/home/mrajner/dr/rysunki/grat_logo-crop.pdf

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

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

site height from model

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 58 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, 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.

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 479 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 **?** 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 **?**

Definition at line 455 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 **?**
- huang **?**
- 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 aggf.f90.

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

Bouger plate computation see eq. page 288.

?

Definition at line 501 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 470 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 **?** first derivative (respective to column height) according to equation 26 in **?** 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 **?** Compute air density for given altitude for standard atmosphere

using formulae 12 in **?**

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 **?**

Definition at line 301 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,* 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 **?** or **?** for details. Uses formulae 5 from **?**. 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 **?**

Parameters

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

Definition at line 369 of file aggf.f90.

5.2.2.12 subroutine 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 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 61 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 46 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 92 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

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)
- 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) **moreverbose**
- type(file) **refpres**
- type(file), dimension(:), allocatable **model**
- 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)"

5.7.1 Detailed Description

Definition at line 8 of file get_cmd_line.f90.

5.7.2 Member Function/Subroutine Documentation

5.7.2.1 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 497 of file get_cmd_line.f90.

5.7.2.2 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 169 of file get_cmd_line.f90.

5.7.2.3 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 847 of file get_cmd_line.f90.

5.7.2.4 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 771 of file get_cmd_line.f90.

5.7.2.5 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 685 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 18 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 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)

5.9.1 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) **?**

Definition at line 10 of file mod_data.f90.

5.9.2 Member Function/Subroutine Documentation

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

Check the return code from netCDF manipulation.

from **?**

Definition at line 216 of file mod_data.f90.

5.9.2.2 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 *level,* 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 **?**

Get value can

Definition at line 237 of file mod_data.f90.

5.9.2.3 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 25 of file mod_data.f90.

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

Unpack variable.

from **?**

Definition at line 198 of file mod_data.f90.

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

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

5.10 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**

5.10.1 Detailed Description

Definition at line 1 of file mod_green.f90.

5.10.2 Member Function/Subroutine Documentation

5.10.2.1 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 179 of file mod_green.f90.

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

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

5.11 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)

5.11.1 Detailed Description

Definition at line 1 of file mod_polygon.f90.

5.11.2 Member Function/Subroutine Documentation

5.11.2.1 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 **?** adopted to grat and Fortran90 syntax From original description

Definition at line 82 of file mod_polygon.f90.

5.11.2.2 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 **?** slightly modified

Definition at line 196 of file mod_polygon.f90.

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

Reads polygon data.

inspired by spotl **?**

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

5.12 get_cmd_line::polygon_data Type Reference

Public Attributes

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

5.12.1 Detailed Description

Definition at line 29 of file get_cmd_line.f90.

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

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

5.13 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**

5.13.1 Detailed Description

Definition at line 34 of file get_cmd_line.f90.

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

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

5.14 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.

5.14.1 Detailed Description

Definition at line 9 of file mod_green.f90.

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

- /home/mrajner/src/grat/src/mod_green.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 71 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/src/aggf.f90 File Reference

This module contains utitlities for computing Atmospheric Gravity Green Functions.

Data Types

- module aggf

6.1.1 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.

6.2 aggf.f90

```
! ===============================================================================
!> \file
!! \brief This module contains utitlities for computing
!! Atmospheric Gravity Green Functions
!!
!! In this module there are several subroutines for computing
!! AGGF and standard atmosphere parameters
! ===============================================================================
module aggf

  use constants
  implicit none

contains

! ===============================================================================
!> \brief Compute first derivative of AGGF with respect to temperature
!! for specific angular distance (psi)
!!
!! optional argument define (-dt;-dt) range
!! See equation 19 in \cite Huang05
!! Same simple method is applied for aggf(gn) if \c aggf optional parameter
!! is set to \c .true.
!! \warning Please do not use \c aggf=.true. this option was added only
!! for testing some numerical routines
! ===============================================================================
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. !< Default value
  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

! ==============================================================================
!> Wczytuje tablice danych AGGF
!! \li merriam \cite Merriam92
!! \li huang   \cite Huang05
!! \li rajner  \cite Rajnerdr
!!
!! This is just quick solution for \c example_aggf program
!! in \c grat see the more general routine \c parse_green()
! ==============================================================================
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


! ==============================================================================
!> This subroutine computes the value of atmospheric gravity green functions
!! (AGGF) on the basis of spherical distance (psi)
! ==============================================================================
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       !< spherical distance from site
 [degree]
  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         !< station height                 [km]
     (default=0)
  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 )

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

      !> direct derivative of equation 20 \cite Huang05
      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
      !> first derivative (respective to column height)
      !! according to equation 26 in \cite Huang05
      !! micro Gal / hPa / km
      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 GN
        !! micro Gal / hPa
        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 you put the optional parameter \c if_normalization=.false.
  !! this block will be skipped
  !! by default the normalization is applied according to \cite Merriam92
  if ( (.not.present(if_normalization)) .or. (if_normalization)) then
    aggf_val= psir * aggf_val * 1e5  / p0
  endif

end subroutine

! ==============================================================================
!> Compute air density for given altitude for standard atmosphere
!!
!! using formulae 12 in \cite Huang05
! ==============================================================================
subroutine standard_density ( height , rho , t_zero ,fels_type )

  implicit none
  real(dp) , intent(in)  ::  height !< height [km]
  real(dp) , intent(in), optional  :: t_zero !< if this parameter is given
  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

! =============================================================================
!> \brief Computes pressure [hPa] for specific height
!!
!! See \cite US1976 or  \cite Huang05 for details.
!! Uses formulae 5 from \cite Huang05.
!! Simplified method if optional argument if_simplificated = .true.
! =============================================================================
subroutine standard_pressure (height, pressure , &
          p_zero , t_zero , h_zero,  if_simplificated ,fels_type , inverted)
  implicit none
  real(dp) , intent(in)            :: height
  real(dp) , intent(in) , optional :: t_zero , p_zero , h_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 , sfc_height , sfc_temperature , sfc_gravity , alpha , sfc_pressure

  sfc_temperature = t0
  sfc_pressure = p0
  sfc_height = 0.
  sfc_gravity = g0

  if (present(h_zero)) then
    sfc_height = h_zero
    call standard_temperature(sfc_height , sfc_temperature )
    call standard_temperature(sfc_height , sfc_temperature )
    call standard_gravity(sfc_height , sfc_gravity )
  endif

  if (present(p_zero)) sfc_pressure = p_zero
  if (present(t_zero)) sfc_temperature = t_zero

  lambda = r_air * sfc_temperature / sfc_gravity

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

! =============================================================================
! > This will transfer pressure beetween different height using barometric
! formulae
! ==============================================================================
!> \warning OBSOLETE ROUTINE -- use standard_pressure instead with optional args
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

! ==============================================================================
!> \brief Compute gravity acceleration of the Earth
!! for the specific height using formula
!!
!! see \cite US1976
! ==============================================================================
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


! ==============================================================================
!> \brief Compute geometric height from geopotential heights
! ==============================================================================
real(sp) function geop2geom (geopotential_height)
  real (sp) :: geopotential_height

  geop2geom = geopotential_height * (r0 / ( r0 + geopotential_height ) )
end function


! ==============================================================================
!> Iterative computation of surface temp. from given height using bisection
!! method
! ==============================================================================
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
! ==============================================================================
!> \brief 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 \argument fels_type
!! \cite Fels86
! ==============================================================================
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
    !< \li US standard atmosphere (default)
    !! \li tropical
    !! \li subtropical_summer
    !! \li subtropical_winter
    !! \li subarctic_summer
    !! \li subarctic_winter
  real(dp) :: aux , cn , t
  integer :: i,indeks
  real , dimension (10) :: z,c,d

  !< Read into memory the parameters of temparature height profiles
  !! for standard atmosphere
  !! From \cite Fels86
  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

!==============================================================================
!> \brief 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 \cite Merriam92
!==============================================================================
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


!==============================================================================
!> \brief returns numbers of arguments for n times denser size
!!
!! i.e. * * * *  -->  * . . * . . * . . * (3 times denser)
!==============================================================================
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

!==============================================================================
!> \brief Bouger plate computation
!!
!==============================================================================
real(dp) function bouger ( R_opt )
  real(dp), optional :: r_opt !< height of point above the cylinder
  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
! ==============================================================================
!> \brief Bouger plate computation
!! see eq. page 288 \cite Warburton77
! ==============================================================================
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
```

