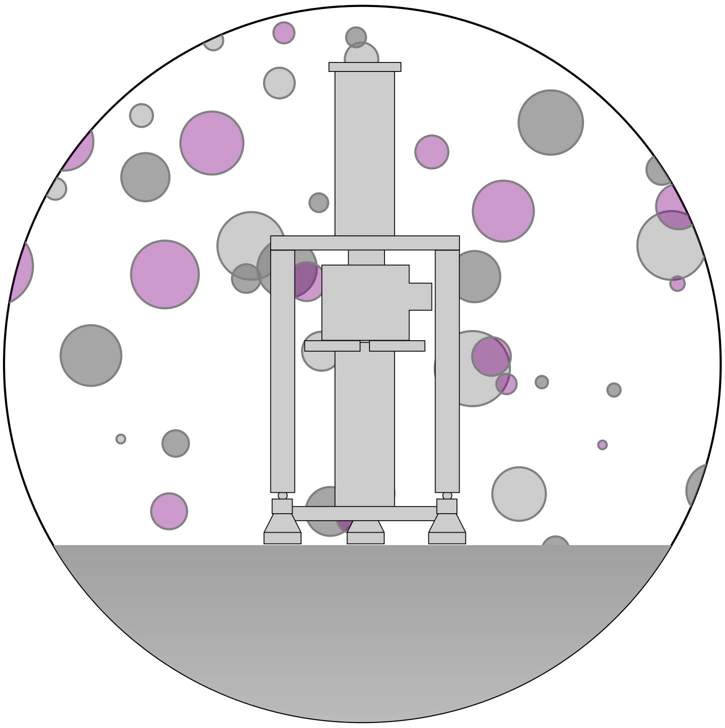


grat v. 1.0 Manual

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 more easy.

Version

v. 1.0

Date

2012-12-12

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 yours (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.

Chapter 2

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

Chapter 3

Data Type Index

3.1 Data Types List

Here are the data types with brief descriptions:

Chapter 4

File Index

4.1 File List

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

Chapter 5

Data Type Documentation

5.1 get_cmd_line::additional_info Type Reference

Public Attributes

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

5.1.1 Detailed Description

Definition at line 57 of file get_cmd_line.f90.

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

- /home/mrajner/src/grat/src/get_cmd_line.f90

5.2 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, 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 geometric_height (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.

5.2.1 Detailed Description

Definition at line 9 of file aggf.f90.

5.2.2 Member Function/Subroutine Documentation

5.2.2.1 real(dp) function aggf::bouger (real(dp), optional *R_opt*)

Bouger plate computation.

Parameters

r_opt	height of point above the cylinder

Definition at line 469 of file aggf.f90.

5.2.2.2 subroutine 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 aggf.f90.

5.2.2.3 subroutine 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 aggf.f90.

5.2.2.4 real function 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 445 of file aggf.f90.

5.2.2.5 subroutine 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 Rajner [2013]

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

Definition at line 66 of file aggf.f90.

5.2.2.6 real(dp) function aggf::simple_def (real(dp) *R*)

Bouger plate computation see eq. page 288.

Warburton and Goodkind [1977]

Definition at line 491 of file aggf.f90.

5.2.2.7 integer function 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 460 of file aggf.f90.

5.2.2.8 subroutine 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 aggf.f90.

5.2.2.9 subroutine 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 291 of file aggf.f90.

5.2.2.10 subroutine 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*, 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 aggf.f90.

5.2.2.11 subroutine 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

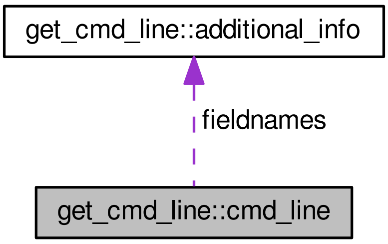
Definition at line 359 of file aggf.f90.

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

- /home/mrajner/src/grat/src/aggf.f90

5.3 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**

5.3.1 Detailed Description

Definition at line 60 of file get_cmd_line.f90.

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

- /home/mrajner/src/grat/src/get_cmd_line.f90

5.4 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]

5.4.1 Detailed Description

Definition at line 5 of file constants.f90.

5.4.2 Member Function/Subroutine Documentation

5.4.2.1 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.

5.4.2.2 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.

5.4.2.3 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.

5.4.2.4 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:

- /home/mrajner/src/grat/src/constants.f90

5.5 get_cmd_line::dateandmjd Type Reference

Public Attributes

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

5.5.1 Detailed Description

Definition at line 45 of file get_cmd_line.f90.

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

- /home/mrajner/src/grat/src/get_cmd_line.f90

5.6 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

5.6.1 Detailed Description

Definition at line 91 of file get_cmd_line.f90.

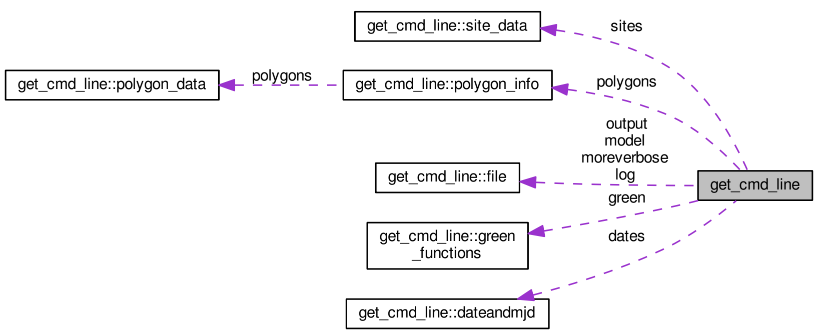
The documentation for this type was generated from the following file:

- /home/mrajner/src/grat/src/get_cmd_line.f90

5.7 get_cmd_line Module Reference

This module sets the initial values for parameters reads from command line and gives help.

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)

  Counts occurence of character (separator, default comma) in string.
- 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)
- 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**
- 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) **moreverbose**
- type(file), dimension(:),
  allocatable **model**
- character(len=40), dimension(5) **model_names** = ["pressure_surface"
- logical if_verbose = .false.

  whether print all information
- logical inverted_barometer = .true.

  whether print all information
- character(50), dimension(2) interpolation_names = ["nearest"

  Logical parameters for easy operation.
- 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)"

5.7.1 Detailed Description

This module sets the initial values for parameters reads from command line and gives help.

Definition at line 5 of file get_cmd_line.f90.

5.7.2 Member Function/Subroutine Documentation

5.7.2.1 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 170 of file get_cmd_line.f90.

5.7.2.2 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 779 of file get_cmd_line.f90.

5.7.2.3 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 703 of file get_cmd_line.f90.

5.7.2.4 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 617 of file get_cmd_line.f90.

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

- /home/mrajner/src/grat/src/get_cmd_line.f90

5.8 get_cmd_line::green_functions Type Reference

Public Attributes

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

5.8.1 Detailed Description

Definition at line 17 of file get_cmd_line.f90.

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

- /home/mrajner/src/grat/src/get_cmd_line.f90

5.9 mapcon_util Module Reference

Public Member Functions

- subroutine **mapaascii2mapablv**

5.9.1 Detailed Description

Definition at line 1 of file mapcon_util.f90.

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

- /home/mrajner/src/grat/src/mapcon_util.f90

5.10 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, method)

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

5.10.1 Detailed Description

This modele gives routines to read, and write data.

The netCDF format is widely used in geosciences. 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.

5.10.2 Member Function/Subroutine Documentation

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

Check the return code from netCDF manipulation.

from net

Definition at line 214 of file mod_data.f90.

5.10.2.2 subroutine mod_data::get_value (type(file), intent(in) *model,* real(sp), intent(in) *lat,* real(sp), intent(in) *lon,* real(sp), intent(out) *val,* integer, intent(in), optional *method*)

Returns the value from model file.

if it is first call it loads the model into memory inspired by spotl Agnew [1997]

Definition at line 231 of file mod_data.f90.

5.10.2.3 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 27 of file mod_data.f90.

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

Unpack variable.

from net

Definition at line 196 of file mod_data.f90.

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

- /home/mrajner/src/grat/src/mod_data.f90

5.11 mod_green Module Reference

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, denserdist, denseraz)
- subroutine **convolve_moreverbose** (site, azimuth, distance)

5.11.1 Detailed Description

Definition at line 1 of file mod_green.f90.

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

- /home/mrajner/src/grat/src/mod_green.f90

5.12 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)

5.12.1 Detailed Description

Definition at line 1 of file mod_polygon.f90.

5.12.2 Member Function/Subroutine Documentation

5.12.2.1 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.

5.12.2.2 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.

