

grat<sup>v. 1.0</sup> Manual



# Contents

<b>1</b>	<b>Grat overview</b>	<b>1</b>
1.1	Purpose . . . . .	1
<b>2</b>	<b>Todo List</b>	<b>3</b>
<b>3</b>	<b>Data Type Index</b>	<b>5</b>
3.1	Data Types List . . . . .	5
<b>4</b>	<b>File Index</b>	<b>7</b>
4.1	File List . . . . .	7
<b>5</b>	<b>Data Type Documentation</b>	<b>9</b>
5.1	get_cmd_line::additional_info Type Reference . . . . .	9
5.1.1	Detailed Description . . . . .	9
5.2	aggf Module Reference . . . . .	9
5.2.1	Detailed Description . . . . .	10
5.2.2	Member Function/Subroutine Documentation . . . . .	10
5.2.2.1	bouger . . . . .	10
5.2.2.2	compute_aggf . . . . .	10
5.2.2.3	compute_aggfdt . . . . .	11
5.2.2.4	gn_thin_layer . . . . .	11
5.2.2.5	read_tabulated_green . . . . .	11
5.2.2.6	simple_def . . . . .	11
5.2.2.7	size_ntimes_denser . . . . .	11
5.2.2.8	standard_density . . . . .	11
5.2.2.9	standard_gravity . . . . .	12
5.2.2.10	standard_pressure . . . . .	12
5.2.2.11	standard_temperature . . . . .	12
5.3	get_cmd_line::cmd_line Type Reference . . . . .	13
5.3.1	Detailed Description . . . . .	13
5.4	constants Module Reference . . . . .	13
5.4.1	Detailed Description . . . . .	14
5.4.2	Member Function/Subroutine Documentation . . . . .	14

5.4.2.1	ispline	14
5.4.2.2	jd	14
5.4.2.3	spline	15
5.4.2.4	spline_interpolation	15
5.5	get_cmd_line::dateandmjd Type Reference	15
5.5.1	Detailed Description	15
5.6	get_cmd_line::file Type Reference	15
5.6.1	Detailed Description	16
5.7	get_cmd_line Module Reference	16
5.7.1	Detailed Description	18
5.7.2	Member Function/Subroutine Documentation	18
5.7.2.1	intro	18
5.7.2.2	is_numeric	18
5.7.2.3	parse_dates	19
5.7.2.4	read_site_file	19
5.8	get_cmd_line::green_functions Type Reference	19
5.8.1	Detailed Description	19
5.9	mapcon_util Module Reference	19
5.9.1	Detailed Description	19
5.10	mod_data Module Reference	20
5.10.1	Detailed Description	20
5.10.2	Member Function/Subroutine Documentation	20
5.10.2.1	check	20
5.10.2.2	get_value	20
5.10.2.3	put_grd	21
5.10.2.4	unpack_netcdf	21
5.11	mod_green Module Reference	21
5.11.1	Detailed Description	21
5.12	mod_polygon Module Reference	21
5.12.1	Detailed Description	21
5.12.2	Member Function/Subroutine Documentation	22
5.12.2.1	chkgon	22
5.12.2.2	ncross	22
5.12.2.3	read_polygon	22
5.13	get_cmd_line::polygon_data Type Reference	22
5.13.1	Detailed Description	22
5.14	get_cmd_line::polygon_info Type Reference	23
5.14.1	Detailed Description	23
5.15	get_cmd_line::site_data Type Reference	23
5.15.1	Detailed Description	23

<b>6 File Documentation</b>	<b>25</b>
6.1 /home/mrajner/src/grat/polygon/polygon_map.sh File Reference . . . . .	25
6.1.1 Detailed Description . . . . .	25
6.1.2 Variable Documentation . . . . .	26
6.1.2.1 last . . . . .	26
6.2 polygon_map.sh . . . . .	26
6.3 /home/mrajner/src/grat/src/aggf.f90 File Reference . . . . .	28
6.3.1 Detailed Description . . . . .	28
6.4 aggf.f90 . . . . .	28
6.5 /home/mrajner/src/grat/src/constants.f90 File Reference . . . . .	34
6.5.1 Detailed Description . . . . .	34
6.6 constants.f90 . . . . .	34
6.7 /home/mrajner/src/grat/src/example_aggf.f90 File Reference . . . . .	38
6.7.1 Detailed Description . . . . .	39
6.7.2 Function/Subroutine Documentation . . . . .	39
6.7.2.1 aux_heights . . . . .	39
6.7.2.2 compare_fels_profiles . . . . .	39
6.7.2.3 simple_atmospheric_model . . . . .	40
6.7.2.4 standard1976 . . . . .	40
6.8 example_aggf.f90 . . . . .	40
6.9 /home/mrajner/src/grat/src/grat.f90 File Reference . . . . .	45
6.9.1 Detailed Description . . . . .	46
6.10 grat.f90 . . . . .	46
<b>A Polygon</b>	<b>49</b>
<b>B Interpolation</b>	<b>51</b>



# Chapter 1

## Grat overview

### 1.1 Purpose

This program was created to make computation of atmospheric gravity correction more easy.

#### Version

v. 1.0

#### Date

2012-12-12

#### Author

Marcin Rajner  
Politechnika Warszawska  
(Warsaw University of Technology)

#### Warning

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





## Chapter 2

## Todo List

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

give source

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

mjd!

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

give source

**Subprogram `get_cmd_line::is_numeric`** (string)

Add source name



## Chapter 3

# Data Type Index

### 3.1 Data Types List

Here are the data types with brief descriptions:

get_cmd_line::additional_info . . . . .	9
aggf . . . . .	9
get_cmd_line::cmd_line . . . . .	13
constants . . . . .	13
get_cmd_line::dateandmjd . . . . .	15
get_cmd_line::file . . . . .	15
get_cmd_line This module sets the initial values for parameters reads from command line and gives help . . .	16
get_cmd_line::green_functions . . . . .	19
mapcon_util . . . . .	19
mod_data This modele gives routines to read, and write data . . . . .	20
mod_green . . . . .	21
mod_polygon . . . . .	21
get_cmd_line::polygon_data . . . . .	22
get_cmd_line::polygon_info . . . . .	23
get_cmd_line::site_data . . . . .	23



## Chapter 4

# File Index

### 4.1 File List

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

/home/mrajner/src/grat/data/ispd/ <b>download.sh</b> . . . . .	??
/home/mrajner/src/grat/data/ispd/ <b>extract_data.f90</b> . . . . .	??
/home/mrajner/src/grat/data/ispd/ <b>location_map.sh</b> . . . . .	??
/home/mrajner/src/grat/data/landsea/ <b>landsea.sh</b> . . . . .	??
/home/mrajner/src/grat/data/ncep_reanalysis/ <b>download.sh</b> . . . . .	??
<b>interpolation_ilustration.sh</b> . . . . .	??
<b>polygon_ilustration.sh</b> . . . . .	??
/home/mrajner/src/grat/polygon/ <b>baltyk.sh</b> . . . . .	??
/home/mrajner/src/grat/polygon/ <b>polygon_map.sh</b>	
Make map of polygon(s) . . . . .	26
/home/mrajner/src/grat/ <b>aggf.f90</b>	
This module contains utilities for computing Atmospheric Gravity Green Functions . . . . .	28
/home/mrajner/src/grat/ <b>constants.f90</b>	
This module define some constant values used . . . . .	34
/home/mrajner/src/grat/src/ <b>example_aggf.f90</b>	
This program shows some example of using AGGF module . . . . .	40
/home/mrajner/src/grat/src/ <b>get_cmd_line.f90</b> . . . . .	??
/home/mrajner/src/grat/src/ <b>grat.f90</b> . . . . .	46
/home/mrajner/src/grat/src/ <b>ignewt.f90</b> . . . . .	??
/home/mrajner/src/grat/src/ <b>mapa.sh</b> . . . . .	??
/home/mrajner/src/grat/src/ <b>mapa_indsea.f90</b> . . . . .	??
/home/mrajner/src/grat/src/ <b>mapcon_util.f90</b> . . . . .	??
/home/mrajner/src/grat/src/ <b>mod_data.f90</b> . . . . .	??
/home/mrajner/src/grat/src/ <b>mod_green.f90</b> . . . . .	??
/home/mrajner/src/grat/src/ <b>mod_polygon.f90</b> . . . . .	??
/home/mrajner/src/grat/src/ <b>obsolte.f90</b> . . . . .	??
/home/mrajner/src/grat/src/ <b>obsoltes.f90</b> . . . . .	??
/home/mrajner/src/grat/src/ <b>polygon_check.f90</b> . . . . .	??
/home/mrajner/src/grat/src/ <b>press2spotl.f90</b> . . . . .	??
/home/mrajner/src/grat/src/ <b>symuluj_dane.f90</b> . . . . .	??
/home/mrajner/src/grat/src/ <b>value_check.f90</b> . . . . .	??



## Chapter 5

# Data Type Documentation

### 5.1 `get_cmd_line::additional_info` Type Reference

#### Public Attributes

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

#### 5.1.1 Detailed Description

Definition at line 57 of file `get_cmd_line.f90`.

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

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

### 5.2 `aggf` Module Reference

#### Public Member Functions

- subroutine `compute_aggfdt` (`psi`, `aggfdt`, `delta_`, `aggf`)  
*Compute first derivative of AGGF with respect to temperature for specific angular distance (psi)*
- subroutine `read_tabulated_green` (`table`, `author`)  
*Wczytuje tablice danych AGGF.*
- subroutine `compute_aggf` (`psi`, `aggf_val`, `hmin`, `hmax`, `dh`, `if_normalization`, `t_zero`, `h`, `first_derivative_h`, `first_derivative_z`, `fels_type`)  
*This subroutine computes the value of atmospheric gravity green functions (AGGF) on the basis of spherical distance (psi)*
- subroutine `standard_density` (`height`, `rho`, `t_zero`, `fels_type`)  
*first derivative (respective to station height) micro Gal height / km*
- subroutine `standard_pressure` (`height`, `pressure`, `p_zero`, `t_zero`, `if_simplified`, `fels_type`, `inverted`)  
*Computes pressure [hPa] for specific height.*
- subroutine `transfer_pressure` (`height1`, `height2`, `pressure1`, `pressure2`, `temperature`, `polish_meteo`)
- subroutine `standard_gravity` (`height`, `g`)  
*Compute gravity acceleration of the Earth for the specific height using formula.*
- real(sp) function `geometric_height` (`geopotential_height`)  
*Compute geometric height from geopotential heights.*
- subroutine `surface_temperature` (`height`, `temperature1`, `temperature2`, `fels_type`, `tolerance`)

- Iterative computation of surface temp. from given height using bisection method.*
- subroutine **standard\_temperature** (height, temperature, t\_zero, fels\_type)  
*Compute standard temperature [K] for specific height [km].*
- real function **gn\_thin\_layer** (psi)  
*Compute AGGF GN for thin layer.*
- integer function **size\_ntimes\_denser** (size\_original, ndenser)  
*returns numbers of arguments for n times denser size*
- real(dp) function **bouger** (R\_opt)  
*Bouger plate computation.*
- real(dp) function **simple\_def** (R)  
*Bouger plate computation see eq. page 288.*

### 5.2.1 Detailed Description

Definition at line 9 of file **aggf.f90**.

### 5.2.2 Member Function/Subroutine Documentation

#### 5.2.2.1 real(dp) function aggf::bouger ( real(dp), optional R\_opt )

Bouger plate computation.

##### Parameters

<i>r_opt</i>	height of point above the cylinder
--------------	------------------------------------

Definition at line 469 of file **aggf.f90**.

#### 5.2.2.2 subroutine aggf::compute\_aggf ( real(dp), intent(in) psi, real(dp), intent(out) aggf\_val, real(dp), intent(in), optional hmin, real(dp), intent(in), optional hmax, real(dp), intent(in), optional dh, logical, intent(in), optional if\_normalization, real(dp), intent(in), optional t\_zero, real(dp), intent(in), optional h, logical, intent(in), optional first\_derivative\_h, logical, intent(in), optional first\_derivative\_z, character (len=\*), intent(in), optional fels\_type )

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

##### Parameters

in	<i>psi</i>	spherical distance from site [degree]
in	<i>h</i>	station height [km] (default=0)

##### Parameters

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

Definition at line 110 of file **aggf.f90**.



