   do i =1, size (accepted_switch)
00271     if (switch(2:2).eq.accepted_switch(i)) if_switch_program=.
      true.
00272   enddo
00273 end function
00274
00275 ! ========================================================================
00276 !> This subroutine counts the command line arguments and parse appropriately
00277 ! ========================================================================
00278 subroutine parse_option (cmd_line_entry , program_calling)
00279   type(cmd_line),intent(in):: cmd_line_entry
00280   character(len=*), optional :: program_calling
00281   integer :: i
00282
00283   ! all the command line option are stored in tmp file and later its decide
00284   ! if it is written to STDOUT , log_file or nowwhere
00285     select case (cmd_line_entry%switch)
00286     case ('-h')
00287       call print_help(program_calling)
00288       call exit
00289     case ('-v')
00290       call print_version(program_calling)
00291       call exit()
00292     case ('-V')
00293       if_verbose = .true.
00294       write(fileunit_tmp, form_62) 'verbose mode' ,trim(log%name)
00295       if (len(trim(cmd_line_entry%field(1))).gt.0) then
00296         log%if = .true.
00297         log%name = trim(cmd_line_entry%field(1))
00298         write(fileunit_tmp, form_62) 'the log file was set:' ,log%name
00299       endif
00300     case ('-S')
00301       ! check if format is proper for site
00302       ! i,e. -Sname,B,L[,H]
00303       if (.not. allocated(sites)) then
00304         if (  is_numeric(cmd_line_entry%field(2)) &
00305         .and.is_numeric(cmd_line_entry%field(3)) &
00306         .and.index(cmd_line_entry%field(1), "/" ).eq.0 &
00307         .and.(.not.cmd_line_entry%field(1).eq. "Rg" ) &
00308         ) then
00309             allocate (sites(1))
00310             sites(1)%name = trim(cmd_line_entry%field(1))
00311             read ( cmd_line_entry%field(2) , * ) sites(1)%lat
00312             if (abs(sites(1)%lat).gt.90.) &
00313               sites(1)%lat = sign(90.,sites(1)%lat)
00314             read ( cmd_line_entry%field(3) , * ) sites(1)%lon
00315             if (sites(1)%lon.ge.360.) sites(1)%lon = mod(sites(1)%lon,360.)
00316             if (is_numeric(cmd_line_entry%field(4) ) ) then
00317               read ( cmd_line_entry%field(4) , * ) sites(1)%height
00318             endif
00319             write(fileunit_tmp, form_62) 'the site was set (BLH):' , &
00320               sites(1)%name, sites(1)%lat , sites(1)%lon , sites(1)%height
00321         else
00322           ! or read sites from file
```

```
00323           if (file_exists(cmd_line_entry%field(1) ) ) then
00324             write(fileunit_tmp, form_62) 'the site file was set:' , &
00325               cmd_line_entry%field(1)
00326             call read_site_file(cmd_line_entry%field(1))
00327           elseif(index(cmd_line_entry%field(1), "/" ).ne.0 &
00328               .or.cmd_line_entry%field(1).eq."Rg")  then
00329             call parse_gmt_like_boundaries(
      cmd_line_entry )
00330           else
00331             call print_warning( "site" , fileunit_tmp)
00332           endif
00333         endif
00334       else
00335         call print_warning( "repeated" , fileunit_tmp)
00336       endif
00337     case ("-I")
00338       !> \todo add maximum minimum distances for integration
00339       write( fileunit_tmp , form_62 , advance="no" ) "interpolation method was
       set:"
00340       do i = 1 , cmd_line_entry%fields
00341         if (is_numeric(cmd_line_entry%field(i))) then
00342           read ( cmd_line_entry%field(i) , * ) model(i)%interpolation
00343           write(fileunit_tmp , '(a10,x,$)' ) interpolation_names(model(i)
      %interpolation)
00344           if (model(i)%interpolation.gt.size(interpolation_names)) then
00345             model(i)%interpolation=1
00346           endif
00347         endif
00348       enddo
00349       write(fileunit_tmp , *)
00350     case ("-L")
00351       !> \todo make it mulitichoice: -Lfile:s,file2:b ...
00352       write (fileunit_tmp , form_62) "printing additional information"
00353 !      allocate(moreverbose(cmd_line_entry%fields))
00354 !      print *,size(moreverbose),"XXXX"
00355 !      do i = 1, cmd_line_entry%fields
00356 !        call get_model_info (moreverbose (i) , cmd_line_entry , i )
00357 !        write (fileunit_tmp , form_62) "file: ", moreverbose(i)%name
00358 !      enddo
00359 !      write (fileunit_tmp , form_62) "what: ", moreverbose%names(1)
00360 !      if (len(moreverbose%name).gt.0 .and. moreverbose%name.ne."") then
00361 !        open (newunit = moreverbose%unit , file = moreverbose%name , action =
       "write" )
00362 !      endif
00363     case ("-B")
00364       if (cmd_line_entry%field(1).eq."N" ) inverted_barometer=.false.
00365     case ("-R")
00366       if (cmd_line_entry%field(1).eq."+" ) refpres%if = .true.
00367     case ('-D')
00368       call parse_dates( cmd_line_entry )
00369     case ('-F')
00370       allocate(model(cmd_line_entry%fields))
00371       do i = 1, cmd_line_entry%fields
00372         call get_model_info(model(i) , cmd_line_entry , i )
00373       enddo
00374     case ("-G")
00375       !> \todo when no given take defaults
00376       call parse_green(cmd_line_entry)
00377     case ('-M')
00378       !> \todo rozbudować
00379       method = cmd_line_entry%field(1)
00380       write(fileunit_tmp, form_62), 'method was set: ' , method
00381     case ('-o')
00382       output%if=.true.
00383       output%name=cmd_line_entry%field(1)
00384       write(fileunit_tmp, form_62), 'output file was set: ' , output%name
00385       if (len(output%name).gt.0.and. output%name.ne."") then
00386         open (newunit = output%unit , file = output%name , action = "write"
       )
00387       endif
00388     case ('-P')
00389       do i = 1 , 2 !size(cmd_line_entry%field)
00390         polygons(i)%name=cmd_line_entry%field(i)
00391         if (file_exists((polygons(i)%name))) then
00392           write(fileunit_tmp, form_62), 'polygon file was set: ' , polygons(i)
      %name
00393           polygons(i)%if=.true.
00394           ! todo
00395 !          call read_polygon (polygons(i))
00396         else
00397           write(fileunit_tmp, form_62), 'file do not exist. Polygon file was
       IGNORED'
00398         endif
00399       enddo
00400     case default
00401       write(fileunit_tmp,form_62), "unknown argument: IGNORING"
00402     end select
```

```
00403     return
00404 end subroutine
00405
00406 ! ================================================================================
00407 !> This subroutine parse -G option i.e. reads Greens function
00408 ! ================================================================================
00409 subroutine parse_green ( cmd_line_entry)
00410   type (cmd_line)  :: cmd_line_entry
00411   character (60) :: filename
00412   integer :: i , iunit , io_status , lines ,  ii
00413   integer :: fields(2)= [1,2]
00414   real (sp) , allocatable , dimension(:) :: tmp
00415
00416     write(fileunit_tmp , form_62) "Green function file was set:"
00417     allocate (green(cmd_line_entry%fields))
00418
00419     do i = 1 , cmd_line_entry%fields
00420
00421       if (i.eq.6) then
00422         if (is_numeric(cmd_line_entry%field(i))) then
00423           read( cmd_line_entry%field(i), *) denser(1)
00424           if (is_numeric(cmd_line_entry%fieldnames(i)%names(1))) then
00425             read( cmd_line_entry%fieldnames(i)%names(1), *) denser(2)
00426           endif
00427           return
00428          endif
00429       endif
00430
00431       if (.not.file_exists(cmd_line_entry%field(i)) &
00432       .and. (.not. cmd_line_entry%field(i).eq."merriam" &
00433       .and. .not. cmd_line_entry%field(i).eq."huang" &
00434       .and. .not. cmd_line_entry%field(i).eq."rajner" )) then
00435         cmd_line_entry%field(i)="merriam"
00436       endif
00437
00438       !> change the paths accordingly
00439       if (cmd_line_entry%field(i).eq."merriam") then
00440         filename="/home/mrajner/src/grat/dat/merriam_green.dat"
00441         if (i.eq.1) fields = [1,2]
00442         if (i.eq.2) fields = [1,3]
00443         if (i.eq.3) fields = [1,4]
00444         if (i.eq.4) fields = [1,4]
00445         if (i.eq.5) fields = [1,6]
00446       elseif(cmd_line_entry%field(i).eq."huang") then
00447         filename="/home/mrajner/src/grat/dat/huang_green.dat"
00448         if (i.eq.1) fields = [1,2]
00449         if (i.eq.2) fields = [1,3]
00450         if (i.eq.3) fields = [1,4]
00451         if (i.eq.4) fields = [1,5]
00452         if (i.eq.5) fields = [1,6]
00453       elseif(cmd_line_entry%field(i).eq."rajner") then
00454         filename="/home/mrajner/src/grat/dat/rajner_green.dat"
00455         if (i.eq.1) fields = [1,2]
00456         if (i.eq.2) fields = [1,3]
00457         if (i.eq.3) fields = [1,4]
00458         if (i.eq.4) fields = [1,5]
00459         if (i.eq.5) fields = [1,6]
00460       elseif(file_exists(cmd_line_entry%field(i))) then
00461         filename = cmd_line_entry%field(i)
00462         if (size(cmd_line_entry%fieldnames).ne.0 .and. allocated(cmd_line_entry
      %fieldnames(i)%names)) then
00463           do ii=1, 2
00464             if(is_numeric(cmd_line_entry%fieldnames(i)%names(ii) ) )
      then
00465               read( cmd_line_entry%fieldnames(i)%names(ii), *) fields(ii)
00466             endif
00467           enddo
00468         endif
00469       endif
00470
00471       allocate(tmp(max(fields(1),fields(2))))
00472       lines = 0
00473       open ( newunit =iunit,file=filename,action="read")
00474       do
00475         call skip_header(iunit)
00476         read (iunit , * , iostat = io_status)
00477         if (io_status == iostat_end) exit
00478         lines = lines + 1
00479       enddo
00480       allocate (green(i)%distance(lines))
00481       allocate (green(i)%data(lines))
00482       rewind(iunit)
00483       lines = 0
00484       do
00485         call skip_header(iunit)
00486         lines = lines + 1
00487         read (iunit , * , iostat = io_status) tmp
```

```
00488         if (io_status == iostat_end) exit
00489         green(i)%distance(lines) = tmp(fields(1))
00490         green(i)%data(lines)     = tmp(fields(2))
00491       enddo
00492       deallocate(tmp)
00493       close(iunit)
00494       if (cmd_line_entry%field(i).eq."merriam" .and. i.eq.4) then
00495         green(i)%data = green(i)%data * (-1.)
00496       endif
00497       if (cmd_line_entry%field(i).eq."huang" .and. (i.eq.3.or.i.eq.4)) then
00498         green(i)%data = green(i)%data * 1000.
00499       endif
00500       write(fileunit_tmp , form_63) trim(green_names(i)), &
00501         trim(cmd_line_entry%field(i)),":", fields
00502     enddo
00503 end subroutine
00504
00505 ! ================================================================================
00506 !> Counts occurence of character (separator, default comma) in string
00507 ! ================================================================================
00508 integer function count_separator (dummy , separator)
00509   character(*) , intent(in) ::dummy
00510   character(1), intent(in), optional  :: separator
00511   character(1)  :: sep
00512   character(:), allocatable :: dummy2
00513   integer :: i
00514
00515   dummy2=dummy
00516   sep = ","
00517   if (present(separator)) sep = separator
00518   count_separator=0
00519   do
00520     i = index(dummy2, sep)
00521     if (i.eq.0) exit
00522     dummy2 = dummy2(i+1:)
00523     count_separator=count_separator+1
00524   enddo
00525 end function
00526
00527
00528 ! ================================================================================
00529 !> This subroutine fills the fields of command line entry for every input arg
00530 ! ================================================================================
00531 subroutine get_cmd_line_entry (dummy , cmd_line_entry ,
       program_calling )
00532   character(*) :: dummy
00533   character(:), allocatable :: dummy2
00534   type (cmd_line),intent(out) :: cmd_line_entry
00535   character(1) :: separator=","
00536   character(len=*) , intent(in) , optional :: program_calling
00537   integer :: i , j , ii , jj
00538
00539   cmd_line_entry%switch = dummy(1:2)
00540   write(fileunit_tmp, form_61) , dummy
00541   if (.not.if_switch_program(program_calling, cmd_line_entry
      %switch)) then
00542     write ( fileunit_tmp , form_62 ) "this switch is IGNORED by program "//
      program_calling
00543     return
00544   endif
00545
00546
00547   dummy=dummy(3:)
00548
00549   cmd_line_entry%fields = count_separator(dummy) + 1
00550   allocate(cmd_line_entry%field (cmd_line_entry%fields) )
00551
00552   ! if ":" separator is present in command line allocate
00553   ! additional array for fieldnames
00554   if (count_separator(dummy, ":" ).ge.1) then
00555     allocate(cmd_line_entry%fieldnames (cmd_line_entry%fields) )
00556   endif
00557   do i = 1 , cmd_line_entry%fields
00558     j = index(dummy, separator)
00559     cmd_line_entry%field(i) = dummy(1:j-1)
00560     if (i.eq.cmd_line_entry%fields) cmd_line_entry%field(i)=dummy
00561     dummy=dummy(j+1:)
00562
00563     ! separate field and fieldnames
00564     if ( index(cmd_line_entry%field(i),":").ne.0 ) then
00565       dummy2 = trim(cmd_line_entry%field(i))//":"
00566       allocate ( cmd_line_entry%fieldnames(i)%names(count_separator
      (dummy2,":") - 1 ))
00567       do ii = 1, size(cmd_line_entry%fieldnames(i)%names)+1
00568         jj = index(dummy2, ":")
00569         if (ii.eq.1) then
00570           cmd_line_entry%field(i) = dummy2(1:jj-1)
```

