grat Manual

Marcin Rajner
Politechnika Warszawska | Warsaw University of Technology

Contents

1 grat overview **1**

1.1 Purpose . . . 1

1.2 Usage . . . 1

2 grat **3**

3 ilustration **5**

4 External resources **9**

5 polygon_check **11**

6 value_check **13**

7 Todo List **15**

8 Data Type Index **17**

8.1 Data Types List . . . 17

9 File Index **19**

9.1 File List . . . 19

10 Data Type Documentation **21**

10.1 mod_cmdline::additional_info Type Reference . . . 21

10.1.1 Detailed Description . . . 21

10.2 mod_cmdline::cmd_line Type Reference . . . 21

10.2.1 Detailed Description . . . 21

10.3 mod_cmdline::dateandmjd Type Reference . . . 22

10.3.1 Detailed Description . . . 22

10.4 mod_cmdline::file Type Reference . . . 22

10.4.1 Detailed Description . . . 22

10.5 mod_cmdline::green_functions Type Reference . . . 23

10.5.1 Detailed Description . . . 23

10.6 mod_aggf Module Reference . . . 23

10.6.1 Detailed Description . . . 23

10.6.2 Member Function/Subroutine Documentation . . . 24
10.6.2.1 bouger . . . 24
10.6.2.2 compute_aggf . . . 24
10.6.2.3 compute_aggfdt . . . 24
10.6.2.4 geop2geom . . . 25
10.6.2.5 gn_thin_layer . . . 25
10.6.2.6 read_tabulated_green . . . 25
10.6.2.7 simple_def . . . 25
10.6.2.8 size_ntimes_denser . . . 26
10.6.2.9 standard_density . . . 26
10.6.2.10 standard_gravity . . . 26
10.6.2.11 standard_pressure . . . 26
10.6.2.12 standard_temperature . . . 26
10.7 mod_cmdline Module Reference . . . 27
10.7.1 Detailed Description . . . 29
10.7.2 Member Function/Subroutine Documentation . . . 29
10.7.2.1 check_if_switch_or_minus . . . 29
10.7.2.2 count_separator . . . 29
10.7.2.3 dataname . . . 29
10.7.2.4 if_minimum_args . . . 30
10.7.2.5 intro . . . 30
10.7.2.6 mod_cmdline_entry . . . 30
10.7.2.7 nmodels . . . 31
10.7.2.8 parse_dates . . . 31
10.7.2.9 parse_green . . . 31
10.7.2.10 print_version . . . 31
10.7.2.11 read_site_file . . . 32
10.7.2.12 string2date . . . 32
10.8 mod_constants Module Reference . . . 32
10.8.1 Detailed Description . . . 33
10.9 mod_data Module Reference . . . 33
10.9.1 Detailed Description . . . 33
10.9.2 Member Function/Subroutine Documentation . . . 34
10.9.2.1 check . . . 34
10.9.2.2 nctime2date . . . 34
10.9.2.3 unpack_netcdf . . . 34
10.10 mod_green Module Reference . . . 34
10.10.1 Detailed Description . . . 35
10.10.2 Member Function/Subroutine Documentation . . . 35
10.10.2.1 convolve . . . 35

10.10.2.2 getgrf . . . 35
10.10.2.3 spher_area . . . 36
10.11 mod_polygon Module Reference . . . 36
10.11.1 Detailed Description . . . 36
10.11.2 Member Function/Subroutine Documentation . . . 36
10.11.2.1 chkgon . . . 36
10.11.2.2 ncross . . . 37
10.11.2.3 read_polygon . . . 37
10.12 mod_utilities Module Reference . . . 37
10.12.1 Detailed Description . . . 38
10.12.2 Member Function/Subroutine Documentation . . . 38
10.12.2.1 count_records_to_read . . . 38
10.12.2.2 d2r . . . 38
10.12.2.3 file_exists . . . 39
10.12.2.4 invmjd . . . 39
10.12.2.5 is_numeric . . . 39
10.12.2.6 ispline . . . 39
10.12.2.7 jd . . . 40
10.12.2.8 mjd . . . 40
10.12.2.9 ntokens . . . 40
10.12.2.10 r2d . . . 41
10.12.2.11 spher_trig . . . 41
10.12.2.12 spher_trig_inverse . . . 41
10.12.2.13 spline . . . 42
10.12.2.14 spline_interpolation . . . 42
10.13 mod_cmdline::polygon_data Type Reference . . . 42
10.13.1 Detailed Description . . . 42
10.14 mod_cmdline::polygon_info Type Reference . . . 42
10.14.1 Detailed Description . . . 43
10.15 mod_green::result Type Reference . . . 43
10.15.1 Detailed Description . . . 43
10.16 mod_cmdline::site_data Type Reference . . . 43
10.16.1 Detailed Description . . . 43

11 File Documentation **45**

11.1 grat/doc/interpolation_ilustration.sh File Reference . . . 45
11.1.1 Detailed Description . . . 45
11.1.2 Variable Documentation . . . 45
11.1.2.1 interp . . . 45
11.2 interpolation_ilustration.sh . . . 45

11.3 grat/src/grat.f90 File Reference . . . 46
11.3.1 Detailed Description . . . 46
11.4 grat.f90 . . . 46
11.5 grat/src/mod_aggf.f90 File Reference . . . 48
11.5.1 Detailed Description . . . 48
11.6 mod_aggf.f90 . . . 48
11.7 grat/src/mod_cmdline.f90 File Reference . . . 55
11.7.1 Detailed Description . . . 55
11.8 mod_cmdline.f90 . . . 55
11.9 grat/src/mod_green.f90 File Reference . . . 69
11.9.1 Detailed Description . . . 69
11.10 mod_green.f90 . . . 70
11.11 grat/src/real_vs_standard.f90 File Reference . . . 75
11.11.1 Detailed Description . . . 75
11.12 real_vs_standard.f90 . . . 75
11.13 grat/src/value_check.f90 File Reference . . . 76
11.13.1 Detailed Description . . . 77
11.14 value_check.f90 . . . 77
11.15 grat/tmp/compar.sh File Reference . . . 78
11.15.1 Detailed Description . . . 78
11.16 compar.sh . . . 78

12 Example Documentation **81**
12.1 example_aggf.f90 . . . 81
12.2 grat_usage.sh . . . 87

A Polygon **89**

B Interpolation **91**

Chapter 1

grat overview

1.1 Purpose

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

Version

TESTING!

Date

2013-01-12

Author

Marcin Rajner
Politechnika Warszawska | Warsaw University of Technology

Warning

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

Attention

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

1.2 Usage

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

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

- grat

- value_check
- polygon_check

Chapter 2

grat

```
grat [-h] [-v] [-S[[site_name],latitude,longitude[,height]]|[sites_file]|[
      Rlonmin/lonmax/latmin/latmax[,lonresolution[,latresolution]]]] [-V[log_file]] [-L[
      filename]:what,[filename2]:what] [-Ppolygon_file[:+-][,polygon_file[:+-]]] [-I[1|2]
      ] [-I[1|2]] [-G todo] [-D[yyyy[mm[dd[hh[mm[ss]]]]][,yyy[,interval]]]] [-Q[+-]]

Summary of available options for program grat
  -h  help
      -h
      prints summary of available option and exit
      optional parameter
      default: help=.false.

  -v  version
      -v
      print version and author and exit
      optional parameter
      default: version=.false.

  -S  site coordinates
      -S[[site_name],latitude,longitude[,height]]|[sites_file]|[Rlonmin/lonmax/
      latmin/latmax[,lonresolution[,latresolution]]]
      you can give information about sites you want include in computation in
      three different ways
        1 -S [site_name], lat , lon , height
            example:
              -S JOZE, 52.1, 21.3 , 110
            or
              -S , 52.1, 21.3
        2 -S file_name
            where in the file you put space separated: name lat lon [heihght]
            all records with bad specification will be ignored
        3 -S Rlonmin/lonmax/latmin/latmax[,lonresolution]
      lat in decimal degrees (+ north | - south)
      lon in decimal degrees <-180,360)
      height in meters (orthometric)
      obligatory parameter
      default: height=0

  -V  verbose
      -V[log_file]
      prints settings to log_file if specified or to STDOUT
      default: verbose=.false.

  -L  more verbose
      -L[filename]:what,[filename2]:what
      prints out additional information depending on specification
      optional parameter
      default: moreverbose=.false.
      fields: n - nearest
              b - bilinear
              s - statistic (short)
              G - greens function

  -P  polygon(s)
      -Ppolygon_file[:+-][,polygon_file[:+-]]
        you can overrid settings in polygon file
        -P polygon_file : +
      optional parameter

  -I  interpolation
      -I[1|2]
      specify the interpolation scheme for data
      Default: -I1
```

```
    optional parameter

-F  todo
    @dataname
      SP surface  pressure
      VP vertical pressure
      LS landsea  mask

-G  green functions
    -G todo

    optional parameter
    default: green function from Merriam 1992 !todo

-D  specify dates
    -D[yyyy[mm[dd[hh[mm[ss]]]]][,yyy[,interval]]]
    specify date
      -D 20110304050600
    or dates range and interval [hours]
      -D 20110304050600 , 201105 , 6
    If you ommit part of date specification the programm assumes as follow
      month=01; day=01 hour=00; minute=00; second=00;
      therfore
      -D 201204 , 2013
      is equal to
      -D 201204000000, 20130101000000, 6
    you can select reverse order
      -D 20110304050600   , 201105           , -6
      -D  201105          , 20110304050600   ,  6
    Default: first time field in data , for interval 6 hours
    optional parameter

-Q  use reference pressure
    -Q[+-]
```

Chapter 3

ilustration

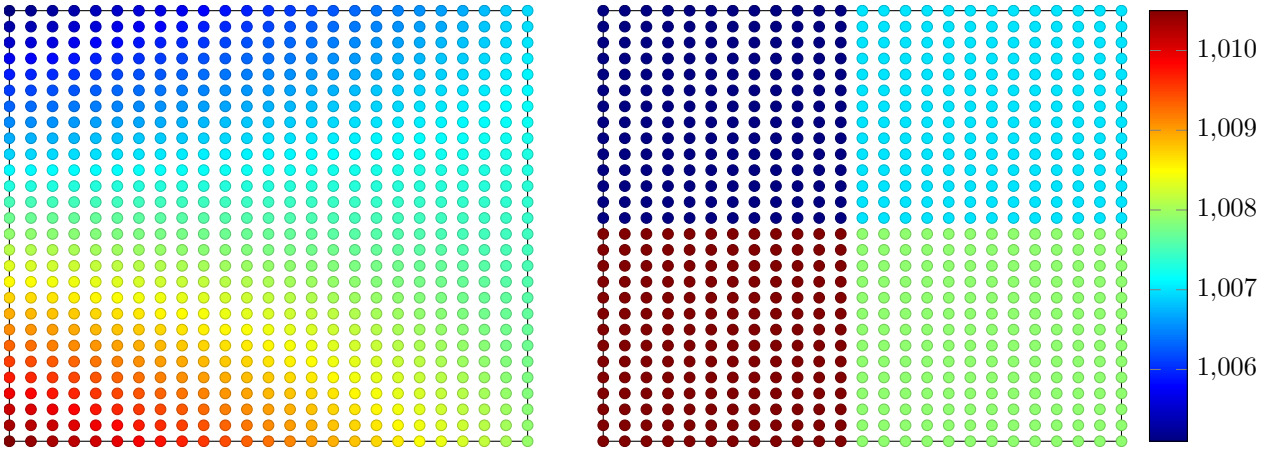
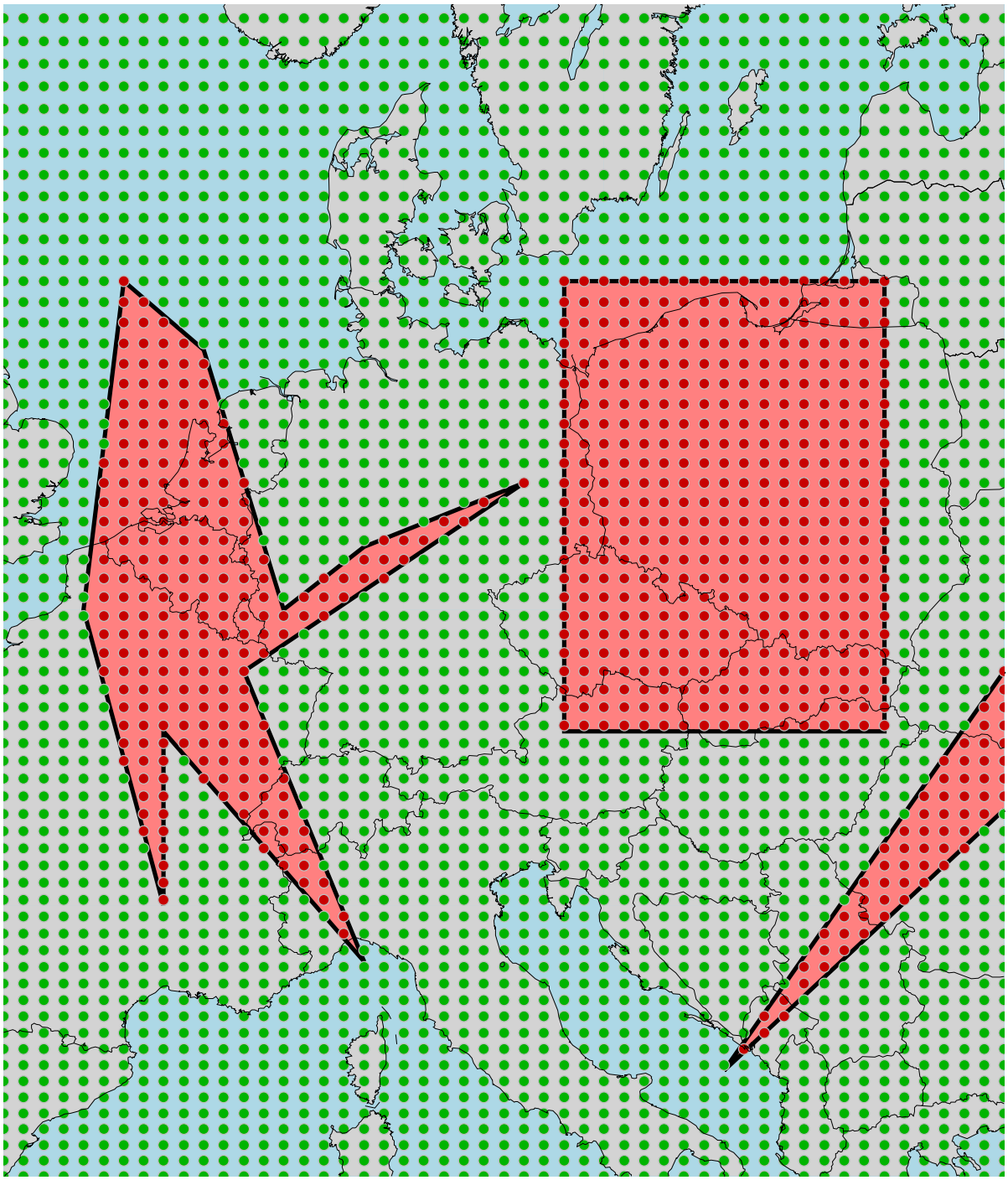
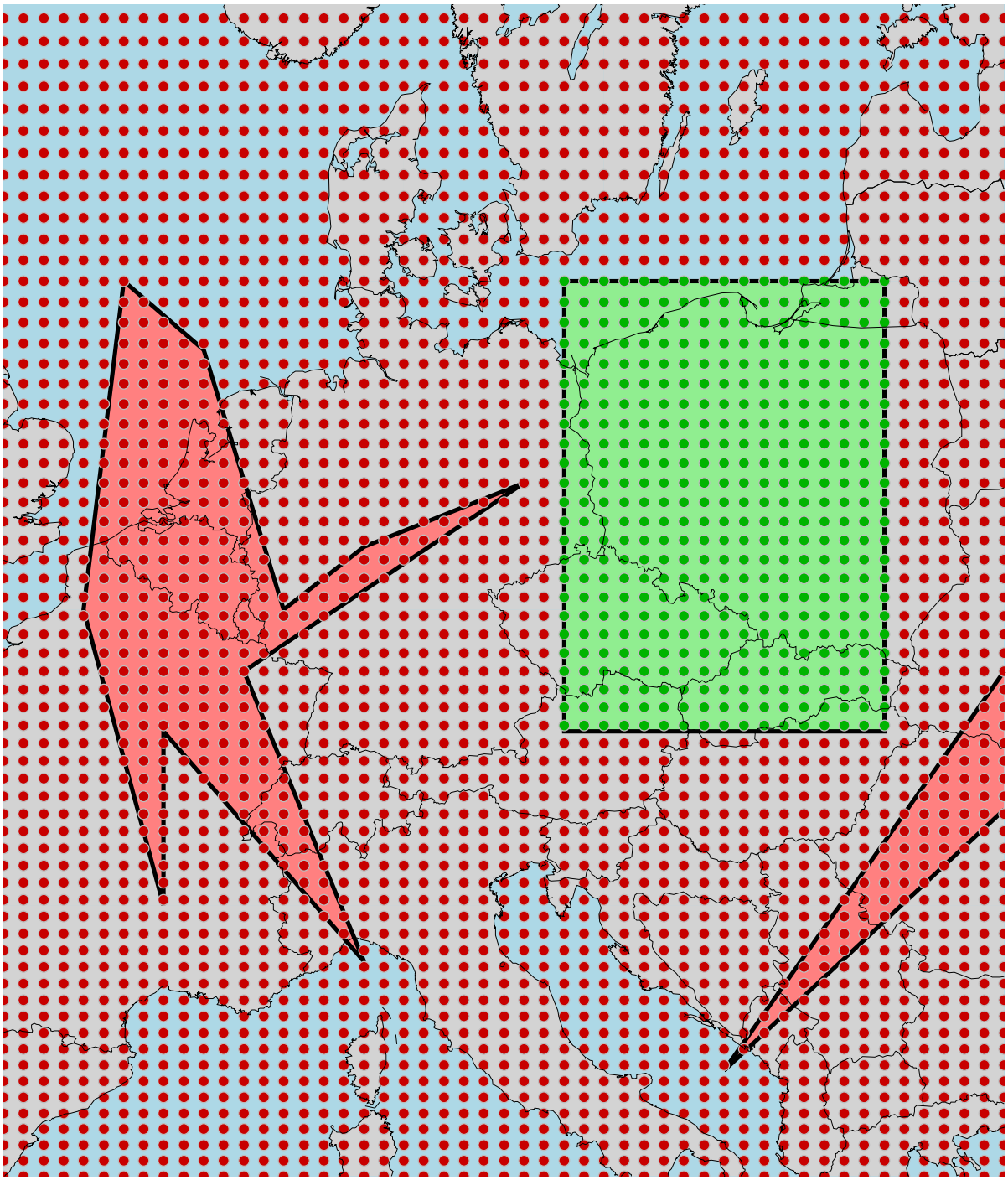
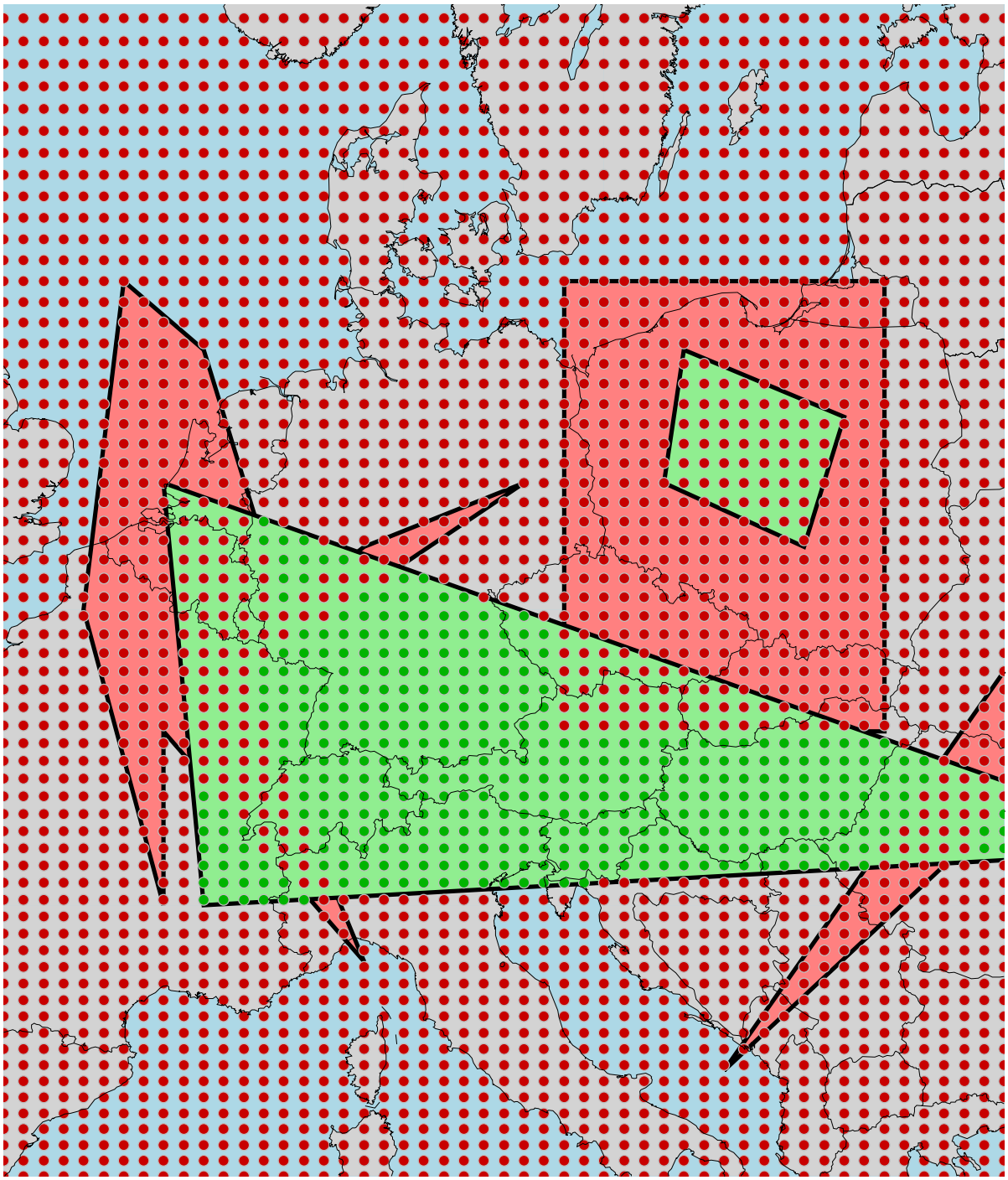