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

This module define some constant values used.

Data Types

- module constants

6.3.1 Detailed Description

This module define some constant values used.

Definition in file constants.f90.

6.4 constants.f90

```
! ==============================================================================
!> \file
!! This module define some constant values used
! ==============================================================================
module constants

  implicit none
  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)
    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          !< mean density of Earth [kg/m3]

contains

! ==============================================================================
!> For given vectors x1, y1 and x2, y2 it gives x2interpolated for x1
!!
!! uses \c ispline and \c spline subroutines
```

```
! ==============================================================================
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

! ==============================================================================
!> This subroutine was taken from
!! \todo give source
! ==============================================================================
!  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


! ==============================================================================
!> This subroutine was taken from
!! \todo give source
! ==============================================================================
!======================================================================
! 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

! ==============================================================================
!> taken from ArkM http://www.tek-tips.com/viewthread.cfm?qid=1688013
! ==============================================================================
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

! ==============================================================================
!> This routine skips the lines with comment chars (default '#')
!! from opened files (unit) to read
! ==============================================================================
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

!> downloaded from  http://aa.usno.navy.mil/faq/docs/jd_formula.php
!! \todo mjd!
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
```

6.5 /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** ()

6.5.1 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.

6.5.2 Function/Subroutine Documentation

6.5.2.1 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.

6.5.2.2 subroutine example_aggf::compare_fels_profiles ( )

Compare different vertical temperature profiles.

Using tables and formula from **?**

Definition at line 192 of file example_aggf.f90.

6.5.2.3 subroutine example_aggf::simple_atmospheric_model ( )

Reproduces data to Fig.∼3 in.

?

Definition at line 39 of file example_aggf.f90.

6.5.2.4 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.

6.6 example_aggf.f90

```
00001 ! ============================================================================
00002 !> \file
00003 !! \brief This program shows some example of using AGGF module
00004 !! \author Marcin Rajner
00005 !! \date 20121108
00006 !!
00007 !! The examples are in contained subroutines
00008 ! ============================================================================
00009 program example_aggf
00010
00011   !> module with subroutines for calculating Atmospheric Gravity Green
       Fucntions
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 ! =============================================================================
00037 !> \brief Reproduces data to Fig.~3 in \cite Warburton77
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
      simple_def(r) * 1e8
  enddo

end subroutine
! ==============================================================================
!> \brief Compare tabulated green functions from different authors
! ==============================================================================
subroutine compare_tabulated_green_functions ()
  integer :: i , j , file_unit , ii , iii
  real(dp), dimension(:,:), allocatable :: table , results
  real(dp), dimension(:,:), allocatable :: parameters
  real(dp), dimension(:), allocatable :: x1, y1 ,x2 , y2 , x, y , x_interpolated, y_interpolated
  integer :: how_many_denser
  character(len=255), dimension(3) :: authors
  integer , dimension(3) :: columns

  authors=["rajner", "merriam" , "huang"]
  ! selected columns for comparison in appropriate tables
  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

! =============================================================================
!> \brief Compute AGGF and derivatives
! =============================================================================
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

! ============================================================================
!> \brief Compare different vertical temperature profiles impact on AGGF
! ============================================================================
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


! ============================================================================
!> \brief Compare different vertical temperature profiles
!!
!! Using tables and formula from \cite Fels86
! ============================================================================
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) )
```

```fortran

  ! 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

! ============================================================================
!> \brief Computes AGGF for different site height (h)
! ============================================================================
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

! ============================================================================
!> \brief This computes AGGF for different surface temperature
! ============================================================================
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

! ==============================================================================
!> \brief This computes AGGFDT for different dT
! ==============================================================================
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))
    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

! ==============================================================================
!> \brief This computes AGGF for different height integration step
! ==============================================================================
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
```

```fortran

! ============================================================================
!> \brief This computes standard atmosphere parameters
!!
!! It computes temperature, gravity, pressure, pressure (simplified formula)
!! density for given height
! ============================================================================
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

! ============================================================================
!> \brief This computes relative values of AGGF for different atmosphere
!! height integration
! ============================================================================
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

      !> Relative value of aggf depending on integration height
      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

! ============================================================================
!> \brief Auxiliary subroutine -- height sampling for semilog plot
! ============================================================================
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
```

6.7 /home/mrajner/src/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

6.7.1 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.