```
00571         else
00572           cmd_line_entry%fieldnames(i)%names(ii-1) = dummy2(1:jj-1)
00573         endif
00574         dummy2 = dummy2(jj+1:)
00575       enddo
00576     endif
00577   enddo
00578   call parse_option(cmd_line_entry , program_calling =
      program_calling)
00579 end subroutine
00580
00581 ! ================================================================================
00582 !> This subroutine fills the model info
00583 ! ================================================================================
00584 subroutine get_model_info ( model , cmd_line_entry , field)
00585   type(cmd_line),intent(in):: cmd_line_entry
00586   type(file),intent(inout):: model
00587   integer :: field , i
00588
00589   model%name = trim(cmd_line_entry%field(field))
00590   if (model%name.eq."") return
00591   if ( file_exists(model%name) ) then
00592     write (fileunit_tmp , form_62) , trim(model_names(field) )
00593     write(fileunit_tmp, form_63), trim(model%name)
00594
00595     do i =1 , size (model%names)
00596       if (size(cmd_line_entry%fieldnames).gt.0) then
00597         if (i.le.size (cmd_line_entry%fieldnames(field)%names) &
00598           .and. cmd_line_entry%fieldnames(field)%names(i).ne."" &
00599           ) then
00600           model%names(i) = cmd_line_entry%fieldnames(field)%names(i)
00601         endif
00602       endif
00603       write(fileunit_tmp, form_63, advance="no") , trim( model%names(i))
00604     enddo
00605     model%if=.true.
00606     write(fileunit_tmp, form_63)
00607   elseif(is_numeric(model%name)) then
00608     model%if_constant_value=.true.
00609     read (model%name , * ) model%constant_value
00610     write (fileunit_tmp , form_62) , trim(model_names(field) )
00611     write(fileunit_tmp, form_63), 'constant value was set: ' , model
      %constant_value
00612     model%if_constant_value=.true.
00613   else
00614     write (fileunit_tmp , form_63 ) "no (correct) model in field: ", field
00615   endif
00616 end subroutine
00617
00618
00619 ! ================================================================================
00620 !> This subroutine checks if given limits for model are proper
00621 ! ================================================================================
00622 subroutine parse_gmt_like_boundaries ( cmd_line_entry
       )
00623   implicit none
00624   real(sp) :: limits (4) , resolution (2) =[1,1]
00625   real(sp) :: range_lon , range_lat , lat , lon
00626   character(10) :: dummy
00627   integer :: i , ii
00628   type (cmd_line) , intent (in) :: cmd_line_entry
00629   character(:) ,allocatable :: text
00630   integer :: n_lon , n_lat
00631
00632   text = cmd_line_entry%field(1)
00633
00634   do i=1,3
00635     if ( is_numeric(text(1:index(text, "/"))) ) then
00636       read ( text(1:index(text, "/")) , * )  limits(i)
00637     else
00638       if (text.eq."Rg" ) then
00639         limits=[0. , 360. , -90 , 90. ]
00640       endif
00641     endif
00642     text=text(index(text,"/")+1:)
00643   enddo
00644
00645   if ( is_numeric(text(1:)) ) then
00646     read ( text(1:) , * )  limits(4)
00647   else
00648     call print_warning("boundaries")
00649   endif
00650
00651   do i = 1 ,2
00652     if (limits(i).lt. -180. .or. limits(i).gt.360. ) then
00653       call print_warning("boundaries")
00654     else
```

```
00655       if (limits(i).lt.0.) limits(i)=limits(i)+360.
00656     endif
00657   enddo
00658   do i =3,4
00659     if (limits(i).lt. -90. .or. limits(i).gt.90. ) then
00660       call print_warning("boundaries")
00661     endif
00662   enddo
00663   if (limits(3).gt.limits(4)) then
00664     call print_warning("boundaries")
00665   endif
00666
00667   if (is_numeric(cmd_line_entry%field(2) ) ) then
00668     read (cmd_line_entry%field(2) , * ) resolution(1)
00669     resolution(2) = resolution(1)
00670   endif
00671   if (is_numeric(cmd_line_entry%field(3) ) ) then
00672     read (cmd_line_entry%field(3) , * ) resolution(2)
00673   endif
00674
00675   range_lon=limits(2) - limits(1)
00676   if (range_lon.lt.0) range_lon = range_lon + 360.
00677   range_lat=limits(4) - limits(3)
00678   n_lon = floor( range_lon / resolution(1)) + 1
00679   n_lat = floor( range_lat / resolution(2)) + 1
00680   allocate (sites( n_lon * n_lat ) )
00681
00682   do i = 1 , n_lon
00683     lon = limits(1) + (i-1) * resolution(1)
00684     if (lon.ge.360.) lon = lon - 360.
00685     do ii = 1 , n_lat
00686       lat = limits(3) + (ii-1) * resolution(2)
00687       sites( (i-1) * n_lat + ii  )%lon = lon
00688       sites( (i-1) * n_lat + ii  )%lat = lat
00689     enddo
00690   enddo
00691
00692 end subroutine
00693
00694 ! ============================================================================
00695 !> Read site list from file
00696 !!
00697 !! checks for arguments and put it into array \c sites
00698 ! ============================================================================
00699 subroutine read_site_file ( file_name )
00700   character(len=*) , intent(in) ::  file_name
00701   integer :: io_status , i , good_lines = 0 , number_of_lines = 0 , nloop
00702   character(len=255) ,dimension(4)  :: dummy
00703   character(len=255) :: line_of_file
00704   type(site_data) :: aux
00705
00706
00707
00708     open ( newunit = fileunit_site , file = file_name, &
00709             iostat = io_status ,status = "old" , action="read" )
00710
00711     ! two loops, first count good lines and print rejected
00712     ! second allocate array of sites and read coordinates into it
00713       nloops: do nloop = 1, 2
00714       if (nloop.eq.2) allocate(sites(good_lines))
00715       if (number_of_lines.ne.good_lines) then
00716         call print_warning("site_file_format")
00717       endif
00718       good_lines=0
00719       line_loop:do
00720         read ( fileunit_site , '(a)' , iostat = io_status ) line_of_file
00721         if ( io_status == iostat_end)  exit  line_loop
00722         number_of_lines = number_of_lines + 1
00723         ! we need at least 3 parameter for site (name , B , L )
00724         if (ntokens(line_of_file).ge.3) then
00725           ! but no more than 4 parameters (name , B , L, H)
00726           if (ntokens(line_of_file).gt.4) then
00727             read ( line_of_file , * ) dummy(1:4)
00728           else
00729             read ( line_of_file , * ) dummy(1:3)
00730             ! if site height was not given we set it to zero
00731             dummy(4)="0."
00732           endif
00733         endif
00734         ! check the values given
00735         if(    is_numeric(trim(dummy(2)))   &
00736           .and.is_numeric(trim(dummy(3)))   &
00737           .and.is_numeric(trim(dummy(4)))   &
00738           .and.ntokens(line_of_file).ge.3 ) then
00739
00740           aux%name= trim(dummy(1))
00741           read( dummy(2),*) aux%lat
```

```
00742           read(dummy(3),*) aux%lon
00743           read(dummy(4),*) aux%height
00744
00745 !          ! todo
00746           if (aux%lat.ge.-90 .and. aux%lat.le.90) then
00747             if (aux%lon.ge.-180 .and. aux%lon.le.360) then
00748               good_lines=good_lines+1
00749               if (nloop.eq.2) then
00750                 sites(good_lines)%name= trim(dummy(1))
00751                 read(dummy(2),*) sites(good_lines)%lat
00752                 read(dummy(3),*) sites(good_lines)%lon
00753                 read(dummy(4),*) sites(good_lines)%height
00754               endif
00755             else
00756               if (nloop.eq.2) write ( fileunit_tmp, form_63) "rejecting (lon
       limits):" , line_of_file
00757             endif
00758           else
00759             if (nloop.eq.2) write ( fileunit_tmp, form_63) "rejecting (lat
       limits):" , line_of_file
00760           endif
00761
00762         else
00763           ! print it only once
00764           if (nloop.eq.2) then
00765             write ( fileunit_tmp, form_63) "rejecting (args):      " ,
      line_of_file
00766           endif
00767         endif
00768       enddo line_loop
00769       if (nloop.eq.1) rewind(fileunit_site)
00770     enddo nloops
00771
00772   ! if longitude <-180, 180> change to <0,360) domain
00773   do i =1 , size (sites)
00774     if (sites(i)%lon.lt.0) sites(i)%lon= sites(i)%lon + 360.
00775     if (sites(i)%lon.eq.360) sites(i)%lon= 0.
00776   enddo
00777 end subroutine
00778
00779
00780 ! ================================================================================
00781 !> Parse date given as 20110503020103  to yy mm dd hh mm ss and mjd
00782 !!
00783 !! \warning decimal seconds are not allowed
00784 ! ================================================================================
00785 subroutine parse_dates (cmd_line_entry )
00786   type(cmd_line) cmd_line_entry
00787   integer , dimension(6) :: start , stop
00788   real (sp) :: step =6. ! step in hours
00789   integer :: i
00790
00791   call string2date(cmd_line_entry%field(1), start)
00792   write (fileunit_tmp , form_62) "start date:" , start
00793   if (cmd_line_entry%field(2).eq."".or.cmd_line_entry%fields.le.1) then
00794     stop = start
00795   else
00796     call string2date(cmd_line_entry%field(2), stop )
00797     write (fileunit_tmp , form_62) "stop date: " , stop
00798   endif
00799   if (is_numeric(cmd_line_entry%field(3)).and.cmd_line_entry%fields
      .ge.3) then
00800     read(cmd_line_entry%field(3),*) step
00801     write (fileunit_tmp , form_62) "interval [h]:" , step
00802   endif
00803
00804   allocate (dates( int( ( mjd(stop) - mjd(start) ) / step * 24. + 1 ) ))
00805   do i = 1 , size(dates)
00806     dates(i)%mjd = mjd(start) + ( i -1 ) * step / 24.
00807     call invmjd( dates(i)%mjd , dates(i)%date)
00808   enddo
00809 end subroutine
00810
00811 subroutine string2date ( string , date )
00812   integer , dimension(6) ,intent(out):: date
00813   character (*) , intent(in) :: string
00814   integer :: start_char , end_char , j
00815
00816   ! this allow to specify !st Jan of year simple as -Dyyyy
00817   date = [2000 , 1 , 1 , 0 ,0 ,0]
00818
00819   start_char = 1
00820   do j = 1 , 6
00821     if (j.eq.1) then
00822       end_char=start_char+3
00823     else
00824       end_char=start_char+1
```