Figure 3.1: example







Chapter 4

External resources

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

Chapter 5

polygon_check

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

```
polygon_check [-h] [-v] [-S[[site_name],latitude,longitude[,height]]|[
      sites_file]|[Rlonmin/lonmax/latmin/latmax[,lonresolution[,latresolution]]]] [-V[log_file]
      ] [-L[filename]:what,[filename2]:what] [-Ppolygon_file[:+-][,polygon_file[:+-]]]
       [-I[1|2]]

Summary of available options for program polygon_check
  -h  help
      -h
      prints summary of available option and exit
      optional parameter
      default: help=.false.

  -v  version
      -v
      print version and author and exit
      optional parameter
      default: version=.false.

  -S  site coordinates
      -S[[site_name],latitude,longitude[,height]]|[sites_file]|[Rlonmin/lonmax/
      latmin/latmax[,lonresolution[,latresolution]]]
      you can give information about sites you want include in computation in
      three different ways
        1 -S [site_name], lat , lon , height
            example:
              -S JOZE, 52.1, 21.3 , 110
            or
              -S , 52.1, 21.3
        2 -S file_name
            where in the file you put space separated: name lat lon [heihght]
            all records with bad specification will be ignored
        3 -S Rlonmin/lonmax/latmin/latmax[,lonresolution]
      lat in decimal degrees (+ north | - south)
      lon in decimal degrees <-180,360)
      height in meters (orthometric)
      obligatory parameter
      default: height=0

  -V  verbose
      -V[log_file]
      prints settings to log_file if specified or to STDOUT
      default: verbose=.false.

  -L  more verbose
      -L[filename]:what,[filename2]:what
      prints out additional information depending on specification
      optional parameter
      default: moreverbose=.false.
      fields: n - nearest
              b - bilinear
              s - statistic (short)
              G - greens function

  -P  polygon(s)
      -Ppolygon_file[:+-][,polygon_file[:+-]]
        you can overrid settings in polygon file
        -P polygon_file : +
      obligatory parameter

  -I  interpolation
      -I[1|2]
```

```
specify the interpolation scheme for data
Default: -I1
optional parameter
```

Chapter 6

value_check

```
value_check [-h] [-v] [-S[[site_name],latitude,longitude[,height]]|[sites_file]
      |[Rlonmin/lonmax/latmin/latmax[,lonresolution[,latresolution]]]] [-V[log_file]]
      [-L[filename]:what,[filename2]:what] [-Ppolygon_file[:+-][,polygon_file[:+-]]] [
      -I[1|2]] [-I[1|2]] [-D[yyyy[mm[dd[hh[mm[ss]]]]][,yyy[,interval]]]]

Summary of available options for program value_check
  -h  help
      -h
      prints summary of available option and exit
      optional parameter
      default: help=.false.

  -v  version
      -v
      print version and author and exit
      optional parameter
      default: version=.false.

  -S  site coordinates
      -S[[site_name],latitude,longitude[,height]]|[sites_file]|[Rlonmin/lonmax/
      latmin/latmax[,lonresolution[,latresolution]]]
      you can give information about sites you want include in computation in
      three different ways
        1 -S [site_name], lat , lon , height
            example:
              -S JOZE, 52.1, 21.3 , 110
            or
              -S , 52.1, 21.3
        2 -S file_name
            where in the file you put space separated: name lat lon [heihght]
            all records with bad specification will be ignored
        3 -S Rlonmin/lonmax/latmin/latmax[,lonresolution]
      lat in decimal degrees (+ north | - south)
      lon in decimal degrees <-180,360)
      height in meters (orthometric)
      obligatory parameter
      default: height=0

  -V  verbose
      -V[log_file]
      prints settings to log_file if specified or to STDOUT
      default: verbose=.false.

  -L  more verbose
      -L[filename]:what,[filename2]:what
      prints out additional information depending on specification
      optional parameter
      default: moreverbose=.false.
      fields: n - nearest
              b - bilinear
              s - statistic (short)
              G - greens function

  -P  polygon(s)
      -Ppolygon_file[:+-][,polygon_file[:+-]]
        you can overrid settings in polygon file
        -P polygon_file : +

  -I  interpolation
      -I[1|2]
      specify the interpolation scheme for data
      Default: -I1
      optional parameter
```

```
-F  todo
    @dataname
      SP surface  pressure
      VP vertical pressure
      LS landsea  mask

-D  specify dates
    -D[yyyy[mm[dd[hh[mm[ss]]]]][,yyy[,interval]]]
    specify date
      -D 20110304050600
    or dates range and interval [hours]
      -D 20110304050600 , 201105 , 6
    If you ommit part of date specification the programm assumes as follow
      month=01; day=01 hour=00; minute=00; second=00;
      therfore
      -D 201204 , 2013
      is equal to
      -D 201204000000, 20130101000000, 6
    you can select reverse order
      -D 20110304050600   , 201105          , -6
      -D  201105          , 20110304050600   ,  6
    Default: first time field in data , for interval 6 hours
    optional parameter
```

Date

2013-01-09

Author

M. Rajner

Date

2013-03-19 added -P (if point is excluded all values are zero)

Chapter 7

Todo List

Subprogram mod_cmdline::if_minimum_args (program_calling)

Make it compact (if errror ....)

Subprogram mod_cmdline::parse_green (cmd_line_entry)

add maximum minimum distances for integration

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

when no given take defaults

rozbudować

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

mjd!

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

find url Original description below: ==========
Calculate the coefficients b(i), c(i), and d(i), i=1,2,...,n for cubic spline interpolation s(x) = y(i) + b(i)∗(x-x(i)) + c(i)∗(x-x(i))∗∗2 + d(i)∗(x-x(i))∗∗3 for x(i) <= x <= x(i+1) Alex G: January 2010

File real_vs_standard.f90

put description

Chapter 8

Data Type Index

8.1 Data Types List

Here are the data types with brief descriptions:

mod_cmdline::additional_info . . . 21
mod_cmdline::cmd_line . . . 21
mod_cmdline::dateandmjd . . . 22
mod_cmdline::file . . . 22
mod_cmdline::green_functions . . . 23
mod_aggf . . . 23
mod_cmdline . . . 27
mod_constants
    Define constant values . . . 32
mod_data
    This modele gives routines to read, and write data . . . 33
mod_green . . . 34
mod_polygon
    Some routines to deal with inclusion or exclusion of polygons . . . 36
mod_utilities . . . 37
mod_cmdline::polygon_data . . . 42
mod_cmdline::polygon_info . . . 42
mod_green::result . . . 43
mod_cmdline::site_data . . . 43

Chapter 9

File Index

9.1 File List

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

grat/**mapa.sh** . . . . . **??**
grat/dat/**help.hlp** . . . . . **??**
grat/doc/**grat.hlp** . . . . . **??**
grat/doc/interpolation_ilustration.sh 45
grat/doc/**polygon_check.hlp** . . . . . **??**
grat/doc/**polygon_ilustration.sh** . . . . . **??**
grat/doc/**value_check.hlp** . . . . . **??**
grat/examples/**example_aggf.f90** . . . . . **??**
grat/examples/**grat_usage.sh** . . . . . **??**
grat/polygon/**baltyk.sh** . . . . . **??**
grat/polygon/**polygon_map.sh** . . . . . **??**
grat/src/**.obsolete.f90** . . . . . **??**
grat/src/**barometric_formula.f90** . . . . . **??**
grat/src/grat.f90 46
grat/src/**joinnc.f90** . . . . . **??**
grat/src/mod_aggf.f90
This module contains utitlities for computing Atmospheric Gravity Green Functions 48
grat/src/mod_cmdline.f90
This module sets the initial values for parameters reads from command line and gives help it allows to specify commands with or without spaces therefore it is convienient to use with auto completion of names 55
grat/src/**mod_constants.f90** . . . . . **??**
grat/src/**mod_data.f90** . . . . . **??**
grat/src/mod_green.f90
Module 70
grat/src/**mod_polygon.f90** . . . . . **??**
grat/src/**mod_utilities.f90** . . . . . **??**
grat/src/**polygon_check.f90** . . . . . **??**
grat/src/real_vs_standard.f90
This program 75
grat/src/value_check.f90 77
grat/tmp/compar.sh 78

Chapter 10

Data Type Documentation

10.1 mod_cmdline::additional_info Type Reference

Public Attributes

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

10.1.1 Detailed Description

Definition at line 60 of file mod_cmdline.f90.

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

- grat/src/mod_cmdline.f90

10.2 mod_cmdline::cmd_line Type Reference

Collaboration diagram for mod_cmdline::cmd_line:

Public Attributes

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

10.2.1 Detailed Description

Definition at line 63 of file mod_cmdline.f90.

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

- grat/src/mod_cmdline.f90

10.3 mod_cmdline::dateandmjd Type Reference

Public Attributes

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

10.3.1 Detailed Description

Definition at line 48 of file mod_cmdline.f90.

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

- grat/src/mod_cmdline.f90

10.4 mod_cmdline::file Type Reference

Public Attributes

- character(:), allocatable **name**
- character(len=50), dimension(5) **names** = ["z"
- character(len=40) **dataname**
- integer **unit** = output_unit
- logical **if** = .false.
- logical **first_call** = .true.
- real(dp), dimension(4) **limits**
- real(dp), dimension(:), allocatable **lat**
- real(dp), dimension(:), allocatable **lon**
- real(dp), dimension(:), allocatable **time**
- real(dp), dimension(:), allocatable **level**
- integer, dimension(:,:),
  allocatable **date**
- real(dp), dimension(2) **latrange**
- real(dp), dimension(2) **lonrange**
- logical **if_constant_value**
- real(dp) **constant_value**
- real(dp), dimension(:,:,:),
  allocatable data

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

10.4.1 Detailed Description

Definition at line 94 of file mod_cmdline.f90.

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

- grat/src/mod_cmdline.f90

10.5 mod_cmdline::green_functions Type Reference

Public Attributes

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

10.5.1 Detailed Description

Definition at line 18 of file mod_cmdline.f90.

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

- grat/src/mod_cmdline.f90

10.6 mod_aggf Module Reference

Public Member Functions

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

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

  Wczytuje tablice danych AGGF.
- subroutine, public 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, public standard_density (height, rho, t_zero, fels_type)

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

  Computes pressure [hPa] for specific height.
- subroutine, public standard_gravity (height, g)

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

  Compute geometric height from geopotential heights.
- subroutine, public standard_temperature (height, temperature, t_zero, fels_type)

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

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

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

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

  Bouger plate computation.

10.6.1 Detailed Description

Definition at line 9 of file mod_aggf.f90.

10.6.2 Member Function/Subroutine Documentation

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

Bouger plate computation.

Parameters

r_opt	height of point above the cylinder

Definition at line 464 of file mod_aggf.f90.

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

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

Parameters

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

Parameters

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

Definition at line 115 of file mod_aggf.f90.

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

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

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

Warning

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

Author

M. Rajner

Date

2013-03-19

Definition at line 35 of file mod_aggf.f90.

10.6.2.4 real(dp) function, public mod_aggf::geop2geom (real (dp) *geopotential_height*)

Compute geometric height from geopotential heights.

Author

M. Rajner

Date

2013-03-19

Definition at line 296 of file mod_aggf.f90.

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

Compute AGGF GN for thin layer.

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

Author

M. Rajner

Date

2013-03-19

Definition at line 439 of file mod_aggf.f90.

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

Wczytuje tablice danych AGGF.

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

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

Definition at line 70 of file mod_aggf.f90.

10.6.2.7 real(dp) function, public mod_aggf::simple_def (real(dp) *R*)

Bouger plate computation.

see eq. page 288 Warburton and Goodkind [1977]

Date

2013-03-18

Author

M. Rajner

Definition at line 489 of file mod_aggf.f90.