5.12.2.3 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:

- /home/mrajner/src/grat/src/mod_polygon.f90

5.13 get_cmd_line::polygon_data Type Reference

Public Attributes

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

5.13.1 Detailed Description

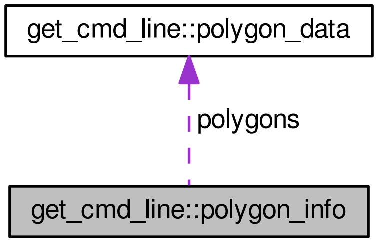
Definition at line 28 of file get_cmd_line.f90.

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

- /home/mrajner/src/grat/src/get_cmd_line.f90

5.14 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**

5.14.1 Detailed Description

Definition at line 33 of file get_cmd_line.f90.

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

- /home/mrajner/src/grat/src/get_cmd_line.f90

5.15 get_cmd_line::site_data Type Reference

Public Attributes

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

5.15.1 Detailed Description

Definition at line 70 of file get_cmd_line.f90.

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

- /home/mrajner/src/grat/src/get_cmd_line.f90

Chapter 6

File Documentation

6.1 /home/mrajner/src/grat/polygon/polygon_map.sh File Reference

Make map of polygon(s)

Functions/Subroutines

- then **shift** ((OPTIND-1)) OTHERARGS

Variables

- __pad0__

    If there are no command line argument then stop with error.
- echo **$FILE**
- d set **x**
- **DEBUG** = true
- v **VERBOSE** = true
- h **usage**
- **exit**
- R **R** = "-R$OPTARG"
- o **output** = "$OPTARG"
- p **POINTSFILE** = "$OPTARG"
- esac done **if** [-z $FILE]
- then echo Not enough cmd line **parameters**
- then echo you set the **verbose**
- then echo you set the $last $i p **last**
- then get_R fi **A** = "+"]
- then **color** = "#" | sed -n -e $(($last+1))

6.1.1 Detailed Description

Make map of polygon(s) This scrips

Author

Marcin Rajner

Date

03.11.2012 This scripts need GMT to be installed Wessel and Smith [1998] The .pdf suffix will be given for output file

Definition in file polygon_map.sh.

6.1.2 Variable Documentation

6.1.2.1 then echo you set the $last $i p last

Initial value:

```
$(($last+$i+2))
  done | minmax -C | awk '{print $1-1, $2+1, $3-1,$4+1}' | sed 's/\s/\
  R=-R$R
}


if [ -z $R ]
```

Definition at line 108 of file polygon_map.sh.

6.2 polygon_map.sh

```
#!/bin/bash -
## \file
## \brief Make map of polygon(s)
##
## This scrips
## \author Marcin Rajner
## \date 03.11.2012
## This scripts need GMT to be installed \cite Wessel98
## The \c .pdf suffix will be given for output file
# ================================================================================


## If there are no command line argument then stop with error
: ${1?"Try: $0 -h"}

## This function read in polygon file and return informations for plot
get_information(){

  ## Get the number of polygons
  number_of_polygons=$(cat $FILE | grep -v "#" | awk 'NR==1{ print $_ + 0 }'  )
  last=2

  ## initialize counter
  count=0
  ## loop over all polygons
  while [ $count -lt $number_of_polygons ]
  do

    ## Get the number of polygon points and the polygon action (incl/excl)
    ## and save in the array
    number_of_points=(${number_of_points[*]} $(cat $FILE | grep -v "#" | awk "NR==${last} { print \$_ + 0 }"))
    if_include=(${if_include[*]} $(cat $FILE | grep -v "#" | awk "NR==$(($last+1)){ print \$_}"))
    let last=$last+${number_of_points[$count]}+2
    let count++
  done
}

usage()
{
DESCRIPTION="This program generate the map of polygon. It requires Generic Mapping Tools command available"
echo "
usage: $0 options

$DESCRIPTION

OPTIONS:
```

```
  -h      Show this message
  -v      Verbose (default no verbose)
  -d      Debug (set debugging mode)
  -f      file [required]
  -R      GMT specific range (e.g. -R10/30/30/50)
  -o      output file
"
}

VERBOSE=
DEBUG=
FILE=
OTHERARGS=

while getopts "vhdR:f:o:p:" flag
do
  case "$flag" in
    f) FILE="$OPTARG";echo $FILE ;;
    d) set -x ; DEBUG=true;;
    v) VERBOSE=true ;;
    h) usage ; exit ;;
    R) R="-R$OPTARG" ;;
    o) output="$OPTARG" ;;
    p) POINTSFILE="$OPTARG" ;;
  esac
#  echo "$flag" $OPTIND $OPTARG
done

if [ -z $FILE ]; then
  shift $((OPTIND-1))
  OTHERARGS="$@"
fi

echo $FILE

# todo ! from command line
if [ -z $FILE ] || [ -z $output ] ; then
  echo "Not enough cmd line parameters... , try $0 -h"
  exit
fi



if [ -n "$VERBOSE" ] ; then
  echo "you set the verbose: $VERBOSE"
  echo "not recognized parameters args $OTHERARGS"
fi
echo "creating map for: $FILE ..."

get_information $FILE
echo "Number of polygons:    " $number_of_polygons
echo "Number of points:      " ${number_of_points[*]}
echo "Include[+]/exclude[-]: " ${if_include[*]}

#cat $FILE |grep -v "#" |nl

function get_R(){
  last=3
  R=$(for i in ${number_of_points[*]}
  do
    cat $FILE | grep -v "#" | sed -n -e $(($last+1)),$(($last+$i))p
    last=$(($last+$i+2))
  done | minmax -C | awk '{print $1-1, $2+1, $3-1,$4+1}' | sed 's/\s/\//g')
  R=-R$R
}


if [ -z $R ]; then
  get_R
fi

A="-A999"

  gmtset FRAME_WIDTH=0.01c
#  psbasemap $R -K -JM20+ -X0 -Y0 -B100 > $output.ps
pscoast $R -Slightblue -Glightgray  -K -Di $A -J  > $output.ps
  last=3
  for i in $(seq 0 $((${#number_of_points[*]}-1)))
  do
    if [ ${if_include[$(($i))]} = "+" ]; then
      color=lightgreen
    else
      color=lightred
    fi
    cat $FILE | grep -v "#" | sed -n -e $(($last+1)),$(($last+${
number_of_points[$i]}))p \
      | psxy  -R -J -K -O -A -W2p -L -G$color >> $output.ps
```

```
    last=$(($last+${number_of_points[$i]}+2))
  done

  if [ -z $POINTSFILE ] ; then
    echo "no points file given"
  else
    makecpt -Cjet -T0.1/0.9/0.2 |sed 's/^B.*/B 200 0 0/' |sed 's/^F.*/F 0 180
 0/' > points.cpt
    cat $POINTSFILE | awk "{print \$1 , \$2 ,\$(3)}" | psxy $R -J -Sc5p -
Cpoints.cpt -Gred -W0.41p/gray -O -K -V  >> $output.ps
  fi


  pscoast $R -O -Di $A -J -W -N1thin >> $output.ps

ps2raster $output.ps -Tf -P -A
#evince $output.ps







exit 0
```

6.3 /home/mrajner/src/grat/src/aggf.f90 File Reference

This module contains utitlities for computing Atmospheric Gravity Green Functions.

Data Types

- module aggf

6.3.1 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 aggf.f90.

6.4 aggf.f90

```
!
 ============================================================================
!
 ============================================================================
module aggf

  use constants
  implicit none

contains

!
 ============================================================================
!
 ============================================================================
subroutine compute_aggfdt ( psi , aggfdt , delta_ , aggf )
  implicit none
  real(dp) , intent (in) :: psi
  real(dp) , intent (in) , optional :: delta_
  logical , intent (in) , optional :: aggf
  real(dp) , intent (out) :: aggfdt
  real(dp) :: deltat , aux , h_

  deltat = 10.
  if (present( delta_) )  deltat = delta_
  if (present( aggf ) .and. aggf ) then
    h_ = 0.001 ! default if we compute dggfdh using this routine
      if (present( delta_) )  h_ = deltat
    call compute_aggf( psi , aux , h = + h_ )
```

```
    aggfdt = aux
    call compute_aggf( psi , aux , h= -h_ )
    aggfdt = aggfdt - aux
    aggfdt = aggfdt / ( 2. * h_ )
  else
    call compute_aggf( psi , aux , t_zero = t0 + deltat )
    aggfdt = aux
    call compute_aggf( psi , aux , t_zero = t0 - deltat )
    aggfdt = aggfdt - aux
    aggfdt = aggfdt / ( 2. * deltat)
  endif



end subroutine

! ==============================================================================
! ==============================================================================
subroutine read_tabulated_green ( table , author )
  real(dp), intent (inout),dimension(:,:), allocatable :: table
  character ( len = * ) , intent (in) , optional       :: author
  integer                                              :: i , j
  integer                                              :: rows , columns , file_unit
  character (len=255)                                  :: file_name

  rows      = 85
  columns   = 6
  file_name = '../dat/merriam_green.dat'

  if ( present(author) ) then
    if ( author .eq. "huang" ) then
      rows    = 80
      columns = 5
      file_name = '../dat/huang_green.dat'
    elseif( author .eq. "rajner" ) then
      rows    = 85
      columns = 5
      file_name = '../dat/rajner_green.dat'
    elseif( author .eq. "merriam" ) then
    else
      write ( * , * ) 'cannot find specified tables, using merriam instead'
    endif
  endif

  if (allocated (table) ) deallocate (table)
  allocate ( table( rows , columns ) )

  open (newunit = file_unit , file = file_name , action='read', status='old')

  call skip_header(file_unit)

  do i = 1 , rows
    read (file_unit,*) ( table( i , j ), j = 1 , columns )
  enddo
  close(file_unit)
end subroutine


! ==============================================================================
! ==============================================================================
subroutine compute_aggf (psi , aggf_val , hmin , hmax , dh , if_normalization, &
                    t_zero , h ,  first_derivative_h , first_derivative_z , fels_type )
  implicit none
  real(dp), intent(in)          :: psi
  real(dp), intent(in),optional :: 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)
                                   h
  logical, intent(in), optional :: if_normalization , first_derivative_h , first_derivative_z
  character (len=*) , intent(in), optional  :: fels_type
  real(dp), intent(out)          :: aggf_val
  real(dp)                       :: r , z , psir , da , dz , rho , h_min , h_max , h_station , j_aux