5.2.2.3 subroutine `aggf::compute_aggfdt` ( `real(dp)`, `intent(in)` *psi*, `real(dp)`, `intent(out)` *aggfdt*, `real(dp)`, `intent(in)`, optional *delta\_*, logical, `intent(in)`, optional *aggf* )

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

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

#### Warning

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

Definition at line 27 of file `aggf.f90`.

5.2.2.4 real function `aggf::gn_thin_layer` ( `real(dp)`, `intent(in)` *psi* )

Compute AGGF GN for thin layer.

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

Definition at line 445 of file `aggf.f90`.

5.2.2.5 subroutine `aggf::read_tabulated_green` ( `real(dp)`, `dimension(:, :)`, `intent(inout)`, allocatable *table*, character ( len = \* ), `intent(in)`, optional *author* )

Wczytuje tablice danych AGGF.

- merriam [Merriam \[1992\]](#)
- huang [Huang et al. \[2005\]](#)
- rajner [Rajner \[2013\]](#)

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

Definition at line 66 of file `aggf.f90`.

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

Bouger plate computation see eq. page 288.

[Warburton and Goodkind \[1977\]](#)

Definition at line 491 of file `aggf.f90`.

5.2.2.7 integer function `aggf::size_ntimes_denser` ( integer, `intent(in)` *size\_original*, integer, `intent(in)` *ndenser* )

returns numbers of arguments for n times denser size

i.e. \* \* \* \* -> \* . . \* . . \* . . \* (3 times denser)

Definition at line 460 of file `aggf.f90`.

5.2.2.8 subroutine `aggf::standard_density` ( `real(dp)`, `intent(in)` *height*, `real(dp)`, `intent(out)` *rho*, `real(dp)`, `intent(in)`, optional *t\_zero*, character(len = 22), optional *fels\_type* )

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

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

using formulae 12 in [Huang et al. \[2005\]](#)

#### Parameters

in	<i>height</i>	height [km]
in	<i>t_zero</i>	if this parameter is given

Definition at line 194 of file [aggf.f90](#).

#### 5.2.2.9 subroutine aggf::standard\_gravity ( real(dp), intent(in) *height*, real(dp), intent(out) *g* )

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

see [Comitee on extension of the Standard Atmosphere \[1976\]](#)

Definition at line 291 of file [aggf.f90](#).

#### 5.2.2.10 subroutine aggf::standard\_pressure ( real(dp), intent(in) *height*, real(dp), intent(out) *pressure*, real(dp), intent(in), optional *p\_zero*, real(dp), intent(in), optional *t\_zero*, logical, intent(in), optional *if\_simplified*, character(len = 22), optional *fels\_type*, logical, intent(in), optional *inverted* )

Computes pressure [hPa] for specific height.

See [Comitee on extension of the Standard Atmosphere \[1976\]](#) or [Huang et al. \[2005\]](#) for details. Uses formulae 5 from [Huang et al. \[2005\]](#). Simplified method if optional argument `if_simplified = .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 predefined temperature profiles can be set using optional argument `fels_type` [Fels \[1986\]](#)

#### Parameters

in	<i>fels_type</i>	<ul style="list-style-type: none"> <li>• US standard atmosphere (default)</li> <li>• tropical</li> <li>• subtropical_summer</li> <li>• subtropical_winter</li> <li>• subarctic_summer</li> <li>• subarctic_winter</li> </ul>
----	------------------	--

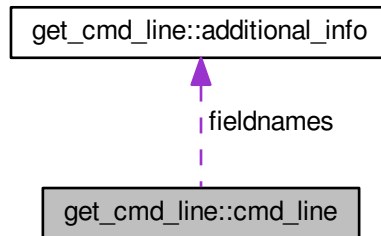
Definition at line 359 of file [aggf.f90](#).

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

- [/home/mrajner/src/grat/src/aggf.f90](#)

## 5.3 get\_cmd\_line::cmd\_line Type Reference

Collaboration diagram for get\_cmd\_line::cmd\_line:



### Public Attributes

- character(2) **switch**
- integer **fields**
- character(len=255), dimension(:), allocatable **field**
- type(**additional\_info**), dimension(:), allocatable **fieldnames**

#### 5.3.1 Detailed Description

Definition at line 60 of file [get\\_cmd\\_line.f90](#).

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

- [/home/mrajner/src/grat/src/get\\_cmd\\_line.f90](#)

## 5.4 constants Module Reference

### Public Member Functions

- subroutine [spline\\_interpolation](#) (x, y, x\_interpolated, y\_interpolated)  
*For given vectors x1, y1 and x2, y2 it gives x2interpolated for x1.*
- subroutine [spline](#) (x, y, b, c, d, n)  
*This subroutine was taken from.*
- real function [ispline](#) (u, x, y, b, c, d, n)  
*This subroutine was taken from.*
- integer function [ntokens](#) (line)  
*taken from ArkM <http://www.tek-tips.com/viewthread.cfm?qid=1688013>*
- subroutine [skip\\_header](#) (unit, comment\_char\_optional)  
*This routine skips the lines with comment chars (default '#') from opened files (unit) to read.*
- real function [jd](#) (year, month, day, hh, mm, ss)  
*downloaded from [http://aa.usno.navy.mil/faq/docs/jd\\_formula.php](http://aa.usno.navy.mil/faq/docs/jd_formula.php)*

- real(dp) function **mjd** (date)
- subroutine **invmj**d (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](http://aa.usno.navy.mil/faq/docs/jd_formula.php)

**Todo** mjd!

Definition at line 253 of file [constants.f90](#).

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

This subroutine was taken from.

**Todo** give source

Definition at line 68 of file `constants.f90`.

5.4.2.4 subroutine `constants::spline_interpolation` ( `real(dp)`, `dimension(:)`, `intent(in)`, allocatable *x*, `real(dp)`, `dimension(:)`, `intent(in)`, allocatable *y*, `real(dp)`, `dimension(:)`, `intent(in)`, allocatable *x\_interpolated*, `real(dp)`, `dimension(:)`, `intent(out)`, allocatable *y\_interpolated* )

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

uses `ispline` and `spline` subroutines

Definition at line 28 of file `constants.f90`.

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

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

## 5.5 `get_cmd_line::dateandmjd` Type Reference

### Public Attributes

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

### 5.5.1 Detailed Description

Definition at line 45 of file `get_cmd_line.f90`.

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

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

## 5.6 `get_cmd_line::file` Type Reference

### Public Attributes

- `character(:)`, allocatable **name**
- `character(len=50)`, `dimension(5)` **names** = [ "z"
- integer **unit** = `output_unit`
- logical **if** = `.false.`
- logical **first\_call** = `.true.`
- `real(sp)`, `dimension(4)` **limits**
- `real(sp)`, `dimension(:)`, allocatable **lat**
- `real(sp)`, `dimension(:)`, allocatable **lon**
- `real(sp)`, `dimension(:)`, allocatable **time**
- `real(sp)`, `dimension(:)`, allocatable **level**
- integer, `dimension(:, :)`, allocatable **date**

- real(sp), dimension(2) **latrange**
- real(sp), dimension(2) **lonrange**
- logical **if\_constant\_value**
- real(sp) **constant\_value**
- real(sp), dimension(:, :, :),  
allocatable **data**  
*4 dimension - lat , lon , level , mjd*
- integer **ncid**
- integer **interpolation** = 1

### 5.6.1 Detailed Description

Definition at line 91 of file [get\\_cmd\\_line.f90](#).

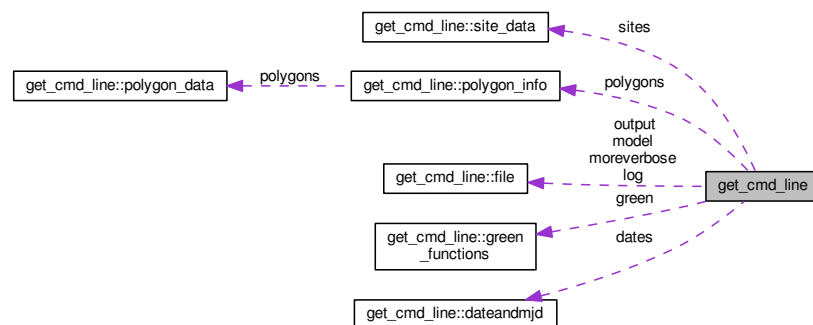
The documentation for this type was generated from the following file:

- [/home/mrajner/src/grat/src/get\\_cmd\\_line.f90](#)

## 5.7 get\_cmd\_line Module Reference

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

Collaboration diagram for `get_cmd_line`:



## Data Types

- type [additional\\_info](#)
- type [cmd\\_line](#)
- type [dateandmjd](#)
- type [file](#)
- type [green\\_functions](#)
- type [polygon\\_data](#)
- type [polygon\\_info](#)
- type [site\\_data](#)

## Public Member Functions

- subroutine **intro** (program\_calling)  
*This subroutine counts the command line arguments.*
- subroutine **if\_minimum\_args** (program\_calling)  
*Check if at least all obligatory command line arguments were given if not print warning.*
- logical function **if\_switch\_program** (program\_calling, switch)  
*This function is true if switch is used by calling program or false if it is not.*
- subroutine **parse\_option** (cmd\_line\_entry, program\_calling)  
*This subroutine counts the command line arguments and parse appropriately.*
- subroutine **parse\_green** (cmd\_line\_entry)  
*This subroutine parse -G option i.e. reads Greens function.*
- integer function **count\_separator** (dummy, separator)  
*Counts occurrence of character (separator, default comma) in string.*
- subroutine **get\_cmd\_line\_entry** (dummy, cmd\_line\_entry, program\_calling)  
*This subroutine fills the fields of command line entry for every input arg.*
- subroutine **get\_model\_info** (model, cmd\_line\_entry, field)
- subroutine **parse\_gmt\_like\_boundaries** (cmd\_line\_entry)  
*This subroutine checks if given limits for model are proper.*
- subroutine **read\_site\_file** (file\_name)  
*Read site list from file.*
- subroutine **parse\_dates** (cmd\_line\_entry)  
*Parse date given as 20110503020103 to yy mm dd hh mm ss and mjd.*
- subroutine **string2date** (string, date)
- logical function **is\_numeric** (string)  
*Auxiliary function.*
- logical function **file\_exists** (string)  
*Check if file exists , return logical.*
- real(dp) function **d2r** (degree)  
*degree -> radian*
- real(dp) function **r2d** (radian)  
*radian -> degree*
- subroutine **print\_version** (program\_calling)  
*Print version of program depending on program calling.*
- subroutine **print\_settings** (program\_calling)  
*Print settings.*
- subroutine **print\_help** (program\_calling)
- subroutine **print\_warning** (warn, unit)
- integer function **nmodels** (model)  
*Counts number of properly specified models.*

## Public Attributes

- type(**green\_functions**),  
dimension(:), allocatable **green**
- type(**polygon\_info**), dimension(2) **polygons**
- real(kind=4) **cpu\_start**
- real(kind=4) **cpu\_finish**  
*for time execution of program*
- type(**dateandmjd**), dimension(:),  
allocatable **dates**

- type(**site\_data**), dimension(:),  
allocatable **sites**
- integer **fileunit\_tmp**  
*unit of scratch file*
- integer, dimension(8) **execution\_date**  
*To give time stamp of execution.*
- character(len=2) **method** = "2D"  
*computation method*
- character(:), allocatable **filename\_site**
- integer **fileunit\_site**
- type(**file**) **log**
- type(**file**) **output**
- type(**file**) **moreverbose**
- type(**file**), dimension(:),  
allocatable **model**
- character(len=40), dimension(5) **model\_names** = ["pressure\_surface"]
- logical **if\_verbose** = .false.  
*whether print all information*
- logical **inverted\_barometer** = .true.  
*whether print all information*
- character(50), dimension(2) **interpolation\_names** = [ "nearest"  
*Logical parameters for easy operation.*
- character(len=255), parameter **form\_header** = '(60("#"))'
- character(len=255), parameter **form\_separator** = '(60("-"))'
- character(len=255), parameter **form\_inheader** = '("("#,1x,a56,1x,("#"))'
- character(len=255), parameter **form\_60** = "(a,100(1x,g0))"
- character(len=255), parameter **form\_61** = "(2x,a,100(1x,g0))"
- character(len=255), parameter **form\_62** = "(4x,a,100(1x,g0))"
- character(len=255), parameter **form\_63** = "(6x,100(x,g0))"
- character(len=255), parameter **form\_64** = "(4x,4x,a,4x,a)"

### 5.7.1 Detailed Description

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

Definition at line 5 of file [get\\_cmd\\_line.f90](#).