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

returns numbers of arguments for n times denser size

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

Definition at line 454 of file mod_aggf.f90.

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

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

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

using formulae 12 in Huang et al. [2005]

Date

2013-03-18

Author

M. Rajner

Parameters

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

Definition at line 201 of file mod_aggf.f90.

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

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

see Comitee on extension of the Standard Atmosphere [1976]

Definition at line 281 of file mod_aggf.f90.

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

Computes pressure [hPa] for specific height.

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

Definition at line 224 of file mod_aggf.f90.

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

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

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

- US standard atmosphere (default)
- tropical
- subtropical_summer
- subtropical_winter
- subarctic_summer
- subarctic_winter

Definition at line 358 of file mod_aggf.f90.

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

- grat/src/mod_aggf.f90

10.7 mod_cmdline Module Reference

Collaboration diagram for mod_cmdline:

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, accepted_switch)

  This subroutine counts the command line arguments.
- logical function check_if_switch_or_minus (dummy)

  Check if - starts new option in command line or is just a minus in command line entry.
- subroutine if_minimum_args (program_calling)

  Check if at least all obligatory command line arguments were given if not print error and exit.
- 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 – Greens function.
- integer function count_separator (dummy, separator)

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

  This subroutine fills the fields of command line entry for every input arg.

- subroutine get_model_info (model, cmd_line_entry, field)

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

  P.
- 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)

  Convert dates given as string to integer (6 elements)
- subroutine **sprawdzdate** (mjd)
- 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.
- character(len=40) function dataname (abbreviation)

  Attach full dataname by abbreviation.

Public Attributes

- type(green_functions), dimension(:), allocatable **green**
- integer, dimension(2) **denser** = [1
- type(polygon_info), dimension(2) **polygons**
- real(dp) **cpu_start**
- real(dp) **cpu_finish**
- type(dateandmjd), dimension(:), allocatable **dates**
- type(site_data), dimension(:), allocatable **sites**
- integer fileunit_tmp

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

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

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

- character(50), dimension(2) **interpolation_names** = ["nearest"
- character(len=255), parameter **form_header** = '(60("#"))'
- character(len=255), parameter **form_separator** = '("#",59("-"))'
- character(len=255), parameter **form_inheader** = '(("#"),1x,a56,1x,("#"))'
- character(len=255), parameter **form_60** = "(a,100(1x,g0))"
- character(len=255), parameter **form_61** = "(2x,a,100(1x,g0))"
- character(len=255), parameter **form_62** = "(4x,a,100(1x,g0))"
- character(len=255), parameter **form_63** = "(6x,100(x,g0))"
- character(len=255), parameter **form_64** = "(4x,4x,a,4x,a)"

10.7.1 Detailed Description

Definition at line 8 of file mod_cmdline.f90.

10.7.2 Member Function/Subroutine Documentation

10.7.2.1 logical function mod_cmdline::check_if_switch_or_minus (character(∗) *dummy*)

Check if - starts new option in command line or is just a minus in command line entry.

if after '-' is space or number or ',' or ':' (field separators) do not start next option for command line If switch return .true. otherwise return .false

Author

M. Rajner

Date

2013-03-19

Definition at line 241 of file mod_cmdline.f90.

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

change the paths accordingly

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

Definition at line 577 of file mod_cmdline.f90.

10.7.2.3 character(len=40) function mod_cmdline::dataname (character(len=2) *abbreviation*)

Attach full dataname by abbreviation.

Date

2013-03-21

Author

M. Rajner

Definition at line 1161 of file mod_cmdline.f90.

10.7.2.4 subroutine mod_cmdline::if_minimum_args (character (∗), intent(in) *program_calling*)

Check if at least all obligatory command line arguments were given if not print error and exit.

Date

2013-03-15

Author

M. Rajner

Todo Make it compact (if errror ....)

Definition at line 262 of file mod_cmdline.f90.

10.7.2.5 subroutine mod_cmdline::intro (character(len=∗), intent(in) *program_calling,* character(len=∗), intent(in), optional *accepted_switch*)

This subroutine counts the command line arguments.

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

Date

2012-12-20

Author

M. Rajner

Date

2013-03-19 parsing negative numbers after space fixed (-S -11... was previously treated as two cmmand line entries, now only -? non-numeric terminates input argument)

Definition at line 180 of file mod_cmdline.f90.

10.7.2.6 subroutine mod_cmdline::mod_cmdline_entry (character(∗) *dummy,* type (cmd_line), intent(out) *cmd_line_entry,* character(len=∗), intent(in), optional *program_calling*)

This subroutine fills the fields of command line entry for every input arg.

Author

M. Rajner

Date

2013-03-21

Definition at line 603 of file mod_cmdline.f90.

10.7.2.7 integer function mod_cmdline::nmodels (type(file), dimension (:), allocatable *model*)

Counts number of properly specified models.

Date

2013-03-15

Author

M. Rajner

Definition at line 1144 of file mod_cmdline.f90.

10.7.2.8 subroutine mod_cmdline::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 864 of file mod_cmdline.f90.

10.7.2.9 subroutine mod_cmdline::parse_green (type (cmd_line) *cmd_line_entry*)

This subroutine parse -G option – Greens function.

Todo add maximum minimum distances for integration

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

Todo when no given take defaults

Todo rozbudować

This subroutines takes the -G argument specified as follows: -G

Author

M. Rajner

Date

2013-03-06

Definition at line 475 of file mod_cmdline.f90.

10.7.2.10 subroutine mod_cmdline::print_version (character(*) *program_calling*)

Print version of program depending on program calling.

Author

M. Rajner

Date

2013-03-06

Definition at line 962 of file mod_cmdline.f90.

10.7.2.11 subroutine mod_cmdline::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 777 of file mod_cmdline.f90.

10.7.2.12 subroutine mod_cmdline::string2date (character (*), intent(in) *string,* integer, dimension(6), intent(out) *date*)

Convert dates given as string to integer (6 elements)

20110612060302 –> [2011 , 6 , 12 , 6 , 3 , 2 you can omit

Warning

decimal seconds are not allowed

Definition at line 909 of file mod_cmdline.f90.

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

- grat/src/mod_cmdline.f90

10.8 mod_constants Module Reference

Define constant values.

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]
- real(dp) **earth_mass** = 5.97219e24
- real(dp) **geocentric_constant** = 398600.4419

10.8.1 Detailed Description

Define constant values.

This module define some constant values oftenly used.

Author

M. Rajner

Date

2013-03-04

Definition at line 8 of file mod_constants.f90.

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

- grat/src/mod_constants.f90

10.9 mod_data Module Reference

This modele gives routines to read, and write data.

Public Member Functions

- subroutine, public read_netcdf (model)
  Read netCDF file into memory.
- subroutine, public 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, public get_value (model, lat, lon, val, level, method)
  Returns the value from model file.
- real(dp) function **bilinear** (x, y, aux)

10.9.1 Detailed Description

This modele gives routines to read, and write data.

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

Author

M. Rajner

Date

2013-03-04

Definition at line 12 of file mod_data.f90.

10.9.2 Member Function/Subroutine Documentation

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

Check the return code from netCDF manipulation.

Author

From netcdf website net

Date

2013-03-04

Definition at line 233 of file mod_data.f90.

10.9.2.2 subroutine mod_data::nctime2date (type (file) *model*)

Change time in netcdf to dates.

Author

M. Rajner

Date

2013-03-04

Definition at line 135 of file mod_data.f90.

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

Unpack variable.

from net

Definition at line 211 of file mod_data.f90.

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

- grat/src/mod_data.f90

10.10 mod_green Module Reference

Collaboration diagram for mod_green:

Data Types

- type result

Public Member Functions

- subroutine green_unification (green, green_common, denser)

  Unification:

- subroutine spher_area (distance, ddistance, azstp, area, method)

  Calculate area of spherical segment.
- subroutine convolve (site, green, results, denserdist, denseraz)

  Perform convolution.
- subroutine **convolve_moreverbose** (latin, lonin, azimuth, azstep, distance, distancestep)
- subroutine **wczytaj_linie_informacyjne**
- subroutine **plot2green** (green_file)
- subroutine **green2plot** (green_file)
- subroutine getgrf (num, ntot, ngr, fingrd)

  chapter 4.1 of spotl manual

Public Attributes

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

10.10.1 Detailed Description

Definition at line 3 of file mod_green.f90.

10.10.2 Member Function/Subroutine Documentation

10.10.2.1 subroutine mod_green::convolve (type(site_data), intent(in) *site,* type(green_functions), dimension(:), allocatable *green,* type (result), intent(out) *results,* integer, intent(in) *denserdist,* integer, intent(in) *denseraz*)

Perform convolution.

Date

2013-03-15

Author

M. Rajner

Definition at line 96 of file mod_green.f90.

10.10.2.2 subroutine mod_green::getgrf (integer *num,* integer *ntot,* integer *ngr,* character∗1 *fingrd*)

chapter 4.1 of spotl manual

?

Date

2013-03-15

Author

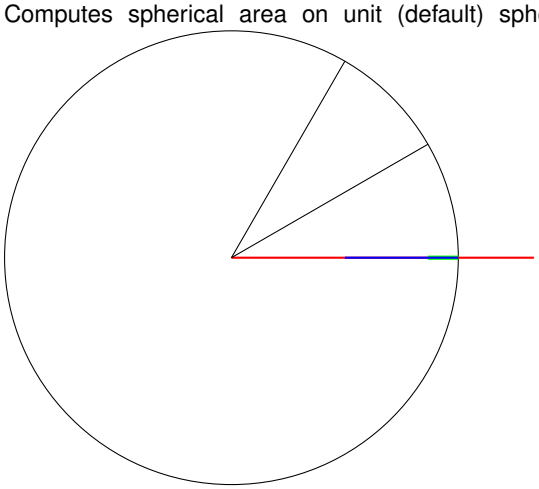
M. Rajner

Definition at line 415 of file mod_green.f90.

10.10.2.3 subroutine mod_green::spher_area (real(dp), intent(in) *distance,* real(dp), intent(in) *ddistance,* real(dp), intent(in) *azstp,* real(dp), intent(out) *area,* integer, intent(in), optional *method*)

Calculate area of spherical segment.

Computes spherical area on unit (default) sphere given by distance from station and azimuth angle xxx



Definition at line 76 of file mod_green.f90.

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

- grat/src/mod_green.f90

10.11 mod_polygon Module Reference

Some routines to deal with inclusion or exclusion of polygons.

Public Member Functions

- subroutine, public read_polygon (polygon)

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

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

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

10.11.1 Detailed Description

Some routines to deal with inclusion or exclusion of polygons.

Author

M.Rajner

Date

2012-12-20
2013-03-19 added overriding of poly use bt command line like in **?**

Definition at line 9 of file mod_polygon.f90.

10.11.2 Member Function/Subroutine Documentation

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

Check if point is in closed polygon.

From spotl Agnew [1997] adopted to grat and Fortran90 syntax From original description returns iok=0 if

1. there is any polygon (of all those read in) in which the coordinate should not fall, and it does or
2. the coordinate should fall in at least one polygon (of those read in) and it does not otherwise returns iok=1

Author

D.C. Agnew Agnew [1996]
adopted by Marcin Rajner

Date

2013-03-04

Definition at line 105 of file mod_polygon.f90.

10.11.2.2 integer function mod_polygon::ncross (real(dp), intent(in) *x1,* real(dp), intent(in) *y1,* real(dp), intent(in) *x2,* real(dp), 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 220 of file mod_polygon.f90.

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

Reads polygon data.

inspired by spotl Agnew [1997]

Definition at line 23 of file mod_polygon.f90.

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

- grat/src/mod_polygon.f90

10.12 mod_utilities Module Reference

Public Member Functions

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

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

- subroutine, public spline (x, y, b, c, d, n)

  Compute coefficients for spline interpolation.
- real(dp) function, public ispline (u, x, y, b, c, d, n, method)

  Evaluates the cubic spline interpolatione.
- integer function, public ntokens (line)

  This function counts the word in line separated with space or multispaces.
- subroutine, public skip_header (unit, comment_char_optional)

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

  Compute Julian date for given date.
- real(dp) function, public mjd (date)

  MJD from date.
- subroutine, public invmjd (mjd, date)

  Compute date from given Julian Day.
- logical function, public is_numeric (string)

  Check if argument is numeric.
- logical function, public file_exists (string)

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

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

  radian -> degree
- subroutine, public spher_trig (latin, lonin, distance, azimuth, latout, lonout)

  This soubroutine gives the latitude and longitude of the point at the specified distance and azimuth from site latitude and longitude.
- subroutine, public spher_trig_inverse (lat1, lon1, lat2, lon2, distance, azimuth, haversine)

  For given coordinates for two points on sphere calculate distance and azimuth in radians.
- subroutine, public count_records_to_read (file_name, rows, columns, comment_char)

  Count rows and (or) columns of file.

10.12.1 Detailed Description

Definition at line 1 of file mod_utilities.f90.

10.12.2 Member Function/Subroutine Documentation

10.12.2.1 subroutine, public mod_utilities::count_records_to_read (character(∗) *file_name,* integer, intent(out), optional *rows,* integer, intent(out), optional *columns,* character(len=1), intent(in), optional *comment_char*)

Count rows and (or) columns of file.

You can also specify the comment sign to ignore in data file. The number of columns is set to maximum of number of columns in consecutive rows.

Date

2013-03-10

Author

M. Rajner

Definition at line 504 of file mod_utilities.f90.

10.12.2.2 real(dp) function, public mod_utilities::d2r (real(dp), intent(in) *degree*)

degree -> radian

This function convert values given in decimal degrees to radians.

Author

M. Rajner

Date

2013-03-04

Definition at line 398 of file mod_utilities.f90.

10.12.2.3 logical function, public mod_utilities::file_exists (character(len=*), intent(in) *string*)

Check if file exists.

Logical function checking if given file exists.

Author

M. Rajner (based on www)

Date

2013-03-04

Definition at line 377 of file mod_utilities.f90.

10.12.2.4 subroutine, public mod_utilities::invmjd (real(dp), intent(in) *mjd,* integer, dimension (6), intent(out) *date*)

Compute date from given Julian Day.

This subroutine computes date (as an six elements integer array) from Modified Julian Day

Date

2013-03-04

Definition at line 321 of file mod_utilities.f90.

10.12.2.5 logical function, public mod_utilities::is_numeric (character(len=*), intent(in) *string*)

Check if argument is numeric.

Author

Taken from www

Date

2013-03-19

Definition at line 360 of file mod_utilities.f90.

10.12.2.6 real (dp) function, public mod_utilities::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,* character(∗), optional *method*)

Evaluates the cubic spline interpolatione.

Function ispline evaluates the cubic spline interpolation at point z ispline = y(i)+b(i)∗(u-x(i))+c(i)∗(u-x(i))∗∗2+d(i)∗(u-x(i))∗∗3

where x(i) <= u <= x(i+1)

input.. u = the abscissa at which the spline is to be evaluated x, y = the arrays of given data points b, c, d = arrays of spline coefficients computed by spline n = the number of data points output: ispline = interpolated value at point u

Date

2013-03-10

Author

M. Rajner

added optional parameter method

Definition at line 156 of file mod_utilities.f90.

10.12.2.7 real(dp) function, public mod_utilities::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*)

Compute Julian date for given date.

Compute Julian Day (not MJD!). Seconds as integer!

Author

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

Todo mjd!

Date

2013-03-04

Definition at line 273 of file mod_utilities.f90.

10.12.2.8 real(dp) function, public mod_utilities::mjd (integer, dimension (6), intent(in) *date*)

MJD from date.

Compute Modified Julian date for given date. Iput is six element array of !integers. Seconds also as integers!

Date

2013-03-04

Definition at line 295 of file mod_utilities.f90.

10.12.2.9 integer function, public mod_utilities::ntokens (character, dimension(*), intent(in) *line*)

This function counts the word in line separated with space or multispaces.

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

Definition at line 214 of file mod_utilities.f90.

10.12.2.10 real(dp) function, public mod_utilities::r2d (real(dp), intent(in) *radian*)

radian -> degree

This function convert values given in radians to decimal degrees.

Author

Marcin Rajner

Date

2013-03-04

Definition at line 412 of file mod_utilities.f90.

10.12.2.11 subroutine, public mod_utilities::spher_trig (real(dp), intent(in) *latin,* real(dp), intent(in) *lonin,* real(dp), intent(in) *distance,* real(dp), intent(in) *azimuth,* real(dp), intent(out) *latout,* real(dp), intent(out) *lonout*)

This soubroutine gives the latitude and longitude of the point at the specified distance and azimuth from site latitude and longitude.

all parameters in decimal degree

Author

D.C. Agnew Agnew [1996]

Date

2012

Author

M. Rajner - modification

Date

2013-03-06

Definition at line 429 of file mod_utilities.f90.

10.12.2.12 subroutine, public mod_utilities::spher_trig_inverse (real(dp), intent(in) *lat1,* real(dp), intent(in) *lon1,* real(dp), intent(in) *lat2,* real(dp), intent(in) *lon2,* real(dp), intent(out) *distance,* real(dp), intent(out) *azimuth,* logical, intent(in), optional *haversine*)

For given coordinates for two points on sphere calculate distance and azimuth in radians.