```

```
  h_min = 0.
  h_max = 60.
  dz    = 0.0001 !mrajner 2012-11-08 13:49
  h_station = 0.

  if ( present(hmin) ) h_min    = hmin
  if ( present(hmax) ) h_max    = hmax
  if ( present(  dh) )    dz    = dh
  if ( present(   h) ) h_station = h


  psir = psi * pi / 180.

  da = 2 * pi * r0**2 * ( 1 - cos(1. *pi/180.) )


  aggf_val=0.
  do z = h_min , h_max , dz

    r = ( ( r0 + z )**2 + (r0 + h_station)**2 &
      - 2.*(r0 + h_station ) *(r0+z)*cos(psir) )**(0.5)
    call standard_density( z , rho , t_zero = t_zero , fels_type = fels_type )

    if ( present( first_derivative_h) .and. first_derivative_h ) then

      !! see equation 22, 23 in \cite Huang05
      !J_aux =  (( r0 + z )**2)*(1.-3.*((cos(psir))**2)) -2.*(r0 + h_station )**2  &
      !  + 4.*(r0+h_station)*(r0+z)*cos(psir)
      ! aggf_val =  aggf_val -  rho * (  J_aux  /  r**5  ) * dz

      j_aux = (2.* (r0 ) - 2 * (r0 +z )*cos(psir)) / (2. * r)
      j_aux =  -r - 3 * j_aux * ((r0+z)*cos(psir) - r0)
      aggf_val =  aggf_val +  rho * (  j_aux  /  r**4  ) * dz
    else
      if ( present( first_derivative_z) .and. first_derivative_z ) then
        if (z.eq.h_min) then
            aggf_val = aggf_val  &
              + rho*( ((r0 + z)*cos(psir) - ( r0 + h_station ) ) / ( r**3 ) )
        endif
      else
        aggf_val = aggf_val  &
         + rho * ( ( (r0 + z ) * cos( psir ) - ( r0 + h_station ) ) / ( r**3 ) ) * dz
      endif
    endif
  enddo

  aggf_val = -g * da * aggf_val * 1e8 * 1000

  if ( (.not.present(if_normalization)) .or. (if_normalization)) then
    aggf_val= psir * aggf_val * 1e5  / p0
  endif

end subroutine

! ==============================================================================
! ==============================================================================
subroutine standard_density ( height , rho , t_zero ,fels_type )

  implicit none
  real(dp) , intent(in)  ::  height
  real(dp) , intent(in), optional  :: t_zero
  character(len = 22) , optional :: fels_type
  !! surface temperature is set to this value,
  !! otherwise the T0 for standard atmosphere is used
  real(dp) , intent(out) :: rho
  real(dp)  :: p ,t

  call standard_pressure(height , p , t_zero = t_zero, fels_type=fels_type)
  call standard_temperature(height , t , t_zero = t_zero, fels_type=fels_type)

  ! pressure in hPa --> Pa
  rho= 100 * p / ( r_air * t )
end subroutine

! ==============================================================================
! ==============================================================================
subroutine standard_pressure (height, pressure , &
          p_zero , t_zero ,  if_simplificated ,fels_type , inverted)
  implicit none
```

```
  real(dp) , intent(in)             :: height
  real(dp) , intent(in) , optional :: t_zero , p_zero
  character(len = 22) , optional :: fels_type
  logical          , intent(in) , optional :: if_simplificated
  logical          , intent(in) , optional :: inverted
  real(dp), intent(out) :: pressure
  real(dp) ::  lambda , temperature , g , alpha , sfc_pressure

  sfc_pressure = p0
  if (present(p_zero)) sfc_pressure = p_zero

  call standard_temperature( height,temperature,t_zero=t_zero, &
                              fels_type =fels_type)
  call standard_gravity( height , g )

  lambda = r_air * temperature / g

  if (present(if_simplificated) .and. if_simplificated ) then
    ! use simplified formulae
    alpha = -6.5
    pressure = sfc_pressure * ( 1 + alpha / t0 * height ) ** ( -g0 / (r_air * alpha / 1000 ) )
  else
    ! use precise formulae
    pressure = sfc_pressure  * exp( -1000. * height / lambda )
  endif
  if (present(inverted).and.inverted) then
    pressure = sfc_pressure  / ( exp( -1000. * height / lambda ) )
  endif
end subroutine

! ==============================================================================
! > This will transfer pressure beetween different height using barometric
! formulae
! ==============================================================================
subroutine transfer_pressure (height1 , height2 , pressure1 , pressure2 , &
  temperature , polish_meteo )
  real (dp) , intent (in) :: height1 , height2 , pressure1
  real (dp) , intent (in), optional :: temperature
  real (dp) :: sfc_temp , sfc_pres
  logical , intent (in), optional :: polish_meteo
  real(dp) , intent(out) :: pressure2

  sfc_temp = t0


  ! formulae used to reduce press to sfc in polish meteo service
  if (present(polish_meteo) .and. polish_meteo) then
    sfc_pres = exp(log(pressure1) + 2.30259 * height1*1000. &
      /(18400.*(1+0.00366*((temperature-273.15) + 0.0025*height1*1000.)))  )
  else
  ! different approach
    if(present(temperature) ) then
      call surface_temperature( height1 , temperature , sfc_temp )
    endif
    call standard_pressure(height1 , sfc_pres , t_zero=sfc_temp , &
      inverted=.true. , p_zero = pressure1 )
  endif

  ! move from sfc to height2
  call standard_pressure(height2 , pressure2 , t_zero=sfc_temp , &
    p_zero = sfc_pres )
end subroutine

! ==============================================================================
! ==============================================================================
subroutine standard_gravity ( height , g )
  implicit none
  real(dp), intent(in)  :: height
  real(dp), intent(out) :: g

  g= g0 * ( r0 / ( r0 + height ) )**2
end subroutine


! ==============================================================================
! ==============================================================================
real(sp) function geometric_height (geopotential_height)
  real (sp) :: geopotential_height

  geometric_height = geopotential_height * (r0 / ( r0 + geopotential_height ) )
end function
```

```


! ============================================================================
! ============================================================================
subroutine surface_temperature (height , temperature1 , &
  temperature2, fels_type , tolerance)
  real(dp) , intent(in)  :: height , temperature1
  real(dp) , intent(out) :: temperature2
  real(dp) :: temp(3) , temp_ (3) , tolerance_ = 0.1
  character (len=*) , intent(in), optional  :: fels_type
  real(sp) , intent(in), optional  :: tolerance
  integer :: i

  if (present(tolerance)) tolerance_ = tolerance

  ! searching limits
  temp(1)=t0-150
  temp(3)=t0+ 50

  do
    temp(2)= ( temp(1) + temp(3) ) /2.

    do i = 1,3
      call standard_temperature(height , temp_(i) , t_zero=temp(i) , fels_type = fels_type )
    enddo

    if (abs(temperature1 - temp_(2) ) .lt. tolerance_ ) then
      temperature2 = temp(2)
      return
    endif

    if ( (temperature1 - temp_(1) ) * (temperature1 - temp_(2) ) .lt.0 ) then
      temp(3) = temp(2)
    elseif( (temperature1 - temp_(3) ) * (temperature1 - temp_(2) ) .lt.0 ) then
      temp(1) = temp(2)
    else
      stop "surface_temp"
    endif
  enddo
end subroutine
! ============================================================================
! ============================================================================
subroutine standard_temperature ( height , temperature , t_zero , fels_type )
  real(dp) , intent(in)  :: height
  real(dp) , intent(out) :: temperature
  real(dp) , intent(in), optional  :: t_zero
  character (len=*) , intent(in), optional  :: fels_type

  real(dp) :: aux , cn , t
  integer :: i,indeks
  real , dimension (10) :: z,c,d


  z = (/11.0 , 20.1 , 32.1 , 47.4 , 51.4 , 71.7 , 85.7, 100.0, 200.0, 300.0/)
  c = (/-6.5,   0.0,   1.0,   2.75,  0.0,  -2.75, -1.97,  0.0,   0.0,   0.0/)
  d = (/ 0.3,   1.0,   1.0,   1.0,   1.0,   1.0,   1.0,   1.0,   1.0,   1.0/)
  t = t0

  if ( present(fels_type)) then
    if (fels_type .eq. "US1976" ) then
    elseif(fels_type .eq. "tropical" ) then
      z=(/ 2.0  , 3.0, 16.5 , 21.5 , 45.0 , 51.0, 70.0 , 100.0 , 200.0 , 300.0 /)
      c=(/-6.0 , -4.0, -6.7  , 4.0  , 2.2  , 1.0, -2.8  , -0.27  , 0.0   , 0.0 /)
      d=(/ 0.5  , 0.5 , 0.3  , 0.5  , 1.0  , 1.0 , 1.0   , 1.0   , 1.0   , 1.0 /)
      t=300.0
    elseif(fels_type .eq. "subtropical_summer" ) then
      z = (/ 1.5  , 6.5  , 13.0 , 18.0 , 26.0 , 36.0  , 48.0 ,  50.0 ,  70.0 , 100.0  /)
      c = (/-4.0 , -6.0  , -6.5  , 0.0  , 1.2  , 2.2  ,  2.5  ,  0.0   ,-3.0 ,-0.025/)
      d = (/ 0.5  , 1.0   , 0.5  , 0.5  , 1.0  , 1.0   , 2.5   , 0.5   , 1.0 , 1.0  /)
      t = 294.0
    elseif(fels_type .eq. "subtropical_winter" ) then
      z = (/ 3.0  ,10.0  , 19.0 , 25.0 , 32.0 , 44.5  , 50.0 ,  71.0 ,  98.0 , 200.0 /)
      c = (/-3.5 , -6.0  , -0.5  , 0.0  , 0.4  , 3.2  ,  1.6  , -1.8   , 0.7 , 0.0 /)
      d = (/ 0.5  , 0.5   , 1.0  , 1.0  , 1.0  , 1.0   , 1.0   , 1.0   , 1.0 , 1.0 /)
```

```
      t = 272.2
    elseif(fels_type .eq. "subarctic_summer" ) then
      z = (/  4.7 , 10.0 , 23.0 , 31.8 , 44.0 , 50.2  , 69.2 , 100.0 , 200.0 ,
 300.0 /)
      c = (/ -5.3 , -7.0 ,  0.0 ,  1.4 ,  3.0 , 0.7  , -3.3  , -0.2   , 0.0 ,
 0.0 /)
      d = (/  0.5 ,  0.3 ,  1.0 ,  1.0 ,  2.0 , 1.0   , 1.5   , 1.0   , 1.0 ,
 1.0 /)
      t = 287.0
    elseif(fels_type .eq. "subarctic_winter" ) then
      z = (/  1.0 ,  3.2 ,  8.5 , 15.5 , 25.0 , 30.0 , 35.0 , 50.0 , 70.0 , 100
.0 /)
      c = (/  3.0 , -3.2 , -6.8 ,  0.0 , -0.6 ,  1.0 ,  1.2 ,  2.5 , -0.7 ,  -1
.2 /)
      d = (/  0.4 ,  1.5 ,  0.3 ,  0.5 ,  1.0 ,  1.0 ,  1.0 ,  1.0 ,  1.0 ,   1
.0 /)
      t = 257.1
    else
      print * ,
"unknown fels_type argument: &        using US standard atmosphere 1976
 instead"
    endif
  endif

  if (present(t_zero) ) then
    t=t_zero
  endif

  do i=1,10
    if (height.le.z(i)) then
      indeks=i
      exit
    endif
  enddo

  aux = 0.
  do i = 1 , indeks
    if (i.eq.indeks) then
      cn = 0.
    else
      cn = c(i+1)
    endif
      aux = aux + d(i) * ( cn - c(i) )  * log( cosh( (height - z(i)) / d(i) ) /
 cosh(z(i)/d(i)) )
  enddo
  temperature = t + c(1) * height/2. + aux/2.
end subroutine

!
 ============================================================================
!
 ============================================================================
real function gn_thin_layer (psi)
  implicit none
  real(dp) , intent(in) :: psi
  real(dp) :: psir

  psir = psi * pi / 180.
  gn_thin_layer = 1.627 * psir / sin( psir / 2. )
end function


!
 ============================================================================
!
 ============================================================================
integer function size_ntimes_denser (size_original, ndenser)
  integer, intent(in) :: size_original , ndenser
  size_ntimes_denser= (size_original - 1 ) * (ndenser +1 ) +
1
end function

!
 ============================================================================
!
 ============================================================================
real(dp) function bouger ( R_opt )
  real(dp), optional :: r_opt
  real(dp) :: aux
  real(dp) :: r
  real(dp) ::  h = 8.84 ! scale height of standard atmosphere

  aux = 1

  if (present( r_opt ) ) then
    r = r_opt
    aux = h  + r  - sqrt(  r**2  + (h/2. ) ** 2 )
```

```
    bouger = 2 * pi * g  * aux
  else
    aux = h
    bouger = 2 * pi * g * aux
    return
  endif
end function
!
 ===============================================================================
!
 ===============================================================================
real(dp) function simple_def (R)
  real(dp) :: r ,delta

  delta = 0.22e-11 * r

  simple_def = g0 / r0 * delta * ( 2. - 3./2. * rho_crust / rho_earth
 &
    -3./4. * rho_crust / rho_earth * sqrt(2* (1. )) ) * 1000
end function

!polish_meteo

end module
```