### 5.7.2 Member Function/Subroutine Documentation

#### 5.7.2.1 subroutine `get_cmd_line::intro` ( `character(len=*) program_calling` )

This subroutine counts the command line arguments.

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

Definition at line 170 of file [get\\_cmd\\_line.f90](#).

#### 5.7.2.2 logical function `get_cmd_line::is_numeric` ( `character(len=*)`, `intent(in) string` )

Auxiliary function.

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

**Todo** Add source name

Definition at line 779 of file [get\\_cmd\\_line.f90](#).



## 5.7.2.3 subroutine get\_cmd\_line::parse\_dates ( type(cmd\_line) cmd\_line\_entry )

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

## Warning

decimal seconds are not allowed

Definition at line 703 of file [get\\_cmd\\_line.f90](#).

## 5.7.2.4 subroutine get\_cmd\_line::read\_site\_file ( character(len=\*), intent(in) file\_name )

Read site list from file.

checks for arguments and put it into array `sites`

Definition at line 617 of file [get\\_cmd\\_line.f90](#).

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

- /home/mrajner/src/grat/src/get\_cmd\_line.f90

## 5.8 get\_cmd\_line::green\_functions Type Reference

### Public Attributes

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

### 5.8.1 Detailed Description

Definition at line 17 of file [get\\_cmd\\_line.f90](#).

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

- /home/mrajner/src/grat/src/get\_cmd\_line.f90

## 5.9 mapcon\_util Module Reference

### Public Member Functions

- subroutine **mapaascii2mapablv**

### 5.9.1 Detailed Description

Definition at line 1 of file [mapcon\\_util.f90](#).

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

- /home/mrajner/src/grat/src/mapcon\_util.f90

## 5.10 mod\_data Module Reference

This module gives routines to read, and write data.

### Public Member Functions

- subroutine **put\_grd** (model, time, level, filename\_opt)  
*Put netCDF COARDS compliant.*
- subroutine **read\_netcdf** (model)  
*Read netCDF file into memory.*
- subroutine **get\_variable** (model, date)  
*Get values from netCDF file for specified variables.*
- subroutine **nctime2date** (model)  
*Change time in netcdf to dates.*
- subroutine **get\_dimension** (model, i)  
*Get dimension, allocate memory and fill with values.*
- subroutine **unpack\_netcdf** (model)  
*Unpack variable.*
- subroutine **check** (status)  
*Check the return code from netCDF manipulation.*
- subroutine **get\_value** (model, lat, lon, val, method)  
*Returns the value from model file.*
- real function **bilinear** (x, y, aux)
- subroutine **invspt** (alp, del, b, rlong)

### 5.10.1 Detailed Description

This module gives routines to read, and write data.

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

Definition at line 10 of file **mod\_data.f90**.

### 5.10.2 Member Function/Subroutine Documentation

#### 5.10.2.1 subroutine mod\_data::check ( integer, intent(in) status )

Check the return code from netCDF manipulation.

from **net**

Definition at line 214 of file **mod\_data.f90**.

#### 5.10.2.2 subroutine mod\_data::get\_value ( type(file), intent(in) model, real(sp), intent(in) lat, real(sp), intent(in) lon, real(sp), intent(out) val, integer, intent(in), optional method )

Returns the value from model file.

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

Definition at line 231 of file **mod\_data.f90**.

5.10.2.3 subroutine mod\_data::put\_grd ( type (file) *model*, integer *time*, integer *level*, character (\*), intent(in), optional *filename\_opt* )

Put netCDF COARDS compliant.

for GMT drawing

Definition at line 27 of file [mod\\_data.f90](#).

5.10.2.4 subroutine mod\_data::unpack\_netcdf ( type(file) *model* )

Unpack variable.

from [net](#)

Definition at line 196 of file [mod\\_data.f90](#).

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

- /home/mrajner/src/grat/src/mod\_data.f90

## 5.11 mod\_green Module Reference

### Public Member Functions

- subroutine **green\_unification** (green, green\_common, denser)
- subroutine **spher\_area** (distance, ddistance, azstp, area)
- subroutine **spher\_trig** (latin, lonin, distance, azimuth, latout, lonout)
- subroutine **convolve** (site, green, denserdist, denseraz)
- subroutine **convolve\_moreverbose** (site, azimuth, distance)

#### 5.11.1 Detailed Description

Definition at line 1 of file [mod\\_green.f90](#).

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

- /home/mrajner/src/grat/src/mod\_green.f90

## 5.12 mod\_polygon Module Reference

### Public Member Functions

- subroutine [read\\_polygon](#) (polygon)  
*Reads polygon data.*
- subroutine [chkgon](#) (rlong, rlat, polygon, iok)  
*check if point is in closed polygon*
- integer function **if\_inpoly** (x, y, coords)
- integer function [ncross](#) (x1, y1, x2, y2)  
*finds whether the segment from point 1 to point 2 crosses the negative x-axis or goes through the origin (this is the signed crossing number)*

#### 5.12.1 Detailed Description

Definition at line 1 of file [mod\\_polygon.f90](#).

### 5.12.2 Member Function/Subroutine Documentation

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

check if point is in closed polygon

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

Definition at line 82 of file `mod_polygon.f90`.

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

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

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

taken from spotl [Agnew \[1997\]](#) slightly modified

Definition at line 196 of file `mod_polygon.f90`.

5.12.2.3 subroutine `mod_polygon::read_polygon` ( `type(polygon_info)` *polygon* )

Reads polygon data.

inspired by spotl [Agnew \[1997\]](#)

Definition at line 12 of file `mod_polygon.f90`.

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

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

## 5.13 `get_cmd_line::polygon_data` Type Reference

### Public Attributes

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

### 5.13.1 Detailed Description

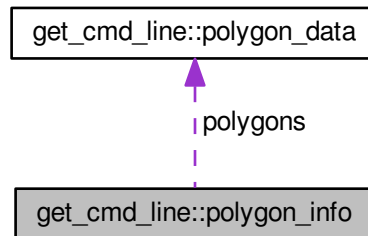
Definition at line 28 of file `get_cmd_line.f90`.

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

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

## 5.14 `get_cmd_line::polygon_info` Type Reference

Collaboration diagram for `get_cmd_line::polygon_info`:



### Public Attributes

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

#### 5.14.1 Detailed Description

Definition at line 33 of file `get_cmd_line.f90`.

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

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

## 5.15 `get_cmd_line::site_data` Type Reference

### Public Attributes

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

#### 5.15.1 Detailed Description

Definition at line 70 of file `get_cmd_line.f90`.

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

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



## Chapter 6

# File Documentation

### 6.1 /home/mrajner/src/grat/polygon/polygon\_map.sh File Reference

Make map of polygon(s)

#### Functions/Subroutines

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

#### Variables

- `__pad0__`  
*If there are no command line argument then stop with error.*
- echo **\$FILE**
- d set **x**
- **DEBUG** = true
- v **VERBOSE** = true
- h **usage**
- **exit**
- R **R** = "-R\$OPTARG"
- o **output** = "\$OPTARG"
- p **POINTSFILE** = "\$OPTARG"
- esac done **if** [-z \$FILE]
- then echo Not enough cmd line **parameters**
- then echo you set the **verbose**
- then echo you set the \$last \$i p **last**
- then get\_R fi **A** = "+" ]
- then **color** = "#" | sed -n -e \$(((\$last+1))

#### 6.1.1 Detailed Description

Make map of polygon(s) This scrips

#### Author

Marcin Rajner

**Date**

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

Definition in file [polygon\\_map.sh](#).

**6.1.2 Variable Documentation****6.1.2.1 then echo you set the \$last \$i p last****Initial value:**

```

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

if [ -z $R ]

```

Definition at line 108 of file [polygon\\_map.sh](#).

**6.2 polygon\_map.sh**

```

00001 #!/bin/bash -
00002 ## \file
00003 ## \brief Make map of polygon(s)
00004 ##
00005 ## This scrips
00006 ## \author Marcin Rajner
00007 ## \date 03.11.2012
00008 ## This scripts need GMT to be installed \cite Wessel98
00009 ## The \c .pdf suffix will be given for output file
00010 #
00011 =====
00012
00013 ## If there are no command line argument then stop with error
00014 : ${1?"Try: $0 -h"}
00015
00016 ## This function read in polygon file and return informations for plot
00017 get_information(){
00018
00019     ## Get the number of polygons
00020     number_of_polygons=$(cat $FILE | grep -v "#" | awk 'NR==1{ print $_ + 0 }' )
00021     last=2
00022
00023     ## initialize counter
00024     count=0
00025     ## loop over all polygons
00026     while [ $count -lt $number_of_polygons ]
00027     do
00028
00029         ## Get the number of polygon points and the polygon action (incl/excl)
00030         ## and save in the array
00031         number_of_points=({number_of_points[*]} $(cat $FILE | grep -v "#" | awk "
NR==${last} { print \$_ + 0 }"))
00032         if_include=({if_include[*]} $(cat $FILE | grep -v "#" | awk "
NR==${($last+1)} { print \$_ }"))
00033         let last=$last+${number_of_points[$count]}+2
00034         let count++
00035     done
00036 }
00037
00038 usage()
00039 {
00040     DESCRIPTION="This program generate the map of polygon. It requires Generic
Mapping Tools command available"
00041     echo "
00042     usage: $0 options
00043
00044     $DESCRIPTION
00045
00046     OPTIONS:

```



```

00047 -h      Show this message
00048 -v      Verbose (default no verbose)
00049 -d      Debug (set debugging mode)
00050 -f      file [required]
00051 -R      GMT specific range (e.g. -R10/30/30/50)
00052 -o      output file
00053 "
00054 }
00055
00056 VERBOSE=
00057 DEBUG=
00058 FILE=
00059 OTHERARGS=
00060
00061 while getopts "vhdR:f:o:p:" flag
00062 do
00063     case "$flag" in
00064         f) FILE="$OPTARG";echo $FILE ;;
00065         d) set -x ; DEBUG=true;;
00066         v) VERBOSE=true ;;
00067         h) usage ; exit ;;
00068         R) R="-R$OPTARG" ;;
00069         o) output="$OPTARG" ;;
00070         p) POINTSFILE="$OPTARG" ;;
00071     esac
00072     # echo "$flag" $OPTIND $OPTARG
00073 done
00074
00075 if [ -z $FILE ]; then
00076     shift $((OPTIND-1))
00077     OTHERARGS="$@"
00078 fi
00079
00080 echo $FILE
00081
00082 # todo ! from command line
00083 if [ -z $FILE ] || [ -z $output ] ; then
00084     echo "Not enough cmd line parameters... , try $0 -h"
00085     exit
00086 fi
00087
00088
00089
00090 if [ -n "$VERBOSE" ] ; then
00091     echo "you set the verbose: $VERBOSE"
00092     echo "not recognized parameters args $OTHERARGS"
00093 fi
00094 echo "creating map for: $FILE ..."
00095
00096 get_information $FILE
00097 echo "Number of polygons:      " $number_of_polygons
00098 echo "Number of points:        " ${number_of_points[*]}
00099 echo "Include[+]/exclude[-]: " ${if_include[*]}
00100
00101 #cat $FILE |grep -v "#" |nl
00102
00103 function get_R(){
00104     last=3
00105     R=$(for i in ${number_of_points[*]}
00106     do
00107         cat $FILE | grep -v "#" | sed -n -e $((($last+1)),$(($last+$i)))p
00108         last=$((($last+$i+2))
00109     done | minmax -C | awk '{print $1-1, $2+1, $3-1,$4+1}' | sed 's/\s/\\/g')
00110     R=-R$R
00111 }
00112
00113
00114 if [ -z $R ]; then
00115     get_R
00116 fi
00117
00118 A="-A999"
00119
00120 gmtset FRAME_WIDTH=0.01c
00121 # psbasemap $R -K -JM20+ -X0 -Y0 -B100 > $output.ps
00122 pscoast $R -Slightblue -Glightgray -K -Di $A -J > $output.ps
00123 last=3
00124 for i in $(seq 0 $(( ${number_of_points[*]}-1 )) )
00125 do
00126     if [ ${if_include[${i}]} = "+" ]; then
00127         color=lightgreen
00128     else
00129         color=lightred
00130     fi
00131     cat $FILE | grep -v "#" | sed -n -e $((($last+1)),$(($last+${
00132         number_of_points[${i}]}))p \
00133         | psxy -R -J -K -O -A -W2p -L -G$color >> $output.ps

```

```

00133     last=$((last+${number_of_points[$i]}+2))
00134 done
00135
00136 if [ -z $POINTSFILE ] ; then
00137     echo "no points file given"
00138 else
00139     makecpt -Cjet -T0.1/0.9/0.2 |sed 's/^B.*/B 200 0 0/' |sed 's/^F.*/F 0 180
00140 0/' > points.cpt
00141     cat $POINTSFILE | awk "{print \$1 , \$2 ,\$(3)}" | psxy $R -J -Sc5p -
00142     Cpoints.cpt -Gred -W0.4lp/gray -O -K -V >> $output.ps
00143 fi
00144
00145 pscoast $R -O -Di $A -J -W -Nlthin >> $output.ps
00146
00147 ps2raster $output.ps -Tf -P -A
00148 #evince $output.ps
00149
00150
00151
00152
00153
00154 exit 0

```

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

This module contains utilities for computing Atmospheric Gravity Green Functions.