```
00825     endif
00826       if (is_numeric(string(start_char : end_char) )) then
00827       read(string(start_char : end_char),*) date(j)
00828     endif
00829     start_char=end_char+1
00830   enddo
00831
00832 end subroutine
00833
00834
00835 !subroutine sprawdzdate(mjd)
00836 !  real:: mjd
00837 !    if
       (mjd.gt.jd(data_uruchomienia(1),data_uruchomienia(2),data_uruchomienia(3),data_uruchomienia(4),data_uruchomienia(5),dat
00838 !      write (*,'(4x,a)') "Data późniejsza niż dzisiaj. KOŃCZĘ!"
00839 !      call exit
00840 !    elseif (mjd.lt.jd(1980,1,1,0,0,0)) then
00841 !      write (*,'(4x,a)') "Data wcześniejsza niż 1980-01-01. KOŃCZĘ!"
00842 !      call exit
00843 !    endif
00844 !    if (.not.log_E) then
00845 !      data_koniec=data_poczatek
00846 !      mjd_koniec=mjd_poczatek
00847 !    endif
00848 !    if (mjd_koniec.lt.mjd_poczatek) then
00849 !      write (*,*) "Data końcowa większa od początkowej. KOŃCZĘ!"
00850 !      write (*,form_64) "Data końcowa większa od początkowej. KOŃCZĘ!"
00851 !    endif
00852 !end subroutine
00853
00854 ! ================================================================================
00855 !> Auxiliary function
00856 !!
00857 !! check if argument given as string is valid number
00858 !! Taken from www
00859 !! \todo Add source name
00860 ! ================================================================================
00861 function is_numeric(string)
00862   implicit none
00863   character(len=*), intent(in) :: string
00864   logical :: is_numeric
00865   real :: x
00866   integer :: e
00867   read(string,*,iostat=e) x
00868   is_numeric = e == 0
00869 end function
00870
00871
00872 ! ================================================================================
00873 !> Check if file exists , return logical
00874 ! ================================================================================
00875 logical function file_exists(string)
00876   implicit none
00877   character(len=*), intent(in) :: string
00878   logical :: exists
00879   real :: x
00880   integer :: e
00881   if (string =="") then
00882     file_exists=.false.
00883     return
00884   endif
00885   inquire(file=string, exist=exists)
00886   file_exists=exists
00887 end function
00888
00889
00890 ! ================================================================================
00891 !> degree -> radian
00892 ! ================================================================================
00893 real(dp) function d2r (degree)
00894   real(dp) , intent (in) :: degree
00895   d2r= pi / 180.0 * degree
00896 end function
00897
00898 ! ================================================================================
00899 !> radian -> degree
00900 ! ================================================================================
00901 real(dp) function r2d ( radian )
00902   real(dp), intent (in) :: radian
00903   r2d= 180. / pi * radian
00904 end function
00905
00906 ! ================================================================================
00907 !> Print version of program depending on program calling
00908 ! ================================================================================
00909 subroutine print_version (program_calling)
00910   implicit none
```

```
00911   character(*) :: program_calling
00912   write(log%unit , form_header )
00913   if (program_calling.eq."grat" ) then
00914     write(log%unit,form_inheader ) , 'grat v. 1.0'
00915     write(log%unit,form_inheader ) , 'Last modification: 20120910'
00916   elseif(program_calling.eq."polygon_check") then
00917     write(log%unit,form_inheader ) , 'polygon_check v. 1.0'
00918     write(log%unit,form_inheader ) , 'Last modification: 20120910'
00919     write(log%unit,form_inheader ) , ''
00920     write(log%unit,form_inheader ) , 'Check if given point (given with -S)'
00921     write(log%unit,form_inheader ) ,
00922 'is included or excluded usig &      specific polygon file'
00923   elseif(program_calling.eq."value_check") then
00924     write(log%unit,form_inheader ) , 'value_check v. 1.0'
00925     write(log%unit,form_inheader ) , 'Last modification: 20120910'
00926     write(log%unit,form_inheader ) , ''
00927     write(log%unit,form_inheader ) , 'Check data value for given point (given
       with -S)'
00928   endif
00929   write(log%unit,form_inheader ) , ''
00930   write(log%unit,form_inheader ) , 'Marcin Rajner'
00931   write(log%unit,form_inheader ) , 'Warsaw University of Technology'
00932   write(log%unit , form_header )
00933 end subroutine
00934
00935 ! ========================================================================
00936 !> Print settings
00937 ! ========================================================================
00938 subroutine print_settings ( program_calling )
00939   implicit none
00940   logical :: exists
00941   character (len=255):: dummy
00942   integer :: io_status , j
00943   character(*) :: program_calling
00944
00945   call print_version( program_calling = program_calling)
00946   call date_and_time( values = execution_date )
00947   write(log%unit,
00948 '("Program started:",1x,i4,2("-",i2.2), &
      1x,i2.2,2(":",i2.2),1x,"(",SP,i3.2,"h UTC)")'),          &
00949     execution_date(1:3),execution_date(5:7),execution_date(4)/60
00950   write(log%unit, form_separator)
00951
00952   inquire(fileunit_tmp, exist=exists)
00953   if (exists) then
00954     write (log%unit, form_60 ) 'Summary of command line arguments'
00955
00956     !-----------------------------------------------------
00957     ! Cmd line summary (from scratch file)
00958     !-----------------------------------------------------
00959     rewind(fileunit_tmp)
00960     do
00961       read(fileunit_tmp,'(a80)', iostat = io_status ) dummy
00962       if ( io_status == iostat_end)  exit
00963       write (log%unit, '(a80)') dummy
00964     enddo
00965
00966     !-----------------------------------------------------
00967     ! Site summary
00968     !-----------------------------------------------------
00969     write(log%unit, form_separator)
00970     write(log%unit, form_60 ) "Processing:", size(sites), "sites"
00971     write(log%unit, '(2x,a,t16,3a15)') "Name" , "lat [deg]" , "lon [deg]" ,"H
       [m]"
00972     do j = 1,size(sites)
00973       write(log%unit, '(2x,a,t16,3f15.4)') &
00974         sites(j)%name, sites(j)%lat, sites(j)%lon , sites(j)%height
00975       if (j.eq.10) exit
00976     enddo
00977     if (size(sites).gt.10) write(log%unit , form_62 ) &
00978       "and", size(sites)-10, "more"
00979
00980     !-----------------------------------------------------
00981     ! Computation method summary
00982     !-----------------------------------------------------
00983     if (program_calling.eq."grat" ) then
00984       write(log%unit, form_separator)
00985       write(log%unit, form_60 ) "Method used:", method
00986     endif
00987
00988     write(log%unit, form_separator)
00989     write(log%unit, form_60 ) "Interpolation data:", &
00990     interpolation_names(model%interpolation)(1:7)
00991
00992
00993
00994   endif
```

```
00995 end subroutine
00996
00997 subroutine print_help (program_calling)
00998   character(*) :: program_calling
00999   integer :: help_unit , io_stat
01000   character(500)::line
01001   character(255)::syntax
01002   logical:: if_print_line = .false., if_optional=.true.
01003
01004   if_print_line=.false.
01005
01006   open(newunit=help_unit, file="~/src/grat/dat/help.hlp", action="read",
      status="old")
01007
01008
01009
01010   write (log%unit ,"(a)" , advance="no" ) program_calling
01011   ! first loop - print only syntax with squre brackets if parameter is optional
01012   do
01013     read (help_unit , '(a)', iostat=io_stat) line
01014     if ((io_stat==iostat_end .or. line(1:1) == "-") .and. if_print_line ) then
01015       if (if_optional) write(log%unit, '(a)' , advance="no") " ["
01016       if (if_optional) write(log%unit, '(a)' , advance="no") trim(syntax)
01017       if (if_optional) write(log%unit, '(a)' , advance="no") "]"
01018     endif
01019     if (io_stat==iostat_end) then
01020       write(log%unit, *) ""
01021       if_print_line = .false.
01022       exit
01023     endif
01024     if(line(1:1)=="-") then
01025       if(if_switch_program(program_calling , line(1:2) )) then
01026         if_print_line = .true.
01027       else
01028         if(line(1:1)=="-") if_print_line=.false.
01029       endif
01030     endif
01031
01032     if (line(5:13) == "optional " .and. (line(2:2) == program_calling(1:1) .or.
       line(2:2)=="")) then
01033       if_optional=.true.
01034     elseif(line(5:13) == "mandatory") then
01035       if_optional=.false.
01036     endif
01037     if (line(2:2)=="s") then
01038       syntax = trim(adjustl(line(3:)))
01039     endif
01040   enddo
01041   rewind(help_unit)
01042
01043   write(log%unit , form_60) , 'Summary of available options for program '//
      program_calling
01044   ! second loop - print informations
01045   do
01046     read (help_unit , '(a)', iostat=io_stat) line
01047     if (io_stat==iostat_end) exit
01048
01049     if(line(1:1)=="-") then
01050       if(if_switch_program(program_calling , line(1:2) )) then
01051         if_print_line = .true.
01052         write (log%unit , form_61 ) trim(line)
01053       else
01054         if(line(1:1)=="-") if_print_line=.false.
01055       endif
01056     elseif(line(2:2)==program_calling(1:1) .or. line(2:2)=="s") then
01057       if (if_print_line) then
01058         write (log%unit , form_61 ) "  "//trim(line(3:))
01059       endif
01060     elseif(line(2:2)=="") then
01061       if (if_print_line) write (log%unit , form_61 ) trim(line)
01062     endif
01063   enddo
01064   close(help_unit)
01065
01066 end subroutine
01067
01068 subroutine print_warning (  warn , unit)
01069  implicit none
01070  character (len=*)  :: warn
01071  integer , optional :: unit
01072  integer :: def_unit
01073
01074  def_unit=fileunit_tmp
01075  if (present(unit) ) def_unit=unit
01076
01077  if (warn .eq. "site_file_format") then
01078   write(def_unit, form_63) "Some records were rejected"
```

```
01079   write(def_unit, form_63) "you should specify for each line at least 3[4]
       parameters in free format:"
01080   write(def_unit, form_63) "name lat lon [H=0] (skipped)"
01081 elseif(warn .eq. "boundaries") then
01082   write(def_unit, form_62) "something wrong with boundaries. IGNORED"
01083 elseif(warn .eq. "site") then
01084   write(def_unit, form_62) "something wrong with -S specification. IGNORED"
01085 elseif(warn .eq. "repeated") then
01086   write(def_unit, form_62) "reapeted specification. IGNORED"
01087 elseif(warn .eq. "dates") then
01088   write(def_unit, form_62) "something wrong with date format -D. IGNORED"
01089 endif
01090 end subroutine
01091
01092
01093 ! ============================================================================
01094 !> Counts number of properly specified models
01095 ! ============================================================================
01096 integer function nmodels (model)
01097   type(file) , allocatable, dimension (:) :: model
01098   integer :: i
01099
01100   nmodels = 0
01101
01102   do i = 1 , size (model)
01103     if (model(i)%if) nmodels =nmodels + 1
01104     if (model(i)%if_constant_value) nmodels =nmodels + 1
01105   enddo
01106 end function
01107
01108 end module get_cmd_line
```

11.7 grat/src/grat.f90 File Reference

Functions/Subroutines

- program **grat**

11.7.1 Detailed Description

Definition in file grat.f90.

11.8 grat.f90

```
00001 !
       ============================================================================
00002 !> \file
00003 !! \mainpage grat overview
00004 !! \section Purpose
00005 !! This program was created to make computation of atmospheric gravity
00006 !! correction easier. Still developing. Consider visiting later...
00007 !!
00008 !! \version TESTING!
00009 !! \date 2013-01-12
00010 !! \author Marcin Rajner\n
00011 !! Politechnika Warszawska | Warsaw University of Technology
00012 !!
00013 !! \warning This program is written in Fortran90 standard but uses some
       featerus
00014 !! of 2003 specification (e.g., \c 'newunit='). It was also written
00015 !! for <tt>Intel Fortran Compiler</tt> hence some commands can be unavailable
00016 !! for other compilers (e.g., \c <integer_parameter> for \c IO statements. This
       should be
00017 !! easily modifiable according to your output needs.
00018 !! Also you need to have \c iso_fortran_env module available to guess the
       number
00019 !! of output_unit for your compiler.
00020 !! When you don't want a \c log_file and you don't switch \c verbose all
00021 !! unneceserry information whitch are normally collected goes to \c /dev/null
00022 !! file. This is *nix system default trash. For other system or file system
00023 !! organization, please change this value in \c get_cmd_line module.
00024 !!
00025 !! \attention
00026 !! \c grat and value_check needs a \c netCDF library \cite netcdf
```