6.8 get_cmd_line.f90

```
! =============================================================================
!> \file
!! \brief 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
! =============================================================================
module get_cmd_line
```

```
use iso_fortran_env
use constants

implicit none

!----------------------------------------------------
! Greens function
!----------------------------------------------------

type green_functions
  real(dp),allocatable,dimension(:) :: distance
  real(dp),allocatable,dimension(:) :: data
  logical  :: if
end type
type(green_functions), allocatable , dimension(:) :: green
integer :: denser(2) = [1,1]

!----------------------------------------------------
! polygons
!----------------------------------------------------
type polygon_data
  logical :: use
  real(sp), allocatable , dimension (:,:) :: coords
end type

type polygon_info
  integer :: unit
  character(:), allocatable  :: name
  type(polygon_data) , dimension (:) , allocatable :: polygons
  logical :: if
end type

type(polygon_info) , dimension (2) :: polygons

!----------------------------------------------------
! dates
!----------------------------------------------------
type dateandmjd
  real(dp) :: mjd
  integer,dimension (6) :: date
end type

real(kind=4) :: cpu_start , cpu_finish  !< for time execution of program
type(dateandmjd) , allocatable,dimension (:) :: dates


!----------------------------------------------------
! command line entry
!----------------------------------------------------
type additional_info
  character (len=55) ,allocatable ,dimension(:) :: names
end type
type cmd_line
  character(2) :: switch
  integer ::      fields
  character (len=255) ,allocatable ,dimension(:) :: field
  type (additional_info), allocatable , dimension(:) :: fieldnames
end type

!---------------------------------------------------
! site information
!---------------------------------------------------
type site_data
 character(:), allocatable :: name
 real(sp)                  :: lat,lon,height
end type

type(site_data) , allocatable , dimension(:) :: sites

! various
!----------------------------------------------------
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

!----------------------------------------------------
! Site names file
!----------------------------------------------------
character(:), allocatable &
        :: filename_site
integer :: fileunit_site


type file
  character(:), allocatable &
        :: name
```

```
    ! varname , lonname,latname,levelname , timename
    character(len=50) :: names(5) = [ "z", "lon", "lat","level","time"]

    integer :: unit = output_unit

    ! if file was determined
    logical :: if =.false.

    ! to read into only once
    logical :: first_call =.true.

    ! boundary of model e , w ,s ,n
    real(sp):: limits(4)

!     resolution of model in lon lat
!    real(sp):: resolution(2)

    real(sp) , allocatable ,dimension(:) :: lat , lon , time ,level
    integer , allocatable , dimension(:,: ) :: date

    real (sp), dimension(2) :: latrange , lonrange

    ! todo
    logical :: if_constant_value
    real(sp):: constant_value

    ! data
    !> 4 dimension - lat , lon , level , mjd
    ! todo
    real(sp) , allocatable , dimension (:,:,:) :: data

    ! netcdf identifiers
    integer :: ncid
    integer :: interpolation = 1
  end type

  ! External files
  type(file) ::  log  , output , moreverbose , refpres
  type(file) , allocatable, dimension (:) :: model

  character (len =40) :: model_names (5) = ["pressure_surface" , &
    "temperature_surface" , "topography" , "landsea" , "pressure levels" ]


  character(len=5) :: green_names(5) = [ "GN   ", "GN/dt", "GN/dh","GN/dz","GE
  "]


  ! Verbose information and the output for \c log_file
  logical :: if_verbose  = .false.  !< whether print all information
  logical :: inverted_barometer  = .true.

  character (50) :: interpolation_names (2) &
    = [ "nearest" , "bilinear" ]

  !----------------------------------------------------
  ! For preety printing
  !----------------------------------------------------
  character(len=255), parameter ::  &
    form_header    = ’(60("#"))’ , &
    form_separator = ’(60("-"))’ , &
    form_inheader  = ’(("#"),1x,a56,1x,("#"))’ , &
    form_60        = "(a,100(1x,g0))",          &
    form_61        = "(2x,a,100(1x,g0))",    &
    form_62        = "(4x,a,100(1x,g0))",       &
    form_63        = "(6x,100(x,g0))",       &
    form_64        = "(4x,4x,a,4x,a)"


contains
! ============================================================================
!> This subroutine counts the command line arguments
!!
!! Depending on command line options set all initial parameters and reports it
! ============================================================================
subroutine intro (program_calling)
  implicit none
  integer :: i, j
  character(len=255) :: dummy, dummy2,arg
  character(len=*) :: program_calling
  type(cmd_line) :: cmd_line_entry

  if(iargc().eq.0) then
    write(output_unit , ’(a)’ ) , ’Short description: ./’//program_calling//’
 -h’
    call exit
  else
```

```
    open(newunit=fileunit_tmp,status='scratch')
    write (fileunit_tmp,form_61) "command invoked"
    call get_command(dummy)
    write (fileunit_tmp,form_62) trim(dummy)
    do i = 1 , iargc()
      call get_command_argument(i,dummy)
      ! allow specification like '-F file' and '-Ffile'
      call get_command_argument(i+1,dummy2)
      if (dummy(1:1).eq."-") then
        arg = trim(dummy)
      else
        arg=trim(arg)//trim(dummy)
      endif
      if(dummy2(1:1).eq."-".or.i.eq.iargc()) then
         call get_cmd_line_entry(arg, cmd_line_entry , program_calling = program_calling)
      endif
    enddo

    call if_minimum_args( program_calling = program_calling )

    ! Where and if to log the additional information
    if (log%if) then
      ! if file name was given then automaticall switch verbose mode
      if_verbose = .true.
      open (newunit = log%unit, file = log%name , action = "write" )
    else
      ! if you don't specify log file, or not switch on verbose mode
      ! all additional information will go to trash
      ! Change  /dev/null accordingly if your file system does not
      ! support this name
      if (.not.if_verbose) then
        open (newunit=log%unit, file = "/dev/null", action = "write" )
      endif
    endif
  endif
end subroutine

! ==============================================================================
!> Check if at least all obligatory command line arguments were given
!! if not print warning
! ==============================================================================
subroutine if_minimum_args ( program_calling )
  implicit none
  character (*) , intent(in) :: program_calling

  if (program_calling.eq."grat" ) then

    if (size(sites) .eq. 0) then
      write(error_unit, * ) "ERROR:", program_calling
      write(error_unit, * ) "ERROR:", "no sites!"
      call exit
    endif
  elseif(program_calling.eq."polygon_check" ) then
  endif
end subroutine

! ==============================================================================
!> This function is true if switch is used by calling program or false if it
!! is not
! ==============================================================================
logical function if_switch_program (program_calling , switch )
  implicit none
  character(len=*), intent (in) :: program_calling
  character(len=*),  intent (in) :: switch
  character, dimension(:) , allocatable :: accepted_switch
  integer :: i

  ! default
  if_switch_program=.false.

  ! depending on program calling decide if switch is permitted
  if (program_calling.eq."grat") then
    allocate( accepted_switch(15) )
    accepted_switch = [ "V" , "f" , "S", "B" , "L" , "G" , "P" , "p", &
      "o" , "F" , "I" , "D" , "d" , "v" , "h"   ]
  elseif(program_calling.eq."polygon_check") then
    allocate( accepted_switch(12) )
    accepted_switch = [ "V" , "f" , "A", "B" , "L" , "P" , "o", "S" , &
      "h" , "v" , "I" , "i"]
  elseif(program_calling.eq."value_check") then
    allocate( accepted_switch(9) )
    accepted_switch = [ "V" , "F" , "o", "S" , "h" , "v" , "I" , "D" , "L"]
  else
    if_switch_program=.true.
    return
  endif
```

```

 ! loop trough accepted switches
 do i =1, size (accepted_switch)
  if (switch(2:2).eq.accepted_switch(i)) if_switch_program=.
true.
 enddo
end function

! ============================================================================
!> This subroutine counts the command line arguments and parse appropriately
! ============================================================================
subroutine parse_option (cmd_line_entry , program_calling)
 type(cmd_line),intent(in):: cmd_line_entry