### Data Types

- module [aggf](#)

### 6.3.1 Detailed Description

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

Definition in file [aggf.f90](#).

## 6.4 aggf.f90

```

00001 !
00002 =====
00008 !
00009 =====
00009 module aggf
00010
00011     use constants
00012     implicit none
00013
00014 contains
00015
00016 !
00017 =====
00026 !
00027 =====
00027 subroutine compute_aggfdt ( psi , aggfdt , delta_ , aggf )
00028     implicit none
00029     real(dp) , intent (in) :: psi
00030     real(dp) , intent (in) , optional :: delta_
00031     logical , intent (in) , optional :: aggf
00032     real(dp) , intent (out) :: aggfdt
00033     real(dp) :: deltat , aux , h_
00034
00035     deltat = 10.
00036     if (present( delta_ ) ) deltat = delta_
00037     if (present( aggf ) .and. aggf ) then
00038         h_ = 0.001 ! default if we compute dggfdh using this routine
00039         if (present( delta_ ) ) h_ = deltat
00040         call compute_aggf( psi , aux , h = + h_ )

```

```

00041     aggfdt = aux
00042     call compute_aggf( psi , aux , h= -h_ )
00043     aggfdt = aggfdt - aux
00044     aggfdt = aggfdt / ( 2. * h_ )
00045   else
00046     call compute_aggf( psi , aux , t_zero = t0 + deltat )
00047     aggfdt = aux
00048     call compute_aggf( psi , aux , t_zero = t0 - deltat )
00049     aggfdt = aggfdt - aux
00050     aggfdt = aggfdt / ( 2. * deltat )
00051   endif
00052
00053
00054
00055 end subroutine
00056
00057 !
=====
00065 !
=====
00066 subroutine read_tabulated_green ( table , author )
00067   real(dp), intent (inout), dimension(:, :), allocatable :: table
00068   character ( len = * ) , intent (in) , optional          :: author
00069   integer                                     :: i , j
00070   integer                                     :: rows , columns ,
file_unit
00071   character (len=255)                               :: file_name
00072
00073   rows      = 85
00074   columns   = 6
00075   file_name = '../dat/merriam_green.dat'
00076
00077   if ( present(author) ) then
00078     if ( author .eq. "huang" ) then
00079       rows      = 80
00080       columns   = 5
00081       file_name = '../dat/huang_green.dat'
00082     elseif( author .eq. "rajner" ) then
00083       rows      = 85
00084       columns   = 5
00085       file_name = '../dat/rajner_green.dat'
00086     elseif( author .eq. "merriam" ) then
00087     else
00088       write ( * , * ) 'cannot find specified tables, using merriam instead'
00089     endif
00090   endif
00091
00092   if (allocated (table) ) deallocate (table)
00093   allocate ( table( rows , columns ) )
00094
00095   open (newunit = file_unit , file = file_name , action='read' , status='old')
00096
00097   call skip_header(file_unit)
00098
00099   do i = 1 , rows
00100     read (file_unit,*) ( table( i , j ) , j = 1 , columns )
00101   enddo
00102   close(file_unit)
00103 end subroutine
00104
00105
00106 !
=====
00109 !
=====
00110 subroutine compute_aggf (psi , aggf_val , hmin , hmax , dh ,
if_normalization, &
00111     t_zero , h , first_derivative_h , first_derivative_z ,
fels_type )
00112   implicit none
00113   real(dp), intent(in)                :: psi
00114   real(dp), intent(in), optional :: hmin , & !< minimum height, starting point
[km] (default=0)
00115     hmax , & !< maximum height. ending point [km]
00116     dh , & !< integration step [km]
00117     t_zero , & !< temperature at the surface [K]
00118     h
00119   logical, intent(in), optional :: if_normalization , first_derivative_h ,
first_derivative_z
00120   character (len=*) , intent(in), optional :: fels_type
00121   real(dp), intent(out)                :: aggf_val
00122   real(dp)                             :: r , z , psir , da , dz , rho , h_min , h_max
, h_station , j_aux
00123

```

```

00124 h_min = 0.
00125 h_max = 60.
00126 dz = 0.0001 !mrajner 2012-11-08 13:49
00127 h_station = 0.
00128
00129 if ( present(hmin) ) h_min = hmin
00130 if ( present(hmax) ) h_max = hmax
00131 if ( present( dh ) ) dz = dh
00132 if ( present( h ) ) h_station = h
00133
00134
00135 psir = psi * pi / 180.
00136
00137 da = 2 * pi * r0**2 * ( 1 - cos(1. *pi/180.) )
00138
00139
00140 aggf_val=0.
00141 do z = h_min , h_max , dz
00142
00143 r = ( ( r0 + z )**2 + (r0 + h_station)**2 &
00144 - 2.*(r0 + h_station) * (r0+z)*cos(psir) )**(0.5)
00145 call standard_density( z , rho , t_zero = t_zero ,
fels_type = fels_type )
00146
00149 if ( present( first_derivative_h ) .and. first_derivative_h ) then
00150
00151 !! see equation 22, 23 in \cite Huang05
00152 !J_aux = (( r0 + z )**2)*(1.-3.*((cos(psir))**2)) -2.*(r0 + h_station
)**2 &
00153 ! + 4.*(r0+h_station)*(r0+z)*cos(psir)
00154 ! aggf_val = aggf_val - rho * ( J_aux / r**5 ) * dz
00155
00157 j_aux = (2.* (r0 ) - 2 * (r0 +z )*cos(psir)) / (2. * r)
00158 j_aux = -r - 3 * j_aux * ((r0+z)*cos(psir) - r0)
00159 aggf_val = aggf_val + rho * ( j_aux / r**4 ) * dz
00160 else
00164 if ( present( first_derivative_z ) .and. first_derivative_z ) then
00165 if (z.eq.h_min) then
00166 aggf_val = aggf_val &
00167 + rho*( ((r0 + z)*cos(psir) - ( r0 + h_station ) ) / ( r**3 ) )
00168 endif
00169 else
00172 aggf_val = aggf_val &
00173 + rho * ( ( (r0 + z ) * cos( psir ) - ( r0 + h_station ) ) / ( r**3 )
) * dz
00174 endif
00175 endif
00176 enddo
00177
00178 aggf_val = -g * da * aggf_val * 1e8 * 1000
00179
00183 if ( (.not.present(if_normalization)) .or. (if_normalization)) then
00184 aggf_val= psir * aggf_val * 1e5 / p0
00185 endif
00186
00187 end subroutine
00188
00189 !
=====
00193 !
=====
00194 subroutine standard_density ( height , rho , t_zero ,fels_type
)
00195
00196 implicit none
00197 real(dp) , intent(in) :: height
00198 real(dp) , intent(in), optional :: t_zero
00199 character(len = 22) , optional :: fels_type
00200 !! surface temperature is set to this value,
00201 !! otherwise the T0 for standard atmosphere is used
00202 real(dp) , intent(out) :: rho
00203 real(dp) :: p ,t
00204
00205 call standard_pressure(height , p , t_zero = t_zero,
fels_type=fels_type)
00206 call standard_temperature(height , t , t_zero = t_zero,
fels_type=fels_type)
00207
00208 ! pressure in hPa --> Pa
00209 rho= 100 * p / ( r_air * t )
00210 end subroutine
00211
00212 ! =====
00218 ! =====
00219 subroutine standard_pressure (height, pressure , &
00220 p_zero , t_zero , if_simplified ,fels_type , inverted)
00221 implicit none

```

```

00222 real(dp) , intent(in)          :: height
00223 real(dp) , intent(in) , optional :: t_zero , p_zero
00224 character(len = 22) , optional :: fels_type
00225 logical          , intent(in) , optional :: if_simplified
00226 logical          , intent(in) , optional :: inverted
00227 real(dp), intent(out) :: pressure
00228 real(dp) :: lambda , temperature , g , alpha , sfc_pressure
00229
00230 sfc_pressure = p0
00231 if (present(p_zero)) sfc_pressure = p_zero
00232
00233 call standard_temperature( height,temperature,t_zero=
t_zero, &
00234                          fels_type=fels_type)
00235 call standard_gravity( height , g )
00236
00237 lambda = r_air * temperature / g
00238
00239 if (present(if_simplified) .and. if_simplified ) then
00240   ! use simplified formulae
00241   alpha = -6.5
00242   pressure = sfc_pressure * ( 1 + alpha / t0 * height ) ** ( -g0 / (r_air *
alpha / 1000 ) )
00243 else
00244   ! use precise formulae
00245   pressure = sfc_pressure * exp( -1000. * height / lambda )
00246 endif
00247 if (present(inverted).and.inverted) then
00248   pressure = sfc_pressure / ( exp( -1000. * height / lambda ) )
00249 endif
00250 end subroutine
00251
00252 ! =====
00253 ! > This will transfer pressure between different height using barometric
00254 ! formulae
00255 ! =====
00256 subroutine transfer_pressure (height1 , height2 , pressure1 , pressure2 , &
00257 temperature , polish_meteo )
00258 real (dp) , intent (in) :: height1 , height2 , pressure1
00259 real (dp) , intent (in), optional :: temperature
00260 real (dp) :: sfc_temp , sfc_pres
00261 logical , intent (in), optional :: polish_meteo
00262 real(dp) , intent(out) :: pressure2
00263
00264 sfc_temp = t0
00265
00266 ! formulae used to reduce press to sfc in polish meteo service
00267 if (present(polish_meteo) .and. polish_meteo) then
00268   sfc_pres = exp(log(pressure1) + 2.30259 * height1*1000. &
00269 / (18400.*(1+0.00366*(temperature-273.15) + 0.0025*height1*1000.))) )
00270 else
00271   ! different approach
00272   if(present(temperature) ) then
00273     call surface_temperature( height1 , temperature ,
00274 sfc_temp )
00275   endif
00276   call standard_pressure(height1 , sfc_pres , t_zero=
sfc_temp , &
00277 inverted=.true. , p_zero = pressure1 )
00278   endif
00279   ! move from sfc to height2
00280   call standard_pressure(height2 , pressure2 , t_zero=sfc_temp
00281 , &
00282 p_zero = sfc_pres )
00283 end subroutine
00284
00285 ! =====
00286 ! =====
00290 ! =====
00291 subroutine standard_gravity ( height , g )
00292 implicit none
00293 real(dp), intent(in) :: height
00294 real(dp), intent(out) :: g
00295
00296 g= g0 * ( r0 / ( r0 + height ) )**2
00297 end subroutine
00298
00299
00300 ! =====
00301 ! =====
00302 ! =====
00303 real(sp) function geometric_height (geopotential_height)
00304 real (sp) :: geopotential_height
00305
00306 geometric_height = geopotential_height * (r0 / ( r0 +
geopotential_height ) )
00307 end function