6.5 /home/mrajner/src/grat/src/constants.f90 File Reference

This module define some constant values used.

Data Types

- module constants

6.5.1 Detailed Description

This module define some constant values used.

Definition in file constants.f90.

6.6 constants.f90

```
!
 ===============================================================================
!
 ===============================================================================
module constants

  implicit none
  integer , parameter :: dp = 8
  integer , parameter :: sp = 4
  real(dp) , parameter :: &
    T0          = 288.15,     &  !< surface temperature for standard atmosphere
 [K] (15 degC)
    g0          = 9.80665,    &  !< mean gravity on the Earth [m/s2]
    r0          = 6356.766,   &  !< Earth radius (US Std. atm. 1976)  [km]
    p0          = 1013.25,    &  !< surface pressure for standard Earth [hPa]
    G           = 6.672e-11,  &  !< Cavendish constant \f$[m^3/kg/s^2]\f$
    R_air       = 287.05,     &  !< dry air constant  [J/kg/K]
    pi          = 4*atan(1.), &  !< pi = 3.141592... [ ]
    rho_crust   = 2670  , &     !< mean density of crust [kg/m3]
    rho_earth   = 5500

contains

!
 ===============================================================================
!
 ===============================================================================
subroutine spline_interpolation(x,y, x_interpolated,
 y_interpolated)
  implicit none
  real(dp) , allocatable , dimension (:) ,intent(in) :: x, y, x_interpolated
```

```
  real(dp) , allocatable , dimension (:) , intent(out) :: y_interpolated
  real(dp) , dimension (:) , allocatable :: b, c, d
  integer :: i

  allocate (b(size(x)))
  allocate (c(size(x)))
  allocate (d(size(x)))
  allocate (y_interpolated(size(x_interpolated)))

  call spline( x , y, b , c, d, size(x))

  do i=1, size(x_interpolated)
     y_interpolated(i) = ispline(x_interpolated(i) , x , y , b , c , d ,
 size (x) )
  enddo

end subroutine

!
 ==============================================================================
!
 ==============================================================================
!  Calculate the coefficients b(i), c(i), and d(i), i=1,2,...,n
!  for cubic spline interpolation
!  s(x) = y(i) + b(i)*(x-x(i)) + c(i)*(x-x(i))**2 + d(i)*(x-x(i))**3
!  for  x(i) <= x <= x(i+1)
!  Alex G: January 2010
!----------------------------------------------------------------------
!  input..
!  x = the arrays of data abscissas (in strictly increasing order)
!  y = the arrays of data ordinates
!  n = size of the arrays xi() and yi() (n>=2)
!  output..
!  b, c, d  = arrays of spline coefficients
!  comments ...
!  spline.f90 program is based on fortran version of program spline.f
!  the accompanying function fspline can be used for interpolation
!
 ==============================================================================
subroutine spline (x, y, b, c, d, n)
  implicit none
  integer n
  real(dp) :: x(n), y(n), b(n), c(n), d(n)
  integer i, j, gap
  real ::  h

  gap = n-1
  ! check input
  if ( n < 2 ) return
  if ( n < 3 ) then
    b(1) = (y(2)-y(1))/(x(2)-x(1))    ! linear interpolation
    c(1) = 0.
    d(1) = 0.
    b(2) = b(1)
    c(2) = 0.
    d(2) = 0.
    return
  end if
  !
  ! step 1: preparation
  !
  d(1) = x(2) - x(1)
  c(2) = (y(2) - y(1))/d(1)
  do i = 2, gap
    d(i) = x(i+1) - x(i)
    b(i) = 2.0*(d(i-1) + d(i))
    c(i+1) = (y(i+1) - y(i))/d(i)
    c(i) = c(i+1) - c(i)
  end do
  !
  ! step 2: end conditions
  !
  b(1) = -d(1)
  b(n) = -d(n-1)
  c(1) = 0.0
  c(n) = 0.0
  if(n /= 3) then
    c(1) = c(3)/(x(4)-x(2)) - c(2)/(x(3)-x(1))
    c(n) = c(n-1)/(x(n)-x(n-2)) - c(n-2)/(x(n-1)-x(n-3))
    c(1) = c(1)*d(1)**2/(x(4)-x(1))
    c(n) = -c(n)*d(n-1)**2/(x(n)-x(n-3))
  end if
  !
  ! step 3: forward elimination
  !
  do i = 2, n
    h = d(i-1)/b(i-1)
```

```
    b(i) = b(i) - h*d(i-1)
    c(i) = c(i) - h*c(i-1)
  end do
  !
  ! step 4: back substitution
  !
  c(n) = c(n)/b(n)
  do j = 1, gap
    i = n-j
    c(i) = (c(i) - d(i)*c(i+1))/b(i)
  end do
  !
  ! step 5: compute spline coefficients
  !
  b(n) = (y(n) - y(gap))/d(gap) + d(gap)*(c(gap) + 2.0*c(n))
  do i = 1, gap
    b(i) = (y(i+1) - y(i))/d(i) - d(i)*(c(i+1) + 2.0*c(i))
    d(i) = (c(i+1) - c(i))/d(i)
    c(i) = 3.*c(i)
  end do
  c(n) = 3.0*c(n)
  d(n) = d(n-1)
end subroutine spline


!
==============================================================================
!
==============================================================================
!======================================================================
! function ispline evaluates the cubic spline interpolation at point z
! ispline = y(i)+b(i)*(u-x(i))+c(i)*(u-x(i))**2+d(i)*(u-x(i))**3
! where  x(i) <= u <= x(i+1)
!----------------------------------------------------------------------
! input..
! u       = the abscissa at which the spline is to be evaluated
! x, y    = the arrays of given data points
! b, c, d = arrays of spline coefficients computed by spline
! n       = the number of data points
! output:
! ispline = interpolated value at point u
!=======================================================================
function ispline(u, x, y, b, c, d, n)
implicit none
real ispline
integer n
real(dp)::  u, x(n), y(n), b(n), c(n), d(n)
integer ::  i, j, k
real :: dx

! if u is ouside the x() interval take a boundary value (left or right)
if(u <= x(1)) then
  ispline = y(1)
  return
end if
if(u >= x(n)) then
  ispline = y(n)
  return
end if

!*
!  binary search for for i, such that x(i) <= u <= x(i+1)
!*
i = 1
j = n+1
do while (j > i+1)
  k = (i+j)/2
  if(u < x(k)) then
    j=k
    else
    i=k
   end if
end do
!*
!  evaluate spline interpolation
!*
dx = u - x(i)
ispline = y(i) + dx*(b(i) + dx*(c(i) + dx*d(i)))
end function ispline

!
==============================================================================
!
==============================================================================
integer function ntokens(line)
character,intent(in):: line*(*)
integer i, n, toks
```

```

i = 1;
n = len_trim(line)
toks = 0
ntokens = 0
do while(i <= n)
   do while(line(i:i) == ' ')
     i = i + 1
     if (n < i) return
   enddo
   toks = toks + 1
   ntokens = toks
   do
     i = i + 1
     if (n < i) return
     if (line(i:i) == ' ') exit
   enddo
enddo
end function ntokens

! ==============================================================================
! ==============================================================================
subroutine skip_header ( unit , comment_char_optional )
  use iso_fortran_env
  implicit none
  integer , intent (in) :: unit
  character (len = 1) , optional :: comment_char_optional
  character (len = 60 ) :: dummy
  character (len = 1)  :: comment_char
  integer :: io_stat

  if (present( comment_char_optional ) ) then
    comment_char = comment_char_optional
  else
    comment_char = '#'
  endif

  read ( unit, * , iostat = io_stat) dummy
  if(io_stat == iostat_end) return

  do while ( dummy(1:1) .eq. comment_char )
    read ( unit, * , iostat = io_stat ) dummy
    if(io_stat == iostat_end) return
  enddo
  backspace(unit)
end subroutine

real function jd (year,month,day, hh,mm,ss)
  implicit none
  integer, intent(in) ::  year,month,day
  integer, intent(in) :: hh,mm, ss
  integer :: i , j , k
  i= year
  j= month
  k= day
  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.) &
  + mm/(24.*60.) +ss/(24.*60.*60.) ! - 2400000.5
  return
end function

!subroutine gdate (jd, year,month,day,hh,mm,ss)
!  !! modyfikacja mrajner 20120922
!  !! pobrane  http://aa.usno.navy.mil/faq/docs/jd_formula.php
!  implicit none
!  real, intent(in):: jd
!  real :: aux
!  integer,intent(out) :: year,month,day,hh,mm,ss
!  integer :: i,j,k,l,n

!  l= int((jd+68569))
!  n= 4*l/146097
!  l= l-(146097*n+3)/4
!  i= 4000*(l+1)/1461001
!  l= l-1461*i/4+31
!  j= 80*l/2447
!  k= l-2447*j/80
!  l= j/11
!  j= j+2-12*l
!  i= 100*(n-49)+i+l

!  year= i
!  month= j
!  day= k

```

```
!  aux= jd - int(jd) + 0.0001/86400 ! ostatni argument zapewnia poprawe
!                                    ! jeżeli ss jest integer
!  hh= aux*24
!  mm= aux*24*60    - hh*60
!  ss= aux*24*60*60 - hh*60*60 - mm*60
!end subroutine
real(dp) function mjd  (date)
  implicit none
  integer ,intent(in) :: date (6)
  integer :: aux (6)
  integer :: i , k
  real(dp) :: dayfrac

  aux=date
  if ( aux(2) .le.  2) then
      aux(1)  = date(1) -  1
      aux(2) = date(2) +  12
  endif
  i = aux(1)/100
  k = 2 - i + int(i/4);
  mjd = int(365.25 * aux(1) ) - 679006
  dayfrac =  aux(4) / 24. + date(5)/(24. * 60. ) + date(6)/(24. * 3600. )
  mjd = mjd + int(30.6001*( aux(2) + 1)) + date(3) + k + dayfrac
end function

subroutine invmjd (mjd , date)
  implicit none
  real(dp), intent (in) :: mjd
  integer , intent (out):: date (6)
  integer :: t1 ,t4 , h , t2 , t3 , ih1 , ih2
  real(dp) :: dayfrac

  date =0

  t1 = 1+ int(mjd) + 2400000
  t4 = mjd - int(mjd);
   h = int((t1 - 1867216.25)/36524.25);
  t2 = t1 + 1 + h - int(h/4)
  t3 = t2 - 1720995
  ih1 = int((t3 -122.1)/365.25)
  t1 = int(365.25 * ih1)
  ih2 = int((t3 - t1)/30.6001);
  date(3) = (t3 - t1 - int(30.6001 * ih2)) + t4;
  date(2) = ih2 - 1;
  if (ih2 .gt. 13) date(2) = ih2 - 13
   date(1) = ih1
  if (date(2).le. 2) date(1) = date(1) + 1

  dayfrac = mjd - int(mjd) + 1./ (60*60*1000)
  date(4) = int(dayfrac * 24. )
  date(5) = ( dayfrac - date(4) / 24. ) * 60 * 24
  date(6) = ( dayfrac - date(4) / 24. - date(5)/(24.*60.)   ) * 60 * 24 *60
  if (date(6) .eq. 60 ) then
    date(6)=0
    date(5)=date(5) + 1
  endif
end subroutine

end module constants
```

6.7 /home/mrajner/src/grat/src/example_aggf.f90 File Reference

This program shows some example of using AGGF module.

Functions/Subroutines

- program **example_aggf**
- subroutine simple_atmospheric_model ()

    Reproduces data to Fig.~3 in.
- subroutine compare_tabulated_green_functions ()

    Compare tabulated green functions from different authors.
- subroutine compute_tabulated_green_functions ()

    Compute AGGF and derivatives.
- subroutine aggf_resp_fels_profiles ()

Compare different vertical temperature profiles impact on AGGF.