 character(len=*), optional :: program_calling
 integer :: i

 ! all the command line option are stored in tmp file and later its decide
 ! if it is written to STDOUT , log_file or nowhere
  select case (cmd_line_entry%switch)
  case ('-h')
   call print_help(program_calling)
   call exit
  case ('-v')
   call print_version(program_calling)
   call exit()
  case ('-V')
   if_verbose = .true.
   write(fileunit_tmp, form_62) 'verbose mode' ,trim(log%name)
   if (len(trim(cmd_line_entry%field(1))).gt.0) then
    log%if = .true.
    log%name = trim(cmd_line_entry%field(1))
    write(fileunit_tmp, form_62) 'the log file was set:' ,log%name
   endif
  case ('-S')
   ! check if format is proper for site
   ! i,e. -Sname,B,L[,H]
   if (.not. allocated(sites)) then
    if (  is_numeric(cmd_line_entry%field(2)) &
    .and.is_numeric(cmd_line_entry%field(3)) &
    .and.index(cmd_line_entry%field(1), "/" ).eq.0 &
    .and.(.not.cmd_line_entry%field(1).eq. "Rg" ) &
    ) then
      allocate (sites(1))
      sites(1)%name = trim(cmd_line_entry%field(1))
      read ( cmd_line_entry%field(2) , * ) sites(1)%lat
      if (abs(sites(1)%lat).gt.90.) &
       sites(1)%lat = sign(90.,sites(1)%lat)
      read ( cmd_line_entry%field(3) , * ) sites(1)%lon
      if (sites(1)%lon.ge.360.) sites(1)%lon = mod(sites(1)%lon,360.)
      if (is_numeric(cmd_line_entry%field(4) ) ) then
       read ( cmd_line_entry%field(4) , * ) sites(1)%height
      endif
      write(fileunit_tmp, form_62) 'the site was set (BLH):' , &
       sites(1)%name, sites(1)%lat , sites(1)%lon , sites(1)%height
    else
     ! or read sites from file
     if (file_exists(cmd_line_entry%field(1) ) ) then
      write(fileunit_tmp, form_62) 'the site file was set:' , &
       cmd_line_entry%field(1)
      call read_site_file(cmd_line_entry%field(1))
     elseif(index(cmd_line_entry%field(1), "/" ).ne.0 &
       .or.cmd_line_entry%field(1).eq."Rg")  then
      call parse_gmt_like_boundaries(
cmd_line_entry )
     else
      call print_warning( "site" , fileunit_tmp)
     endif
    endif
   else
    call print_warning( "repeated" , fileunit_tmp)
   endif
  case ("-I")
   write( fileunit_tmp , form_62 , advance="no" ) "interpolation method was
 set:"
   do i = 1 , cmd_line_entry%fields
    if (is_numeric(cmd_line_entry%field(i))) then
     read ( cmd_line_entry%field(i) , * ) model(i)%interpolation
     write(fileunit_tmp , '(a10,x,$)' ) interpolation_names(model(i)
%interpolation)
     if (model(i)%interpolation.gt.size(interpolation_names)) then
      model(i)%interpolation=1
     endif
    endif
   enddo
   write(fileunit_tmp , *)
  case ("-L")
   moreverbose%if=.true.
```

```
      moreverbose%name=cmd_line_entry%field(1)
      moreverbose%names(1) = cmd_line_entry%fieldnames(1)%names(1)
      write (fileunit_tmp , form_62) "printing additional information"
      if (len(moreverbose%name).gt.0 .and. moreverbose%name.ne."") then
        open (newunit = moreverbose%unit , file = moreverbose%name , action
 = "write" )
      endif
    case ("-B")
      if (cmd_line_entry%field(1).eq."N" ) inverted_barometer=.false.
    case (’-D’)
      call parse_dates( cmd_line_entry )
    case (’-F’)
      allocate(model(cmd_line_entry%fields))
      do i = 1, cmd_line_entry%fields
        call get_model_info(model(i) , cmd_line_entry , i )
      enddo
    case ("-G")
      call parse_green(cmd_line_entry)
    case (’-M’)
      method = cmd_line_entry%field(1)
      write(fileunit_tmp, form_62), ’method was set: ’ , method
    case (’-o’)
      output%if=.true.
      output%name=cmd_line_entry%field(1)
      write(fileunit_tmp, form_62), ’output file was set: ’ , output%name
      if (len(output%name).gt.0.and. output%name.ne."") then
        open (newunit = output%unit , file = output%name , action = "write"
 )
      endif
    case (’-P’)
      do i = 1 , 2 !size(cmd_line_entry%field)
        polygons(i)%name=cmd_line_entry%field(i)
        if (file_exists((polygons(i)%name))) then
          write(fileunit_tmp, form_62), ’polygon file was set: ’ , polygons(i)
%name
          polygons(i)%if=.true.
          ! todo
!          call read_polygon (polygons(i))
        else
          write(fileunit_tmp, form_62), ’file do not exist. Polygon file was
 IGNORED’
        endif
      enddo
    case default
      write(fileunit_tmp,form_62), "unknown argument: IGNORING"
    end select
    return
end subroutine

! ==============================================================================
!> This subroutine parse -G option i.e. reads Greens function
! ==============================================================================
subroutine parse_green ( cmd_line_entry)
  type (cmd_line)  :: cmd_line_entry
  character (60) :: filename
  integer :: i , iunit , io_status , lines ,  ii
  integer :: fields(2)= [1,2]
  real (sp) , allocatable , dimension(:) :: tmp

    write(fileunit_tmp , form_62) "Green function file was set:"
    allocate (green(cmd_line_entry%fields))

    do i = 1 , cmd_line_entry%fields

      if (i.eq.6) then
        if (is_numeric(cmd_line_entry%field(i))) then
          read( cmd_line_entry%field(i), *) denser(1)
          if (is_numeric(cmd_line_entry%fieldnames(i)%names(1))) then
            read( cmd_line_entry%fieldnames(i)%names(1), *) denser(2)
          endif
          return
         endif
      endif

      if (.not.file_exists(cmd_line_entry%field(i)) &
      .and. (.not. cmd_line_entry%field(i).eq."merriam" &
      .and. .not. cmd_line_entry%field(i).eq."huang" &
      .and. .not. cmd_line_entry%field(i).eq."rajner" )) then
        cmd_line_entry%field(i)="merriam"
      endif


      !> change the paths accordingly
      if (cmd_line_entry%field(i).eq."merriam") then
        filename="/home/mrajner/src/grat/dat/merriam_green.dat"
        if (i.eq.1) fields = [1,2]
        if (i.eq.2) fields = [1,3]
        if (i.eq.3) fields = [1,4]
        if (i.eq.4) fields = [1,4]
        if (i.eq.5) fields = [1,6]
      elseif(cmd_line_entry%field(i).eq."huang") then
        filename="/home/mrajner/src/grat/dat/huang_green.dat"
        if (i.eq.1) fields = [1,2]
        if (i.eq.2) fields = [1,3]
        if (i.eq.3) fields = [1,4]
        if (i.eq.4) fields = [1,5]
        if (i.eq.5) fields = [1,6]
      elseif(cmd_line_entry%field(i).eq."rajner") then
        filename="/home/mrajner/src/grat/dat/rajner_green.dat"
        if (i.eq.1) fields = [1,2]
        if (i.eq.2) fields = [1,3]
        if (i.eq.3) fields = [1,4]
        if (i.eq.4) fields = [1,5]
        if (i.eq.5) fields = [1,6]
      elseif(file_exists(cmd_line_entry%field(i))) then
        filename = cmd_line_entry%field(i)
        if (size(cmd_line_entry%fieldnames).ne.0 .and. allocated(cmd_line_entry%fieldnames(i)%names)) then
          do ii=1, 2
            if(is_numeric(cmd_line_entry%fieldnames(i)%names(ii) ) ) then
              read( cmd_line_entry%fieldnames(i)%names(ii), *) fields(ii)
            endif
          enddo
        endif
      endif

      allocate(tmp(max(fields(1),fields(2))))
      lines = 0
      open ( newunit =iunit,file=filename,action="read")
      do
        call skip_header(iunit)
        read (iunit , * , iostat = io_status)
        if (io_status == iostat_end) exit
        lines = lines + 1
      enddo
      allocate (green(i)%distance(lines))
      allocate (green(i)%data(lines))
      rewind(iunit)
      lines = 0
      do
        call skip_header(iunit)
        lines = lines + 1
        read (iunit , * , iostat = io_status) tmp
        if (io_status == iostat_end) exit
        green(i)%distance(lines) = tmp(fields(1))
        green(i)%data(lines)     = tmp(fields(2))
      enddo
      deallocate(tmp)
      close(iunit)
      if (cmd_line_entry%field(i).eq."merriam" .and. i.eq.4) then
        green(i)%data = green(i)%data * (-1.)
      endif
      if (cmd_line_entry%field(i).eq."huang" .and. (i.eq.3.or.i.eq.4)) then
        green(i)%data = green(i)%data * 1000.
      endif
      write(fileunit_tmp , form_63) trim(green_names(i)), &
        trim(cmd_line_entry%field(i)),":", fields
    enddo
end subroutine

! ==============================================================================
!> Counts occurence of character (separator, default comma) in string
! ==============================================================================
integer function count_separator (dummy , separator)
  character(*) , intent(in) ::dummy
  character(1), intent(in), optional  :: separator
  character(1)  :: sep
  character(:), allocatable :: dummy2
  integer :: i

  dummy2=dummy
  sep = ","
  if (present(separator)) sep = separator
  count_separator=0
  do
    i = index(dummy2, sep)
    if (i.eq.0) exit
    dummy2 = dummy2(i+1:)
    count_separator=count_separator+1
  enddo
end function