```

```

00308
00309
00310 ! =====
00313 ! =====
00314 subroutine surface_temperature (height , temperature1 , &
00315     temperature2, fels_type , tolerance)
00316     real(dp) , intent(in) :: height , temperature1
00317     real(dp) , intent(out) :: temperature2
00318     real(dp) :: temp(3) , temp_ (3) , tolerance_ = 0.1
00319     character (len=*) , intent(in), optional :: fels_type
00320     real(sp) , intent(in), optional :: tolerance
00321     integer :: i
00322
00323     if (present(tolerance)) tolerance_ = tolerance
00324
00325     ! searching limits
00326     temp(1)=t0-150
00327     temp(3)=t0+ 50
00328
00329     do
00330         temp(2)= ( temp(1) + temp(3) ) /2.
00331
00332         do i = 1,3
00333             call standard_temperature(height , temp_(i) , t_zero=
00334 temp(i) , fels_type = fels_type )
00335         enddo
00336
00337         if (abs(temperature1 - temp_(2) ) .lt. tolerance_ ) then
00338             temperature2 = temp(2)
00339             return
00340         endif
00341
00342         if ( (temperature1 - temp_(1) ) * (temperature1 - temp_(2) ) .lt.0 ) then
00343             temp(3) = temp(2)
00344         elseif( (temperature1 - temp_(3) ) * (temperature1 - temp_(2) ) .lt.0 )
00345 then
00346             temp(1) = temp(2)
00347         else
00348             stop "surface_temp"
00349         endif
00350     enddo
00351 end subroutine
00352 ! =====
00353 ! =====
00354
00355 subroutine standard_temperature ( height , temperature ,
00356     t_zero , fels_type )
00357     real(dp) , intent(in) :: height
00358     real(dp) , intent(out) :: temperature
00359     real(dp) , intent(in), optional :: t_zero
00360     character (len=*) , intent(in), optional :: fels_type
00361
00362     real(dp) :: aux , cn , t
00363     integer :: i,indeks
00364     real , dimension (10) :: z,c,d
00365
00366     z = (/11.0 , 20.1 , 32.1 , 47.4 , 51.4 , 71.7 , 85.7, 100.0, 200.0, 300.0/)
00367     c = (/ -6.5, 0.0, 1.0, 2.75, 0.0, -2.75, -1.97, 0.0, 0.0, 0.0/)
00368     d = (/ 0.3, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0/)
00369     t = t0
00370
00371     if ( present(fels_type)) then
00372         if (fels_type .eq. "US1976" ) then
00373             z= (/ 2.0 , 3.0, 16.5 , 21.5 , 45.0 , 51.0, 70.0 , 100.0 , 200.0 , 300.0
00374 /)
00375             c= (/ -6.0 , -4.0, -6.7 , 4.0 , 2.2 , 1.0, -2.8 , -0.27 , 0.0 , 0.0
00376 /)
00377             d= (/ 0.5 , 0.5 , 0.3 , 0.5 , 1.0 , 1.0 , 1.0 , 1.0 , 1.0 , 1.0
00378 /)
00379             t=300.0
00380         elseif(fels_type .eq. "tropical" ) then
00381             z = (/ 1.5 , 6.5 , 13.0 , 18.0 , 26.0 , 36.0 , 48.0 , 50.0 , 70.0 ,
00382 100.0
00383 /)
00384             c = (/ -4.0 , -6.0 , -6.5 , 0.0 , 1.2 , 2.2 , 2.5 , 0.0 , -3.0
00385 , -0.025/)
00386             d = (/ 0.5 , 1.0 , 0.5 , 0.5 , 1.0 , 1.0 , 2.5 , 0.5 , 1.0
00387 , 1.0
00388 /)
00389             t = 294.0
00390         elseif(fels_type .eq. "subtropical_summer" ) then
00391             z = (/ 3.0 , 10.0 , 19.0 , 25.0 , 32.0 , 44.5 , 50.0 , 71.0 , 98.0 ,
00392 200.0
00393 /)
00394             c = (/ -3.5 , -6.0 , -0.5 , 0.0 , 0.4 , 3.2 , 1.6 , -1.8 , 0.7
00395 , 0.0
00396 /)
00397             d = (/ 0.5 , 0.5 , 1.0 , 1.0 , 1.0 , 1.0 , 1.0 , 1.0 , 1.0 , 1.0
00398 , 1.0
00399 /)

```

```

00398      t = 272.2
00399      elseif(fels_type .eq. "subarctic_summer" ) then
00400      z = (/ 4.7 , 10.0 , 23.0 , 31.8 , 44.0 , 50.2 , 69.2 , 100.0 , 200.0 ,
300.0 /)
00401      c = (/ -5.3 , -7.0 , 0.0 , 1.4 , 3.0 , 0.7 , -3.3 , -0.2 , 0.0 ,
0.0 /)
00402      d = (/ 0.5 , 0.3 , 1.0 , 1.0 , 2.0 , 1.0 , 1.5 , 1.0 , 1.0 ,
1.0 /)
00403      t = 287.0
00404      elseif(fels_type .eq. "subarctic_winter" ) then
00405      z = (/ 1.0 , 3.2 , 8.5 , 15.5 , 25.0 , 30.0 , 35.0 , 50.0 , 70.0 , 100
.0 /)
00406      c = (/ 3.0 , -3.2 , -6.8 , 0.0 , -0.6 , 1.0 , 1.2 , 2.5 , -0.7 , -1
.2 /)
00407      d = (/ 0.4 , 1.5 , 0.3 , 0.5 , 1.0 , 1.0 , 1.0 , 1.0 , 1.0 , 1
.0 /)
00408      t = 257.1
00409      else
00410      print * ,
00411      "unknown fels_type argument: &          using US standard atmosphere 1976
instead"
00412      endif
00413      endif
00414
00415      if (present(t_zero) ) then
00416      t=t_zero
00417      endif
00418
00419      do i=1,10
00420      if (height.le.z(i)) then
00421      indeks=i
00422      exit
00423      endif
00424      enddo
00425
00426      aux = 0.
00427      do i = 1 , indeks
00428      if (i.eq.indeks) then
00429      cn = 0.
00430      else
00431      cn = c(i+1)
00432      endif
00433      aux = aux + d(i) * ( cn - c(i) ) * log( cosh( (height - z(i)) / d(i) ) /
cosh(z(i)/d(i)) )
00434      enddo
00435      temperature = t + c(1) * height/2. + aux/2.
00436      end subroutine
00437
00438      !
=====
00444      !
=====
00445      real function gn_thin_layer (psi)
00446      implicit none
00447      real(dp) , intent(in) :: psi
00448      real(dp) :: psir
00449
00450      psir = psi * pi / 180.
00451      gn_thin_layer = 1.627 * psir / sin( psir / 2. )
00452      end function
00453
00454
00455      !
=====
00459      !
=====
00460      integer function size_ntimes_denser (size_original, ndenser)
00461      integer, intent(in) :: size_original , ndenser
00462      size_ntimes_denser= (size_original - 1 ) * (ndenser +1 ) +
1
00463      end function
00464
00465      !
=====
00468      !
=====
00469      real(dp) function bouger ( R_opt )
00470      real(dp), optional :: r_opt
00471      real(dp) :: aux
00472      real(dp) :: r
00473      real(dp) :: h = 8.84 ! scale height of standard atmosphere
00474
00475      aux = 1
00476
00477      if (present( r_opt ) ) then
00478      r = r_opt
00479      aux = h + r - sqrt( r**2 + (h/2. ) ** 2 )

```

```

00480     bouger = 2 * pi * g * aux
00481   else
00482     aux = h
00483     bouger = 2 * pi * g * aux
00484     return
00485   endif
00486 end function
00487 !
=====
00490 !
=====
00491 real(dp) function simple_def (R)
00492   real(dp) :: r ,delta
00493
00494   delta = 0.22e-11 * r
00495
00496   simple_def = g0 / r0 * delta * ( 2. - 3./2. * rho_crust / rho_earth
&
00497     -3./4. * rho_crust / rho_earth * sqrt(2* (1. )) ) * 1000
00498 end function
00499
00500 !polish_meteo
00501
00502 end module

```

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

This module define some constant values used.

### Data Types

- module `constants`

#### 6.5.1 Detailed Description

This module define some constant values used.

Definition in file `constants.f90`.

## 6.6 constants.f90

```

00001 !
=====
00004 !
=====
00005 module constants
00006
00007   implicit none
00008   integer , parameter :: dp = 8
00009   integer , parameter :: sp = 4
00010   real(dp) , parameter :: &
00011     T0      = 288.15, & !< surface temperature for standard atmosphere
[K] (15 degC)
00012     g0      = 9.80665, & !< mean gravity on the Earth [m/s2]
00013     r0      = 6356.766, & !< Earth radius (US Std. atm. 1976) [km]
00014     p0      = 1013.25, & !< surface pressure for standard Earth [hPa]
00015     G       = 6.672e-11, & !< Cavendish constant \f$[m^3/kg/s^2]\f$
00016     R_air   = 287.05, & !< dry air constant [J/kg/K]
00017     pi      = 4*atan(1.), & !< pi = 3.141592... [ ]
00018     rho_crust = 2670 , & !< mean density of crust [kg/m3]
00019     rho_earth = 5500
00020
00021 contains
00022
00023 !
=====
00027 !
=====
00028 subroutine spline_interpolation(x,y, x_interpolated,
y_interpolated)
00029   implicit none
00030   real(dp) , allocatable , dimension (:),intent(in) :: x, y, x_interpolated

```



```

00031  real(dp) , allocatable , dimension (:) , intent(out) :: y_interpolated
00032  real(dp) , dimension (:) , allocatable :: b, c, d
00033  integer :: i
00034
00035  allocate (b(size(x)))
00036  allocate (c(size(x)))
00037  allocate (d(size(x)))
00038  allocate (y_interpolated(size(x_interpolated)))
00039
00040  call spline( x , y , b , c , d , size(x))
00041
00042  do i=1, size(x_interpolated)
00043    y_interpolated(i) = ispline(x_interpolated(i) , x , y , b , c , d ,
00044      size (x) )
00045  enddo
00046 end subroutine
00047
00048 !
=====
00051 !
=====
00052 ! Calculate the coefficients b(i), c(i), and d(i), i=1,2,...,n
00053 ! for cubic spline interpolation
00054 ! s(x) = y(i) + b(i)*(x-x(i)) + c(i)*(x-x(i))**2 + d(i)*(x-x(i))**3
00055 ! for x(i) <= x <= x(i+1)
00056 ! Alex G: January 2010
00057 !-----
00058 ! input..
00059 ! x = the arrays of data abscissas (in strictly increasing order)
00060 ! y = the arrays of data ordinates
00061 ! n = size of the arrays xi() and yi() (n>=2)
00062 ! output..
00063 ! b, c, d = arrays of spline coefficients
00064 ! comments ...
00065 ! spline.f90 program is based on fortran version of program spline.f
00066 ! the accompanying function fspline can be used for interpolation
00067 !
=====
00068 subroutine spline (x, y, b, c, d, n)
00069  implicit none
00070  integer n
00071  real(dp) :: x(n), y(n), b(n), c(n), d(n)
00072  integer i, j, gap
00073  real :: h
00074
00075  gap = n-1
00076  ! check input
00077  if ( n < 2 ) return
00078  if ( n < 3 ) then
00079    b(1) = (y(2)-y(1))/(x(2)-x(1)) ! linear interpolation
00080    c(1) = 0.
00081    d(1) = 0.
00082    b(2) = b(1)
00083    c(2) = 0.
00084    d(2) = 0.
00085    return
00086  end if
00087  !
00088  ! step 1: preparation
00089  !
00090  d(1) = x(2) - x(1)
00091  c(2) = (y(2) - y(1))/d(1)
00092  do i = 2, gap
00093    d(i) = x(i+1) - x(i)
00094    b(i) = 2.0*(d(i-1) + d(i))
00095    c(i+1) = (y(i+1) - y(i))/d(i)
00096    c(i) = c(i+1) - c(i)
00097  end do
00098  !
00099  ! step 2: end conditions
00100  !
00101  b(1) = -d(1)
00102  b(n) = -d(n-1)
00103  c(1) = 0.0
00104  c(n) = 0.0
00105  if(n /= 3) then
00106    c(1) = c(3)/(x(4)-x(2)) - c(2)/(x(3)-x(1))
00107    c(n) = c(n-1)/(x(n)-x(n-2)) - c(n-2)/(x(n-1)-x(n-3))
00108    c(1) = c(1)*d(1)**2/(x(4)-x(1))
00109    c(n) = -c(n)*d(n-1)**2/(x(n)-x(n-3))
00110  end if
00111  !
00112  ! step 3: forward elimination
00113  !
00114  do i = 2, n
00115    h = d(i-1)/b(i-1)