- subroutine compare_fels_profiles ()

  Compare different vertical temperature profiles.
- subroutine aggf_resp_h ()

  Computes AGGF for different site height (h)
- subroutine aggf_resp_t ()

  This computes AGGF for different surface temperature.
- subroutine aggfdt_resp_dt ()

  This computes AGGFDT for different dT.
- subroutine aggf_resp_dz ()

  This computes AGGF for different height integration step.
- subroutine standard1976

  This computes standard atmosphere parameters.
- subroutine aggf_resp_hmax ()

  This computes relative values of AGGF for different atmosphere height integration.
- subroutine aux_heights (table)

  Relative value of aggf depending on integration height.
- subroutine **aggf_thin_layer** ()

6.7.1 Detailed Description

This program shows some example of using AGGF module.

Author

Marcin Rajner

Date

20121108

The examples are in contained subroutines

Definition in file example_aggf.f90.

6.7.2 Function/Subroutine Documentation

6.7.2.1 subroutine example_aggf::aux_heights (real(dp), dimension (:), intent(inout), allocatable *table*)

Relative value of aggf depending on integration height.

Auxiliary subroutine – height sampling for semilog plot

Definition at line 459 of file example_aggf.f90.

6.7.2.2 subroutine example_aggf::compare_fels_profiles ( )

Compare different vertical temperature profiles.

Using tables and formula from Fels [1986]

Definition at line 192 of file example_aggf.f90.

6.7.2.3 subroutine example_aggf::simple_atmospheric_model ( )

Reproduces data to Fig.∼3 in.

Warburton and Goodkind [1977]

Definition at line 39 of file example_aggf.f90.

6.7.2.4 subroutine example_aggf::standard1976 ( )

This computes standard atmosphere parameters.

It computes temperature, gravity, pressure, pressure (simplified formula) density for given height

Definition at line 387 of file example_aggf.f90.