```

```
! ==============================================================================
!> This subroutine fills the fields of command line entry for every input arg
! ==============================================================================
subroutine get_cmd_line_entry (dummy , cmd_line_entry , program_calling )
  character(*) :: dummy
  character(:), allocatable :: dummy2
  type (cmd_line),intent(out) :: cmd_line_entry
  character(1) :: separator=","
  character(len=*) , intent(in) , optional :: program_calling
  integer :: i , j , ii , jj

  cmd_line_entry%switch = dummy(1:2)
  write(fileunit_tmp, form_61) , dummy
  if (.not.if_switch_program(program_calling, cmd_line_entry%switch)) then
    write ( fileunit_tmp , form_62 ) "this switch is IGNORED by program "//program_calling
    return
  endif


  dummy=dummy(3:)

  cmd_line_entry%fields = count_separator(dummy) + 1
  allocate(cmd_line_entry%field (cmd_line_entry%fields) )

  ! if ":" separator is present in command line allocate
  ! additional array for fieldnames
  if (count_separator(dummy, ":" ).ge.1) then
    allocate(cmd_line_entry%fieldnames (cmd_line_entry%fields) )
  endif
  do i = 1 , cmd_line_entry%fields
    j = index(dummy, separator)
    cmd_line_entry%field(i) = dummy(1:j-1)
    if (i.eq.cmd_line_entry%fields) cmd_line_entry%field(i)=dummy
    dummy=dummy(j+1:)

    ! separate field and fieldnames
    if ( index(cmd_line_entry%field(i),":").ne.0 ) then
      dummy2 = trim(cmd_line_entry%field(i))//":"
      allocate ( cmd_line_entry%fieldnames(i)%names(count_separator(dummy2,":") - 1 ))
      do ii = 1, size(cmd_line_entry%fieldnames(i)%names)+1
        jj = index(dummy2, ":")
        if (ii.eq.1) then
          cmd_line_entry%field(i) = dummy2(1:jj-1)
        else
          cmd_line_entry%fieldnames(i)%names(ii-1) = dummy2(1:jj-1)
        endif
        dummy2 = dummy2(jj+1:)
      enddo
    endif
  enddo
  call parse_option(cmd_line_entry , program_calling = program_calling)
end subroutine

subroutine get_model_info ( model , cmd_line_entry , field)
  type(cmd_line),intent(in):: cmd_line_entry
  type(file),intent(inout):: model
  integer :: field , i

  model%name = trim(cmd_line_entry%field(field))
  if (model%name.eq."") return
  if ( file_exists(model%name) ) then
    write (fileunit_tmp , form_62) , trim(model_names(field) )
    write(fileunit_tmp, form_63), trim(model%name)

    do i =1 , size (model%names)
      if (size(cmd_line_entry%fieldnames).gt.0) then
        if (i.le.size (cmd_line_entry%fieldnames(field)%names) &
          .and. cmd_line_entry%fieldnames(field)%names(i).ne."" &
          ) then
          model%names(i) = cmd_line_entry%fieldnames(field)%names(i)
        endif
      endif
      write(fileunit_tmp, form_63, advance="no") , trim( model%names(i))
    enddo
    model%if=.true.
    write(fileunit_tmp, form_63)
  elseif(is_numeric(model%name)) then
    model%if_constant_value=.true.
    read (model%name , * ) model%constant_value
    write (fileunit_tmp , form_62) , trim(model_names(field) )
    write(fileunit_tmp, form_63), 'constant value was set: ' , model%constant_value
    model%if_constant_value=.true.
  else
    write (fileunit_tmp , form_63 ) "no (correct) model in field: ", field
  endif
end subroutine


! ==============================================================================
!> This subroutine checks if given limits for model are proper
! ==============================================================================
subroutine parse_gmt_like_boundaries ( cmd_line_entry )
  implicit none
  real(sp) :: limits (4) , resolution (2) =[1,1]
  real(sp) :: range_lon , range_lat , lat , lon
  character(10) :: dummy
  integer :: i , ii
  type (cmd_line) , intent (in) :: cmd_line_entry
  character(:) ,allocatable :: text
  integer :: n_lon , n_lat

  text = cmd_line_entry%field(1)

  do i=1,3
    if ( is_numeric(text(1:index(text, "/"))) ) then
      read ( text(1:index(text, "/")) , * )  limits(i)
    else
      if (text.eq."Rg" ) then
        limits=[0. , 360. , -90 , 90. ]
      endif
    endif
    text=text(index(text,"/")+1:)
  enddo

  if ( is_numeric(text(1:)) ) then
    read ( text(1:) , * )  limits(4)
  else
    call print_warning("boundaries")
  endif

  do i = 1 ,2
    if (limits(i).lt. -180. .or. limits(i).gt.360. ) then
      call print_warning("boundaries")
    else
      if (limits(i).lt.0.) limits(i)=limits(i)+360.
    endif
  enddo
  do i =3,4
    if (limits(i).lt. -90. .or. limits(i).gt.90. ) then
      call print_warning("boundaries")
    endif
  enddo
  if (limits(3).gt.limits(4)) then
    call print_warning("boundaries")
  endif

  if (is_numeric(cmd_line_entry%field(2) ) ) then
    read (cmd_line_entry%field(2) , * ) resolution(1)
    resolution(2) = resolution(1)
  endif
  if (is_numeric(cmd_line_entry%field(3) ) ) then
    read (cmd_line_entry%field(3) , * ) resolution(2)
  endif

  range_lon=limits(2) - limits(1)
  if (range_lon.lt.0) range_lon = range_lon + 360.
  range_lat=limits(4) - limits(3)
  n_lon = floor( range_lon / resolution(1)) + 1
  n_lat = floor( range_lat / resolution(2)) + 1
  allocate (sites( n_lon * n_lat ) )

  do i = 1 , n_lon
    lon = limits(1) + (i-1) * resolution(1)
    if (lon.ge.360.) lon = lon - 360.
    do ii = 1 , n_lat
      lat = limits(3) + (ii-1) * resolution(2)
      sites( (i-1) * n_lat + ii  )%lon = lon
      sites( (i-1) * n_lat + ii  )%lat = lat
    enddo
  enddo

end subroutine

! ==============================================================================
!> Read site list from file
!!
!! checks for arguments and put it into array \c sites
```

```
! ============================================================================
subroutine read_site_file ( file_name )
  character(len=*) , intent(in) ::  file_name
  integer :: io_status , i , good_lines = 0 , number_of_lines = 0 , nloop
  character(len=255) ,dimension(4)  :: dummy
  character(len=255) :: line_of_file
  type(site_data) :: aux



    open ( newunit = fileunit_site , file = file_name, &
            iostat = io_status ,status = "old" , action="read" )

    ! two loops, first count good lines and print rejected
    ! second allocate array of sites and read coordinates into it
      nloops: do nloop = 1, 2
      if (nloop.eq.2) allocate(sites(good_lines))
      if (number_of_lines.ne.good_lines) then
        call print_warning("site_file_format")