```

```

00116      b(i) = b(i) - h*d(i-1)
00117      c(i) = c(i) - h*c(i-1)
00118  end do
00119  !
00120  ! step 4: back substitution
00121  !
00122  c(n) = c(n)/b(n)
00123  do j = 1, gap
00124      i = n-j
00125      c(i) = (c(i) - d(i)*c(i+1))/b(i)
00126  end do
00127  !
00128  ! step 5: compute spline coefficients
00129  !
00130  b(n) = (y(n) - y(gap))/d(gap) + d(gap)*(c(gap) + 2.0*c(n))
00131  do i = 1, gap
00132      b(i) = (y(i+1) - y(i))/d(i) - d(i)*(c(i+1) + 2.0*c(i))
00133      d(i) = (c(i+1) - c(i))/d(i)
00134      c(i) = 3.*c(i)
00135  end do
00136  c(n) = 3.0*c(n)
00137  d(n) = d(n-1)
00138  end subroutine spline
00139
00140
00141  !
00144  !
=====
00145  !=====
00146  ! function ispline evaluates the cubic spline interpolation at point z
00147  ! ispline = y(i)+b(i)*(u-x(i))+c(i)*(u-x(i))**2+d(i)*(u-x(i))**3
00148  ! where x(i) <= u <= x(i+1)
00149  !-----
00150  ! input..
00151  ! u      = the abscissa at which the spline is to be evaluated
00152  ! x, y    = the arrays of given data points
00153  ! b, c, d = arrays of spline coefficients computed by spline
00154  ! n      = the number of data points
00155  ! output:
00156  ! ispline = interpolated value at point u
00157  !=====
00158  function ispline(u, x, y, b, c, d, n)
00159  implicit none
00160  real ispline
00161  integer n
00162  real(dp):: u, x(n), y(n), b(n), c(n), d(n)
00163  integer :: i, j, k
00164  real :: dx
00165
00166  ! if u is outside the x() interval take a boundary value (left or right)
00167  if(u <= x(1)) then
00168      ispline = y(1)
00169      return
00170  end if
00171  if(u >= x(n)) then
00172      ispline = y(n)
00173      return
00174  end if
00175
00176  !*
00177  ! binary search for i, such that x(i) <= u <= x(i+1)
00178  !*
00179  i = 1
00180  j = n+1
00181  do while (j > i+1)
00182      k = (i+j)/2
00183      if(u < x(k)) then
00184          j=k
00185      else
00186          i=k
00187      end if
00188  end do
00189  !*
00190  ! evaluate spline interpolation
00191  !*
00192  dx = u - x(i)
00193  ispline = y(i) + dx*(b(i) + dx*(c(i) + dx*d(i)))
00194  end function ispline
00195
00196  !
=====
00198  !
=====
00199  integer function ntokens(line)
00200  character,intent(in):: line(*)
00201  integer i, n, toks

```

```

00202
00203 i = 1;
00204 n = len_trim(line)
00205 toks = 0
00206 ntokens = 0
00207 do while(i <= n)
00208   do while(line(i:i) == ' ')
00209     i = i + 1
00210     if (n < i) return
00211   enddo
00212   toks = toks + 1
00213   ntokens = toks
00214   do
00215     i = i + 1
00216     if (n < i) return
00217     if (line(i:i) == ' ') exit
00218   enddo
00219 enddo
00220 end function ntokens
00221
00222 !
=====
00225 !
=====
00226 subroutine skip_header ( unit , comment_char_optional )
00227   use iso_fortran_env
00228   implicit none
00229   integer , intent (in) :: unit
00230   character (len = 1) , optional :: comment_char_optional
00231   character (len = 60 ) :: dummy
00232   character (len = 1) :: comment_char
00233   integer :: io_stat
00234
00235   if (present( comment_char_optional ) ) then
00236     comment_char = comment_char_optional
00237   else
00238     comment_char = '#'
00239   endif
00240
00241   read ( unit, * , iostat = io_stat) dummy
00242   if(io_stat == iostat_end) return
00243
00244   do while ( dummy(1:1) .eq. comment_char )
00245     read ( unit, * , iostat = io_stat ) dummy
00246     if(io_stat == iostat_end) return
00247   enddo
00248   backspace(unit)
00249 end subroutine
00250
00253 real function jd (year,month,day, hh,mm,ss)
00254   implicit none
00255   integer, intent(in) :: year,month,day
00256   integer, intent(in) :: hh,mm, ss
00257   integer :: i , j , k
00258   i= year
00259   j= month
00260   k= day
00261   jd= k-32075+1461*(i+4800+(j-14)/12)/4+367*(j-2-(j-14)/12*12)/12-3*((i+4900+
(j-14)/12)/100)/4 + (hh/24.) &
00262   + mm/(24.*60.) +ss/(24.*60.*60.) ! - 2400000.5
00263   return
00264 end function
00265
00266 !subroutine gdate (jd, year,month,day,hh,mm,ss)
00267 ! !! modyfikacja mrajner 20120922
00268 ! !! pobrane http://aa.usno.navy.mil/faq/docs/jd_formula.php
00269 ! implicit none
00270 ! real, intent(in):: jd
00271 ! real :: aux
00272 ! integer,intent(out) :: year,month,day,hh,mm,ss
00273 ! integer :: i,j,k,l,n
00274
00275 ! l= int((jd+68569))
00276 ! n= 4*1/146097
00277 ! l= 1-(146097*n+3)/4
00278 ! i= 4000*(l+1)/1461001
00279 ! l= 1-1461*i/4+31
00280 ! j= 80*1/2447
00281 ! k= 1-2447*j/80
00282 ! l= j/11
00283 ! j= j+2-12*l
00284 ! i= 100*(n-49)+i+1
00285
00286 ! year= i
00287 ! month= j
00288 ! day= k
00289

```

```

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

```

## 6.7 /home/mrajner/src/grat/src/example\_aggf.f90 File Reference

This program shows some example of using AGGF module.

### Functions/Subroutines

- program **example\_aggf**
- subroutine **simple\_atmospheric\_model** ()  
*Reproduces data to Fig.~3 in.*
- subroutine **compare\_tabulated\_green\_functions** ()  
*Compare tabulated green functions from different authors.*
- subroutine **compute\_tabulated\_green\_functions** ()  
*Compute AGGF and derivatives.*
- subroutine **aggf\_resp\_fels\_profiles** ()

*Compare different vertical temperature profiles impact on AGGF.*

- subroutine `compare_fels_profiles` ()  
*Compare different vertical temperature profiles.*
- subroutine `aggf_resp_h` ()  
*Computes AGGF for different site height (h)*
- subroutine `aggf_resp_t` ()  
*This computes AGGF for different surface temperature.*
- subroutine `aggfdt_resp_dt` ()  
*This computes AGGFDT for different dT.*
- subroutine `aggf_resp_dz` ()  
*This computes AGGF for different height integration step.*
- subroutine `standard1976`  
*This computes standard atmosphere parameters.*
- subroutine `aggf_resp_hmax` ()  
*This computes relative values of AGGF for different atmosphere height integration.*
- subroutine `aux_heights` (table)  
*Relative value of aggf depending on integration height.*
- subroutine `aggf_thin_layer` ()

### 6.7.1 Detailed Description

This program shows some example of using AGGF module.

#### Author

Marcin Rajner

#### Date

20121108

The examples are in contained subroutines

Definition in file `example_aggf.f90`.

### 6.7.2 Function/Subroutine Documentation

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

Relative value of aggf depending on integration height.

Auxiliary subroutine – height sampling for semilog plot

Definition at line 459 of file `example_aggf.f90`.

#### 6.7.2.2 subroutine `example_aggf::compare_fels_profiles` ( )

Compare different vertical temperature profiles.

Using tables and formula from `Fels` [1986]

Definition at line 192 of file `example_aggf.f90`.

### 6.7.2.3 subroutine example\_aggf::simple\_atmospheric\_model ( )

Reproduces data to Fig.~3 in.

Warburton and Goodkind [1977]

Definition at line 39 of file example\_aggf.f90.

### 6.7.2.4 subroutine example\_aggf::standard1976 ( )

This computes standard atmosphere parameters.

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

Definition at line 387 of file example\_aggf.f90.

## 6.8 example\_aggf.f90

```

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

```

```

00067
00068   how_many_denser=0
00069
00070   ! reference author
00071   call read_tabulated_green(table , author = authors(1) )
00072   allocate (results(size_ntimes_denser(size(table(:,1))),
how_many_denser) , 0 : size(authors) ))
00073
00074   ! fill abscissa in column 0
00075   ii = 1
00076   do i = 1 , size (table(:,1) ) - 1
00077     do j = 0 , how_many_denser
00078       results(ii,0) = table(i,1 ) + j * (table(i+1, 1) -table(i,1) ) / (
how_many_denser + 1 )
00079       ii=ii+1
00080     enddo
00081   enddo
00082   ! and the last element
00083   results( size (results(:,0) ) , 0) = table( size(table(:,1)) ,1 )
00084
00085   ! take it as main for all series
00086   allocate(x_interpolated( size ( results(:,0))))
00087   x_interpolated = results(:,0)
00088
00089   open (newunit = file_unit , file = "../examples/compare_aggf.dat", action=
"write")
00090
00091   ! for every author
00092   do i= 1, size(authors)
00093     print * , trim( authors( i ) )
00094     call read_tabulated_green(table , author = authors(i) )
00095     allocate(x( size (table(:,1))))
00096     allocate(y( size (table(:,2))))
00097     x = table(:,1)
00098     y = table(:, columns(i))
00099     call spline_interpolation( x , y , x_interpolated,
y_interpolated )
00100     if (i.gt.1) then
00101       y_interpolated = ( y_interpolated - results(:,1) ) / results(:,1) * 100.
00102     endif
00103
00104     results(:, i ) = y_interpolated
00105     deallocate(x,y)
00106   enddo
00107
00108   write (file_unit , '( <size(results(1,:))>f20.5)' ) ( results(i , :) , i = 1 ,
size(results( :,1)) )
00109   close(file_unit)
00110 end subroutine
00111
00112 ! =====
00114 ! =====
00115 subroutine compute_tabulated_green_functions
()
00116   integer :: i , file_unit
00117   real(dp) :: val_aggf , val_aggfdt , val_aggfdh , val_aggfdz
00118   real(dp), dimension(:,:) , allocatable :: table , results
00119
00120   ! Get the spherical distances from Merriam92
00121   call read_tabulated_green( table , author = "merriam")
00122
00123   open ( newunit = file_unit, &
00124         file = '../dat/rajner_green.dat', &
00125         action = 'write' &
00126         )
00127
00128   ! print header
00129   write ( file_unit,*) '# This is set of AGGF computed using module ', &
00130   'aggf from grat software'
00131   write ( file_unit,*) '# Normalization according to Merriam92'
00132   write ( file_unit,*) '# Marcin Rajner'
00133   write ( file_unit,*) '# For detail see www.geo.republika.pl'
00134   write ( file_unit,'(10(a23))' ) '#psi[deg]', &
00135   'GN[microGal/hPa]' , 'GN/dT[microGal/hPa/K]' , &
00136   'GN/dh[microGal/hPa/km]' , 'GN/dz[microGal/hPa/km]'
00137
00138   do i= 1, size(table(:,1))
00139     call compute_aggf( table(i,1) , val_aggf )
00140     call compute_aggfdt( table(i,1) , val_aggfdt )
00141     call compute_aggf( table(i,1) , val_aggfdh , first_derivative_h
=.true. )
00142     call compute_aggf( table(i,1) , val_aggfdz , first_derivative_z
=.true. )
00143     write ( file_unit, '(10(e23.5))' ) &
00144     table(i,1) , val_aggf , val_aggfdt , val_aggfdh , val_aggfdz
00145   enddo
00146   close(file_unit)