Input coordinates ub

Author

M. Rajner

Date

2013-03-04 for small spherical distances you should always use havesine=.true.

Definition at line 458 of file mod_utilities.f90.

10.12.2.13 subroutine, public mod_utilities::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*)

Compute coefficients for spline interpolation.

From web sources

Todo find url Original description below: ==========
Calculate the coefficients b(i), c(i), and d(i), i=1,2,...,n for cubic spline interpolation s(x) = y(i) + b(i)∗(x-x(i)) + c(i)∗(x-x(i))∗∗2 + d(i)∗(x-x(i))∗∗3 for x(i) <= x <= x(i+1) Alex G: January 2010

input.. x = the arrays of data abscissas (in strictly increasing order) y = the arrays of data ordinates n = size of the arrays xi() and yi() (n>=2) output.. b, c, d = arrays of spline coefficients comments ... spline.f90 program is based on fortran version of program spline.f

the accompanying function fspline can be used for interpolation

Definition at line 64 of file mod_utilities.f90.

10.12.2.14 subroutine, public mod_utilities::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,* character(∗), optional *method*)

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

uses `ispline` and `spline` subroutines

Definition at line 21 of file mod_utilities.f90.

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

- grat/src/mod_utilities.f90

10.13 mod_cmdline::polygon_data Type Reference

Public Attributes

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

10.13.1 Detailed Description

Definition at line 29 of file mod_cmdline.f90.

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

- grat/src/mod_cmdline.f90

10.14 mod_cmdline::polygon_info Type Reference

Collaboration diagram for mod_cmdline::polygon_info:

Public Attributes

- integer **unit**
- character(:), allocatable **name**
- type(polygon_data), dimension(:), allocatable **polygons**
- logical **if**
- character(1) **pm**

10.14.1 Detailed Description

Definition at line 34 of file mod_cmdline.f90.

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

- grat/src/mod_cmdline.f90

10.15 mod_green::result Type Reference

Public Attributes

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

10.15.1 Detailed Description

Definition at line 12 of file mod_green.f90.

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

- grat/src/mod_green.f90

10.16 mod_cmdline::site_data Type Reference

Public Attributes

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

10.16.1 Detailed Description

Definition at line 73 of file mod_cmdline.f90.

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

- grat/src/mod_cmdline.f90

Chapter 11

File Documentation

11.1 grat/doc/interpolation_ilustration.sh File Reference

Variables

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

11.1.1 Detailed Description

Definition in file interpolation_ilustration.sh.

11.1.2 Variable Documentation

11.1.2.1 then interp

Initial value:

```
2
   else
     interp=1
   fi
    value_check -F /home/mrajner/dat/ncep_reanalysis/pres.sfc.2011.nc@SP:pres

    -S 2.51/4.99/0.05/2.45
```

Definition at line 17 of file interpolation_ilustration.sh.

11.2 interpolation_ilustration.sh

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

```
00015 do
00016    if [ ${co} = "b" ] ; then
00017      interp=2
00018    else
00019      interp=1
00020    fi
00021     value_check -F /home/mrajner/dat/ncep_reanalysis/pres.sfc.2011.nc@SP:pres
         \
00022     -S 2.51/4.99/0.05/2.45,0.091,0.1 -I ${interp} \
00023     -o interp${co}1.dat  -L interpl1.dat :b
00024 done
00025   perl -n -i -e 'print if $. <= 4' interpl1.dat
00026
```

11.3 grat/src/grat.f90 File Reference

Functions/Subroutines

- program **grat**

11.3.1 Detailed Description

Definition in file grat.f90.

11.4 grat.f90

```
00001 !
       ================================================================================
00002 !> \file
00003 !! \mainpage grat overview
00004 !! \section Purpose
00005 !! This program was created to make computation of atmospheric gravity
00006 !! correction easier. Still developing. Consider visiting later...
00007 !!
00008 !! \version TESTING!
00009 !! \date 2013-01-12
00010 !! \author Marcin Rajner\n
00011 !! Politechnika Warszawska | Warsaw University of Technology
00012 !!
00013 !! \warning This program is written in Fortran90 standard but uses some
       featerus
00014 !! of 2003 specification (e.g., \c 'newunit='). It was also written
00015 !! for <tt>Intel Fortran Compiler</tt> hence some commands can be unavailable
00016 !! for other compilers (e.g., \c <integer_parameter> for \c IO statements. This
       should be
00017 !! easily modifiable according to your output needs.
00018 !! Also you need to have \c iso_fortran_env module available to guess the
       number
00019 !! of output_unit for your compiler.
00020 !! When you don't want a \c log_file and you don't switch \c verbose all
00021 !! unneceserry information whitch are normally collected goes to \c /dev/null
00022 !! file. This is *nix system default trash. For other system or file system
00023 !! organization, please change this value in \c mod_cmdline module.
00024 !!
00025 !! \attention
00026 !! \c grat and value_check needs a \c netCDF library \cite netcdf
00027 !!
00028 !! \section Usage
00029 !! After sucsesfull compiling make sure the executables are in your search path
00030 !!
00031 !! There is main program \c grat and some utilities program. For the options
       see
00032 !! the appropriate help:
00033 !!  - \link grat-h grat\endlink
00034 !!  - \link value_check-h value_check\endlink
00035 !!  - \link polygon_check-h polygon_check\endlink
00036 !!
00037 !! \page grat-h grat
00038 !!    \include grat.hlp
00039
00040 !> \page ilustration
00041 !! \image latex /home/mrajner/src/grat/doc/interpolation_ilustration.pdf
       "example"
00042 !! \image latex /home/mrajner/src/grat/doc/mapa1
```

```
00043 !! \image latex /home/mrajner/src/grat/doc/mapa2
00044 !! \image latex /home/mrajner/src/grat/doc/mapa3
00045 !!
00046 !! \image html /home/mrajner/src/grat/doc/interpolation_ilustration.png
       "interpolation example" width=\textwidth
00047 !! \image html /home/mrajner/src/grat/doc/mapa1.png
00048 !! \image html /home/mrajner/src/grat/doc/mapa2.png
00049 !! \image html /home/mrajner/src/grat/doc/mapa3.png
00050
00051 !> \page intro_sec External resources
00052 !!   - <a href="https://code.google.com/p/grat">project page</a> (git
       repository)
00053 !!   - \htmlonly <a href="../latex/refman.pdf">[pdf]</a> version of this
       manual\endhtmlonly
00054 !!   \latexonly
       \href{https://grat.googlecode.com/git/doc/html/index.html}{html} version of this manual\endlatexonly
00055 !! \TODO give source for grant presentation
00056 !!   - <a href="">[pdf]</a> command line options (in Polish)
00057
00058 !> \example example_aggf.f90
00059 !! \example grat_usage.sh
00060 !
       ================================================================================
00061 program grat
00062
00063   use mod_constants , only : dp
00064   use mod_cmdline   , only : cpu_start , cpu_finish,  intro ,
      print_settings , &
00065     polygons , model , refpres, form_separator , log ,dates , sites, output, &
00066     moreverbose, form_60 , form_61, green ,denser
00067   use mod_green     , only : results ,convolve
00068   use mod_polygon   , only : read_polygon
00069   use mod_data      , only : read_netCDF , get_variable
00070
00071   implicit none
00072   real(dp) :: x , y , z , lat ,lon ,val(0:100) !tmp variables
00073   integer :: i , j , ii, iii
00074
00075   ! program starts here with time stamp
00076   call cpu_time(cpu_start)
00077
00078   ! gather cmd line option decide where to put output
00079   call intro(program_calling = "grat")
00080
00081   ! print header to log: version, date and summary of command line options
00082   call print_settings(program_calling = "grat")
00083
00084   ! read polygons
00085   do i =1 , 2
00086    call read_polygon(polygons(i))
00087   enddo
00088
00089   ! read models into memory
00090   do i =1 , size(model)
00091     if (model(i)%if) call read_netcdf(model(i))
00092   enddo
00093
00094   ! todo refpres in get_cmd-line
00095   if (refpres%if) then
00096     refpres%name="/home/mrajner/src/grat/data/refpres/vienna_p0.grd"
00097     call read_netcdf(refpres)
00098   endif
00099
00100
00101   allocate (results(size(sites)*max(size(dates),1)))
00102   iii=0
00103   do j = 1 , max(size (dates),1)
00104     if(size(dates).gt.0)  write(output%unit, '(i4,5(i2.2))', advance ="no")
      dates(j)%date
00105
00106     do ii = 1 , min(2,size(model))
00107       if (model(ii)%if) call get_variable( model(ii) , date = dates(j)%date)
00108     enddo
00109
00110     write(log%unit, form_separator)
00111     write(log%unit, form_60) "Results:"
00112     if (output%if.and.(output%name /= "")) write(log%unit, form_61) "written
       into file:" , trim(output%name)
00113     do i = 1 , size(sites)
00114       write(output%unit, '(2f15.5f)', advance ="no") sites(i)%lat ,sites(i)%lon
00115       iii=iii+1
00116       call convolve(sites(i) , green , results(iii), denserdist =
      denser(1) , denseraz = denser(2))
00117       write (output%unit,'(15f13.5)') , results(iii)%e ,results(iii)%n  ,
      results(iii)%dt , results(iii)%dh, results(iii)%dz
00118     enddo
00119   enddo
```

```
00120
00121
00122   if (moreverbose%if .and. moreverbose%names(1).eq."s") then
00123     print '(15f13.5)', &
00124       results( maxloc( results%e  )  ) %e  - results( minloc( results%e  ) ) %e
      ,  &
00125       results( maxloc( results%n  )  ) %n  - results( minloc( results%n  ) ) %n
      ,  &
00126       results( maxloc( results%dh )  ) %dh - results( minloc( results%dh ) )
     %dh ,  &
00127       results( maxloc( results%dz )  ) %dz - results( minloc( results%dz ) )
     %dz ,  &
00128       results( maxloc( results%dt )  ) %dt - results( minloc( results%dt ) )
     %dt
00129   endif
00130
00131
00132   call cpu_time(cpu_finish)
00133   write(log%unit, '(/,"Execution time:",1x,f16.9," seconds")') cpu_finish -
     cpu_start
00134   write(log%unit, form_separator)
00135 end program
```

11.5 grat/src/mod_aggf.f90 File Reference

This module contains utitlities for computing Atmospheric Gravity Green Functions.

Data Types

- module mod_aggf

11.5.1 Detailed Description

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

Definition in file mod_aggf.f90.

11.6 mod_aggf.f90

```
00001 !
      ================================================================================
00002 !> \file
00003 !! \brief This module contains utitlities for computing
00004 !! Atmospheric Gravity Green Functions
00005 !!
00006 !! In this module there are several subroutines for computing
00007 !! AGGF and standard atmosphere parameters
00008 !
      ================================================================================
00009 module mod_aggf
00010
00011   use mod_constants
00012   implicit none
00013   private
00014
00015   public:: size_ntimes_denser, read_tabulated_green
      , standard_pressure, &
00016          standard_temperature,              bouger,
           simple_def, &
00017               standard_density,     standard_gravity
      ,      compute_aggf, &
00018                compute_aggfdt,        gn_thin_layer,
             geop2geom
00019
00020 contains
00021
00022 !
      ================================================================================
00023 !> Compute first derivative of AGGF with respect to temperature
00024 !! for specific angular distance (psi)
```

```
00025 !!
00026 !! optional argument define (-dt;-dt) range
00027 !! See equation 19 in \cite Huang05
00028 !! Same simple method is applied for aggf(gn) if \c aggf optional parameter
00029 !! is set to \c .true.
00030 !! \warning Please do not use \c aggf=.true. this option was added only
00031 !! for testing some numerical routines
00032 !! \author M. Rajner
00033 !! \date 2013-03-19
00034 !
      ! ============================================================================
00035 subroutine compute_aggfdt ( psi , aggfdt , delta_ , aggf )
00036   real(dp) , intent (in) :: psi
00037   real(dp) , intent (in) , optional :: delta_
00038   logical , intent (in) , optional :: aggf
00039   real(dp) , intent (out) :: aggfdt
00040   real(dp) :: deltat , aux , h_
00041
00042   deltat = 10. !< Default value
00043   if (present( delta_) )  deltat = delta_
00044   if (present( aggf ) .and. aggf ) then
00045     h_ = 0.001 ! default if we compute dggfdh using this routine
00046       if (present( delta_) )  h_ = deltat
00047     call compute_aggf( psi , aux , h = + h_ )
00048     aggfdt = aux
00049     call compute_aggf( psi , aux , h= -h_ )
00050     aggfdt = aggfdt - aux
00051     aggfdt = aggfdt / ( 2. * h_ )
00052   else
00053     call compute_aggf( psi , aux , t_zero = t0 + deltat )
00054     aggfdt = aux
00055     call compute_aggf( psi , aux , t_zero = t0 - deltat )
00056     aggfdt = aggfdt - aux
00057     aggfdt = aggfdt / ( 2. * deltat)
00058   endif
00059 end subroutine
00060
00061 !
      ! ============================================================================
00062 !> Wczytuje tablice danych AGGF
00063 !! \li merriam \cite Merriam92
00064 !! \li huang   \cite Huang05
00065 !! \li rajner  \cite Rajnerdr
00066 !!
00067 !! This is just quick solution for \c example_aggf program
00068 !! in \c grat see the more general routine \c parse_green()
00069 !
      ! ============================================================================
00070 subroutine read_tabulated_green ( table , author )
00071   use mod_utilities, only: skip_header , count_records_to_read
00072   real(dp), intent (inout),dimension(:,:), allocatable :: table
00073   character ( len = * ) , intent (in)                  :: author
00074   integer                                              :: i , j
00075   integer                                              :: rows , columns ,
     file_unit
00076   character (len=255)                                  :: file_name
00077
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     rows      = 85
00088     columns   = 6
00089     file_name = '../dat/merriam_green.dat'
00090   elseif( author .eq. "farrell" ) then
00091     file_name = '/home/mrajner/src/gotic2/data/grn1.data'
00092     call count_records_to_read(file_name, rows = rows, columns = columns)
00093   else
00094     write ( * , * ) 'cannot find specified tables, using merriam instead'
00095   endif
00096
00097   if (allocated (table) ) deallocate (table)
00098   allocate ( table( rows , columns ) )
00099
00100   open (newunit = file_unit , file = file_name , action='read', status='old')
00101
00102   call skip_header(file_unit)
00103
00104   do i = 1 , rows
00105     read (file_unit,*) ( table( i , j ), j = 1 , columns )
00106   enddo
00107   close(file_unit)
```