```
00027 !!
00028 !! \section Usage
00029 !! After sucsesfull compiling make sure the executables are in your search path
00030 !!
00031 !! There is main program \c grat and some utilities program. For the options
       see
00032 !! the appropriate help:
00033 !!  - \link grat-h grat\endlink
00034 !!  - \link value_check-h value_check\endlink
00035 !!  - \link polygon_check-h polygon_check\endlink
00036 !!
00037 !! \page grat-h grat
00038 !!    \include grat.hlp
00039
00040 !> \page ilustration
00041 !! \image latex /home/mrajner/src/grat/doc/interpolation_ilustration.pdf
       "example"
00042 !!
00043 !! \image html /home/mrajner/src/grat/doc/interpolation_ilustration.png
       "interpolation example" width=\textwidth
00044 !! \image html /home/mrajner/src/grat/doc/mapa1.png
00045 !! \image html /home/mrajner/src/grat/doc/mapa2.png
00046 !! \image html /home/mrajner/src/grat/doc/mapa3.png
00047
00048 !> \page intro_sec External resources
00049 !!   - <a href="https://code.google.com/p/grat">project page</a> (git
       repository)
00050 !!   - \htmlonly <a href="../latex/refman.pdf">[pdf]</a> version of this
       manual\endhtmlonly
00051 !!   \latexonly
       \href{https://grat.googlecode.com/git/doc/html/index.html}{html} version of this manual\endlatexonly
00052 !! \TODO give source for grant presentation
00053 !!   - <a href="">[pdf]</a> command line options (in Polish)
00054
00055 !> \example example_aggf.f90
00056 !! \example grat_usage.sh
00057 !
       ================================================================================
00058 program grat
00059   use iso_fortran_env
00060   use get_cmd_line
00061   use mod_polygon
00062   use mod_data
00063   use mod_green
00064
00065
00066   implicit none
00067   real(sp) :: x , y , z , lat ,lon ,val(0:100) !tmp variables
00068   integer :: i , j , ii, iii
00069
00070   !> program starts here with time stamp
00071   call cpu_time(cpu_start)
00072
00073   ! gather cmd line option decide where to put output
00074   call intro( program_calling = "grat" )
00075
00076   ! print header to log: version, date and summary of command line options
00077   call print_settings(program_calling = "grat")
00078
00079   ! read polygons
00080   do i =1 , 2
00081    call read_polygon(polygons(i))
00082   enddo
00083
00084   ! read models into memory
00085   do i =1 , size(model)
00086     if (model(i)%if) call read_netcdf( model(i) )
00087   enddo
00088
00089   ! todo refpres in get_cmd-line
00090   if (refpres%if) then
00091     refpres%name="/home/mrajner/src/grat/data/refpres/vienna_p0.grd"
00092     call read_netcdf(refpres)
00093   endif
00094
00095
00096   allocate (results(size(sites)*max(size(dates),1)))
00097   iii=0
00098   do j = 1 , max(size (dates),1)
00099     if(size(dates).gt.0)  write(output%unit, '(i4,5(i2.2))', advance ="no")
      dates(j)%date
00100
00101     do ii = 1 , min(2,size(model))
00102       if (model(ii)%if) call get_variable( model(ii) , date = dates(j)%date)
00103     enddo
00104
00105     do i = 1 , size(sites)
```

```
00106       write(output%unit, '(2f15.5f)', advance ="no") sites(i)%lat ,sites(i)%lon
00107       iii=iii+1
00108       call convolve(sites(i) , green , results(iii), denserdist = denser(1) ,
      denseraz = denser(2))
00109       write (output%unit,'(15f13.5)') , results(iii)%e ,results(iii)%n  ,
      results(iii)%dt , results(iii)%dh, results(iii)%dz
00110     enddo
00111   enddo
00112
00113
00114   if (moreverbose%if .and. moreverbose%names(1).eq."s") then
00115     print '(15f13.5)', &
00116       results(maxloc(results%e))%e - results(minloc(results%e))%e       ,&
00117       results(maxloc(results%n))%n - results(minloc(results%n))%n       ,&
00118       results(maxloc(results%dh))%dh - results(minloc(results%dh))%dh   ,&
00119       results(maxloc(results%dz))%dz - results(minloc(results%dz))%dz   ,&
00120       results(maxloc(results%dt))%dt - results(minloc(results%dt))%dt
00121   endif
00122
00123
00124   call cpu_time(cpu_finish)
00125   write(log%unit, '(/,"Execution time:",1x,f16.9," seconds")') cpu_finish -
      cpu_start
00126   write(log%unit, form_separator)
00127
00128 end program
```

11.9 grat/src/mod_aggf.f90 File Reference

This module contains utitlities for computing Atmospheric Gravity Green Functions.

Data Types

- module mod_aggf

11.9.1 Detailed Description

This module contains utitlities for computing Atmospheric Gravity Green Functions. In this module there are several subroutines for computing AGGF and standard atmosphere parameters

Definition in file mod_aggf.f90.

11.10 mod_aggf.f90

```
00001 !
      ================================================================================
00002 !> \file
00003 !! \brief This module contains utitlities for computing
00004 !! Atmospheric Gravity Green Functions
00005 !!
00006 !! In this module there are several subroutines for computing
00007 !! AGGF and standard atmosphere parameters
00008 !
      ================================================================================
00009 module mod_aggf
00010
00011   use constants
00012   implicit none
00013
00014 contains
00015
00016 !
      ================================================================================
00017 !> \brief Compute first derivative of AGGF with respect to temperature
00018 !! for specific angular distance (psi)
00019 !!
00020 !! optional argument define (-dt;-dt) range
00021 !! See equation 19 in \cite Huang05
00022 !! Same simple method is applied for aggf(gn) if \c aggf optional parameter
00023 !! is set to \c .true.
00024 !! \warning Please do not use \c aggf=.true. this option was added only
```

```
00025 !! for testing some numerical routines
00026 !
       ================================================================================
00027 subroutine compute_aggfdt ( psi , aggfdt , delta_ , aggf )
00028   implicit none
00029   real(dp) , intent (in) :: psi
00030   real(dp) , intent (in) , optional :: delta_
00031   logical , intent (in) , optional :: aggf
00032   real(dp) , intent (out) :: aggfdt
00033   real(dp) :: deltat , aux , h_
00034
00035   deltat = 10. !< Default value
00036   if (present( delta_) )  deltat = delta_
00037   if (present( aggf ) .and. aggf ) then
00038     h_ = 0.001 ! default if we compute dggfdh using this routine
00039       if (present( delta_) )  h_ = deltat
00040     call compute_aggf( psi , aux , h = + h_ )
00041     aggfdt = aux
00042     call compute_aggf( psi , aux , h= -h_ )
00043     aggfdt = aggfdt - aux
00044     aggfdt = aggfdt / ( 2. * h_ )
00045   else
00046     call compute_aggf( psi , aux , t_zero = t0 + deltat )
00047     aggfdt = aux
00048     call compute_aggf( psi , aux , t_zero = t0 - deltat )
00049     aggfdt = aggfdt - aux
00050     aggfdt = aggfdt / ( 2. * deltat)
00051   endif
00052
00053
00054
00055 end subroutine
00056
00057 !
       ================================================================================
00058 !> Wczytuje tablice danych AGGF
00059 !! \li merriam \cite Merriam92
00060 !! \li huang   \cite Huang05
00061 !! \li rajner  \cite Rajnerdr
00062 !!
00063 !! This is just quick solution for \c example_aggf program
00064 !! in \c grat see the more general routine \c parse_green()
00065 !
       ================================================================================
00066 subroutine read_tabulated_green ( table , author )
00067   real(dp), intent (inout),dimension(:,:), allocatable :: table
00068   character ( len = * ) , intent (in) , optional       :: author
00069   integer                                              :: i , j
00070   integer                                              :: rows , columns ,
      file_unit
00071   character (len=255)                                  :: file_name
00072
00073   rows      = 85
00074   columns   = 6
00075   file_name = '../dat/merriam_green.dat'
00076
00077   if ( present(author) ) then
00078     if ( author .eq. "huang" ) then
00079       rows    = 80
00080       columns = 5
00081       file_name = '../dat/huang_green.dat'
00082     elseif( author .eq. "rajner" ) then
00083       rows    = 85
00084       columns = 5
00085       file_name = '../dat/rajner_green.dat'
00086     elseif( author .eq. "merriam" ) then
00087     else
00088       write ( * , * ) 'cannot find specified tables, using merriam instead'
00089     endif
00090   endif
00091
00092   if (allocated (table) ) deallocate (table)
00093   allocate ( table( rows , columns ) )
00094
00095   open (newunit = file_unit , file = file_name , action='read', status='old')
00096
00097   call skip_header(file_unit)
00098
00099   do i = 1 , rows
00100     read (file_unit,*) ( table( i , j ), j = 1 , columns )
00101   enddo
00102   close(file_unit)
00103 end subroutine
00104
00105
00106 !
       ================================================================================
```

```
00107 !> This subroutine computes the value of atmospheric gravity green functions
00108 !! (AGGF) on the basis of spherical distance (psi)
00109 !
      ! ========================================================================
00110 subroutine compute_aggf (psi , aggf_val , hmin , hmax , dh ,
       if_normalization, &
00111                       t_zero , h ,  first_derivative_h , first_derivative_z ,
      fels_type )
00112   implicit none
00113   real(dp), intent(in)          :: psi       !< spherical distance from site
       [degree]
00114   real(dp), intent(in),optional :: hmin ,  & !< minimum height, starting point
      [km]      (default=0)
00115                                 hmax ,  & !< maximum height. eding point    [km]
           (default=60)
00116                                 dh ,    & !< integration step              [km]
           (default=0.0001 -> 10 cm)
00117                                 t_zero, & !< temperature at the surface     [K]
           (default=288.15=t0)
00118                                 h         !< station height                [km]
           (default=0)
00119   logical, intent(in), optional :: if_normalization , first_derivative_h ,
      first_derivative_z
00120   character (len=*) , intent(in), optional  :: fels_type
00121   real(dp), intent(out)         :: aggf_val
00122   real(dp)                      :: r , z , psir , da , dz , rho , h_min , h_max
       , h_station , j_aux
00123
00124   h_min = 0.
00125   h_max = 60.
00126   dz    = 0.0001 !mrajner 2012-11-08 13:49
00127   h_station = 0.
00128
00129   if ( present(hmin) ) h_min    = hmin
00130   if ( present(hmax) ) h_max    = hmax
00131   if ( present(  dh) )    dz    = dh
00132   if ( present(   h) ) h_station = h
00133
00134
00135   psir = psi * pi / 180.
00136
00137   da = 2 * pi * r0**2 * ( 1 - cos(1. *pi/180.) )
00138
00139
00140   aggf_val=0.
00141   do z = h_min , h_max , dz
00142
00143     r = ( ( r0 + z )**2 + (r0 + h_station)**2 &
00144       - 2.*(r0 + h_station ) *(r0+z)*cos(psir) )**(0.5)
00145     call standard_density( z , rho , t_zero = t_zero ,
      fels_type = fels_type )
00146
00147     !> first derivative (respective to station height)
00148     !> micro Gal height / km
00149     if ( present( first_derivative_h) .and. first_derivative_h ) then
00150
00151       !! see equation 22, 23 in \cite Huang05
00152       !J_aux =  (( r0 + z )**2)*(1.-3.*((cos(psir))**2)) -2.*(r0 + h_station
       )**2  &
00153       !  + 4.*(r0+h_station)*(r0+z)*cos(psir)
00154       ! aggf_val =  aggf_val -  rho * (  J_aux  /  r**5  ) * dz
00155
00156       !> direct derivative of equation 20 \cite Huang05
00157       j_aux = (2.* (r0 ) - 2 * (r0 +z )*cos(psir)) / (2. * r)
00158       j_aux =  -r - 3 * j_aux * ((r0+z)*cos(psir) - r0)
00159       aggf_val =  aggf_val +  rho * (  j_aux  /  r**4  ) * dz
00160     else
00161       !> first derivative (respective to column height)
00162       !! according to equation 26 in \cite Huang05
00163       !! micro Gal / hPa / km
00164       if ( present( first_derivative_z) .and. first_derivative_z ) then
00165         if (z.eq.h_min) then
00166             aggf_val = aggf_val  &
00167               + rho*( ((r0 + z)*cos(psir) - ( r0 + h_station ) ) / ( r**3 ) )
00168         endif
00169       else
00170         !> aggf GN
00171         !! micro Gal / hPa
00172         aggf_val = aggf_val  &
00173          + rho * ( ( (r0 + z ) * cos( psir ) - ( r0 + h_station ) ) / ( r**3 )
      ) * dz
00174       endif
00175     endif
00176   enddo
00177
00178   aggf_val = -g * da * aggf_val * 1e8 * 1000
00179
```