```

```

00147 end subroutine
00148
00149 ! =====
00151 ! =====
00152 subroutine aggf_resp_fels_profiles ()
00153   character (len=255), dimension (6) :: fels_types
00154   real (dp) :: val_aggf
00155   integer :: i, j, file_unit
00156   real(dp), dimension(:,:), allocatable :: table
00157
00158   ! All possible optional arguments for standard_temperature
00159   fels_types = (/ "US1976" , "tropical", &
00160                  "subtropical_summer", "subtropical_winter", &
00161                  "subarctic_summer" , "subarctic_winter"  /)
00162
00163   open ( newunit = file_unit, &
00164          file     = '../examples/aggf_resp_fels_profiles.dat' , &
00165          action   = 'write' &
00166          )
00167
00168   call read_tabulated_green(table)
00169
00170   ! print header
00171   write ( file_unit , '(100(a20))' ) &
00172     'psi', ( trim( fels_types(i) ) , i = 1 , size (fels_types) )
00173
00174   ! print results
00175   do i = 1 , size (table(:,1))
00176     write (file_unit, '(f20.6$)' ) table(i,1)
00177     do j = 1 , size(fels_types)
00178       call compute_aggf(table(i,1), val_aggf ,fels_type=fels_types(
00179 j))
00179       write (file_unit, '(f20.6$)' ) val_aggf
00180     enddo
00181     write(file_unit, *)
00182   enddo
00183   close(file_unit)
00184 end subroutine
00185
00186
00187 ! =====
00191 ! =====
00192 subroutine compare_fels_profiles ()
00193   character (len=255), dimension (6) :: fels_types
00194   real (dp) :: height , temperature
00195   integer :: i , file_unit
00196
00197   ! All possible optional arguments for standard_temperature
00198   fels_types = (/ "US1976" , "tropical", &
00199                  "subtropical_summer", "subtropical_winter", &
00200                  "subarctic_summer" , "subarctic_winter"  /)
00201
00202   open ( newunit = file_unit, &
00203          file     = '../examples/compare_fels_profiles.dat' , &
00204          action   = 'write' &
00205          )
00206
00207   ! Print header
00208   write ( file_unit , '(100(a20))' ) &
00209     'height', ( trim( fels_types(i) ) , i = 1 , size (fels_types) )
00210
00211   ! Print results
00212   do height = 0. , 70. , 1.
00213     write ( file_unit , '(f20.3$)' ) , height
00214     do i = 1 , size (fels_types)
00215       call standard_temperature &
00216         ( height , temperature , fels_type = fels_types(i) )
00217       write ( file_unit , '(f20.3$)' ) , temperature
00218     enddo
00219     write ( file_unit , * )
00220   enddo
00221   close(file_unit)
00222 end subroutine
00223
00224 ! =====
00226 ! =====
00227 subroutine aggf_resp_h ()
00228   real(dp), dimension(:,:), allocatable :: table , results
00229   integer :: i, j, file_unit , ii
00230   real(dp) :: val_aggf
00231
00232   ! Get the spherical distances from Merriam92
00233   call read_tabulated_green( table , author = "merriam")
00234
00235   ! Specify the output table and put station height in first row
00236   allocate ( results( 0 : size (table(:,1)) , 7 ) )
00237   results(0,1) = 1./0 ! Infinity in first header

```



```

00238 results(0,3) = 0.0      ! 0 m
00239 results(0,3) = 0.001    ! 1 m
00240 results(0,4) = 0.01     ! 10 m
00241 results(0,5) = 0.1      ! 100 m
00242 results(0,6) = 1.        ! 1 km
00243 results(0,7) = 10.       ! 10 km
00244
00245 ! write results to file
00246 open ( &
00247     newunit = file_unit, &
00248     file     = '../examples/aggf_resp_h.dat', &
00249     action   = 'write' &
00250 )
00251
00252 write (file_unit, '(8(F20.8))' ) results(0, :)
00253 do i =1 , size (table(:,1))
00254     ! denser sampling
00255     do ii = 0,8
00256         results( i , 1 ) = table(i,1) + ii * (table(i+1,1) - table(i,1)) / 9.
00257         ! only compute for small spherical distances
00258         if (results(i, 1) .gt. 0.2 ) exit
00259         write (file_unit, '(F20.7,$)' ) , results(i,1)
00260         do j = 2 , size(results(1,: ) )
00261             call compute_aggf(results(i,1) , val_aggf, dh=0.0001, h =
00262 results(0,j))
00263             results(i,j) = val_aggf
00264             write (file_unit,'(f20.7,1x,$)' ) results(i,j)
00265         enddo
00266     enddo
00267 enddo
00268 close (file_unit)
00269 end subroutine
00270
00271 ! =====
00273 ! =====
00274 subroutine aggf_resp_t ()
00275     real(dp), dimension(:,:), allocatable :: table , results
00276     integer :: i, j , file_unit
00277     real(dp) :: val_aggf
00278
00279     ! read spherical distances from Merriam
00280     call read_tabulated_green( table )
00281
00282     ! Header in first row with surface temperature [K]
00283     allocate ( results(0 : size (table(:,1)) , 4 ) )
00284     results(0,1) = 1./0
00285     results(0,2) = t0 + 0.
00286     results(0,3) = t0 + 15.0
00287     results(0,4) = t0 + -45.0
00288     do i =1 , size (table(:,1))
00289         results( i , 1 ) = table(i,1)
00290         do j = 2 , 4
00291             call compute_aggf( results(i , 1 ) , val_aggf, dh = 0.00001,
00292 t_zero = results(0, j) )
00293             results(i,j) = val_aggf
00294         enddo
00295     enddo
00296
00297     ! Print results to file
00298     open ( newunit = file_unit , &
00299         file     = '../examples/aggf_resp_t.dat' , &
00300         action   = 'write' )
00301     write (file_unit , '(4F20.5)' ) &
00302         ( results(i,j) , j=1,4) , i = 0, size ( table(:,1) ) )
00303     close (file_unit)
00304 end subroutine
00305 ! =====
00307 ! =====
00308 subroutine aggfdt_resp_dt ()
00309     real(dp), dimension(:,:), allocatable :: table , results
00310     integer :: i, j , file_unit
00311     real(dp) :: val_aggf
00312
00313     ! read spherical distances from Merriam
00314     call read_tabulated_green( table )
00315
00316     ! Header in first row with surface temperature [K]
00317     allocate ( results(0 : size (table(:,1)) , 6 ) )
00318     results(0,1) = 1./0
00319     results(0,2) = 1.
00320     results(0,3) = 5.
00321     results(0,4) = 10.
00322     results(0,5) = 20.
00323     results(0,6) = 50.
00324     do i =1 , size (table(:,1))

```

```

00325     results( i , 1 ) = table(i,1)
00326     do j = 2 , 6
00327     call compute_aggfdt( results(i , 1 ) , val_aggf, results(0, j
) )
00328     results(i,j) = val_aggf
00329     enddo
00330 enddo
00331
00332 ! Print results to file
00333 open ( newunit = file_unit , &
00334       file = '../examples/aggfdt_resp_dt.dat' , &
00335       action = 'write' )
00336 write (file_unit , '(6F20.5)' ) &
00337       ( (results(i,j) , j=1,6) , i = 0, size ( table(:,1) ) )
00338 close (file_unit)
00339 end subroutine
00340
00341 ! =====
00342 ! =====
00343
00344 subroutine aggf_resp_dz ()
00345 real(dp), dimension(:,,:), allocatable :: table , results
00346 integer :: file_unit , i , j
00347 real(dp) :: val_aggf
00348
00349 open ( newunit = file_unit, &
00350       file = '../examples/aggf_resp_dz.dat', &
00351       action='write' )
00352
00353 ! read spherical distances from Merriam
00354 call read_tabulated_green( table )
00355
00356 ! Differences in AGGF(dz) only for small spherical distances
00357 allocate ( results( 0 : 29 , 0: 5 ) )
00358 results = 0.
00359
00360 ! Header in first row [ infty and selected dz follow on ]
00361 results(0,0) = 1./0
00362 results(0,1:5)=(/ 0.0001, 0.001, 0.01, 0.1, 1./)
00363
00364 do i = 1 , size ( results(:,1) ) - 1
00365     results(i,0) = table(i , 1 )
00366     do j = 1 , size (results(1,:)) - 1
00367         call compute_aggf( results(i,0) , val_aggf , dh = results(0,j)
)
00368     results(i, j) = val_aggf
00369     enddo
00370
00371     ! compute relative errors from column 2 for all dz with respect to column 1
00372     results(i,2:) = abs((results(i,2:) - results(i,1)) / results(i,1) * 100 )
00373 enddo
00374
00375 ! write result to file
00376 write ( file_unit , '(<size(results(1,:))>f14.6)' ) &
00377       ((results(i,j), j=0,size(results(1,:)) - 1), i=0,size(results(:,1)) - 1)
00378 close(file_unit)
00379 end subroutine
00380
00381 ! =====
00382 ! =====
00383
00384 subroutine standard1976 !()
00385 real(dp) :: height , temperature , gravity , pressure , pressure2 , density
00386 integer :: file_unit
00387
00388 open ( newunit = file_unit , &
00389       file = '../examples/standard1976.dat', &
00390       action = 'write' )
00391
00392 ! print header
00393 write ( file_unit , '(6(a12))' ) &
00394       'height[km]', 'T[K]', 'g[m/s2]', 'p[hPa]', 'p_simp[hPa]', 'rho[kg/m3]'
00395 do height=0.,98.
00396     call standard_temperature( height , temperature )
00397     call standard_gravity( height , gravity )
00398     call standard_pressure( height , pressure )
00399     call standard_pressure( height , pressure2 ,
if_simplified = .true. )
00400     call standard_density( height , density )
00401     ! print results to file
00402     write( file_unit,'(5f12.5, e12.3)', &
00403           height,temperature , gravity , pressure , pressure2 , density
00404     enddo
00405 close( file_unit )
00406 end subroutine
00407
00408 ! =====
00409 ! =====
00410
00411 subroutine aggf_resp_hmax ()
00412 real (dp) , dimension (10) :: psi

```

```

00416 real (dp) , dimension (:) , allocatable :: heights
00417 real (dp) , dimension (:,:) , allocatable :: results
00418 integer :: file_unit , i , j
00419 real(dp) :: val_aggf
00420
00421 ! selected spherical distances
00422 psi=(/0.000001, 0.000005,0.00001, 1, 2, 3 , 5, 10 , 90 , 180 /)
00423
00424 ! get heights (for nice graph) - call auxiliary subroutine
00425 call aux_heights( heights )
00426
00427 open ( newunit = file_unit , &
00428       file = '../examples/aggf_resp_hmax.dat', &
00429       action = 'write')
00430
00431 allocate ( results( 0:size(heights)-1 , 1+size(psi) ) )
00432
00433 do j=0 , size (results(:,1))
00434     results( j , 1 ) = heights(j)
00435
00436     do i = 1 , size(psi)
00437         call compute_aggf( psi(i) , val_aggf , hmax = heights(j) , dh
00438         = 0.00001 )
00439         results(j,i+1) = val_aggf
00440
00441         if (j.gt.0) then
00442             results(j,i+1) = results(j,i+1) / results(0,i+1) * 100
00443         endif
00444     enddo
00445 enddo
00446
00447 ! print header
00448 write(file_unit , '(a14,SP,100f14.5)' ),"#wys\psi", (psi(j) , j= 1,size(psi))
00449 ! print results
00450 do i=1, size (results(:,1))-1
00451     write(file_unit, '(100f14.3)' ) (results(i,j), j = 1, size(psi)+1 )
00452 enddo
00453 close(file_unit)
00454 end subroutine
00455
00456 ! =====
00458 ! =====
00459 subroutine aux_heights ( table )
00460 real(dp) , dimension (:), allocatable, intent(inout) :: table
00461 real(dp) , dimension (0:1000) :: heights
00462 real(dp) :: height
00463 integer :: i , count_heights
00464
00465 heights(0) =60
00466 i=0
00467 height=-0.001
00468 do while (height.lt.60)
00469     i=i+1
00470     if (height.lt.0.10) then
00471         height=height+2./1000
00472     elseif(height.lt.1) then
00473         height=height+50./1000
00474     else
00475         height=height+1
00476     endif
00477     heights(i)= height
00478     count_heights=i
00479 enddo
00480 allocate ( table( 0 : count_heights ) )
00481 table(0 : count_heights ) = heights( 0 : count_heights )
00482 end subroutine
00483
00484 subroutine aggf_thin_layer ()
00485 integer :: file_unit , i
00486 real(dp) , dimension (:,:), allocatable :: table
00487
00488 ! read spherical distances from Merriam
00489 call read_tabulated_green(table)
00490 do i = 1 , size (table(:,1))
00491     write(*,*) table(i,1:2) , gn_thin_layer(table(i,1))
00492 enddo
00493
00494 end subroutine
00495 end program

```

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

## Functions/Subroutines

- program **grat**

### 6.9.1 Detailed Description

Definition in file [grat.f90](#).