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

```
00180     endif
00181   enddo
00182
00183   aggf_val = -g * da * aggf_val * 1e8 * 1000
00184
00185   !> if you put the optional parameter \c if_normalization=.false.
00186   !! this block will be skipped
00187   !! by default the normalization is applied according to \cite Merriam92
00188   if ( (.not.present(if_normalization)) .or. (if_normalization)) then
00189     aggf_val= psir * aggf_val * 1e5  / p0
00190   endif
00191
00192 end subroutine
00193
00194 !
      ================================================================================
00195 !> Compute air density for given altitude for standard atmosphere
00196 !!
00197 !! using formulae 12 in \cite Huang05
00198 !! \date 2013-03-18
00199 !! \author M. Rajner
00200 !
      ================================================================================
00201 subroutine standard_density ( height , rho , t_zero ,fels_type
      )
00202   real(dp) , intent(in)  ::  height !< height [km]
00203   real(dp) , intent(in), optional  :: t_zero !< if this parameter is given
00204   character(len = 22) , optional :: fels_type
00205   ! surface temperature is set to this value,
00206   ! otherwise the T0 for standard atmosphere is used
00207   real(dp) , intent(out) :: rho
00208   real(dp)  :: p ,t
00209
00210   call standard_pressure(height , p , t_zero = t_zero,
      fels_type=fels_type)
00211   call standard_temperature(height , t , t_zero = t_zero,
      fels_type=fels_type)
00212
00213   ! pressure in hPa --> Pa
00214   rho= 100 * p / ( r_air * t )
00215 end subroutine
00216
00217 ! ================================================================================
00218 !> \brief Computes pressure [hPa] for specific height
00219 !!
00220 !! See \cite US1976 or  \cite Huang05 for details.
00221 !! Uses formulae 5 from \cite Huang05.
00222 !! Simplified method if optional argument if_simplificated = .true.
00223 ! ================================================================================
00224 subroutine standard_pressure (height, pressure , &
00225           p_zero , t_zero , h_zero,  if_simplificated ,fels_type , inverted)
00226   implicit none
00227   real(dp) , intent(in)             :: height
00228   real(dp) , intent(in) , optional :: t_zero , p_zero , h_zero
00229   character(len = 22) , optional :: fels_type
00230   logical          , intent(in) , optional :: if_simplificated
00231   logical          , intent(in) , optional :: inverted
00232   real(dp), intent(out) :: pressure
00233   real(dp) ::  lambda , sfc_height , sfc_temperature , sfc_gravity , alpha ,
      sfc_pressure
00234
00235   sfc_temperature = t0
00236   sfc_pressure = p0
00237   sfc_height = 0.
00238   sfc_gravity = g0
00239
00240   if (present(h_zero)) then
00241     sfc_height = h_zero
00242     call standard_temperature(sfc_height , sfc_temperature
      )
00243     call standard_temperature(sfc_height , sfc_temperature
      )
00244     call standard_gravity(sfc_height , sfc_gravity )
00245   endif
00246
00247   if (present(p_zero)) sfc_pressure = p_zero
00248   if (present(t_zero)) sfc_temperature = t_zero
00249
00250   lambda = r_air * sfc_temperature / sfc_gravity
00251
00252   if (present(if_simplificated) .and. if_simplificated ) then
00253     ! use simplified formulae
00254     alpha = -6.5
00255     pressure = sfc_pressure  &
00256       * ( 1 + alpha / sfc_temperature * (height-sfc_height)) &
00257       ** ( -sfc_gravity / (r_air * alpha / 1000. ) )
00258   else
```

```
00259     ! use precise formulae
00260     pressure = sfc_pressure * exp( -1000. * (height -sfc_height) / lambda )
00261   endif
00262   if (present(inverted).and.inverted) then
00263     pressure = sfc_pressure  / ( exp( -1000. * (height-sfc_height) / lambda ) )
00264   endif
00265
00266
00267   !todo incorporate this
00268
00269 !  Zdunkowski and Bott
00270 !  p(z) = p0 (T0-gamm z )/T0
00271
00272
00273 end subroutine
00274
00275 ! ================================================================================
00276 !> \brief Compute gravity acceleration of the Earth
00277 !! for the specific height using formula
00278 !!
00279 !! see \cite US1976
00280 ! ================================================================================
00281 subroutine standard_gravity ( height , g )
00282   implicit none
00283   real(dp), intent(in)  :: height
00284   real(dp), intent(out) :: g
00285
00286   g= g0 * ( r0 / ( r0 + height ) )**2
00287 end subroutine
00288
00289
00290 ! ================================================================================
00291 !> Compute geometric height from geopotential heights
00292 !!
00293 !! \author M. Rajner
00294 !! \date 2013-03-19
00295 ! ================================================================================
00296 real(dp) function geop2geom (geopotential_height)
00297   real (dp) :: geopotential_height
00298
00299   geop2geom = geopotential_height * (r0 / ( r0 + geopotential_height )
       )
00300 end function
00301
00302
00303 ! ================================================================================
00304 !> Iterative computation of surface temp. from given height using bisection
00305 !! method
00306 ! ================================================================================
00307 subroutine surface_temperature (height , temperature1 , &
00308   temperature2, fels_type , tolerance)
00309   real(dp) , intent(in)  :: height , temperature1
00310   real(dp) , intent(out) :: temperature2
00311   real(dp) :: temp(3) , temp_ (3) , tolerance_ = 0.1
00312   character (len=*) , intent(in), optional  :: fels_type
00313   real(dp) , intent(in), optional  :: tolerance
00314   integer :: i
00315
00316   if (present(tolerance)) tolerance_ = tolerance
00317
00318   ! searching limits
00319   temp(1)=t0-150
00320   temp(3)=t0+ 50
00321
00322   do
00323     temp(2)= ( temp(1) + temp(3) ) /2.
00324
00325     do i = 1,3
00326       call standard_temperature(height , temp_(i) , t_zero=
      temp(i) , fels_type = fels_type )
00327     enddo
00328
00329     if (abs(temperature1 - temp_(2) ) .lt. tolerance_ ) then
00330       temperature2 = temp(2)
00331       return
00332     endif
00333
00334     if ( (temperature1 - temp_(1) ) * (temperature1 - temp_(2) ) .lt.0 ) then
00335       temp(3) = temp(2)
00336     elseif( (temperature1 - temp_(3) ) * (temperature1 - temp_(2) ) .lt.0 )
      then
00337       temp(1) = temp(2)
00338     else
00339       stop "surface_temp"
00340     endif
00341   enddo
00342 end subroutine
```

```
00343 ! ============================================================================
00344 !> \brief Compute standard temperature [K] for specific height [km]
00345 !!
00346 !! if t_zero is specified use this as surface temperature
00347 !! otherwise use T0.
00348 !! A set of predifined temperature profiles ca be set using
00349 !! optional argument \argument fels_type
00350 !! \cite Fels86
00351 !! \li US standard atmosphere (default)
00352 !! \li tropical
00353 !! \li subtropical_summer
00354 !! \li subtropical_winter
00355 !! \li subarctic_summer
00356 !! \li subarctic_winter
00357 !
      ============================================================================
00358 subroutine standard_temperature ( height , temperature ,
      t_zero , fels_type )
00359   real(dp) , intent(in)  :: height
00360   real(dp) , intent(out) :: temperature
00361   real(dp) , intent(in), optional  :: t_zero
00362   character (len=*) , intent(in), optional  :: fels_type
00363   real(dp) :: aux , cn , t
00364   integer :: i,indeks
00365   real(dp) , dimension (10) :: z,c,d
00366
00367   ! Read into memory the parameters of temparature height profiles
00368   ! for standard atmosphere
00369   z = (/11.0 , 20.1 , 32.1 , 47.4 , 51.4 , 71.7 , 85.7, 100.0, 200.0, 300.0/)
00370   c = (/-6.5,   0.0,   1.0,   2.75,  0.0,  -2.75, -1.97,  0.0,   0.0,   0.0/)
00371   d = (/ 0.3,   1.0,   1.0,   1.0,   1.0,   1.0,   1.0,   1.0,   1.0,   1.0/)
00372   t = t0
00373
00374   if ( present(fels_type)) then
00375     if (fels_type .eq. "US1976" ) then
00376     elseif(fels_type .eq. "tropical" ) then
00377       z=(/ 2.0  , 3.0, 16.5 , 21.5 , 45.0 , 51.0, 70.0 , 100.0 , 200.0 , 300.0
      /)
00378       c=(/-6.0 , -4.0, -6.7  , 4.0  , 2.2  , 1.0, -2.8  , -0.27  , 0.0   , 0.0
      /)
00379       d=(/ 0.5  , 0.5 , 0.3  , 0.5  , 1.0  , 1.0 , 1.0   , 1.0   , 1.0   , 1.0
      /)
00380       t=300.0
00381     elseif(fels_type .eq. "subtropical_summer" ) then
00382       z = (/ 1.5  , 6.5  , 13.0 , 18.0 , 26.0 , 36.0  , 48.0 ,  50.0 ,  70.0 ,
      100.0  /)
00383       c = (/-4.0 , -6.0  , -6.5  , 0.0  , 1.2  , 2.2  ,  2.5  ,  0.0   ,-3.0
      ,-0.025/)
00384       d = (/ 0.5  , 1.0   , 0.5  , 0.5  , 1.0  , 1.0   , 2.5   , 0.5   , 1.0
      , 1.0  /)
00385       t = 294.0
00386     elseif(fels_type .eq. "subtropical_winter" ) then
00387       z = (/ 3.0  ,10.0  , 19.0 , 25.0 , 32.0 , 44.5  , 50.0 ,  71.0 ,  98.0 ,
      200.0 /)
00388       c = (/-3.5 , -6.0  , -0.5  , 0.0  , 0.4  , 3.2  ,  1.6  , -1.8   , 0.7
      , 0.0 /)
00389       d = (/ 0.5  , 0.5   , 1.0  , 1.0  , 1.0  , 1.0   , 1.0   , 1.0   , 1.0
      , 1.0 /)
00390       t = 272.2
00391     elseif(fels_type .eq. "subarctic_summer" ) then
00392       z = (/  4.7 , 10.0 , 23.0 , 31.8 , 44.0 , 50.2  , 69.2 , 100.0 , 200.0 ,
      300.0 /)
00393       c = (/ -5.3 , -7.0 ,  0.0 ,  1.4 ,  3.0 , 0.7  , -3.3  , -0.2   , 0.0 ,
       0.0 /)
00394       d = (/  0.5 ,  0.3 ,  1.0 ,  1.0 ,  2.0 , 1.0   , 1.5   , 1.0   , 1.0 ,
       1.0 /)
00395       t = 287.0
00396     elseif(fels_type .eq. "subarctic_winter" ) then
00397       z = (/  1.0 ,  3.2 ,  8.5 , 15.5 , 25.0 , 30.0 , 35.0 , 50.0 , 70.0 , 100
      .0 /)
00398       c = (/  3.0 , -3.2 , -6.8 ,  0.0 , -0.6 ,  1.0 ,  1.2 ,  2.5 , -0.7 ,  -1
      .2 /)
00399       d = (/  0.4 ,  1.5 ,  0.3 ,  0.5 ,  1.0 ,  1.0 ,  1.0 ,  1.0 ,  1.0 ,   1
      .0 /)
00400       t = 257.1
00401     else
00402       print * ,
00403 "unknown fels_type argument: &        using US standard atmosphere 1976
       instead"
00404     endif
00405   endif
00406
00407   if (present(t_zero) ) then
00408     t=t_zero
00409   endif
00410
00411   do i=1,10
```

```
00412     if (height.le.z(i)) then
00413       indeks=i
00414       exit
00415     endif
00416   enddo
00417
00418   aux = 0.
00419   do i = 1 , indeks
00420     if (i.eq.indeks) then
00421       cn = 0.
00422     else
00423       cn = c(i+1)
00424     endif
00425       aux = aux + d(i) * ( cn - c(i) )  * dlog( dcosh( (height - z(i)) / d(i) )
      / dcosh(z(i)/d(i)) )
00426   enddo
00427   temperature = t + c(1) * height/2. + aux/2.
00428 end subroutine
00429
00430 !
      ============================================================================
00431 !> Compute AGGF GN for thin layer
00432 !!
00433 !! Simple function added to provide complete module
00434 !! but this should not be used for atmosphere layer
00435 !! See eq p. 491 in \cite Merriam92
00436 !! \author M. Rajner
00437 !! \date 2013-03-19
00438 !
      ============================================================================
00439 real function gn_thin_layer (psi)
00440   use mod_utilities, only : d2r
00441   real(dp) , intent(in) :: psi
00442   real(dp) :: psir
00443
00444   psir = d2r(psi)
00445   gn_thin_layer = 1.627 * psir / sin( psir / 2. )
00446 end function
00447
00448
00449 !
      ============================================================================
00450 !> \brief returns numbers of arguments for n times denser size
00451 !!
00452 !! i.e. * * * *  -->  * . . * . . * . . * (3 times denser)
00453 !
      ============================================================================
00454 function size_ntimes_denser (size_original, ndenser)
00455   integer :: size_ntimes_denser
00456   integer, intent(in) :: size_original , ndenser
00457   size_ntimes_denser= (size_original - 1 ) * (ndenser +1 ) +
      1
00458 end function
00459
00460 !
      ============================================================================
00461 !> \brief Bouger plate computation
00462 !!
00463 !
      ============================================================================
00464   real(dp) function bouger ( R_opt )
00465   real(dp), optional :: r_opt !< height of point above the cylinder
00466   real(dp) :: aux
00467   real(dp) :: r
00468   real(dp) ::  h = 8.84 ! scale height of standard atmosphere
00469
00470   aux = 1
00471
00472   if (present( r_opt ) ) then
00473     r = r_opt
00474     aux = h  + r  - sqrt(  r**2  + (h/2. ) ** 2 )
00475     bouger = 2 * pi * g  * aux
00476   else
00477     aux = h
00478     bouger = 2 * pi * g * aux
00479     return
00480   endif
00481 end function
00482 !
      ============================================================================
00483 !> Bouger plate computation
00484 !!
00485 !! see eq. page 288 \cite Warburton77
00486 !! \date 2013-03-18
00487 !! \author M. Rajner
00488 !
      ============================================================================
```

```
00489 function simple_def (R)
00490   real(dp) :: r ,delta
00491   real(dp) :: simple_def
00492
00493   delta = 0.22e-11 * r
00494   simple_def = g0 / r0 * delta * ( 2. - 3./2. * rho_crust / rho_earth
      &
00495     -3./4. * rho_crust / rho_earth * sqrt(2* (1. )) ) * 1000
00496 end function
00497
00498 end module
```

11.7 grat/src/mod_cmdline.f90 File Reference

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

Data Types

- module mod_cmdline
- type mod_cmdline::green_functions
- type mod_cmdline::polygon_data
- type mod_cmdline::polygon_info
- type mod_cmdline::dateandmjd
- type mod_cmdline::additional_info
- type mod_cmdline::cmd_line
- type mod_cmdline::site_data
- type mod_cmdline::file

11.7.1 Detailed Description

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

Definition in file mod_cmdline.f90.