      endif
      good_lines=0
      line_loop:do
        read ( fileunit_site , '(a)' , iostat = io_status ) line_of_file
        if ( io_status == iostat_end)  exit  line_loop
        number_of_lines = number_of_lines + 1
        ! we need at least 3 parameter for site (name , B , L )
        if (ntokens(line_of_file).ge.3) then
          ! but no more than 4 parameters (name , B , L, H)
          if (ntokens(line_of_file).gt.4) then
            read ( line_of_file , * ) dummy(1:4)
          else
            read ( line_of_file , * ) dummy(1:3)
            ! if site height was not given we set it to zero
            dummy(4)="0."
          endif
        endif
        ! check the values given
        if(    is_numeric(trim(dummy(2)))   &
          .and.is_numeric(trim(dummy(3)))   &
          .and.is_numeric(trim(dummy(4)))   &
          .and.ntokens(line_of_file).ge.3 ) then

          aux%name= trim(dummy(1))
          read( dummy(2),*) aux%lat
          read(dummy(3),*) aux%lon
          read(dummy(4),*) aux%height

!          ! todo
          if (aux%lat.ge.-90 .and. aux%lat.le.90) then
            if (aux%lon.ge.-180 .and. aux%lon.le.360) then
              good_lines=good_lines+1
              if (nloop.eq.2) then
                sites(good_lines)%name= trim(dummy(1))
                read(dummy(2),*) sites(good_lines)%lat
                read(dummy(3),*) sites(good_lines)%lon
                read(dummy(4),*) sites(good_lines)%height
              endif
            else
              if (nloop.eq.2) write ( fileunit_tmp, form_63) "rejecting (lon limits):" , line_of_file
            endif
          else
            if (nloop.eq.2) write ( fileunit_tmp, form_63) "rejecting (lat limits):" , line_of_file
          endif

        else
          ! print it only once
          if (nloop.eq.2) then
              write ( fileunit_tmp, form_63) "rejecting (args):      " , line_of_file
          endif
        endif
      enddo line_loop
      if (nloop.eq.1) rewind(fileunit_site)
    enddo nloops

  ! if longitude <-180, 180> change to <0,360) domain
  do i =1 , size (sites)
    if (sites(i)%lon.lt.0) sites(i)%lon= sites(i)%lon + 360.
    if (sites(i)%lon.eq.360) sites(i)%lon= 0.
  enddo
end subroutine


! ============================================================================
!> Parse date given as 20110503020103  to yy mm dd hh mm ss and mjd
```

```
!!
!! \warning decimal seconds are not allowed
! ==============================================================================
subroutine parse_dates (cmd_line_entry )
  type(cmd_line) cmd_line_entry
  integer , dimension(6) :: start , stop
  real (sp) :: step =6. ! step in hours
  integer :: i

  call string2date(cmd_line_entry%field(1), start)
  write (fileunit_tmp , form_62) "start date:" , start
  if (cmd_line_entry%field(2).eq."".or.cmd_line_entry%fields.le.1) then
    stop = start
  else
    call string2date(cmd_line_entry%field(2), stop )
    write (fileunit_tmp , form_62) "stop date: " , stop
  endif
  if (is_numeric(cmd_line_entry%field(3)).and.cmd_line_entry%fields
.ge.3) then
    read(cmd_line_entry%field(3),*) step
    write (fileunit_tmp , form_62) "interval [h]:" , step
  endif

  allocate (dates( int( ( mjd(stop) - mjd(start) ) / step * 24. + 1 ) ))
  do i = 1 , size(dates)
    dates(i)%mjd = mjd(start) + ( i -1 ) * step / 24.
    call invmjd( dates(i)%mjd , dates(i)%date)
  enddo
end subroutine

subroutine string2date ( string , date )
  integer , dimension(6) ,intent(out):: date
  character (*) , intent(in) :: string
  integer :: start_char , end_char , j

  ! this allow to specify !st Jan of year simple as -Dyyyy
  date = [2000 , 1 , 1 , 0 ,0 ,0]

  start_char = 1
  do j = 1 , 6
    if (j.eq.1) then
      end_char=start_char+3
    else
      end_char=start_char+1
    endif
      if (is_numeric(string(start_char : end_char) )) then
      read(string(start_char : end_char),*) date(j)
    endif
    start_char=end_char+1
  enddo

end subroutine


!subroutine sprawdzdate(mjd)
!  real:: mjd
!    if
 (mjd.gt.jd(data_uruchomienia(1),data_uruchomienia(2),data_uruchomienia(3),data_uruchomienia(4),data_uruchomienia(5),dat
!      write (*,'(4x,a)') "Data późniejsza niż dzisiaj. KOŃCZĘ!"
!      call exit
!    elseif (mjd.lt.jd(1980,1,1,0,0,0)) then
!      write (*,'(4x,a)') "Data wcześniejsza niż 1980-01-01. KOŃCZĘ!"
!      call exit
!    endif
!    if (.not.log_E) then
!      data_koniec=data_poczatek
!      mjd_koniec=mjd_poczatek
!    endif
!    if (mjd_koniec.lt.mjd_poczatek) then
!      write (*,*) "Data końcowa większa od początkowej. KOŃCZĘ!"
!      write (*,form_64) "Data końcowa większa od początkowej. KOŃCZĘ!"
!    endif
!end subroutine

! ==============================================================================
!> Auxiliary function
!!
!! check if argument given as string is valid number
!! Taken from www
!! \todo Add source name
! ==============================================================================
function is_numeric(string)
  implicit none
  character(len=*), intent(in) :: string
  logical :: is_numeric
  real :: x
  integer :: e
  read(string,*,iostat=e) x
  is_numeric = e == 0
end function


! ==============================================================================
!> Check if file exists , return logical
! ==============================================================================
logical function file_exists(string)
  implicit none
  character(len=*), intent(in) :: string
  logical :: exists
  real :: x
  integer :: e
  if (string =="") then
    file_exists=.false.
    return
  endif
  inquire(file=string, exist=exists)
  file_exists=exists
end function


! ==============================================================================
!> degree -> radian
! ==============================================================================
real(dp) function d2r (degree)
  real(dp) , intent (in) :: degree
  d2r= pi / 180.0 * degree
end function

! ==============================================================================
!> radian -> degree
! ==============================================================================
real(dp) function r2d ( radian )
  real(dp), intent (in) :: radian
  r2d= 180. / pi * radian
end function

! ==============================================================================
!> Print version of program depending on program calling