6.8 example_aggf.f90

```
00001 ! ============================================================================
00008 ! ============================================================================
00009 program example_aggf
00010
00012   use aggf
00013   use constants
00014   implicit none
00015
00016
00017
00018
00019 ! call standard1976 ()
00020 ! call aggf_resp_hmax ()
00021 ! call aggf_resp_dz ()
00022 ! call aggf_resp_t ()
00023 ! call aggf_resp_h ()
00024 ! call aggfdt_resp_dt ()
00025 ! call compare_fels_profiles ()
00026 ! call compute_tabulated_green_functions ()
00027 ! call aggf_thin_layer ()
00028 ! call aggf_resp_fels_profiles ()
00029 ! call compare_tabulated_green_functions ()
00030 ! call simple_atmospheric_model()
00031
00032
00033
00034 contains
00035
00036 ! =============================================================================
00038 ! =============================================================================
00039 subroutine simple_atmospheric_model ()
00040   real(dp) :: r ! - km
00041   integer :: iunit
00042
00043   open (newunit=iunit,file="/home/mrajner/dr/rysunki/simple_approach.dat" ,&
00044     action = "write")
00045     do r = 0. , 25*8
00046 !     iunit = 6
00047     write ( iunit ,  * ) , r , bouger( r_opt= r) * 1e8, & !conversion to
 microGal
00048       simple_def(r) * 1e8
00049   enddo
00050
00051 end subroutine
00052 ! =============================================================================
00054 ! =============================================================================
00055 subroutine compare_tabulated_green_functions
 ()
00056   integer :: i , j , file_unit , ii , iii
00057   real(dp), dimension(:,:), allocatable :: table , results
00058   real(dp), dimension(:,:), allocatable :: parameters
00059   real(dp), dimension(:), allocatable :: x1, y1 ,x2 , y2 , x, y ,
x_interpolated, y_interpolated
00060   integer :: how_many_denser
00061   character(len=255), dimension(3) :: authors
00062   integer , dimension(3) :: columns
00063
00064   authors=["rajner", "merriam" , "huang"]
00065   ! selected columns for comparison in appropriate tables
00066   columns=[2 , 2, 2]
```

```

  how_many_denser=0

  ! reference author
  call read_tabulated_green(table , author = authors(1) )
  allocate (results(size_ntimes_denser(size(table(:,1)), how_many_denser) , 0 : size(authors) ))

  ! fill abscissa in column 0
  ii = 1
  do i = 1 ,  size (table(:,1) ) - 1
    do j = 0 , how_many_denser
      results(ii,0) = table(i,1 ) + j * (table(i+1, 1) -table(i,1) ) / ( how_many_denser + 1 )
      ii=ii+1
    enddo
  enddo
  ! and the last element
  results( size (results(:,0) )  , 0) =  table( size(table(:,1)) ,1 )

  ! take it as main for all series
  allocate(x_interpolated( size ( results(:,0))))
  x_interpolated = results(:,0)

  open (newunit = file_unit , file = "../examples/compare_aggf.dat", action= "write")

  ! for every author
  do i= 1, size(authors)
    print * , trim( authors( i ) )
    call read_tabulated_green(table , author = authors(i) )
    allocate(x( size (table(:,1))))
    allocate(y( size (table(:,2))))
    x = table(:,1)
    y = table(:, columns(i))
    call spline_interpolation( x , y , x_interpolated, y_interpolated )
    if (i.gt.1) then
      y_interpolated = ( y_interpolated - results(:,1) ) / results(:,1)  * 100.
    endif

    results(:, i ) = y_interpolated
    deallocate(x,y)
  enddo

  write (file_unit , '(<size(results(1,:))>f20.5)' ) ( results(i , :) , i = 1 , size(results( :,1)) )
  close(file_unit)
end subroutine

! ============================================================================
! ============================================================================
subroutine compute_tabulated_green_functions ()
  integer :: i , file_unit
  real(dp) :: val_aggf , val_aggfdt ,val_aggfdh, val_aggfdz
  real(dp), dimension(:,:), allocatable :: table , results

  ! Get the spherical distances from Merriam92
  call read_tabulated_green( table , author = "merriam")

  open   ( newunit = file_unit, &
           file    = '../dat/rajner_green.dat', &
           action  = 'write' &
         )

  ! print header
  write ( file_unit,*) '# This is set of AGGF computed using module ', &
  'aggf from grat software'
  write ( file_unit,*) '# Normalization according to Merriam92'
  write ( file_unit,*) '# Marcin Rajner'
  write ( file_unit,*) '# For detail see www.geo.republika.pl'
  write ( file_unit,'(10(a23))')  '#psi[deg]', &
    'GN[microGal/hPa]'       , 'GN/dT[microGal/hPa/K]' , &
    'GN/dh[microGal/hPa/km]' , 'GN/dz[microGal/hPa/km]'

  do i= 1, size(table(:,1))
    call compute_aggf( table(i,1) , val_aggf   )
    call compute_aggfdt( table(i,1) , val_aggfdt )
    call compute_aggf( table(i,1) , val_aggfdh , first_derivative_h =.true. )
    call compute_aggf( table(i,1) , val_aggfdz , first_derivative_z =.true. )
    write ( file_unit, '(10(e23.5))' ) &
      table(i,1) , val_aggf , val_aggfdt , val_aggfdh, val_aggfdz
  enddo
  close(file_unit)
```

```
end subroutine

! ============================================================================
! ============================================================================
subroutine aggf_resp_fels_profiles ()
  character (len=255) ,dimension (6) :: fels_types
  real (dp) :: val_aggf
  integer :: i , j, file_unit
  real(dp), dimension(:,:), allocatable :: table

  ! All possible optional arguments for standard_temperature
  fels_types = (/ "US1976"              , "tropical",    &
                  "subtropical_summer" , "subtropical_winter" , &
                  "subarctic_summer"   , "subarctic_winter"    /)

  open  ( newunit = file_unit, &
          file    = '../examples/aggf_resp_fels_profiles.dat' , &
          action  = 'write' &
        )

  call read_tabulated_green(table)

  ! print header
  write ( file_unit , '(100(a20))' ) &
    'psi', ( trim( fels_types(i) ) , i = 1 , size (fels_types) )

  ! print results
  do i = 1 , size (table(:,1))
    write (file_unit, '(f20.6$)') table(i,1)
    do j = 1 , size(fels_types)
      call compute_aggf(table(i,1), val_aggf ,fels_type=fels_types(j))
      write (file_unit, '(f20.6$)') val_aggf
    enddo
    write(file_unit, *)
  enddo
  close(file_unit)
end subroutine


! ============================================================================
! ============================================================================
subroutine compare_fels_profiles ()
  character (len=255) ,dimension (6) :: fels_types
  real (dp) :: height , temperature
  integer :: i , file_unit

  ! All possible optional arguments for standard_temperature
  fels_types = (/ "US1976"              , "tropical",    &
                  "subtropical_summer" , "subtropical_winter" , &
                  "subarctic_summer"   , "subarctic_winter"    /)

  open  ( newunit = file_unit, &
          file    = '../examples/compare_fels_profiles.dat' , &
          action  = 'write' &
        )

  ! Print header
  write ( file_unit , '(100(a20))' ) &
    'height', ( trim( fels_types(i) ) , i = 1 , size (fels_types) )

  ! Print results
  do height = 0. , 70. , 1.
    write ( file_unit , '(f20.3$)' ) , height
    do i = 1 , size (fels_types)
      call standard_temperature &
        ( height , temperature , fels_type = fels_types(i) )
      write ( file_unit , '(f20.3$)' ),  temperature
    enddo
    write ( file_unit , * )
  enddo
  close(file_unit)
end subroutine

! ============================================================================
! ============================================================================
subroutine aggf_resp_h ()
  real(dp), dimension(:,:), allocatable :: table , results
  integer :: i, j, file_unit , ii
  real(dp) :: val_aggf

  ! Get the spherical distances from Merriam92
  call read_tabulated_green( table , author = "merriam")

  ! Specify the output table and put station height in first row
  allocate ( results( 0 : size (table(:,1)) , 7 ) )
  results(0,1) = 1./0      ! Infinity in first header
```

```
  results(0,3) = 0.0      !   0 m