11.8 mod_cmdline.f90

```
00001 ! ================================================================================
00002 !> \file
00003 !! \brief This module sets the initial values for parameters
00004 !! reads from command line and gives help
00005 !! it allows to specify commands with or without spaces therefore it is
00006 !! convienient to use with auto completion of names
00007 ! ================================================================================
00008 module mod_cmdline
00009
00010   use mod_constants, only: dp
00011   use iso_fortran_env
00012
00013   implicit none
00014
00015   !-----------------------------------------------------
00016   ! Greens function
00017   !-----------------------------------------------------
00018   type green_functions
00019     real(dp),allocatable,dimension(:) :: distance
00020     real(dp),allocatable,dimension(:) :: data
00021     logical  :: if
00022   end type
00023   type(green_functions), allocatable , dimension(:) :: green
00024   integer :: denser(2) = [1,1]
00025
00026   !-----------------------------------------------------
00027   ! polygons
00028   !-----------------------------------------------------
```

```
00029   type polygon_data
00030     logical :: use
00031     real(dp), allocatable , dimension (:,:) :: coords
00032   end type
00033
00034   type polygon_info
00035     integer :: unit
00036     character(:), allocatable  :: name
00037     type(polygon_data) , dimension (:) , allocatable :: polygons
00038     logical :: if
00039     ! global setting (+|-) which override this in polygon file
00040     character(1):: pm
00041   end type
00042
00043   type(polygon_info) , dimension (2) :: polygons
00044
00045   !-----------------------------------------------------
00046   ! dates
00047   !-----------------------------------------------------
00048   type dateandmjd
00049     real(dp) :: mjd
00050     integer,dimension (6) :: date
00051   end type
00052
00053   real(dp) :: cpu_start , cpu_finish
00054   type(dateandmjd) , allocatable,dimension (:) :: dates
00055
00056
00057   !-----------------------------------------------------
00058   ! command line entry
00059   !-----------------------------------------------------
00060   type additional_info
00061     character (len=55) ,allocatable ,dimension(:) :: names
00062   end type
00063   type cmd_line
00064     character(2) :: switch
00065     integer ::      fields
00066     character (len=255) ,allocatable ,dimension(:) :: field
00067     type (additional_info), allocatable , dimension(:) ::
      fieldnames
00068   end type
00069
00070   !-----------------------------------------------------
00071   ! site information
00072   !-----------------------------------------------------
00073   type site_data
00074    character(:), allocatable :: name
00075    real(dp)                  :: lat,lon,height
00076   end type
00077
00078   type(site_data) , allocatable , dimension(:) :: sites
00079
00080   !-----------------------------------------------------
00081   ! various
00082   !-----------------------------------------------------
00083   integer :: fileunit_tmp               !< unit of scratch file
00084   integer,dimension(8):: execution_date !< To give time stamp of execution
00085   character (len = 2) :: method = "2D"  !< computation method
00086
00087   !-----------------------------------------------------
00088   ! Site names file
00089   !-----------------------------------------------------
00090   character(:), allocatable &
00091           :: filename_site
00092   integer :: fileunit_site
00093
00094   type file
00095     character(:), allocatable :: name
00096
00097     ! varname , lonname,latname,levelname , timename
00098     character(len=50) :: names(5) = [ "z", "lon", "lat","level","time"]
00099
00100     !choose with -F filename@XX:pres...
00101     character(len=40) :: dataname
00102
00103     integer :: unit = output_unit
00104
00105     ! if file was determined
00106     logical :: if =.false.
00107
00108     ! to read into only once
00109     logical :: first_call =.true.
00110
00111     ! boundary of model e , w ,s ,n
00112     real(dp):: limits(4)
00113
00114 !     resolution of model in lon lat
```

```
00115 !    real(dp):: resolution(2)
00116
00117     real(dp) , allocatable ,dimension(:) :: lat , lon , time ,level
00118     integer , allocatable , dimension(:,: ) :: date
00119
00120     real (dp), dimension(2) :: latrange , lonrange
00121
00122     ! todo
00123     logical :: if_constant_value
00124     real(dp):: constant_value
00125
00126     ! data
00127     !> 4 dimension - lat , lon , level , mjd
00128     ! todo
00129     real(dp) , allocatable , dimension (:,:,:) :: data
00130
00131     ! netcdf identifiers
00132     integer :: ncid
00133     integer :: interpolation = 1
00134   end type
00135
00136   ! External files
00137   type(file) ::  log  , output , refpres , moreverbose
00138   type(file) , allocatable, dimension (:) :: model
00139
00140   character (len =40) :: model_names (5) = ["pressure_surface" , &
00141     "temperature_surface" , "topography" , "landsea" , "pressure levels" ]
00142
00143
00144   character(len=5) :: green_names(5) = [ "GN   ", "GN/dt", "GN/dh","GN/dz","GE
      "]
00145
00146   logical :: if_verbose  = .false.
00147   logical :: inverted_barometer  = .true.
00148
00149   character (50) :: interpolation_names (2) &
00150     = [ "nearest" , "bilinear" ]
00151
00152   !-----------------------------------------------------
00153   ! For preety printing
00154   !-----------------------------------------------------
00155   character(len=255), parameter ::  &
00156     form_header    = '(60("#"))' , &
00157     form_separator = '("#",59("-"))' , &
00158     form_inheader  = '(("#"),1x,a56,1x,("#"))' , &
00159     form_60        = "(a,100(1x,g0))",          &
00160     form_61        = "(2x,a,100(1x,g0))",    &
00161     form_62        = "(4x,a,100(1x,g0))",       &
00162     form_63        = "(6x,100(x,g0))",       &
00163     form_64        = "(4x,4x,a,4x,a)"
00164
00165
00166 !  private
00167 !  public :: nmodels
00168
00169 contains
00170 ! ================================================================================
00171 !> This subroutine counts the command line arguments
00172 !!
00173 !! Depending on command line options set all initial parameters and reports it
00174 !! \date 2012-12-20
00175 !! \author M. Rajner
00176 !! \date 2013-03-19 parsing negative numbers after space fixed
00177 !!    (-S -11... was previously treated as two cmmand line entries, now only -?

00178 !!    non-numeric terminates input argument)
00179 ! ================================================================================
00180 subroutine intro (program_calling, accepted_switch )
00181   integer :: i, j
00182   character(len=255) :: dummy, dummy2,arg
00183   character(len=*), intent(in) :: program_calling
00184   type(cmd_line) :: cmd_line_entry
00185   character(len=*) , intent (in), optional :: accepted_switch
00186
00187   if(iargc().eq.0) then
00188     write(output_unit , '(a)' ) , 'Short description: ./'//program_calling//'
      -h'
00189     call exit
00190   else
00191     open(newunit=fileunit_tmp,status='scratch')
00192     write (fileunit_tmp,form_61) "command invoked"
00193     call get_command(dummy)
00194     write (fileunit_tmp,form_62) trim(dummy)
00195     do i = 1 , iargc()
00196       call get_command_argument(i,dummy)
00197       ! allows specification like '-F file' and '-Ffile'
00198       ! but  if -[0,9] it is treated as number belonging to switch (-S -2)
```

```
00199       ! but  if -[\s,:] do not start next command line option
00200       call get_command_argument(i+1,dummy2)
00201       if (check_if_switch_or_minus(dummy)) then
00202         arg = trim(dummy)
00203       else
00204         arg=trim(arg)//trim(dummy)
00205       endif
00206       if(check_if_switch_or_minus(dummy2).or.i.eq.iargc
      ()) then
00207          call mod_cmdline_entry(arg, cmd_line_entry ,
      program_calling = program_calling)
00208       endif
00209     enddo
00210
00211     call if_minimum_args( program_calling = program_calling )
00212
00213     ! Where and if to log the additional information
00214     if (log%if) then
00215       ! if file name was given then automaticall switch verbose mode
00216       if_verbose = .true.
00217       open (newunit = log%unit, file = log%name , action = "write" )
00218     else
00219       ! if you don't specify log file, or not switch on verbose mode
00220       ! all additional information will go to trash
00221       ! Change  /dev/null accordingly if your file system does not
00222       ! support this name
00223       if (.not.if_verbose) then
00224         open (newunit=log%unit, file = "/dev/null", action = "write" )
00225       endif
00226     endif
00227   endif
00228 end subroutine
00229
00230 !
       ================================================================================
00231 !> Check if - starts new option in command line or is just a minus in command
00232 !! line entry
00233 !!
00234 !! if after '-' is space or number or ',' or ':' (field separators) do not
       start
00235 !! next option for command line
00236 !! If switch return .true. otherwise return .false
00237 !!
00238 !! \author M. Rajner
00239 !! \date 2013-03-19
00240 !
       ================================================================================
00241 function check_if_switch_or_minus(dummy)
00242   use mod_utilities, only: is_numeric
00243   logical:: check_if_switch_or_minus
00244   character(*) :: dummy
00245
00246   check_if_switch_or_minus = .false.
00247   if (dummy(1:1).eq."-") check_if_switch_or_minus = .
      true.
00248   if (dummy(2:2).eq." ") check_if_switch_or_minus = .
      false.
00249   if (dummy(2:2).eq.",") check_if_switch_or_minus = .
      false.
00250   if (dummy(2:2).eq.":") check_if_switch_or_minus = .
      false.
00251   if (is_numeric(dummy(2:2))) check_if_switch_or_minus
      = .false.
00252 end function
00253
00254 ! ================================================================================
00255 !> Check if at least all obligatory command line arguments were given
00256 !! if not print error and exit
00257 !!
00258 !! \date 2013-03-15
00259 !! \author M. Rajner
00260 !! \todo Make it compact (if errror ....)
00261 ! ================================================================================
00262 subroutine if_minimum_args (program_calling)
00263   character (*) , intent(in) :: program_calling
00264   type(cmd_line) :: cmd_line_entry
00265   character(len=100) :: dummy
00266
00267   ! all programs
00268 !  if (size(sites) .eq. 0) then
00269 !    write(error_unit, * ) "ERROR:", program_calling, " -- no sites!"
00270 !    call exit
00271 !  endif
00272
00273   if (program_calling.eq."grat" ) then
00274     ! for grat set default for Green functions if not given in command line
00275     ! options
```

```
00276     if (.not.allocated(green)) then
00277       dummy="-G,,,"
00278       call mod_cmdline_entry(dummy,cmd_line_entry,
      program_calling="grat")
00279     endif
00280
00281     if (size(model) .eq. 0) then
00282       write(error_unit, * ) "ERROR:", program_calling, " -- model file not
       specified!"
00283       call exit
00284     endif
00285   elseif(program_calling.eq."polygon_check" ) then
00286   endif
00287 end subroutine
00288
00289 ! ========================================================================
00290 !> This function is true if switch is used by calling program or false if it
00291 !! is not
00292 ! ========================================================================
00293 logical function if_switch_program (program_calling , switch )
00294   character(len= *), intent (in) :: program_calling
00295   character(len= *), intent (in) :: switch
00296   character, dimension(:) , allocatable :: accepted_switch
00297   integer :: i
00298
00299   ! default
00300   if_switch_program=.false.
00301
00302   ! depending on program calling decide if switch is permitted
00303   if (program_calling.eq."grat") then
00304     allocate( accepted_switch(17) )
00305     accepted_switch = [ &
00306       "V", "f", "S", "B", "L", "G", "P", "p", &
00307       "o", "F", "I", "D", "L", "v", "h", "R", "Q" &
00308     ]
00309   elseif(program_calling.eq."polygon_check") then
00310     allocate( accepted_switch(13) )
00311     accepted_switch = [ "V" , "f" , "A", "B" , "L" , "P" , "o", "S" , &
00312       "h" , "v" , "I" , "i" , "R" ]
00313   elseif(program_calling.eq."value_check") then
00314     allocate( accepted_switch(11) )
00315     accepted_switch = [ "V" , "F" , "o", "S" , "h" , "v" , "I" , "D" , "L" ,
      "P" , "R" ]
00316   else
00317     if_switch_program=.true.
00318     return
00319   endif
00320
00321   ! loop trough accepted switches
00322   do i =1, size (accepted_switch)
00323     if (switch(2:2).eq.accepted_switch(i)) if_switch_program=.
      true.
00324   enddo
00325 end function
00326
00327 ! ========================================================================
00328 !> This subroutine counts the command line arguments and parse appropriately
00329 ! ========================================================================
00330 subroutine parse_option (cmd_line_entry , program_calling)
00331   use mod_utilities, only : file_exists, is_numeric
00332   type(cmd_line),intent(in):: cmd_line_entry
00333   character(len=*), optional :: program_calling
00334   integer :: i
00335
00336   ! all the command line option are stored in tmp file and later its decide
00337   ! if it is written to STDOUT , log_file or nowwhere
00338     select case (cmd_line_entry%switch)
00339     case ('-h')
00340       call print_help(program_calling)
00341       call exit
00342     case ('-v')
00343       call print_version(program_calling)
00344       call exit()
00345     case ('-V')
00346       if_verbose = .true.
00347       write(fileunit_tmp, form_62) 'verbose mode' ,trim(log%name)
00348       if (len(trim(cmd_line_entry%field(1))).gt.0) then
00349         log%if = .true.
00350         log%name = trim(cmd_line_entry%field(1))
00351         write(fileunit_tmp, form_62) 'the log file was set:' ,log%name
00352       endif
00353     case ('-S','-R')
00354       ! check if format is proper for site
00355       ! i,e. -Sname,B,L[,H]
00356       if (.not. allocated(sites)) then
00357         if (is_numeric(cmd_line_entry%field(2)) &
00358         .and.is_numeric(cmd_line_entry%field(3)) &
```

```
00359         .and.index(cmd_line_entry%field(1), "/" ).eq.0 &
00360         .and.(.not.cmd_line_entry%field(1).eq. "Rg" ) &
00361         ) then
00362             allocate (sites(1))
00363             sites(1)%name = trim(cmd_line_entry%field(1))
00364             read ( cmd_line_entry%field(2) , * ) sites(1)%lat
00365             if (abs(sites(1)%lat).gt.90.) &
00366               sites(1)%lat = sign(90.,sites(1)%lat)
00367             read ( cmd_line_entry%field(3) , * ) sites(1)%lon
00368             if (sites(1)%lon.ge.360.) sites(1)%lon = mod(sites(1)%lon,360.)
00369             if (is_numeric(cmd_line_entry%field(4) ) ) then
00370               read ( cmd_line_entry%field(4) , * ) sites(1)%height
00371             endif
00372             write(fileunit_tmp, form_62) 'the site was set (BLH):' , &
00373               sites(1)%name, real(sites(1)%lat) , &
00374               real(sites(1)%lon) , real(sites(1)%height)
00375         else
00376           ! or read sites from file
00377           if (file_exists(cmd_line_entry%field(1) ) ) then
00378             write(fileunit_tmp, form_62) 'the site file was set:' , &
00379               cmd_line_entry%field(1)
00380             call read_site_file(cmd_line_entry%field(1))
00381           elseif(index(cmd_line_entry%field(1), "/" ).ne.0 &
00382               .or.cmd_line_entry%field(1).eq."Rg")  then
00383             call parse_gmt_like_boundaries(
    cmd_line_entry )
00384           else
00385             call print_warning( "site" , fileunit_tmp)
00386           endif
00387         endif
00388       else
00389         call print_warning( "repeated" , fileunit_tmp)
00390       endif
00391     case ("-I")
00392       !> \todo add maximum minimum distances for integration
00393       write( fileunit_tmp , form_62 , advance="no" ) "interpolation method was
     set:"
00394       do i = 1 , cmd_line_entry%fields
00395         if (is_numeric(cmd_line_entry%field(i))) then
00396           read ( cmd_line_entry%field(i) , * ) model(i)%interpolation
00397           write(fileunit_tmp , '(a10,x,$)' ) interpolation_names(model(i)
    %interpolation)
00398           if (model(i)%interpolation.gt.size(interpolation_names)) then
00399             model(i)%interpolation=1
00400           endif
00401         endif
00402       enddo
00403       write(fileunit_tmp , *)
00404     case ("-L")
00405       !> \todo make it mulitichoice: -Lfile:s,file2:b ...
00406       write (fileunit_tmp , form_62) "printing additional information"
00407 !      allocate(moreverbose(cmd_line_entry%fields))
00408 !      print *,size(moreverbose),"XXXX"
00409 !      do i = 1, cmd_line_entry%fields
00410 !        call get_model_info (moreverbose (i) , cmd_line_entry , i )
00411 !        write (fileunit_tmp , form_62) "file: ", moreverbose(i)%name
00412 !      enddo
00413 !      write (fileunit_tmp , form_62) "what: ", moreverbose%names(1)
00414 !      if (len(moreverbose%name).gt.0 .and. moreverbose%name.ne."") then
00415 !        open (newunit = moreverbose%unit , file = moreverbose%name , action =
     "write" )
00416 !      endif
00417     case ("-B")
00418       if (cmd_line_entry%field(1).eq."N" ) inverted_barometer = .false.
00419     case ("-Q")
00420       if (cmd_line_entry%field(1).eq."+" ) refpres%if = .true.
00421       write (fileunit_tmp , form_62) "Reference pressure was set."
00422     case ('-D')
00423       call parse_dates( cmd_line_entry )
00424     case ('-F')
00425       allocate(model(cmd_line_entry%fields))
00426       do i = 1, cmd_line_entry%fields
00427         call get_model_info(model(i) , cmd_line_entry , i )
00428       enddo
00429     case ("-G")
00430       !> \todo when no given take defaults
00431       call parse_green(cmd_line_entry)
00432     case ('-M')
00433       !> \todo rozbudować
00434       method = cmd_line_entry%field(1)
00435       write(fileunit_tmp, form_62), 'method was set: ' , method
00436     case ('-o')
00437       output%if=.true.
00438       output%name=cmd_line_entry%field(1)
00439       write(fileunit_tmp, form_62), 'output file was set: ' , output%name
00440       if (len(output%name).gt.0.and. output%name.ne."") then
00441         open (newunit = output%unit , file = output%name , action = "write"
```

```
      )
00442       endif
00443     case ('-P')
00444       do i = 1, cmd_line_entry%fields
00445         ! prevent from multiple -P
00446         if (polygons(i)%if) then
00447           call print_warning("repeated", fileunit_tmp)
00448           return
00449         endif
00450         polygons(i)%name=cmd_line_entry%field(i)
00451         if (file_exists((polygons(i)%name))) then
00452           write(fileunit_tmp, form_62), 'polygon file was set: ' , polygons(i)
      %name
00453           polygons(i)%if=.true.
00454           if (allocated(cmd_line_entry%fieldnames)) then
00455             polygons(i)%pm = trim(cmd_line_entry%fieldnames(i)%names(1))
00456           endif
00457         else
00458           write(fileunit_tmp, form_62), 'file do not exist. Polygon file was
       IGNORED'
00459         endif
00460       enddo
00461     case default
00462       write(fileunit_tmp,form_62), "unknown argument: IGNORING"
00463     end select
00464 end subroutine
00465
00466 ! ================================================================================
00467 !> This subroutine parse -G option -- Greens function.
00468 !!
00469 !! This subroutines takes the -G argument specified as follows:
00470 !!   -G
00471 !!
00472 !! \author M. Rajner
00473 !! \date 2013-03-06
00474 ! ================================================================================
00475 subroutine parse_green ( cmd_line_entry)
00476   use mod_utilities, only: file_exists, is_numeric, skip_header
00477   type (cmd_line)  :: cmd_line_entry
00478   character (60) :: filename
00479   integer :: i , iunit , io_status , lines ,  ii
00480   integer :: fields(2)= [1,2]
00481   real (dp) , allocatable , dimension(:) :: tmp
00482
00483     write(fileunit_tmp , form_62) "Green function file was set:"
00484     allocate (green(cmd_line_entry%fields))
00485
00486     do i = 1 , cmd_line_entry%fields
00487
00488       if (i.eq.6) then
00489         if (is_numeric(cmd_line_entry%field(i))) then
00490           read( cmd_line_entry%field(i), *) denser(1)
00491           if (is_numeric(cmd_line_entry%fieldnames(i)%names(1))) then
00492             read( cmd_line_entry%fieldnames(i)%names(1), *) denser(2)
00493           endif
00494           return
00495          endif
00496       endif
00497
00498       if (.not.file_exists(cmd_line_entry%field(i)) &
00499       .and. (.not. cmd_line_entry%field(i).eq."merriam" &
00500       .and. .not. cmd_line_entry%field(i).eq."huang" &
00501       .and. .not. cmd_line_entry%field(i).eq."rajner" )) then
00502         cmd_line_entry%field(i)="merriam"
00503       endif
00504
00505       !> change the paths accordingly
00506       if (cmd_line_entry%field(i).eq."merriam") then
00507         filename="/home/mrajner/src/grat/dat/merriam_green.dat"
00508         if (i.eq.1) fields = [1,2]
00509         if (i.eq.2) fields = [1,3]
00510         if (i.eq.3) fields = [1,4]
00511         if (i.eq.4) fields = [1,4]
00512         if (i.eq.5) fields = [1,6]
00513       elseif(cmd_line_entry%field(i).eq."huang") then
00514         filename="/home/mrajner/src/grat/dat/huang_green.dat"
00515         if (i.eq.1) fields = [1,2]
00516         if (i.eq.2) fields = [1,3]
00517         if (i.eq.3) fields = [1,4]
00518         if (i.eq.4) fields = [1,5]
00519         if (i.eq.5) fields = [1,6]
00520       elseif(cmd_line_entry%field(i).eq."rajner") then
00521         filename="/home/mrajner/src/grat/dat/rajner_green.dat"
00522         if (i.eq.1) fields = [1,2]
00523         if (i.eq.2) fields = [1,3]
00524         if (i.eq.3) fields = [1,4]
00525         if (i.eq.4) fields = [1,5]
```

```
00526         if (i.eq.5) fields = [1,6]
00527       elseif(file_exists(cmd_line_entry%field(i))) then
00528         filename = cmd_line_entry%field(i)
00529         if (size(cmd_line_entry%fieldnames).ne.0 .and. allocated(cmd_line_entry
      %fieldnames(i)%names)) then
00530           do ii=1, 2
00531             if(is_numeric(cmd_line_entry%fieldnames(i)%names(ii) ) ) then
00532               read( cmd_line_entry%fieldnames(i)%names(ii), *) fields(ii)
00533             endif
00534           enddo
00535         endif
00536       endif
00537
00538       allocate(tmp(max(fields(1),fields(2))))
00539       lines = 0
00540       open ( newunit =iunit,file=filename,action="read")
00541       do
00542         call skip_header(iunit)
00543         read (iunit , * , iostat = io_status)
00544         if (io_status == iostat_end) exit
00545         lines = lines + 1
00546       enddo
00547       allocate (green(i)%distance(lines))
00548       allocate (green(i)%data(lines))
00549       rewind(iunit)
00550       lines = 0
00551       do
00552         call skip_header(iunit)
00553         lines = lines + 1
00554         read (iunit , * , iostat = io_status) tmp
00555         if (io_status == iostat_end) exit
00556         green(i)%distance(lines) = tmp(fields(1))
00557         green(i)%data(lines)     = tmp(fields(2))
00558       enddo
00559       deallocate(tmp)
00560       close(iunit)
00561
00562       ! file specific
00563       if (cmd_line_entry%field(i).eq."merriam" .and. i.eq.4) then
00564         green(i)%data = green(i)%data * (-1.)
00565       endif
00566       if (cmd_line_entry%field(i).eq."huang" .and. (i.eq.3.or.i.eq.4)) then
00567         green(i)%data = green(i)%data * 1000.
00568       endif
00569       write(fileunit_tmp , form_63) trim(green_names(i)), &
00570         trim(cmd_line_entry%field(i)),":", fields
00571     enddo
00572 end subroutine
00573
00574 ! ================================================================================