```
00180   !> if you put the optional parameter \c if_normalization=.false.
00181   !! this block will be skipped
00182   !! by default the normalization is applied according to \cite Merriam92
00183   if ( (.not.present(if_normalization)) .or. (if_normalization)) then
00184     aggf_val= psir * aggf_val * 1e5  / p0
00185   endif
00186
00187 end subroutine
00188
00189 !
       ================================================================================
00190 !> Compute air density for given altitude for standard atmosphere
00191 !!
00192 !! using formulae 12 in \cite Huang05
00193 !
       ================================================================================
00194 subroutine standard_density ( height , rho , t_zero ,fels_type
       )
00195
00196   implicit none
00197   real(dp) , intent(in)  ::  height !< height [km]
00198   real(dp) , intent(in), optional  :: t_zero !< if this parameter is given
00199   character(len = 22) , optional :: fels_type
00200   !! surface temperature is set to this value,
00201   !! otherwise the T0 for standard atmosphere is used
00202   real(dp) , intent(out) :: rho
00203   real(dp)  :: p ,t
00204
00205   call standard_pressure(height , p , t_zero = t_zero,
      fels_type=fels_type)
00206   call standard_temperature(height , t , t_zero = t_zero,
      fels_type=fels_type)
00207
00208   ! pressure in hPa --> Pa
00209   rho= 100 * p / ( r_air * t )
00210 end subroutine
00211
00212 ! ================================================================================
00213 !> \brief Computes pressure [hPa] for specific height
00214 !!
00215 !! See \cite US1976 or  \cite Huang05 for details.
00216 !! Uses formulae 5 from \cite Huang05.
00217 !! Simplified method if optional argument if_simplificated = .true.
00218 ! ================================================================================
00219 subroutine standard_pressure (height, pressure , &
00220           p_zero , t_zero , h_zero,  if_simplificated ,fels_type , inverted)
00221   implicit none
00222   real(dp) , intent(in)            :: height
00223   real(dp) , intent(in) , optional :: t_zero , p_zero , h_zero
00224   character(len = 22) , optional :: fels_type
00225   logical          , intent(in) , optional :: if_simplificated
00226   logical          , intent(in) , optional :: inverted
00227   real(dp), intent(out) :: pressure
00228   real(dp) ::  lambda , sfc_height , sfc_temperature , sfc_gravity , alpha ,
      sfc_pressure
00229
00230   sfc_temperature = t0
00231   sfc_pressure = p0
00232   sfc_height = 0.
00233   sfc_gravity = g0
00234
00235   if (present(h_zero)) then
00236     sfc_height = h_zero
00237     call standard_temperature(sfc_height , sfc_temperature
      )
00238     call standard_temperature(sfc_height , sfc_temperature
      )
00239     call standard_gravity(sfc_height , sfc_gravity )
00240   endif
00241
00242   if (present(p_zero)) sfc_pressure = p_zero
00243   if (present(t_zero)) sfc_temperature = t_zero
00244
00245   lambda = r_air * sfc_temperature / sfc_gravity
00246
00247   if (present(if_simplificated) .and. if_simplificated ) then
00248     ! use simplified formulae
00249     alpha = -6.5
00250     pressure = sfc_pressure  &
00251       * ( 1 + alpha / sfc_temperature * (height-sfc_height)) &
00252       ** ( -sfc_gravity / (r_air * alpha / 1000. ) )
00253   else
00254     ! use precise formulae
00255     pressure = sfc_pressure * exp( -1000. * (height -sfc_height) / lambda )
00256   endif
00257   if (present(inverted).and.inverted) then
00258     pressure = sfc_pressure  / ( exp( -1000. * (height-sfc_height) / lambda ) )
```

```
00259   endif
00260 end subroutine
00261
00262 ! ================================================================================
00263 ! > This will transfer pressure beetween different height using barometric
00264 ! formulae
00265 ! ================================================================================
00266 !> \warning OBSOLETE ROUTINE -- use \c standard_pressure() instead with
      optional args
00267 subroutine transfer_pressure (height1 , height2 , pressure1 ,
      pressure2 , &
00268   temperature , polish_meteo )
00269   real (dp) , intent (in) :: height1 , height2 , pressure1
00270   real (dp) , intent (in), optional :: temperature
00271   real (dp) :: sfc_temp , sfc_pres
00272   logical , intent (in), optional :: polish_meteo
00273   real(dp) , intent(out) :: pressure2
00274
00275   sfc_temp = t0
00276
00277   ! formulae used to reduce press to sfc in polish meteo service
00278   if (present(polish_meteo) .and. polish_meteo) then
00279     sfc_pres = exp(log(pressure1) + 2.30259 * height1*1000. &
00280       /(18400.*(1+0.00366*((temperature-273.15) + 0.0025*height1*1000.)))  )
00281   else
00282   ! different approach
00283     if(present(temperature) ) then
00284       call surface_temperature( height1 , temperature ,
      sfc_temp )
00285     endif
00286     call standard_pressure(height1 , sfc_pres , t_zero=
      sfc_temp , &
00287       inverted=.true. , p_zero = pressure1 )
00288   endif
00289
00290   ! move from sfc to height2
00291   call standard_pressure(height2 , pressure2 , t_zero=sfc_temp
      , &
00292     p_zero = sfc_pres )
00293 end subroutine
00294
00295 ! ================================================================================
00296 !> \brief Compute gravity acceleration of the Earth
00297 !! for the specific height using formula
00298 !!
00299 !! see \cite US1976
00300 ! ================================================================================
00301 subroutine standard_gravity ( height , g )
00302   implicit none
00303   real(dp), intent(in)  :: height
00304   real(dp), intent(out) :: g
00305
00306   g= g0 * ( r0 / ( r0 + height ) )**2
00307 end subroutine
00308
00309
00310 ! ================================================================================
00311 !> \brief Compute geometric height from geopotential heights
00312 ! ================================================================================
00313 real(sp) function geop2geom (geopotential_height)
00314   real (sp) :: geopotential_height
00315
00316   geop2geom = geopotential_height * (r0 / ( r0 + geopotential_height )
      )
00317 end function
00318
00319
00320 ! ================================================================================
00321 !> Iterative computation of surface temp. from given height using bisection
00322 !! method
00323 ! ================================================================================
00324 subroutine surface_temperature (height , temperature1 , &
00325   temperature2, fels_type , tolerance)
00326   real(dp) , intent(in)  :: height , temperature1
00327   real(dp) , intent(out) :: temperature2
00328   real(dp) :: temp(3) , temp_ (3) , tolerance_ = 0.1
00329   character (len=*) , intent(in), optional  :: fels_type
00330   real(sp) , intent(in), optional  :: tolerance
00331   integer :: i
00332
00333   if (present(tolerance)) tolerance_ = tolerance
00334
00335   ! searching limits
00336   temp(1)=t0-150
00337   temp(3)=t0+ 50
00338
00339   do
```

```
00340     temp(2)= ( temp(1) + temp(3) ) /2.
00341
00342     do i = 1,3
00343       call standard_temperature(height , temp_(i) , t_zero=
      temp(i) , fels_type = fels_type )
00344     enddo
00345
00346     if (abs(temperature1 - temp_(2) ) .lt. tolerance_ ) then
00347       temperature2 = temp(2)
00348       return
00349     endif
00350
00351     if ( (temperature1 - temp_(1) ) * (temperature1 - temp_(2) ) .lt.0 ) then
00352       temp(3) = temp(2)
00353     elseif( (temperature1 - temp_(3) ) * (temperature1 - temp_(2) ) .lt.0 )
      then
00354       temp(1) = temp(2)
00355     else
00356       stop "surface_temp"
00357     endif
00358   enddo
00359 end subroutine
00360 ! ================================================================================
00361 !> \brief Compute standard temperature [K] for specific height [km]
00362 !!
00363 !! if t_zero is specified use this as surface temperature
00364 !! otherwise use T0.
00365 !! A set of predifined temperature profiles ca be set using
00366 !! optional argument \argument fels_type
00367 !! \cite Fels86
00368 !
      ================================================================================
00369 subroutine standard_temperature ( height , temperature ,
       t_zero , fels_type )
00370   real(dp) , intent(in)  :: height
00371   real(dp) , intent(out) :: temperature
00372   real(dp) , intent(in), optional  :: t_zero
00373   character (len=*) , intent(in), optional  :: fels_type
00374     !< \li US standard atmosphere (default)
00375     !! \li tropical
00376     !! \li subtropical_summer
00377     !! \li subtropical_winter
00378     !! \li subarctic_summer
00379     !! \li subarctic_winter
00380   real(dp) :: aux , cn , t
00381   integer :: i,indeks
00382   real , dimension (10) :: z,c,d
00383
00384   !< Read into memory the parameters of temparature height profiles
00385   !! for standard atmosphere
00386   !! From \cite Fels86
00387   z = (/11.0 , 20.1 , 32.1 , 47.4 , 51.4 , 71.7 , 85.7, 100.0, 200.0, 300.0/)
00388   c = (/-6.5,   0.0,   1.0,   2.75,  0.0,  -2.75, -1.97,  0.0,   0.0,   0.0/)
00389   d = (/ 0.3,   1.0,   1.0,   1.0,   1.0,   1.0,   1.0,   1.0,   1.0,   1.0/)
00390   t = t0
00391
00392   if ( present(fels_type)) then
00393     if (fels_type .eq. "US1976" ) then
00394     elseif(fels_type .eq. "tropical" ) then
00395       z=(/ 2.0  , 3.0, 16.5 , 21.5 , 45.0 , 51.0, 70.0 , 100.0 , 200.0 , 300.0
      /)
00396       c=(/-6.0 , -4.0, -6.7  , 4.0  , 2.2  , 1.0, -2.8  , -0.27  , 0.0   , 0.0
      /)
00397       d=(/ 0.5  , 0.5 , 0.3  , 0.5  , 1.0  , 1.0 , 1.0   , 1.0   , 1.0   , 1.0
      /)
00398       t=300.0
00399     elseif(fels_type .eq. "subtropical_summer" ) then
00400       z = (/ 1.5  , 6.5  , 13.0 , 18.0 , 26.0 , 36.0  , 48.0 ,  50.0 ,  70.0 ,
      100.0  /)
00401       c = (/-4.0 , -6.0  , -6.5  , 0.0  , 1.2  , 2.2  ,  2.5  ,  0.0   ,-3.0
      ,-0.025/)
00402       d = (/ 0.5  , 1.0   , 0.5  , 0.5  , 1.0  , 1.0   , 2.5   , 0.5   , 1.0
      , 1.0  /)
00403       t = 294.0
00404     elseif(fels_type .eq. "subtropical_winter" ) then
00405       z = (/ 3.0  ,10.0  , 19.0 , 25.0 , 32.0 , 44.5  , 50.0 ,  71.0 ,  98.0 ,
      200.0 /)
00406       c = (/-3.5 , -6.0  , -0.5  , 0.0  , 0.4  , 3.2  ,  1.6  , -1.8   , 0.7
      , 0.0 /)
00407       d = (/ 0.5  , 0.5   , 1.0  , 1.0  , 1.0  , 1.0   , 1.0   , 1.0   , 1.0
      , 1.0 /)
00408       t = 272.2
00409     elseif(fels_type .eq. "subarctic_summer" ) then
00410       z = (/  4.7 , 10.0 , 23.0 , 31.8 , 44.0 , 50.2  , 69.2 , 100.0 , 200.0 ,
      300.0 /)
00411       c = (/ -5.3 , -7.0 ,  0.0 ,  1.4 ,  3.0 , 0.7  , -3.3  , -0.2   , 0.0 ,
       0.0 /)
```