  results(0,3) = 0.001    !   1 m
  results(0,4) = 0.01     !  10 m
  results(0,5) = 0.1      ! 100 m
  results(0,6) = 1.       !   1 km
  results(0,7) = 10.      !  10 km

  ! write results to file
  open ( &
    newunit = file_unit, &
    file    = '../examples/aggf_resp_h.dat', &
    action  = 'write' &
    )

  write (file_unit, '(8(F20.8))' ) results(0, :)
  do i =1 , size (table(:,1))
    ! denser sampling
    do ii = 0,8
      results( i , 1 )  = table(i,1) + ii * (table(i+1,1) - table(i,1)) / 9.
      ! only compute for small spherical distances
      if (results(i, 1) .gt. 0.2 ) exit
      write (file_unit, '(F20.7,$)') , results(i,1)
      do j =  2 , size(results(1,: ) )
        call compute_aggf(results(i,1) , val_aggf, dh=0.0001, h =
results(0,j))
        results(i,j) = val_aggf
        write (file_unit,'(f20.7,1x,$)') results(i,j)
      enddo
      write (file_unit,*)
    enddo
  enddo
  close (file_unit)
end subroutine

! ============================================================================
! ============================================================================
subroutine aggf_resp_t ()
  real(dp), dimension(:,:), allocatable :: table , results
  integer :: i, j , file_unit
  real(dp) :: val_aggf

  ! read spherical distances from Merriam
  call read_tabulated_green( table )

  ! Header in first row with surface temperature [K]
  allocate ( results(0 : size (table(:,1)) , 4 ) )
  results(0,1) = 1./0
  results(0,2) = t0 +   0.
  results(0,3) = t0 +  15.0
  results(0,4) = t0 + -45.0
  do i =1 , size (table(:,1))
    results( i , 1 )  = table(i,1)
    do j =  2 , 4
    call compute_aggf( results(i , 1 ) , val_aggf, dh = 0.00001,
t_zero = results(0, j) )
    results(i,j) = val_aggf
    enddo
  enddo

  ! Print results to file
  open ( newunit = file_unit , &
         file    = '../examples/aggf_resp_t.dat' , &
         action  = 'write')
  write (file_unit , '(4F20.5)' ) &
    ( (results(i,j) , j=1,4) , i = 0, size ( table(:,1) ) )
  close (file_unit)
end subroutine

! ============================================================================
! ============================================================================
subroutine aggfdt_resp_dt ()
  real(dp), dimension(:,:), allocatable :: table , results
  integer :: i, j , file_unit
  real(dp) :: val_aggf

  ! read spherical distances from Merriam
  call read_tabulated_green( table )

  ! Header in first row with surface temperature [K]
  allocate ( results(0 : size (table(:,1)) , 6 ) )
  results(0,1) = 1./0
  results(0,2) = 1.
  results(0,3) = 5.
  results(0,4) = 10.
  results(0,5) = 20.
  results(0,6) = 50.
  do i =1 , size (table(:,1))
```

```fortran
    results( i , 1 )  = table(i,1)
    do j =  2 , 6
    call compute_aggfdt( results(i , 1 ) , val_aggf, results(0, j ) )
    results(i,j) = val_aggf
    enddo
  enddo

  ! Print results to file
  open ( newunit = file_unit , &
         file    = '../examples/aggfdt_resp_dt.dat' , &
         action  = 'write')
  write (file_unit , '(6F20.5)' ) &
    ( (results(i,j) , j=1,6) , i = 0, size ( table(:,1) ) )
  close (file_unit)
end subroutine

! ============================================================================
! ============================================================================
subroutine aggf_resp_dz ()
  real(dp), dimension(:,:), allocatable :: table , results
  integer :: file_unit , i , j
  real(dp) :: val_aggf

  open ( newunit = file_unit, &
         file    = '../examples/aggf_resp_dz.dat', &
         action='write')

  ! read spherical distances from Merriam
  call read_tabulated_green( table )

  ! Differences in AGGF(dz) only for small spherical distances
  allocate ( results( 0 : 29 , 0: 5 ) )
  results = 0.

  ! Header in first row [ infty and selected dz follow on ]
  results(0,0) = 1./0
  results(0,1:5)=(/ 0.0001, 0.001, 0.01, 0.1, 1./)

  do i = 1 , size ( results(:,1) ) - 1
    results(i,0) = table(i , 1 )
    do j = 1 , size (results(1,:) ) - 1
    call compute_aggf( results(i,0) , val_aggf , dh = results(0,j) )
    results(i, j) =  val_aggf
    enddo

    ! compute relative errors from column 2 for all dz with respect to column 1
    results(i,2:) = abs((results(i,2:) - results(i,1)) / results(i,1) * 100 )
  enddo

  ! write result to file
  write ( file_unit , '(<size(results(1,:))>f14.6)' ) &
    ((results(i,j), j=0,size(results(1,:)) - 1), i=0,size(results(:,1)) - 1)
  close(file_unit)
end subroutine

! ============================================================================
! ============================================================================
subroutine standard1976  !()
  real(dp) :: height , temperature , gravity , pressure , pressure2 , density
  integer :: file_unit

  open ( newunit = file_unit , &
         file    = '../examples/standard1976.dat', &
         action  = 'write' )
  ! print header
  write ( file_unit , '(6(a12))' ) &
    'height[km]', 'T[K]' , 'g[m/s2]' , 'p[hPa]', 'p_simp[hPa]' , 'rho[kg/m3]'
  do height=0.,98.
    call standard_temperature( height , temperature )
    call standard_gravity( height , gravity )
    call standard_pressure( height , pressure )
    call standard_pressure( height , pressure2 , if_simplificated = .true. )
    call standard_density( height , density )
    ! print results to file
    write( file_unit,'(5f12.5, e12.3)'), &
    height,temperature , gravity , pressure , pressure2 , density
  enddo
  close( file_unit )
end subroutine

! ============================================================================
! ============================================================================
subroutine aggf_resp_hmax ()
  real (dp) , dimension (10) :: psi
```

```
  real (dp) , dimension (:)   , allocatable :: heights
  real (dp) , dimension (:,:) , allocatable :: results
  integer :: file_unit , i , j
  real(dp) :: val_aggf

  ! selected spherical distances
  psi=(/0.000001, 0.000005,0.00001, 1,  2, 3 , 5, 10 , 90 ,  180 /)

  ! get heights (for nice graph) - call auxiliary subroutine
  call aux_heights( heights )

  open ( newunit = file_unit , &
         file    = '../examples/aggf_resp_hmax.dat', &
         action  = 'write')

  allocate ( results( 0:size(heights)-1 , 1+size(psi) ) )

  do j=0 , size (results(:,1))
      results( j , 1 ) = heights(j)

    do i = 1 , size(psi)
      call compute_aggf( psi(i) , val_aggf , hmax = heights(j) , dh
 = 0.00001 )
      results(j,i+1) = val_aggf

      if (j.gt.0) then
        results(j,i+1) = results(j,i+1) / results(0,i+1) * 100
      endif
    enddo
  enddo

  ! print header
  write(file_unit , '(a14,SP,100f14.5)' ),"#wys\psi", (psi(j) , j= 1,size(psi))
  ! print results
  do i=1, size (results(:,1))-1
    write(file_unit, '(100f14.3)' ) (results(i,j), j = 1, size(psi)+1 )
  enddo
  close(file_unit)
end subroutine

! ==========================================================================
! ==========================================================================
subroutine aux_heights ( table )
  real(dp) , dimension (:), allocatable, intent(inout) :: table
  real(dp) , dimension (0:1000) :: heights
  real(dp) :: height
  integer :: i , count_heights

  heights(0) =60
  i=0
  height=-0.001
  do while (height.lt.60)
    i=i+1
    if (height.lt.0.10) then
      height=height+2./1000
    elseif(height.lt.1) then
      height=height+50./1000
    else
      height=height+1
    endif
    heights(i)= height
    count_heights=i
  enddo
  allocate ( table( 0 : count_heights ) )
  table(0 : count_heights ) = heights( 0 : count_heights )
end subroutine

subroutine aggf_thin_layer ()
  integer :: file_unit , i
  real(dp) , dimension (:,:), allocatable :: table

  ! read spherical distances from Merriam
  call read_tabulated_green(table)
  do i = 1 , size (table(:,1))
    write(*,*) table(i,1:2) , gn_thin_layer(table(i,1))
  enddo

end subroutine
end program
```

6.9 /home/mrajner/src/grat/src/grat.f90 File Reference

Functions/Subroutines

- program **grat**

6.9.1 Detailed Description

Definition in file grat.f90.