00575 !> Counts occurence of character (separator, default comma) in string
00576 ! ================================================================================
00577 integer function count_separator (dummy , separator)
00578   character(*) , intent(in) ::dummy
00579   character(1), intent(in), optional  :: separator
00580   character(1)  :: sep
00581   character(:), allocatable :: dummy2
00582   integer :: i
00583
00584   dummy2=dummy
00585   sep = ","
00586   if (present(separator)) sep = separator
00587   count_separator=0
00588   do
00589     i = index(dummy2, sep)
00590     if (i.eq.0) exit
00591     dummy2 = dummy2(i+1:)
00592     count_separator=count_separator+1
00593   enddo
00594 end function
00595
00596
00597 ! ================================================================================
00598 !> This subroutine fills the fields of command line entry for every input arg
00599 !!
00600 !! \author M. Rajner
00601 !! \date 2013-03-21
00602 ! ================================================================================
00603 subroutine mod_cmdline_entry (dummy , cmd_line_entry ,
      program_calling )
00604   character(*) :: dummy
00605   character(:), allocatable :: dummy2
00606   type (cmd_line),intent(out) :: cmd_line_entry
00607   character(1) :: separator=","
00608   character(len=*) , intent(in) , optional :: program_calling
00609   integer :: i , j , ii , jj
00610
```

```
00611   cmd_line_entry%switch = dummy(1:2)
00612   write(fileunit_tmp, form_61) , dummy
00613   if (.not.if_switch_program(program_calling, cmd_line_entry
      %switch)) then
00614     write ( fileunit_tmp , form_62 ) "this switch is IGNORED by program "//
      program_calling
00615     return
00616   endif
00617
00618   dummy=dummy(3:)
00619   cmd_line_entry%fields = count_separator(dummy) + 1
00620   allocate(cmd_line_entry%field (cmd_line_entry%fields) )
00621
00622   ! if ":" separator is present in command line allocate
00623   ! additional array for fieldnames
00624   if (count_separator(dummy, ":" ).ge.1) then
00625     allocate(cmd_line_entry%fieldnames (cmd_line_entry%fields) )
00626   endif
00627   do i = 1 , cmd_line_entry%fields
00628     j = index(dummy, separator)
00629     cmd_line_entry%field(i) = dummy(1:j-1)
00630     if (i.eq.cmd_line_entry%fields) cmd_line_entry%field(i)=dummy
00631     dummy=dummy(j+1:)
00632
00633     ! separate field and fieldnames
00634     if ( index(cmd_line_entry%field(i),":").ne.0 ) then
00635       dummy2 = trim(cmd_line_entry%field(i))//":"
00636       allocate ( cmd_line_entry%fieldnames(i)%names(count_separator
      (dummy2,":") - 1 ))
00637       do ii = 1, size(cmd_line_entry%fieldnames(i)%names)+1
00638         jj = index(dummy2, ":")
00639         if (ii.eq.1) then
00640           cmd_line_entry%field(i) = dummy2(1:jj-1)
00641         else
00642           cmd_line_entry%fieldnames(i)%names(ii-1) = dummy2(1:jj-1)
00643         endif
00644         dummy2 = dummy2(jj+1:)
00645       enddo
00646     endif
00647   enddo
00648   call parse_option(cmd_line_entry , program_calling =
      program_calling)
00649 end subroutine
00650 !
00651 ! ============================================================================
00652 !> This subroutine fills the model info
00653 ! ============================================================================
00654 subroutine get_model_info (model , cmd_line_entry , field)
00655   use mod_utilities, only : file_exists, is_numeric
00656   type(cmd_line),intent(in):: cmd_line_entry
00657   type(file),intent(inout):: model
00658   integer :: field , i , indeks
00659
00660   ! split name and dataname (separated by @ - optional)
00661   model%name = trim(cmd_line_entry%field(field))
00662   model%dataname = "NN"
00663   indeks = index(cmd_line_entry%field(field),'@')
00664   if (indeks.gt.0) then
00665     model%name = trim(cmd_line_entry%field(field)(1:indeks-1))
00666     model%dataname = trim(cmd_line_entry%field(field)(indeks+1:))
00667   endif
00668   if (model%name.eq."") return
00669   write (fileunit_tmp , form_62) , trim(dataname(model%dataname)), &
00670     "("//trim(model%dataname)//")"
00671   if ( file_exists(model%name) ) then
00672     do i =1 , size (model%names)
00673       if (size(cmd_line_entry%fieldnames).gt.0) then
00674         if (i.le.size (cmd_line_entry%fieldnames(field)%names) &
00675           .and. cmd_line_entry%fieldnames(field)%names(i).ne."" &
00676           ) then
00677           model%names(i) = cmd_line_entry%fieldnames(field)%names(i)
00678         endif
00679       endif
00680       write(fileunit_tmp, form_63, advance="no") , trim(model%names(i))
00681     enddo
00682     model%if=.true.
00683     write(fileunit_tmp, form_63)
00684   elseif(is_numeric(model%name)) then
00685     model%if_constant_value=.true.
00686     read (model%name , * ) model%constant_value
00687     write(fileunit_tmp, form_63), 'constant value was set: ' , model
      %constant_value
00688     model%if_constant_value=.true.
00689   else
00690     write (fileunit_tmp , form_63 ) "no (correct) model in field: ", field
00691   endif
00692 end subroutine
```

```
00693
00694
00695 ! ========================================================================
00696 !> P
00697 ! ========================================================================
00698 subroutine parse_gmt_like_boundaries ( cmd_line_entry
       )
00699   use mod_constants, only : dp ,dp
00700   use mod_utilities, only : is_numeric
00701   real(dp) :: limits (4) , resolution (2) =[1,1]
00702   real(dp) :: range_lon , range_lat , lat , lon
00703   character(10) :: dummy
00704   integer :: i , ii
00705   type (cmd_line) , intent (in) :: cmd_line_entry
00706   character(:) ,allocatable :: text
00707   integer :: n_lon , n_lat
00708
00709   text = cmd_line_entry%field(1)
00710
00711   do i=1,3
00712     if ( is_numeric(text(1:index(text, "/"))) ) then
00713       read ( text(1:index(text, "/")) , * )  limits(i)
00714     else
00715       if (text.eq."Rg" ) then
00716         limits=[0. , 360. , -90 , 90. ]
00717         exit
00718       endif
00719     endif
00720     text=text(index(text,"/")+1:)
00721   enddo
00722
00723   if ( is_numeric(text(1:)) ) then
00724     read ( text(1:) , * )  limits(4)
00725   else
00726     call print_warning("boundaries")
00727   endif
00728
00729   do i = 1 ,2
00730     if (limits(i).lt. -180. .or. limits(i).gt.360. ) then
00731       call print_warning("boundaries")
00732     else
00733       if (limits(i).lt.0.) limits(i)=limits(i)+360.
00734     endif
00735   enddo
00736   do i =3,4
00737     if (limits(i).lt. -90. .or. limits(i).gt.90. ) then
00738       call print_warning("boundaries")
00739     endif
00740   enddo
00741   if (limits(3).gt.limits(4)) then
00742     call print_warning("boundaries")
00743   endif
00744
00745   if (is_numeric(cmd_line_entry%field(2) ) ) then
00746     read (cmd_line_entry%field(2) , * ) resolution(1)
00747     resolution(2) = resolution(1)
00748   endif
00749   if (is_numeric(cmd_line_entry%field(3) ) ) then
00750     read (cmd_line_entry%field(3) , * ) resolution(2)
00751   endif
00752
00753   range_lon=limits(2) - limits(1)
00754   if (range_lon.lt.0) range_lon = range_lon + 360.
00755   range_lat=limits(4) - limits(3)
00756   n_lon = floor( range_lon / resolution(1)) + 1
00757   n_lat = floor( range_lat / resolution(2)) + 1
00758   allocate (sites( n_lon * n_lat ) )
00759
00760   do i = 1 , n_lon
00761     lon = limits(1) + (i-1) * resolution(1)
00762     if (lon.ge.360.) lon = lon - 360.
00763     do ii = 1 , n_lat
00764       lat = limits(3) + (ii-1) * resolution(2)
00765       sites( (i-1) * n_lat + ii  )%lon = lon
00766       sites( (i-1) * n_lat + ii  )%lat = lat
00767     enddo
00768   enddo
00769
00770 end subroutine
00771
00772 ! ========================================================================
00773 !> Read site list from file
00774 !!
00775 !! checks for arguments and put it into array \c sites
00776 ! ========================================================================
00777 subroutine read_site_file ( file_name )
00778   use mod_utilities, only: is_numeric, ntokens
```

```
00779   character(len=*) , intent(in) ::  file_name
00780   integer :: io_status , i , good_lines = 0 , number_of_lines = 0 , nloop
00781   character(len=255) ,dimension(4)  :: dummy
00782   character(len=255) :: line_of_file
00783   type(site_data) :: aux
00784
00785
00786
00787     open ( newunit = fileunit_site , file = file_name, &
00788             iostat = io_status ,status = "old" , action="read" )
00789
00790     ! two loops, first count good lines and print rejected
00791     ! second allocate array of sites and read coordinates into it
00792       nloops: do nloop = 1, 2
00793       if (nloop.eq.2) allocate(sites(good_lines))
00794       if (number_of_lines.ne.good_lines) then
00795         call print_warning("site_file_format")
00796       endif
00797       good_lines=0
00798       line_loop:do
00799         read ( fileunit_site , '(a)' , iostat = io_status ) line_of_file
00800         if ( io_status == iostat_end)  exit  line_loop
00801         number_of_lines = number_of_lines + 1
00802         ! we need at least 3 parameter for site (name , B , L )
00803         if (ntokens(line_of_file).ge.3) then
00804           ! but no more than 4 parameters (name , B , L, H)
00805           if (ntokens(line_of_file).gt.4) then
00806             read ( line_of_file , * ) dummy(1:4)
00807           else
00808             read ( line_of_file , * ) dummy(1:3)
00809             ! if site height was not given we set it to zero
00810             dummy(4)="0."
00811           endif
00812         endif
00813         ! check the values given
00814         if(    is_numeric(trim(dummy(2)))   &
00815           .and.is_numeric(trim(dummy(3)))   &
00816           .and.is_numeric(trim(dummy(4)))   &
00817           .and.ntokens(line_of_file).ge.3 ) then
00818
00819           aux%name= trim(dummy(1))
00820           read( dummy(2),*) aux%lat
00821           read(dummy(3),*) aux%lon
00822           read(dummy(4),*) aux%height
00823
00824 !          ! todo
00825           if (aux%lat.ge.-90 .and. aux%lat.le.90) then
00826             if (aux%lon.ge.-180 .and. aux%lon.le.360) then
00827               good_lines=good_lines+1
00828               if (nloop.eq.2) then
00829                 sites(good_lines)%name= trim(dummy(1))
00830                 read(dummy(2),*) sites(good_lines)%lat
00831                 read(dummy(3),*) sites(good_lines)%lon
00832                 read(dummy(4),*) sites(good_lines)%height
00833               endif
00834             else
00835               if (nloop.eq.2) write ( fileunit_tmp, form_63) "rejecting (lon
       limits):" , line_of_file
00836             endif
00837           else
00838             if (nloop.eq.2) write ( fileunit_tmp, form_63) "rejecting (lat
       limits):" , line_of_file
00839           endif
00840
00841         else
00842           ! print it only once
00843           if (nloop.eq.2) then
00844               write ( fileunit_tmp, form_63) "rejecting (args):      " ,
      line_of_file
00845           endif
00846         endif
00847       enddo line_loop
00848       if (nloop.eq.1) rewind(fileunit_site)
00849     enddo nloops
00850
00851   ! if longitude <-180, 180> change to <0,360) domain
00852   do i =1 , size (sites)
00853     if (sites(i)%lon.lt.0) sites(i)%lon= sites(i)%lon + 360.
00854     if (sites(i)%lon.eq.360) sites(i)%lon= 0.
00855   enddo
00856 end subroutine
00857
00858
00859 ! ========================================================================
00860 !> Parse date given as 20110503020103  to yy mm dd hh mm ss and mjd
00861 !!
00862 !! \warning decimal seconds are not allowed
```

```
00863 ! ============================================================================
00864 subroutine parse_dates (cmd_line_entry )
00865   use mod_utilities, only: is_numeric,mjd,invmjd
00866   type(cmd_line) cmd_line_entry
00867   integer , dimension(6) :: start , stop , swap
00868   real (dp) :: step =6. ! step in hours
00869   integer :: i
00870
00871   call string2date(cmd_line_entry%field(1), start)
00872   write (fileunit_tmp , form_62) "start date:" , start
00873   if (cmd_line_entry%field(2).eq."".or.cmd_line_entry%fields.le.1) then
00874     stop = start
00875   else
00876     call string2date(cmd_line_entry%field(2), stop )
00877     write (fileunit_tmp , form_62) "stop date: " , stop
00878   endif
00879   if (is_numeric(cmd_line_entry%field(3)).and.cmd_line_entry%fields.ge.3) then
00880     read(cmd_line_entry%field(3),*) step
00881     write (fileunit_tmp , form_62) "interval [h]:" , step
00882   endif
00883
00884   ! allow that stop is previous than start and list in reverse order
00885   ! chage the sign of step in dates if necessery
00886   if(mjd(stop).lt.mjd(start).and. step.gt.0) step = -step
00887   ! or if step is negative
00888   if(mjd(stop).gt.mjd(start).and. step.lt.0) then
00889     swap=start
00890     start=stop
00891     stop=swap
00892   endif
00893
00894   allocate (dates( int( ( mjd(stop) - mjd(start) ) / step * 24. + 1 ) ))
00895   do i = 1 , size(dates)
00896     dates(i)%mjd = mjd(start) + ( i -1 ) * step / 24.
00897     call invmjd( dates(i)%mjd , dates(i)%date)
00898   enddo
00899 end subroutine
00900
00901
00902 ! ============================================================================
00903 !> Convert dates given as string to integer (6 elements)
00904 !!
00905 !! 20110612060302 --> [2011 , 6 , 12 , 6 , 3 , 2
00906 !! you can omit
00907 !! \warning decimal seconds are not allowed
00908 ! ============================================================================
00909 subroutine string2date ( string , date )
00910   use mod_utilities, only: is_numeric
00911   integer , dimension(6) ,intent(out):: date
00912   character (*) , intent(in) :: string
00913   integer :: start_char , end_char , j
00914
00915   ! this allow to specify !st Jan of year simple as -Dyyyy
00916   date = [2000 , 1 , 1 , 0 ,0 ,0]
00917
00918   start_char = 1
00919   do j = 1 , 6
00920     if (j.eq.1) then
00921       end_char=start_char+3
00922     else
00923       end_char=start_char+1
00924     endif
00925       if (is_numeric(string(start_char : end_char) )) then
00926       read(string(start_char : end_char),*) date(j)
00927     endif
00928     start_char=end_char+1
00929   enddo
00930
00931 end subroutine
00932
00933
00934 ! ============================================================================
00935 ! ============================================================================
00936 subroutine sprawdzdate(mjd)
00937   use mod_utilities
00938   real(dp):: mjd
00939 !    if
     (mjd.gt.jd(data_uruchomienia(1),data_uruchomienia(2),data_uruchomienia(3),data_uruchomienia(4),data_uruchomienia(5),dat
00940       write (*,'(4x,a)') "Data późniejsza niż dzisiaj. KOŃCZĘ!"
00941 !      call exit
00942 !    elseif (mjd.lt.jd(1980,1,1,0,0,0)) then
00943 !      write (*,'(4x,a)') "Data wcześniejsza niż 1980-01-01. KOŃCZĘ!"
00944 !      call exit
00945 !    endif
00946 !    if (.not.log_E) then
00947 !      data_koniec=data_poczatek
00948 !      mjd_koniec=mjd_poczatek
```

```
00949 !    endif
00950 !    if (mjd_koniec.lt.mjd_poczatek) then
00951 !      write (*,*) "Data końcowa większa od początkowej. KOŃCZĘ!"
00952 !      write (*,form_64) "Data końcowa większa od początkowej. KOŃCZĘ!"
00953 !    endif
00954 end subroutine
00955
00956 ! ================================================================================
00957 !> Print version of program depending on program calling
00958 !!
00959 !! \author M. Rajner
00960 !! \date 2013-03-06
00961 ! ================================================================================
00962 subroutine print_version (program_calling)
00963   character(*) :: program_calling
00964   integer :: version_unit , io_stat
00965   character(40) :: version
00966
00967   ! from the file storing version number
00968   open(newunit=version_unit, file =
      '/home/mrajner/src/grat/dat/version.txt', &
00969     action = 'read' , status = 'old')
00970   do
00971     read (version_unit , '(a)' , iostat = io_stat ) version
00972     if (io_stat == iostat_end) exit
00973     if (version(1:2) == ' '//program_calling(1:1)) exit
00974   enddo
00975   write(log%unit , form_header )
00976   write(log%unit,form_inheader ) , trim(program_calling)
00977   write(log%unit,form_inheader ) , trim(version(3:))
00978   write(log%unit,form_inheader ) , ''
00979   write(log%unit,form_inheader ) , 'Marcin Rajner'
00980   write(log%unit,form_inheader ) , 'Warsaw University of Technology'
00981   write(log%unit , form_header )
00982 end subroutine
00983
00984 ! ================================================================================
00985 !> Print settings
00986 ! ================================================================================
00987 subroutine print_settings (program_calling)
00988   logical :: exists
00989   character (len=255):: dummy
00990   integer :: io_status , j
00991   character(*), intent(in), optional :: program_calling
00992
00993   call print_version( program_calling = program_calling)
00994   call date_and_time( values = execution_date )
00995   write(log%unit,
00996 '("Program started:",1x,i4,2("-",i2.2), &
      1x,i2.2,2(":",i2.2),1x,"(",dp,i3.2,"h UTC)")'),          &
00997     execution_date(1:3),execution_date(5:7),execution_date(4)/60
00998   write(log%unit, form_separator)
00999
01000   inquire(fileunit_tmp, exist=exists)
01001   if (exists) then
01002     write (log%unit, form_60 ) 'Summary of command line arguments'
01003
01004     !-----------------------------------------------------
01005     ! Cmd line summary (from scratch file)
01006     !-----------------------------------------------------
01007     rewind(fileunit_tmp)
01008     do
01009       read(fileunit_tmp,'(a80)', iostat = io_status ) dummy
01010       if ( io_status == iostat_end)  exit
01011       write (log%unit, '(a80)') dummy
01012     enddo
01013
01014     !-----------------------------------------------------
01015     ! Site summary
01016     !-----------------------------------------------------
01017     write(log%unit, form_separator)
01018     write(log%unit, form_60 ) "Processing:", size(sites), "site(s)"
01019
01020     if (size(sites).le.15) then
01021       write(log%unit, '(2x,a,t16,3a15)') "Name" , "lat [deg]" , "lon [deg]" ,"H
      [m]"
01022       do j = 1,size(sites)
01023         write(log%unit, '(2x,a,t16,3f15.4)') &
01024           sites(j)%name, sites(j)%lat, sites(j)%lon , sites(j)%height
01025       enddo
01026     endif
01027
01028     !-----------------------------------------------------
01029     ! Computation method summary
01030     !-----------------------------------------------------
01031     if (program_calling.eq."grat" ) then
01032       write(log%unit, form_separator)
```

```
01033       write(log%unit, form_60 ) "Method used:", method
01034     endif
01035
01036     write(log%unit, form_separator)
01037     write(log%unit, form_60 ) "Interpolation data:", &
01038       interpolation_names(model%interpolation)(1:7)
01039   endif
01040 end subroutine
01041
01042 ! ================================================================================
01043 ! ================================================================================
01044 subroutine print_help (program_calling)
01045   character(*) :: program_calling
01046   integer :: help_unit , io_stat
01047   character(500)::line
01048   character(255)::syntax
01049   logical:: if_print_line = .false., if_optional=.true.
01050
01051   if_print_line=.false.
01052
01053   ! change this path according to your settings
01054   open(newunit=help_unit, file="~/src/grat/dat/help.hlp", action="read",
      status="old")
01055
01056   write (log%unit ,"(a)" , advance="no" ) program_calling
01057   ! first loop - print only syntax with squre brackets if parameter is optional
01058   do
01059     read (help_unit , '(a)', iostat=io_stat) line
01060     if ((io_stat==iostat_end .or. line(1:1) == "-") .and. if_print_line ) then
01061       if (if_optional) write(log%unit, '(a)' , advance="no") " ["
01062       if (if_optional) write(log%unit, '(a)' , advance="no") trim(syntax)
01063       if (if_optional) write(log%unit, '(a)' , advance="no") "]"
01064     endif
01065     if (io_stat==iostat_end) then
01066       write(log%unit, *) " "
01067       if_print_line = .false.
01068       exit
01069     endif
01070     if(line(1:1)=="-") then
01071       if(if_switch_program(program_calling , line(1:2) )) then
01072         if_print_line = .true.
01073       else
01074         if(line(1:1)=="-") if_print_line=.false.
01075       endif
01076     endif
01077
01078     if (line(5:13) == "optional " .and. (line(2:2) == program_calling(1:1) .or.
       line(2:2)="")) then
01079       if_optional=.true.
01080     elseif(line(5:13) == "mandatory") then
01081       if_optional=.false.
01082     endif
01083     if (line(2:2)=="s") then
01084       syntax = trim(adjustl(line(3:)))
01085     endif
01086   enddo
01087   rewind(help_unit)
01088
01089   write(log%unit , form_60) , 'Summary of available options for program '//
      program_calling
01090   ! second loop - print informations
01091   do
01092     read (help_unit , '(a)', iostat=io_stat) line
01093     if (io_stat==iostat_end) exit
01094
01095     if(line(1:1)=="-") then
01096       if(if_switch_program(program_calling , line(1:2) )) then
01097         if_print_line = .true.
01098         write (log%unit , form_61 ) trim(line)
01099       else
01100         if(line(1:1)=="-") if_print_line=.false.
01101       endif
01102     elseif(line(2:2)==program_calling(1:1) .or. line(2:2)=="s") then
01103       if (if_print_line) then
01104         write (log%unit , form_61 ) "  "//trim(line(3:))
01105       endif
01106     elseif(line(2:2)=="") then
01107       if (if_print_line) write (log%unit , form_61 ) trim(line)
01108     endif
01109   enddo
01110   close(help_unit)
01111
01112 end subroutine
01113
01114 subroutine print_warning (  warn , unit)
01115   character (len=*)  :: warn
01116   integer , optional :: unit
```

```
01117   integer :: def_unit
01118
01119   def_unit=fileunit_tmp
01120   if (present(unit) ) def_unit=unit
01121
01122   if (warn .eq. "site_file_format") then
01123     write(def_unit, form_63) "Some records were rejected"
01124     write(def_unit, form_63) "you should specify for each line at least 3[4]
      parameters in free format:"
01125     write(def_unit, form_63) "name lat lon [H=0] (skipped)"
01126   elseif(warn .eq. "boundaries") then
01127     write(def_unit, form_62) "something wrong with boundaries. IGNORED"
01128   elseif(warn .eq. "site") then
01129     write(def_unit, form_62) "something wrong with -S|-R specification.
      IGNORED"
01130   elseif(warn .eq. "repeated") then
01131     write(def_unit, form_62) "reapeted specification. IGNORED"
01132   elseif(warn .eq. "dates") then
01133     write(def_unit, form_62) "something wrong with date format -D. IGNORED"
01134   endif
01135 end subroutine
01136
01137
01138 ! ================================================================================
01139 !> Counts number of properly specified models
01140 !!
01141 !! \date 2013-03-15
01142 !! \author M. Rajner
01143 ! ================================================================================
01144 integer function nmodels (model)
01145   type(file) , allocatable, dimension (:) :: model
01146   integer :: i
01147
01148   nmodels = 0
01149   do i = 1 , size (model)
01150     if (model(i)%if) nmodels =nmodels + 1
01151     if (model(i)%if_constant_value) nmodels =nmodels + 1
01152   enddo
01153 end function
01154
01155 ! ================================================================================
01156 !> Attach full dataname by abbreviation
01157 !!
01158 !! \date 2013-03-21
01159 !! \author M. Rajner
01160 ! ================================================================================
01161 function dataname(abbreviation)
01162   character(len=40) :: dataname
01163   character(len=2) :: abbreviation
01164
01165   dataname="unknown"
01166   if (abbreviation.eq."LS") dataname = "Land-sea mask"
01167   if (abbreviation.eq."SP") dataname = "Surface pressure"
01168 end function
01169
01170 end module mod_cmdline
```