```
00412       d = (/  0.5 ,  0.3 ,  1.0 ,  1.0 ,  2.0 , 1.0   , 1.5   , 1.0   , 1.0 ,
      1.0 /)
00413       t = 287.0
00414     elseif(fels_type .eq. "subarctic_winter" ) then
00415       z = (/  1.0 ,  3.2 ,  8.5 , 15.5 , 25.0 , 30.0 , 35.0 , 50.0 , 70.0 , 100
     .0 /)
00416       c = (/  3.0 , -3.2 , -6.8 ,  0.0 , -0.6 ,  1.0 ,  1.2 ,  2.5 , -0.7 ,  -1
     .2 /)
00417       d = (/  0.4 ,  1.5 ,  0.3 ,  0.5 ,  1.0 ,  1.0 ,  1.0 ,  1.0 ,  1.0 ,   1
     .0 /)
00418       t = 257.1
00419     else
00420       print * ,
00421 "unknown fels_type argument: &        using US standard atmosphere 1976
      instead"
00422     endif
00423   endif
00424
00425   if (present(t_zero) ) then
00426     t=t_zero
00427   endif
00428
00429   do i=1,10
00430     if (height.le.z(i)) then
00431       indeks=i
00432       exit
00433     endif
00434   enddo
00435
00436   aux = 0.
00437   do i = 1 , indeks
00438     if (i.eq.indeks) then
00439       cn = 0.
00440     else
00441       cn = c(i+1)
00442     endif
00443       aux = aux + d(i) * ( cn - c(i) )  * log( cosh( (height - z(i)) / d(i) ) /
      cosh(z(i)/d(i)) )
00444   enddo
00445   temperature = t + c(1) * height/2. + aux/2.
00446 end subroutine
00447
00448 !
      ================================================================================
00449 !> \brief Compute AGGF GN for thin layer
00450 !!
00451 !! Simple function added to provide complete module
00452 !! but this should not be used for atmosphere layer
00453 !! See eq p. 491 in \cite Merriam92
00454 !
      ================================================================================
00455 real function gn_thin_layer (psi)
00456   implicit none
00457   real(dp) , intent(in) :: psi
00458   real(dp) :: psir
00459
00460   psir = psi * pi / 180.
00461   gn_thin_layer = 1.627 * psir / sin( psir / 2. )
00462 end function
00463
00464
00465 !
      ================================================================================
00466 !> \brief returns numbers of arguments for n times denser size
00467 !!
00468 !! i.e. * * * *  -->  * . . * . . * . . * (3 times denser)
00469 !
      ================================================================================
00470 integer function size_ntimes_denser (size_original, ndenser)
00471   integer, intent(in) :: size_original , ndenser
00472   size_ntimes_denser= (size_original - 1 ) * (ndenser +1 ) +
     1
00473 end function
00474
00475 !
      ================================================================================
00476 !> \brief Bouger plate computation
00477 !!
00478 !
      ================================================================================
00479 real(dp) function bouger ( R_opt )
00480   real(dp), optional :: r_opt !< height of point above the cylinder
00481   real(dp) :: aux
00482   real(dp) :: r
00483   real(dp) ::  h = 8.84 ! scale height of standard atmosphere
00484
00485   aux = 1
```

```
00486
00487   if (present( r_opt ) ) then
00488     r = r_opt
00489     aux = h  + r  - sqrt(  r**2  + (h/2. ) ** 2 )
00490     bouger = 2 * pi * g  * aux
00491   else
00492     aux = h
00493     bouger = 2 * pi * g * aux
00494     return
00495   endif
00496 end function
00497 !
      ================================================================================
00498 !> \brief Bouger plate computation
00499 !! see eq. page 288 \cite Warburton77
00500 !
      ================================================================================
00501 real(dp) function simple_def (R)
00502   real(dp) :: r ,delta
00503
00504   delta = 0.22e-11 * r
00505
00506   simple_def = g0 / r0 * delta * ( 2. - 3./2. * rho_crust / rho_earth
      &
00507     -3./4. * rho_crust / rho_earth * sqrt(2* (1. )) ) * 1000
00508 end function
00509
00510 !polish_meteo
00511
00512 end module
```

11.11 grat/src/value_check.f90 File Reference

Functions/Subroutines

- program **value_check**

11.11.1 Detailed Description

Definition in file value_check.f90.

11.12 value_check.f90

```
00001 !> \file
00002 !! \mainpage
00003 !! \brief ...put...
00004 !! \page value_check-h value_check
00005 !!    \include value_check.hlp
00006 !!
00007
00008 program value_check
00009   use get_cmd_line
00010   use mod_data
00011   use ieee_arithmetic
00012   implicit none
00013
00014   real (sp) , allocatable , dimension(:) :: val
00015   integer :: i,ii ,j ,start , imodel
00016
00017   call intro(program_calling = "value_check" )
00018   call print_settings(program_calling = "value_check")
00019
00020
00021   do i = 1 , size(model)
00022     if (model(i)%if) call read_netcdf(model(i))
00023   enddo
00024
00025   allocate (val(nmodels(model)))
00026
00027   start =0
00028   if (size(dates).gt.0) start=1
00029
00030   do j = start , size (dates)
00031     do i = 1 , size(model)
```

```
00032       if (model(i)%if) then
00033         call get_variable( model(i) , date = dates(j)%date)
00034       endif
00035     enddo
00036
00037     do i = 1 , size(sites)
00038       ! add time stamp if -D option was specified
00039       if (j.gt.0) then
00040         write (output%unit , '(f15.3,x,i4.4,5(i2.2))' , advance = "no" ) dates(
      j)%mjd , dates(j)%date
00041       endif
00042
00043       imodel = 0
00044       do ii = 1 , size (model)
00045         if (model(ii)%if .or. model(ii)%if_constant_value) then
00046           imodel = imodel + 1
00047           if (model(ii)%if) then
00048             call get_value( model(ii) , sites(i)%lat , sites(i)%lon , val(
      imodel) , method = model(ii)%interpolation )
00049           elseif(model(ii)%if_constant_value) then
00050             val(imodel) = model(ii)%constant_value
00051           endif
00052         endif
00053       enddo
00054
00055       write (output%unit ,   '(30f15.4, 1x)') , sites(i)%lat, sites(i)%lon, val
00056
00057     enddo
00058
00059   enddo
00060
00061 end program
```

11.13 grat/tmp/compar.sh File Reference

11.13.1 Detailed Description

Definition in file compar.sh.

11.14 compar.sh

```
00001 #!/bin/bash -
00002 ## \file
00003 #
      ================================================================================
00004 #          FILE: compar.sh
00005 #         USAGE: ./compar.sh
00006 #   DESCRIPTION:
00007 #       OPTIONS: ---
00008 #        AUTHOR: mrajner
00009 #       CREATED: 13.12.2012 21:15:45 CET
00010 #      REVISION:  ---
00011 #
      ================================================================================
00012
00013 set -o nounset                              # Treat unset variables as an error
00014
00015   WEN="/home/mrajner/pub/2012_wenecja/dane"
00016   SFC="/home/mrajner/src/grat/data/ncep_reanalysis/pres.sfc.2011.nc:pres"
00017   TMP="../data/ncep_reanalysis/air.sig995.2011.nc:air:lon:lat:level:time"
00018   LND="../data/landsea/test.grd:z:x:y"
00019   HGT="../data/topo/ETOPO2v2g_f4.nc:z:x:y"
00020 #  LND="../data/landsea/test_.grd:z:x:y"
00021 #  POL= ../polygon/tmp.poly
00022
00023
00024   numer=354
00025   I=1
00026
00027   TAB=($(sed -ne 2p ${WEN}/szereg_${numer}.txt))
00028   L=$(echo ${TAB[4]}|tr "," " ")
00029   B=$(echo ${TAB[3]}|sed 's/,//')
00030
00031   echo $B $L
00032 #../bin/grat -V -Stmp,${B},${L}  -F${SFC},${TMP},${HGT},${LND}   -Ghuang,huang,
      huang,huang,,1:1     -D20110218,2012  -o${numer}_${I}_5  -I1
00033   grat -V -L:G -Stmp,${B},${L}  -F${SFC},${TMP},${HGT},${LND}   -G,rajner,,,,1:
```

```
     1    -D2011,2012  -o${numer}_${I}_3  -I1
00034 #../bin/grat -V -Stmp,${B},${L}  -F${SFC},${TMP},,${LND} -Bi  -G,,,,,1:1
       -D20110101,20111231  -o${numer}_${I}_3  -I2
00035 #../bin/value_check -V -Stmp,${B},${L}  -F${TMP}     -D20110101,20111231
       -o${numer}_${I}_6  -I2
00036 #../bin/grat  -Stmp,${B},${L}  -F${SFC},${TMP},,${LND} -Bi  -G,,,,,1:1  -L:G
00037 #../bin/grat  -Stmp,${B},${L}  -F${SFC},${TMP},,${LND} -Bi
       -Grajner,rajner,rajner,rajner,,1:1  -L:G
```

Chapter 12

Example Documentation

12.1 example_aggf.f90

```
00001 ! ========================================================================
00002 !! \brief This program shows some example of using AGGF module
00003 !! \author Marcin Rajner
00004 !! \date 20121108
00005 !!
00006 !! The examples are in contained subroutines
00007 ! ========================================================================
00008 program example_aggf
00009
00010   !> module with subroutines for calculating Atmospheric Gravity Green
      Fucntions
00011   use mod_aggf
00012   use constants
00013   implicit none
00014
00015
00016
00017
00018  print *, "...standard1976 ()"
00019  call standard1976()
00020 !print *, "...aggf_resp_hmax ()"
00021 ! call aggf_resp_hmax ()
00022 !print *, "...aggf_resp_dz ()"
00023 ! call aggf_resp_dz ()
00024 !print *, "...aggf_resp_t ()"
00025 ! call aggf_resp_t ()
00026 !print *, "...aggf_resp_h ()"
00027 ! call aggf_resp_h ()
00028 !print *, "...aggfdt_resp_dt ()"
00029 ! call aggfdt_resp_dt ()
00030 !print *, "...compare_fels_profiles ()"
00031 ! call compare_fels_profiles ()
00032 !print *, "...compute_tabulated_green_functions ()"
00033 ! call compute_tabulated_green_functions ()
00034 !print *, "...aggf_thin_layer ()"
00035 ! call aggf_thin_layer ()
00036 !print *, "...aggf_resp_fels_profiles ()"
00037 ! call aggf_resp_fels_profiles ()
00038 !print *, "...compare_tabulated_green_functions ()"
00039 ! call compare_tabulated_green_functions ()
00040 !print *, "...simple_atmospheric_model()"
00041 ! call simple_atmospheric_model()
00042
00043 contains
00044
00045 ! ========================================================================
00046 !> \brief Reproduces data to Fig.~3 in \cite Warburton77
00047 ! ========================================================================
00048 subroutine simple_atmospheric_model ()
00049   real(dp) :: r ! km
00050   integer :: iunit
00051
00052   open (newunit=iunit,file="/home/mrajner/dr/rysunki/simple_approach.dat" ,&
00053     action = "write")
00054     do r = 0. , 25*8
00055     write ( iunit ,  * ) , r , bouger( r_opt= r) * 1e8, & !conversion to
      microGal
00056       simple_def(r) * 1e8
00057   enddo
```