6.10 grat.f90

```
! ==============================================================================
! ==============================================================================
program grat
  use iso_fortran_env
  use get_cmd_line
  use mod_polygon
  use mod_data
  use mod_green


  implicit none
!  character(255) :: dummy
!  real :: del, grav=0. ,cd ,sd, rlato ,rlong , ddist, pole ,cale_pole, normalizacja, cisnienie_stacja, temperatura_stacja
!  integer :: ii , naz , jj , i , j
!  integer , parameter :: minaz =50 !mrajner 2012-10-03 14:24
!  !integer , parameter :: minaz =1
!  integer , parameter :: ile  = 5 !mrajner 2012-10-03 14:24
!  !integer , parameter :: ile   = 1
!  real :: azstp, azstpd, azimuth ,caz ,saz,saztp, caztp,stpfac ,cb , sb ,sg ,cg
!  real ::  xx
!  real :: grav_merriam_e=0. ,grav_merriam_n=0. , grav_merriam_s=0. ,grav_merriam_e_nib=0.
!  real ::grav_merriam_n_t=0.
!  real ::grav_merriam_n_h=0.
!  real :: admit3
!  real,dimension(85) :: b,c,d
!  integer:: przebieg ,licznik
!  real, dimension(6) :: values_interpolowane
!  real , dimension(:,:), allocatable :: tablica
!  integer :: ile_plikow
!  real :: szerokosc_zmienna , dlugosc_zmienna , wysokosc_stacji_etopo2
!  logical :: czy_otworzyc_nowy_plik=.true.


  real(sp) :: x , y , z , lat ,lon ,val !tmp variables
  integer :: i , j
  integer :: d(6)

  call cpu_time(cpu_start)

  ! gather cmd line option decide where to put output
  call intro( program_calling = "grat" )

  ! print header to log: version, date and summary of command line options
  call print_settings(program_calling = "grat")


  ! read models into memory
  do i =1 , size(model)
    if (model(i)%if) call read_netcdf( model(i) )
  enddo



  do j = 1 , size (dates)
      call get_variable( model(1) , date = dates(j)%date)

    do i = 1 , size(sites)
      call get_value(model(1), sites(i)%lat, sites(i)%lon , val)
!       write(output%unit ,  '(f15.4,2x,i4,5i2.2,3f13.4)') ,mjd (dates(j)%date) , dates(j)%date , sites%lat, sites%lon, val
  call convolve(sites(1) , green , denserdist = 0 , denseraz =1)
    enddo
  enddo

```

```
  ! todo wysokosci nad wodą ustaw na 0. Głębokość nie jest interesująca





  call cpu_time(cpu_finish)
  write(log%unit, '(/,"Execution time:",1x,f16.9," seconds")') cpu_finish -
cpu_start
  write(log%unit, form_separator)

end program
```

Appendix A

Polygon

This examples show how the exclusion of selected polygons works

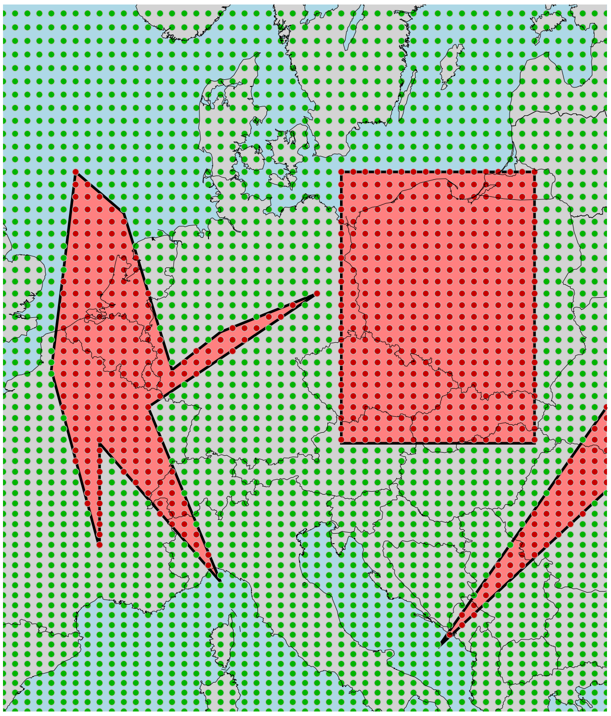


Figure A.1: If only excluded polygons (red area) are given all points falling in it will be excluded (red points) all other will be included

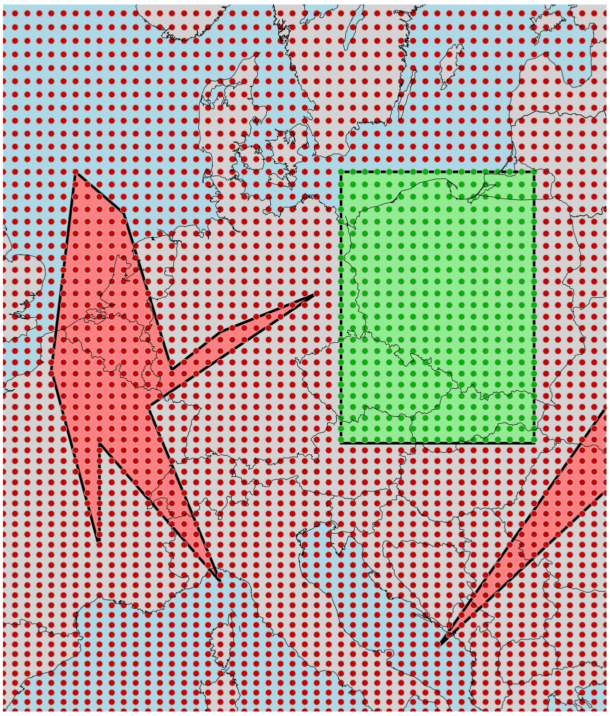


Figure A.2: If at least one included are are given (green area) than all points which not fall into included area will be excluded

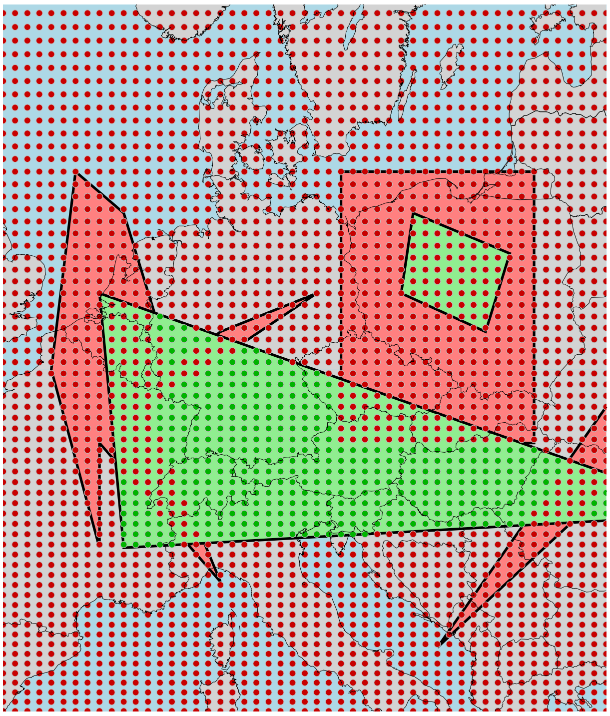


Figure A.3: If there is overlap of polygons the exclusion has higher priority

Appendix B

Interpolation

grat v. 1.0 Manual

Figure B.1: Interpoloation

Bibliography

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.

M. Rajner. *Wyznaczanie atmosferycznych poprawek grawimetrycznych na podstawie numerycznych modeli pogody*. PhD thesis, 2013. URL http://www.geo.republika.pl/pub.

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

P. Wessel and W. H. F. Smith. New, improved version of generic mapping tools released. *EOS Trans. Amer. Geophys. U.*, 79(47):579, 1998.

Index