! ==============================================================================
subroutine print_version (program_calling)
  implicit none
  character(*) :: program_calling
  write(log%unit , form_header )
  if (program_calling.eq."grat" ) then
    write(log%unit,form_inheader ) , 'grat v. 1.0'
    write(log%unit,form_inheader ) , 'Last modification: 20120910'
  elseif(program_calling.eq."polygon_check") then
    write(log%unit,form_inheader ) , 'polygon_check v. 1.0'
    write(log%unit,form_inheader ) , 'Last modification: 20120910'
    write(log%unit,form_inheader ) , ''
    write(log%unit,form_inheader ) , 'Check if given point (given with -S)'
    write(log%unit,form_inheader ) ,
'is included or excluded usig &      specific polygon file'
  elseif(program_calling.eq."value_check") then
    write(log%unit,form_inheader ) , 'value_check v. 1.0'
    write(log%unit,form_inheader ) , 'Last modification: 20120910'
    write(log%unit,form_inheader ) , ''
    write(log%unit,form_inheader ) , 'Check data value for given point (given with -S)'
  endif
  write(log%unit,form_inheader ) , ''
  write(log%unit,form_inheader ) , 'Marcin Rajner'
  write(log%unit,form_inheader ) , 'Politechnika Warszawska'
  write(log%unit,form_inheader ) , '(Warsaw University of Technology)'
  write(log%unit , form_header )
end subroutine

! ==============================================================================
!> Print settings
! ==============================================================================
subroutine print_settings ( program_calling )
  implicit none
  logical :: exists
  character (len=255):: dummy
  integer :: io_status , j
  character(*) :: program_calling

  call print_version( program_calling = program_calling)
  call date_and_time( values = execution_date )
  write(log%unit,
'("Program started:",1x,i4,2("-",i2.2), & 1x,i2.2,2(":",i2.2),1x,"(",SP,i3.2,"h UTC")")',          &
    execution_date(1:3),execution_date(5:7),execution_date(4)/60
  write(log%unit, form_separator)
```

```fortran

  inquire(fileunit_tmp, exist=exists)
  if (exists) then
    write (log%unit, form_60 ) 'Summary of command line arguments'

    !----------------------------------------------------
    ! Cmd line summary (from scratch file)
    !----------------------------------------------------
    rewind(fileunit_tmp)
    do
      read(fileunit_tmp,'(a80)', iostat = io_status ) dummy
      if ( io_status == iostat_end)  exit
      write (log%unit, '(a80)') dummy
    enddo

    !----------------------------------------------------
    ! Site summary
    !----------------------------------------------------
    write(log%unit, form_separator)
    write(log%unit, form_60 ) "Processing:", size(sites), "sites"
    write(log%unit, '(2x,a,t16,3a15)') "Name" , "lat [deg]" , "lon [deg]" ,"H [m]"
    do j = 1,size(sites)
      write(log%unit, '(2x,a,t16,3f15.4)') &
        sites(j)%name, sites(j)%lat, sites(j)%lon , sites(j)%height
      if (j.eq.10) exit
    enddo
    if (size(sites).gt.10) write(log%unit , form_62 ) &
      "and", size(sites)-10, "more"

    !----------------------------------------------------
    ! Computation method summary
    !----------------------------------------------------
    if (program_calling.eq."grat" ) then
      write(log%unit, form_separator)
      write(log%unit, form_60 ) "Method used:", method
    endif

    write(log%unit, form_separator)
    write(log%unit, form_60 ) "Interpolation data:", &
    interpolation_names(model%interpolation)(1:7)



  endif
end subroutine

subroutine print_help (program_calling)
  implicit none
  character(*) :: program_calling
  type help_fields
    character(2) :: switch
    character(255), allocatable,dimension(:) :: description
    character(255):: example=""
  end type
  ! todo change array size
  type(help_fields) help(9)
  integer :: i , j

  help(1)%switch =      "-h"
  allocate(help(1)%description(1))
  help(1)%description(1) = "print help"

  help(2)%switch =      "-v"
!  help(2)%description = "print version"

  help(3)%switch = "-R"
!  help(3)%description = "set limits for regular grid as model input"
!  help(3)%example = "-R0/20/30/40 or -Rg (=R0/360/-90/90) same as GMT"

  help(4)%switch = "-L"
  allocate(help(4)%description(4))
  help(4)%description(1) = "prints additional information"
  help(4)%description(2) = "syntax: -L[filename]"
!    help(4)%example = "-L[filename]"
!  help(4)%example = "todo"//'/'//"fdf"




  write(log%unit , form_60) , 'Summary of available options for program '//program_calling
  do i = 1 , size (help)
  if(if_switch_program(program_calling , help(i)%switch ))then
    write(log%unit , form_61) ,help(i)%switch
    if(allocated(help(i)%description)) then
```

```
      do j = 1 , size(help(i)%description)
        write (log%unit , form_62 ) help(i)%description(j)
      enddo
    endif
!    if (.not.help(i)%example(1:1).eq."") then
!      write(log%unit , form_62) , trim(help(i)%example)
!    endif
  endif
  enddo

end subroutine

subroutine print_warning (  warn , unit)
 implicit none
 character (len=*)  :: warn
 integer , optional :: unit
 integer :: def_unit

 def_unit=fileunit_tmp
 if (present(unit) ) def_unit=unit

 if (warn .eq. "site_file_format") then
  write(def_unit, form_63) "Some records were rejected"
  write(def_unit, form_63) "you should specify for each line at least 3[4] parameters in free format:"
  write(def_unit, form_63) "name lat lon [H=0] (skipped)"
elseif(warn .eq. "boundaries") then
  write(def_unit, form_62) "something wrong with boundaries. IGNORED"
elseif(warn .eq. "site") then
  write(def_unit, form_62) "something wrong with -S specification. IGNORED"
elseif(warn .eq. "repeated") then
  write(def_unit, form_62) "reapeted specification. IGNORED"
elseif(warn .eq. "dates") then
  write(def_unit, form_62) "something wrong with date format -D. IGNORED"
endif
end subroutine


! =============================================================================
!> Counts number of properly specified models
! =============================================================================
integer function nmodels (model)
  type(file) , allocatable, dimension (:) :: model
  integer :: i

  nmodels = 0

  do i = 1 , size (model)
    if (model(i)%if) nmodels =nmodels + 1
    if (model(i)%if_constant_value) nmodels =nmodels + 1
  enddo
end function

end module get_cmd_line
```