```
00058
00059 end subroutine
00060 ! ============================================================================
00061 !> \brief Compare tabulated green functions from different authors
00062 ! ============================================================================
00063 subroutine compare_tabulated_green_functions ()
00064   integer :: i , j , file_unit , ii , iii
00065   real(dp), dimension(:,:), allocatable :: table , results
00066   real(dp), dimension(:,:), allocatable :: parameters
00067   real(dp), dimension(:), allocatable :: x1, y1 ,x2 , y2 , x, y ,
      x_interpolated, y_interpolated
00068   integer :: how_many_denser
00069   character(len=255), dimension(3) :: authors
00070   integer , dimension(3) :: columns
00071
00072   authors=["rajner", "merriam" , "huang"]
00073   ! selected columns for comparison in appropriate tables
00074   columns=[2 , 2, 2]
00075
00076   how_many_denser=0
00077
00078   ! reference author
00079   call read_tabulated_green(table , author = authors(1) )
00080   allocate (results(size_ntimes_denser(size(table(:,1)),
      how_many_denser) , 0 : size(authors) ))
00081
00082   ! fill abscissa in column 0
00083   ii = 1
00084   do i = 1 ,  size (table(:,1) ) - 1
00085     do j = 0 , how_many_denser
00086         results(ii,0) = table(i,1 ) + j * (table(i+1, 1) -table(i,1) ) / (
      how_many_denser + 1 )
00087         ii=ii+1
00088     enddo
00089   enddo
00090   ! and the last element
00091   results( size (results(:,0) )  , 0) =  table( size(table(:,1)) ,1 )
00092
00093   ! take it as main for all series
00094   allocate(x_interpolated( size ( results(:,0))))
00095   x_interpolated = results(:,0)
00096
00097   open (newunit = file_unit , file = "../examples/compare_aggf.dat", action=
      "write")
00098
00099   ! for every author
00100   do i= 1, size(authors)
00101     print * , trim( authors( i ) )
00102     call read_tabulated_green(table , author = authors(i) )
00103     allocate(x( size (table(:,1))))
00104     allocate(y( size (table(:,2))))
00105     x = table(:,1)
00106     y = table(:, columns(i))
00107     call spline_interpolation( x , y , x_interpolated,
      y_interpolated )
00108     if (i.gt.1) then
00109       y_interpolated = ( y_interpolated - results(:,1) ) / results(:,1)  * 100.
00110     endif
00111
00112     results(:, i ) = y_interpolated
00113     deallocate(x,y)
00114   enddo
00115
00116   write (file_unit , '(<size(results(1,:))>f20.5)' ) ( results(i , :) , i = 1 ,
       size(results( :,1)) )
00117   close(file_unit)
00118 end subroutine
00119
00120 ! ============================================================================
00121 !> \brief Compute AGGF and derivatives
00122 ! ============================================================================
00123 subroutine compute_tabulated_green_functions ()
00124   integer :: i , file_unit
00125   real(dp) :: val_aggf , val_aggfdt ,val_aggfdh, val_aggfdz
00126   real(dp), dimension(:,:), allocatable :: table , results
00127
00128   ! Get the spherical distances from Merriam92
00129   call read_tabulated_green( table , author = "merriam")
00130
00131   open  ( newunit = file_unit, &
00132           file    = '../dat/rajner_green.dat', &
00133           action  = 'write' &
00134         )
00135
00136   ! print header
00137   write ( file_unit,*) '# This is set of AGGF computed using module ', &
00138   'aggf from grat software'
```

```
00139   write ( file_unit,*) '# Normalization according to Merriam92'
00140   write ( file_unit,*) '# Marcin Rajner'
00141   write ( file_unit,*) '# For detail see www.geo.republika.pl'
00142   write ( file_unit,'(10(a23))')  '#psi[deg]', &
00143     'GN[microGal/hPa]'       , 'GN/dT[microGal/hPa/K]' , &
00144     'GN/dh[microGal/hPa/km]' , 'GN/dz[microGal/hPa/km]'
00145
00146   do i= 1, size(table(:,1))
00147     call compute_aggf( table(i,1) , val_aggf   )
00148     call compute_aggfdt( table(i,1) , val_aggfdt )
00149     call compute_aggf( table(i,1) , val_aggfdh , first_derivative_h
      =.true. )
00150     call compute_aggf( table(i,1) , val_aggfdz , first_derivative_z
      =.true. )
00151     write ( file_unit, '(10(e23.5))' ) &
00152       table(i,1) , val_aggf , val_aggfdt , val_aggfdh, val_aggfdz
00153   enddo
00154   close(file_unit)
00155 end subroutine
00156
00157 ! ============================================================================
00158 !> \brief Compare different vertical temperature profiles impact on AGGF
00159 ! ============================================================================
00160 subroutine aggf_resp_fels_profiles ()
00161   character (len=255) ,dimension (6) :: fels_types
00162   real (dp) :: val_aggf
00163   integer :: i , j, file_unit
00164   real(dp), dimension(:,:), allocatable :: table
00165
00166   ! All possible optional arguments for standard_temperature
00167   fels_types = (/ "US1976"             , "tropical",   &
00168                   "subtropical_summer" , "subtropical_winter" , &
00169                   "subarctic_summer"   , "subarctic_winter"    /)
00170
00171   open  ( newunit = file_unit, &
00172           file    = '../examples/aggf_resp_fels_profiles.dat' , &
00173           action  = 'write' &
00174         )
00175
00176   call read_tabulated_green(table)
00177
00178   ! print header
00179   write ( file_unit , '(100(a20))' ) &
00180     'psi', ( trim( fels_types(i) ) , i = 1 , size (fels_types) )
00181
00182   ! print results
00183   do i = 1 , size (table(:,1))
00184     write (file_unit, '(f20.6$)') table(i,1)
00185     do j = 1 , size(fels_types)
00186       call compute_aggf(table(i,1), val_aggf ,fels_type=fels_types(
      j))
00187       write (file_unit, '(f20.6$)') val_aggf
00188     enddo
00189     write(file_unit, *)
00190   enddo
00191   close(file_unit)
00192 end subroutine
00193
00194
00195 ! ============================================================================
00196 !> \brief Compare different vertical temperature profiles
00197 !!
00198 !! Using tables and formula from \cite Fels86
00199 ! ============================================================================
00200 subroutine compare_fels_profiles ()
00201   character (len=255) ,dimension (6) :: fels_types
00202   real (dp) :: height , temperature
00203   integer :: i , file_unit
00204
00205   ! All possible optional arguments for standard_temperature
00206   fels_types = (/ "US1976"             , "tropical",   &
00207                   "subtropical_summer" , "subtropical_winter" , &
00208                   "subarctic_summer"   , "subarctic_winter"    /)
00209
00210   open  ( newunit = file_unit, &
00211           file    = '../examples/compare_fels_profiles.dat' , &
00212           action  = 'write' &
00213         )
00214
00215   ! Print header
00216   write ( file_unit , '(100(a20))' ) &
00217     'height', ( trim( fels_types(i) ) , i = 1 , size (fels_types) )
00218
00219   ! Print results
00220   do height = 0. , 70. , 1.
00221     write ( file_unit , '(f20.3$)' ) , height
00222     do i = 1 , size (fels_types)
```

```
00223       call standard_temperature &
00224         ( height , temperature , fels_type = fels_types(i) )
00225       write ( file_unit , '(f20.3$)' ),  temperature
00226     enddo
00227     write ( file_unit , * )
00228   enddo
00229   close(file_unit)
00230 end subroutine
00231
00232 ! ========================================================================
00233 !> \brief Computes AGGF for different site height (h)
00234 ! ========================================================================
00235 subroutine aggf_resp_h ()
00236   real(dp), dimension(:,:), allocatable :: table , results
00237   integer :: i, j, file_unit , ii
00238   real(dp) :: val_aggf
00239
00240   ! Get the spherical distances from Merriam92
00241   call read_tabulated_green( table , author = "merriam")
00242
00243   ! Specify the output table and put station height in first row
00244   allocate ( results( 0 : size (table(:,1)) , 7 ) )
00245   results(0,1) = 1./0     ! Infinity in first header
00246   results(0,3) = 0.0      !   0 m
00247   results(0,3) = 0.001    !   1 m
00248   results(0,4) = 0.01     !  10 m
00249   results(0,5) = 0.1      ! 100 m
00250   results(0,6) = 1.       !   1 km
00251   results(0,7) = 10.      !  10 km
00252
00253   ! write results to file
00254   open ( &
00255     newunit = file_unit, &
00256     file    = '../examples/aggf_resp_h.dat', &
00257     action  = 'write' &
00258     )
00259
00260   write (file_unit, '(8(F20.8))' ) results(0, :)
00261   do i =1 , size (table(:,1))
00262     ! denser sampling
00263     do ii = 0,8
00264       results( i , 1 )  = table(i,1) + ii * (table(i+1,1) - table(i,1)) / 9.
00265       ! only compute for small spherical distances
00266       if (results(i, 1) .gt. 0.2 ) exit
00267       write (file_unit, '(F20.7,$)') , results(i,1)
00268       do j =  2 , size(results(1,: ) )
00269         call compute_aggf(results(i,1) , val_aggf, dh=0.0001, h =
      results(0,j))
00270         results(i,j) = val_aggf
00271         write (file_unit,'(f20.7,1x,$)') results(i,j)
00272       enddo
00273       write (file_unit,*)
00274     enddo
00275   enddo
00276   close (file_unit)
00277 end subroutine
00278
00279 ! ========================================================================
00280 !> \brief This computes AGGF for different surface temperature
00281 ! ========================================================================
00282 subroutine aggf_resp_t ()
00283   real(dp), dimension(:,:), allocatable :: table , results
00284   integer :: i, j , file_unit
00285   real(dp) :: val_aggf
00286
00287   ! read spherical distances from Merriam
00288   call read_tabulated_green( table )
00289
00290   ! Header in first row with surface temperature [K]
00291   allocate ( results(0 : size (table(:,1)) , 4 ) )
00292   results(0,1) = 1./0
00293   results(0,2) = t0 +   0.
00294   results(0,3) = t0 +  15.0
00295   results(0,4) = t0 + -45.0
00296   do i =1 , size (table(:,1))
00297     results( i , 1 )  = table(i,1)
00298     do j =  2 , 4
00299     call compute_aggf( results(i , 1 ) , val_aggf, dh = 0.00001,
      t_zero = results(0, j) )
00300     results(i,j) = val_aggf
00301     enddo
00302   enddo
00303
00304   ! Print results to file
00305   open ( newunit = file_unit , &
00306          file    = '../examples/aggf_resp_t.dat' , &
00307          action  = 'write')
```

```
00308   write (file_unit , '(4F20.5)' ) &
00309     ( (results(i,j) , j=1,4) , i = 0, size ( table(:,1) ) )
00310   close (file_unit)
00311 end subroutine
00312
00313 ! ========================================================================
00314 !> \brief This computes AGGFDT for different dT
00315 ! ========================================================================
00316 subroutine aggfdt_resp_dt ()
00317   real(dp), dimension(:,:), allocatable :: table , results
00318   integer :: i, j , file_unit
00319   real(dp) :: val_aggf
00320
00321   ! read spherical distances from Merriam
00322   call read_tabulated_green( table )
00323
00324   ! Header in first row with surface temperature [K]
00325   allocate ( results(0 : size (table(:,1)) , 6 ) )
00326   results(0,1) = 1./0
00327   results(0,2) = 1.
00328   results(0,3) = 5.
00329   results(0,4) = 10.
00330   results(0,5) = 20.
00331   results(0,6) = 50.
00332   do i =1 , size (table(:,1))
00333     results( i , 1 )  = table(i,1)
00334     do j =  2 , 6
00335     call compute_aggfdt( results(i , 1 ) , val_aggf, results(0, j
      ) )
00336     results(i,j) = val_aggf
00337     enddo
00338   enddo
00339
00340   ! Print results to file
00341   open ( newunit = file_unit , &
00342          file    = '../examples/aggfdt_resp_dt.dat' , &
00343          action  = 'write')
00344   write (file_unit , '(6F20.5)' ) &
00345     ( (results(i,j) , j=1,6) , i = 0, size ( table(:,1) ) )
00346   close (file_unit)
00347 end subroutine
00348
00349 ! ========================================================================
00350 !> \brief This computes AGGF for different height integration step
00351 ! ========================================================================
00352 subroutine aggf_resp_dz ()
00353   real(dp), dimension(:,:), allocatable :: table , results
00354   integer :: file_unit , i , j
00355   real(dp) :: val_aggf
00356
00357   open ( newunit = file_unit, &
00358          file    = '../examples/aggf_resp_dz.dat', &
00359          action='write')
00360
00361   ! read spherical distances from Merriam
00362   call read_tabulated_green( table )
00363
00364   ! Differences in AGGF(dz) only for small spherical distances
00365   allocate ( results( 0 : 29 , 0: 5 ) )
00366   results = 0.
00367
00368   ! Header in first row [ infty and selected dz follow on ]
00369   results(0,0) = 1./0
00370   results(0,1:5)=(/ 0.0001, 0.001, 0.01, 0.1, 1./)
00371
00372   do i = 1 , size ( results(:,1) ) - 1
00373     results(i,0) = table(i , 1 )
00374     do j = 1 , size (results(1,:) ) - 1
00375     call compute_aggf( results(i,0) , val_aggf , dh = results(0,j)
    )
00376     results(i, j) =  val_aggf
00377     enddo
00378
00379     ! compute relative errors from column 2 for all dz with respect to column 1
00380     results(i,2:) = abs((results(i,2:) - results(i,1)) / results(i,1) * 100 )
00381   enddo
00382
00383   ! write result to file
00384   write ( file_unit , '(<size(results(1,:))>f14.6)' ) &
00385     ((results(i,j), j=0,size(results(1,:)) - 1), i=0,size(results(:,1)) - 1)
00386   close(file_unit)
00387 end subroutine
00388
00389 ! ========================================================================
00390 !> \brief This computes standard atmosphere parameters
00391 !!
00392 !! It computes temperature, gravity, pressure, pressure (simplified formula)
```