## 6.10 grat.f90

```

00001 !
00026 !
00027 program grat
00028   use iso_fortran_env
00029   use get_cmd_line
00030   use mod_polygon
00031   use mod_data
00032   use mod_green
00033
00034
00035   implicit none
00036 ! character(255) :: dummy
00037 ! real :: del, grav=0. ,cd ,sd, rlato ,rlong , ddist, pole ,cale_pole,
normalizacja, cisnienie_stacja, temperatura_stacja
00038 ! integer :: ii , naz , jj , i , j
00039 ! integer , parameter :: minaz =50 !mrajner 2012-10-03 14:24
00040 ! !integer , parameter :: minaz =1
00041 ! integer , parameter :: ile = 5 !mrajner 2012-10-03 14:24
00042 ! !integer , parameter :: ile = 1
00043 ! real :: azstp, azstdp, azimuth ,caz ,saz,saztp, caztp,stpfac ,cb , sb ,sg
, cg
00044 ! real :: xx
00045 ! real :: grav_merriam_e=0. ,grav_merriam_n=0. , grav_merriam_s=0.
,grav_merriam_e_nib=0.
00046 ! real ::grav_merriam_n_t=0.
00047 ! real ::grav_merriam_n_h=0.
00048 ! real :: admit3
00049 ! real,dimension(85) :: b,c,d
00050 ! integer:: przebieg ,licznik
00051 ! real, dimension(6) :: values_interpolowane
00052 ! real , dimension(:,::), allocatable :: tablica
00053 ! integer :: ile_plikow
00054 ! real :: szerokosc_zmienna , dlugosc_zmienna , wysokosc_stacji_etopo2
00055 ! logical :: czy_otworzyc_nowy_plik=.true.
00056
00057
00058 real(sp) :: x , y , z , lat ,lon ,val !tmp variables
00059 integer :: i , j
00060 integer :: d(6)
00061
00063 call cpu_time(cpu_start)
00064
00065 ! gather cmd line option decide where to put output
00066 call intro( program_calling = "grat" )
00067
00068 ! print header to log: version, date and summary of command line options
00069 call print_settings(program_calling = "grat")
00070
00071
00072 ! read models into memory
00073 do i =1 , size(model)
00074   if (model(i)%if) call read_netcdf( model(i) )
00075 enddo
00076
00077
00078
00079 do j = 1 , size (dates)
00080   call get_variable( model(1) , date = dates(j)%date)
00081
00082   do i = 1 , size(sites)
00083     call get_value(model(1), sites(i)%lat, sites(i)%lon , val)
00084 ! write(output%unit , ' (f15.4,2x,i4,5i2.2,3f13.4)') ,mjd (dates(j)%date)
, dates(j)%date , sites%lat, sites%lon, val
00085 call convolve(sites(1) , green , denserdist = 0 , denseraz =1)
00086   enddo
00087 enddo
00088

```

```
00089  ! todo wysokosci nad wodą ustaw na 0. Głębokość nie jest interesująca
00090
00091
00092
00093
00094
00095  call cpu_time(cpu_finish)
00096  write(log%unit, '(/"Execution time:",1x,f16.9," seconds")') cpu_finish -
    cpu_start
00097  write(log%unit, form_separator)
00098
00099  end program
```



## Appendix A

### Polygon

This examples show how the exclusion of selected polygons works

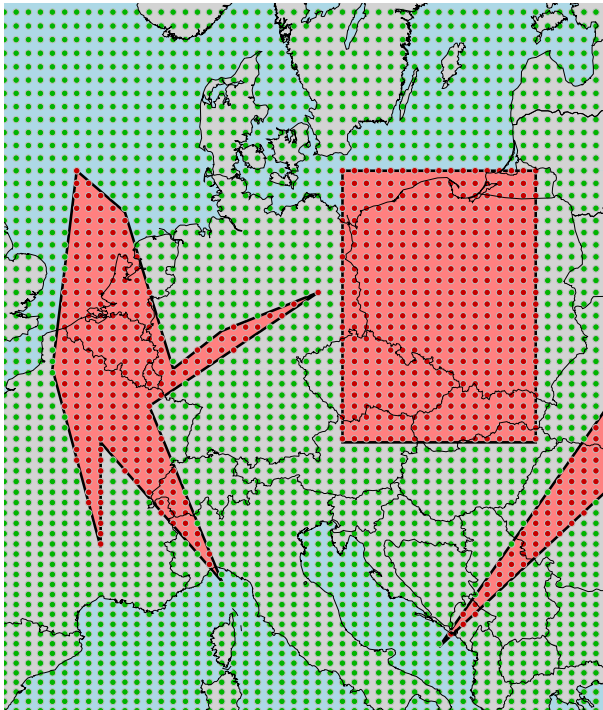


Figure A.1: If only excluded polygons (red area) are given all points falling in it will be excluded (red points) all other will be included

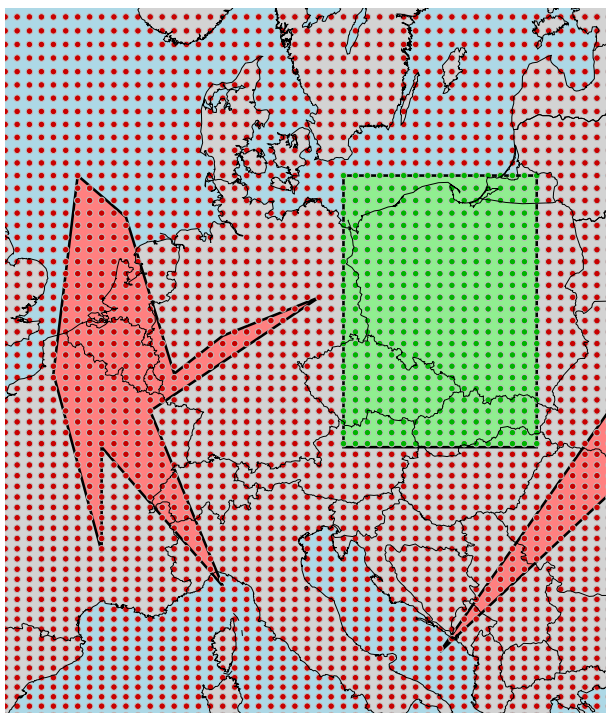


Figure A.2: If at least one included area is given (green area) then all points which do not fall into included area will be excluded

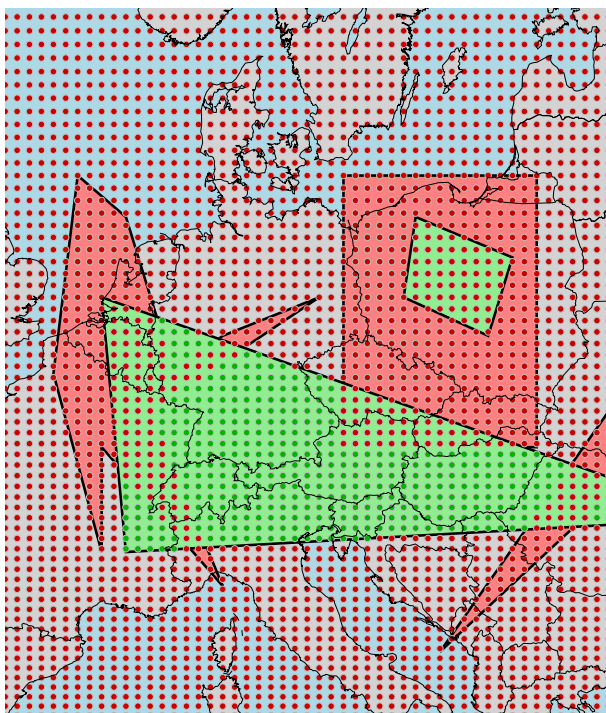


Figure A.3: If there is overlap of polygons the exclusion has higher priority



## Appendix B

# Interpolation



# Bibliography

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

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

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

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

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

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

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

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

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

# Index

/home/mrajner/src/grat/polygon/polygon\_map.sh, 25, 26  
/home/mrajner/src/grat/src/aggf.f90, 28  
/home/mrajner/src/grat/src/constants.f90, 34  
/home/mrajner/src/grat/src/example\_aggf.f90, 38, 40  
/home/mrajner/src/grat/src/grat.f90, 45, 46

aggf, 9  
    bouger, 10  
    compute\_aggf, 10  
    compute\_aggfdt, 10  
    gn\_thin\_layer, 11  
    read\_tabulated\_green, 11  
    simple\_def, 11  
    size\_ntimes\_denser, 11  
    standard\_density, 11  
    standard\_gravity, 12  
    standard\_pressure, 12  
    standard\_temperature, 12

aux\_heights  
    example\_aggf.f90, 39

bouger  
    aggf, 10

check  
    mod\_data, 20

chkgon  
    mod\_polygon, 22

compare\_fels\_profiles  
    example\_aggf.f90, 39

compute\_aggf  
    aggf, 10

compute\_aggfdt  
    aggf, 10

constants, 13  
    ispline, 14  
    jd, 14  
    spline, 14  
    spline\_interpolation, 15

example\_aggf.f90  
    aux\_heights, 39  
    compare\_fels\_profiles, 39  
    simple\_atmospheric\_model, 39  
    standard1976, 40

get\_cmd\_line, 16  
    intro, 18  
    is\_numeric, 18  
    parse\_dates, 18  
    read\_site\_file, 19

get\_cmd\_line::additional\_info, 9  
get\_cmd\_line::cmd\_line, 13  
get\_cmd\_line::dateandmjd, 15  
get\_cmd\_line::file, 15  
get\_cmd\_line::green\_functions, 19  
get\_cmd\_line::polygon\_data, 22  
get\_cmd\_line::polygon\_info, 23  
get\_cmd\_line::site\_data, 23  
get\_value  
    mod\_data, 20  
gn\_thin\_layer  
    aggf, 11

intro  
    get\_cmd\_line, 18

is\_numeric  
    get\_cmd\_line, 18

ispline  
    constants, 14

jd  
    constants, 14

last  
    polygon\_map.sh, 26

mapcon\_util, 19

mod\_data, 20  
    check, 20  
    get\_value, 20  
    put\_grd, 20  
    unpack\_netcdf, 21

mod\_green, 21

mod\_polygon, 21  
    chkgon, 22  
    ncross, 22  
    read\_polygon, 22

ncross  
    mod\_polygon, 22

parse\_dates  
    get\_cmd\_line, 18

polygon\_map.sh  
    last, 26

put\_grd  
    mod\_data, 20

read\_polygon  
    mod\_polygon, 22

read\_site\_file

- get\_cmd\_line, 19
- read\_tabulated\_green
  - aggf, 11
- simple\_atmospheric\_model
  - example\_aggf.f90, 39
- simple\_def
  - aggf, 11
- size\_ntimes\_denser
  - aggf, 11
- spline
  - constants, 14
- spline\_interpolation
  - constants, 15
- standard1976
  - example\_aggf.f90, 40
- standard\_density
  - aggf, 11
- standard\_gravity
  - aggf, 12
- standard\_pressure
  - aggf, 12
- standard\_temperature
  - aggf, 12
- unpack\_netcdf
  - mod\_data, 21