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

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


  implicit none
  real(sp) :: x , y , z , lat ,lon ,val(0:100) !tmp variables
  integer :: i , j , ii, iii

  !> program starts here with time stamp
  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 polygons
  do i =1 , 2
   call read_polygon(polygons(i))
  enddo

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

  refpres%name="/home/mrajner/src/grat/data/refpres/vienna_p0.grd"
  call read_netcdf(refpres)

  allocate (results(size(sites)*max(size(dates),1)))
  iii=0
  do j = 1 , max(size (dates),1)
    if(size(dates).gt.0)  write(output%unit, '(i4,5(i2.2))', advance ="no") dates(j)%date

    do ii = 1 , min(2,size(model))
      if (model(ii)%if) call get_variable( model(ii) , date = dates(j)%date)
    enddo



!todo
    do i = 1 , size(sites)
      write(output%unit, '(2f15.5f)', advance ="no") sites(i)%lat ,sites(i)%lon
      iii=iii+1
      call convolve(sites(i) , green , results(iii), denserdist = denser(1) , denseraz = denser(2))
      write (output%unit,'(15f13.5)') , results(iii)%e ,results(iii)%n  , results(iii)%dt , results(iii)%dh, results(iii)%dz
    enddo
  enddo

! print '(15f13.5)',  results(maxloc (results%e))%e - results(minloc (results%e))%e       ,&
!           results(maxloc (results%n))%n - results(minloc (results%n))%n ,&
!           results(maxloc (results%dh))%dh - results(minloc (results%dh))%dh ,&
!           results(maxloc (results%dz))%dz - results(minloc (results%dz))%dz
```

```
 ,&
!           results(maxloc (results%dt))%dt - results(minloc (results%dt))%dt


  call cpu_time(cpu_finish)
  write(log%unit, '(/,"Execution time:",1x,f16.9," seconds")') cpu_finish - cpu_start
  write(log%unit, form_separator)
! hellow ro
  print * , model(6)%level
  print *
  lat =00
  lon = 00
  call get_value(model(7),lat,lon, val(0))
  do i =1, size(model(6)%level)
  call get_value(model(6),lat,lon, val(i), level = i, method=2)
  enddo
  print  '(30f10.2)', lat , lon , (val(i), i=0,size(model(6)%level))
  print  '(30f10.2)' , lat , lon , (geop2geom(val(i)/1000)*1000., i=0, size(model(6)%level))

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

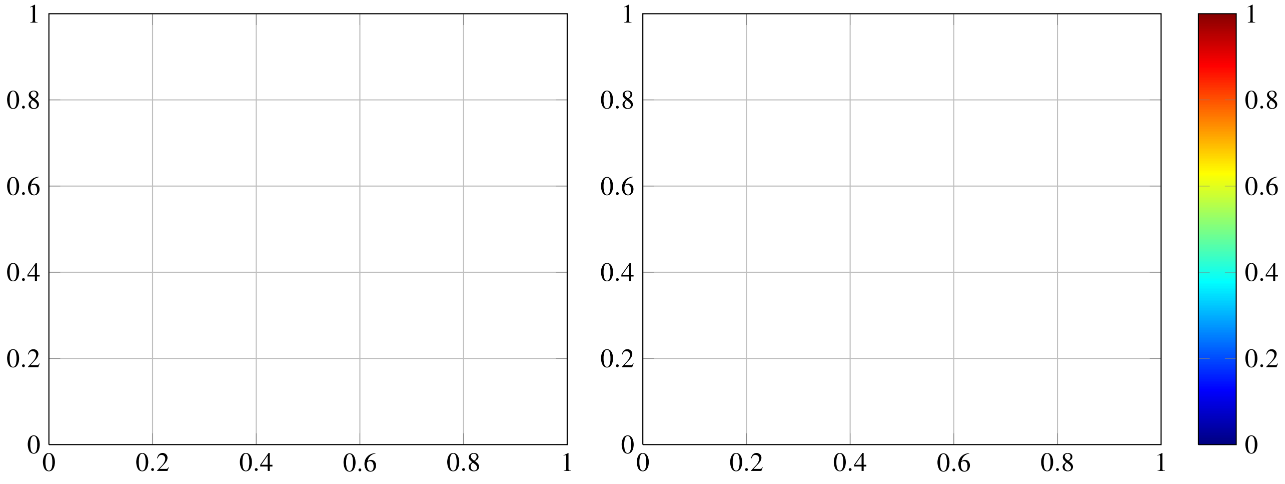

Figure B.1: Interpoloation