```
00393 !! density for given height
00394 ! ============================================================================
00395 subroutine standard1976  !()
00396   real(dp) :: height , temperature , gravity , pressure , pressure2 , density
00397   integer :: file_unit
00398
00399   open ( newunit = file_unit , &
00400          file    = '../examples/standard1976.dat', &
00401          action  = 'write' )
00402   ! print header
00403   write ( file_unit , '(6(a12))' ) &
00404     'height[km]', 'T[K]' , 'g[m/s2]' , 'p[hPa]', 'p_simp[hPa]' , 'rho[kg/m3]'
00405   do height=0.,98.
00406     call standard_temperature( height , temperature )
00407     call standard_gravity( height , gravity )
00408     call standard_pressure( height , pressure )
00409     call standard_pressure( height , pressure2 ,
      if_simplificated = .true. )
00410     call standard_density( height , density )
00411     ! print results to file
00412     write( file_unit,'(5f12.5, e12.3)'), &
00413     height,temperature , gravity , pressure , pressure2 , density
00414   enddo
00415   close( file_unit )
00416 end subroutine
00417
00418 ! ============================================================================
00419 !> \brief This computes relative values of AGGF for different atmosphere
00420 !! height integration
00421 ! ============================================================================
00422 subroutine aggf_resp_hmax ()
00423   real (dp) , dimension (10) :: psi
00424   real (dp) , dimension (:)   , allocatable :: heights
00425   real (dp) , dimension (:,:) , allocatable :: results
00426   integer :: file_unit , i , j
00427   real(dp) :: val_aggf
00428
00429   ! selected spherical distances
00430   psi=(/0.000001, 0.000005,0.00001, 1,  2, 3 , 5, 10 , 90 ,  180 /)
00431
00432   ! get heights (for nice graph) - call auxiliary subroutine
00433   call aux_heights( heights )
00434
00435   open ( newunit = file_unit , &
00436          file    = '../examples/aggf_resp_hmax.dat', &
00437          action  = 'write')
00438
00439   allocate ( results( 0:size(heights)-1 , 1+size(psi) ) )
00440
00441   do j=0 , size (results(:,1))
00442       results( j , 1 ) = heights(j)
00443
00444     do i = 1 , size(psi)
00445       call compute_aggf( psi(i) , val_aggf , hmax = heights(j) )
00446       results(j,i+1) = val_aggf
00447
00448       !> Relative value of aggf depending on integration height
00449       if (j.gt.0) then
00450         results(j,i+1) = results(j,i+1) / results(0,i+1) * 100
00451       endif
00452     enddo
00453   enddo
00454
00455   ! print header
00456   write(file_unit , '(a14,SP,100f14.5)' ),"#wys\psi", (psi(j) , j= 1,size(psi))
00457   ! print results
00458   do i=1, size (results(:,1))-1
00459     write(file_unit, '(100f14.3)' ) (results(i,j), j = 1, size(psi)+1 )
00460   enddo
00461   close(file_unit)
00462 end subroutine
00463
00464 ! ============================================================================
00465 !> \brief Auxiliary subroutine -- height sampling for semilog plot
00466 ! ============================================================================
00467 subroutine aux_heights ( table )
00468   real(dp) , dimension (:), allocatable, intent(inout) :: table
00469   real(dp) , dimension (0:1000) :: heights
00470   real(dp) :: height
00471   integer :: i , count_heights
00472
00473   heights(0) =60
00474   i=0
00475   height=-0.001
00476   do while (height.lt.60)
00477     i=i+1
00478     if (height.lt.0.10) then
```

```
00479       height=height+2./1000
00480     elseif(height.lt.1) then
00481       height=height+50./1000
00482     else
00483       height=height+1
00484     endif
00485     heights(i)= height
00486     count_heights=i
00487   enddo
00488   allocate ( table( 0 : count_heights ) )
00489   table(0 : count_heights ) = heights( 0 : count_heights )
00490 end subroutine
00491
00492 subroutine aggf_thin_layer ()
00493   integer :: file_unit , i
00494   real(dp) , dimension (:,:), allocatable :: table
00495
00496   ! read spherical distances from Merriam
00497   call read_tabulated_green(table)
00498   do i = 1 , size (table(:,1))
00499     write(*,*) table(i,1:2) , gn_thin_layer(table(i,1))
00500   enddo
00501
00502 end subroutine
00503 end program
```

12.2 grat_usage.sh

```
#!/bin/bash -
#
      ================================================================================
#          FILE: grat_usage.sh
#         USAGE: ./grat_usage.sh
#        AUTHOR: mrajner
#       CREATED: 12.01.2013 16:44:52 CET
#
      ================================================================================

set -o nounset                              # Treat unset variables as an error

# after successfully source compilation you should be able to run this command
# make sure the grat command can be found in your executables path

  grat \
    -S JOZE,52.1,21.1,110 \
    -F ../data/ncep_reanalysis/pres.sfc.2011.nc:pres \
    -G rajner \
    -D 201101,2012

    # specify the station: name,lat[decDeg],lon[decDeg],height[m]

# The spaces are not mandatory. The program searches for the next switch
       (starting with "-")
# or field separator "," ":"
# thus the commands below are equal:

# grat -F ../file , file2: field1  :field2 ,
# grat -F../file,file2:field1:field2,

# this is extreemly useful if one use <TAB> completion for path and filenames
```

Appendix A

Polygon

This examples show how the exclusion of selected polygons works

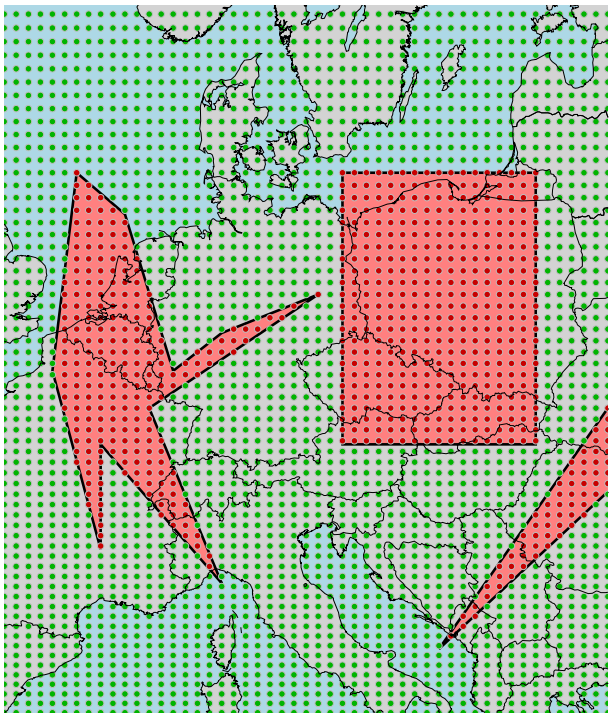


Figure A.1: If only excluded polygons (red area) are given all points falling in it will be excluded (red points) all other will be included

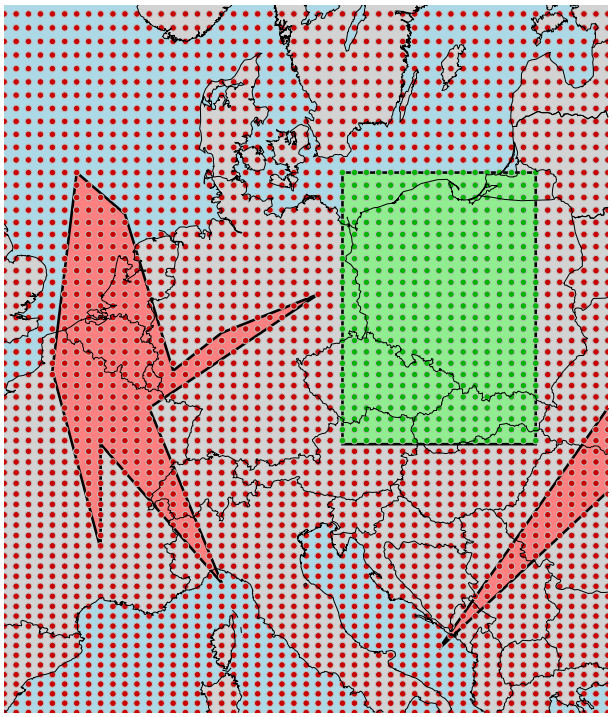


Figure A.2: If at least one included are are given (green area) than all points which not fall into included area will be excluded

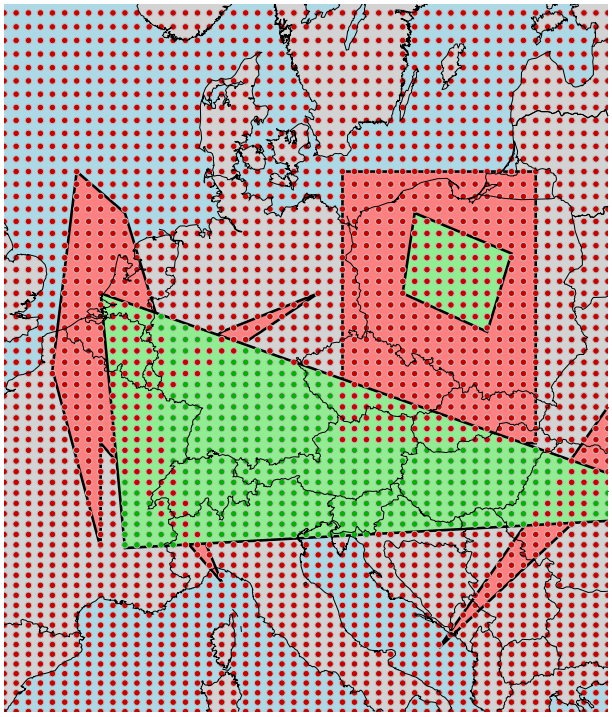


Figure A.3: If there is overlap of polygons the exclusion has higher priority

Appendix B

Interpolation

Figure B.1: Interpoloation

Bibliography

netcdf. URL https://www.unidata.ucar.edu/software/netcdf/.

D. C. Agnew. NLOADF: a program for computing ocean-tide loading. *J. Geophys. Res.*, 102:5109–5110, 1997.

COESA Comitee on extension of the Standard Atmosphere. U.S. Standard Atmosphere, 1976. Technical report, 1976.

S. B. Fels. Analytic Representations of Standard Atmosphere Temperature Profiles. *Journal of Atmospheric Sciences*, 43:219–222, January 1986. doi: 10.1175/1520-0469(1986)043<0219:AROSAT>2.0.CO;2.

Y. Huang, J. Guo, C. Huang, and X. Hu. Theoretical computation of atmospheric gravity green's functions. *Chinese Journal of Geophysics*, 48(6):1373–1380, 2005.

J. B. Merriam. Atmospheric pressure and gravity. *Geophysical Journal International*, 109(3):488–500, 1992. ISSN 1365-246X. doi: 10.1111/j.1365-246X.1992.tb00112.x. URL http://dx.doi.org/10.1111/j.1365-246X.1992.tb00112.x.

R. J. Warburton and J. M. Goodkind. The influence of barometeic-pressure variations on gravity. *Geophys. J. R. Astron. Society*, 48:281–292, 1977.

Index