11.9 grat/src/mod_green.f90 File Reference

module

Data Types

- module mod_green
- type mod_green::result

11.9.1 Detailed Description

module

Definition in file mod_green.f90.

11.10 mod_green.f90

```
00001 !> \file
00002 !! module
00003 module mod_green
00004
00005   use mod_constants, only: dp
00006   implicit none
00007 !  private
00008 !  public :: results
00009
00010
00011   real(dp), allocatable , dimension(:,:)  :: green_common
00012   type result
00013     real(dp) :: n=0. , dt=0. ,e=0. , dh=0.,dz=0.
00014   end type
00015   type (result), allocatable, dimension(:) :: results
00016 !
00017 !
00018 !
00019 contains
00020
00021 ! ================================================================================
00022 !> Unification:
00023 ! ================================================================================
00024 subroutine green_unification (green , green_common , denser)
00025   use mod_constants , only : dp
00026   use mod_cmdline, only: moreverbose, method ,  green_functions
00027   use mod_aggf, only: size_ntimes_denser
00028   use mod_utilities, only:spline_interpolation
00029
00030   type(green_functions), allocatable , dimension(:) , intent(in)
       :: green
00031   integer, optional :: denser
00032   integer :: i , ndenser , j ,ii
00033   real(dp), allocatable , dimension(:) :: x , y , dist
00034   real(dp), allocatable , dimension(:,:) , intent(out) :: green_common
00035
00036
00037   ndenser=0
00038   if (present(denser)) ndenser = denser
00039
00040   allocate (x(size_ntimes_denser(size(green(1)%distance),
      ndenser)-1))
00041   allocate(dist(size(x)))
00042   ii=0
00043   do i = 1 , size(green(1)%distance)-1
00044     do j = 0 , ndenser
00045       ii=ii+1
00046       x(ii) = green(1)%distance (i)  + (j +1./2.) * (green(1)%distance (i+1) -
      green(1)%distance (i) ) / ( ndenser + 1 )
00047       dist(ii) = (green(1)%distance (i+1) -green(1)%distance (i) ) / ( ndenser
      + 1 )
00048     enddo
00049   enddo
00050   ! x(size(x)) = green(1)%distance(size(green(1)%distance))
00051   allocate(green_common(size(x) , 7))
00052   green_common(:,1) = x
00053   green_common(:,2) = dist
00054   do i = 1 , 5
00055     if (size(green).ge.i .and. allocated(green(i)%distance)) then
00056       call spline_interpolation(green(i)%distance , green(i)%data, x , y)
00057       green_common(:,i+2) = y
00058     else
00059       green_common(:,i+2) = 0
00060     endif
00061   enddo
00062   if (moreverbose%if.and. moreverbose%names(1).eq."G") then
00063     write(moreverbose%unit , '(7F13.6)' ) (green_common(i,:), i =1,ubound(
      green_common,1))
00064   endif
00065 end subroutine
00066
00067 ! ================================================================================
00068 !> Calculate area of spherical segment
00069 !!
00070 !! Computes spherical area on unit (default) sphere given by
00071 !! distance from station and azimuth angle
00072 !! xxx
00073 !! \image latex /home/mrajner/src/grat/doc/rysunki/spher_area.pdf
00074 !! \image html /home/mrajner/src/grat/doc/rysunki/spher_area.png
00075 ! ================================================================================
00076 subroutine spher_area (distance ,ddistance, azstp,  area, method )
00077   use mod_constants, only: dp, sp
00078   use mod_utilities, only: d2r, r2d
00079   real(dp), intent(out) :: area
```

```
00080   real(dp), intent(in)  :: distance,ddistance
00081   real(dp), intent(in)  :: azstp
00082   integer , intent(in), optional :: method
00083
00084
00085   area =  (-cos(d2r(distance+ddistance/2.)) &
00086            + cos(d2r(distance-ddistance/2.)))*d2r(azstp)
00087
00088 end subroutine
00089
00090 ! ========================================================================
00091 !> Perform convolution
00092 !!
00093 !! \date 2013-03-15
00094 !! \author M. Rajner
00095 ! ========================================================================
00096 subroutine convolve (site ,  green , results, denserdist , denseraz  )
00097   use mod_constants, only: pi , dp, t0
00098   use mod_cmdline , only: site_data, green_functions , moreverbose ,
       &
00099     inverted_barometer , model , polygons , refpres , method
00100   use mod_utilities, only: d2r , spher_trig
00101   use mod_data, only: get_value
00102   use mod_polygon, only: chkgon
00103
00104   type(site_data) , intent(in) :: site
00105   type(green_functions), allocatable , dimension(:) :: green
00106   integer , intent (in) :: denserdist , denseraz
00107   real(dp) :: latin , lonin
00108   integer ::  ndenser , igreen  , iazimuth , nazimuth
00109   real(dp) :: azimuth
00110   real(dp) :: lat , lon , area
00111   real(dp) :: val(4) , ref_p
00112   integer :: i , iok(2) , npoints
00113   real(dp) :: normalize
00114   type (result) ,intent(out)  :: results
00115
00116
00117   if (.not.allocated(green_common))  then
00118     call green_unification(green , green_common , denser =
      denserdist-1)
00119   endif
00120
00121   npoints=0
00122   do igreen = 1 ,size(green_common(:,1))
00123     nazimuth = max(int(360*sin(d2r(green_common(igreen,1)))),100) * denseraz
00124     do iazimuth  = 1 , nazimuth
00125       npoints = npoints + 1
00126       azimuth = (iazimuth - 1) * 360./nazimuth
00127
00128       ! get lat and lon of point
00129       call spher_trig( site%lat , site%lon , green_common(igreen,1) , azimuth ,
       lat , lon)
00130
00131       ! get values of model
00132       do i = 1 , size(model)
00133         if(model(i)%if) then
00134           call get_value(model(i) , lat , lon , val(i) , level=1, method =model
      (i)%interpolation)
00135         else
00136           val(i) = 0.
00137         endif
00138       enddo
00139
00140       if (refpres%if) then
00141        call get_value(refpres , lat , lon , ref_p , method =1)
00142      else
00143        ref_p=0.
00144      endif
00145
00146       ! get polygons
00147       do i = 1 , 2
00148         if (polygons(i)%if) then
00149           call chkgon( lon, lat , polygons(i) , iok(i) )
00150         else
00151           iok(i)=1
00152         endif
00153       end do
00154
00155       ! calculate area using spherical formulae
00156       if (val(1).ne.0) then
00157         call spher_area(green_common(igreen,1) , green_common(igreen,
      2), dble(360./nazimuth) , area)
00158
00159         ! force topography to zero over oceans
00160         if (val(4).eq.0.and.val(3).lt.0) val(3) = 0.
00161
```

```
00162         ! normalization according to Merriam (1992)
00163         normalize= 1. / ( 2. * pi * ( 1. - cos( d2r(dble(1.)) ) ) * d2r(
      green_common(igreen,1)) *1.e5  )
00164
00165        ! elastic part
00166         ! if the cell is not over sea and inverted barometer assumption was not
       set
00167         ! and is not excluded by polygon
00168         if ((.not.((val(4).eq.0.and.inverted_barometer).or. iok(2).eq.0)).or.
      size(model).lt.4) then
00169           results%e = results%e + (val(1) / 100. -ref_p) * green_common(igreen,
      7) * area * normalize
00170         endif
00171   !       print*, results%e , inverted_barometer ,
       .not.((val(4).eq.0.and.inverted_barometer).or. iok(2).eq.0) ,val(4)
00172   !       stop
00173
00174         ! newtonian part
00175         if(.not. iok(1).eq.0) then
00176          results%n = results%n   + (val(1)/ 100.-ref_p) * green_common(igreen,3
      ) * area * normalize
00177
00178          if (model(2)%if.and.size(model).ge.2) then
00179             results%dt = results%dt + (val(1)/ 100.-ref_p) * &
00180               (green_common(igreen,4)*(val(2)-t0) ) * area * normalize
00181           endif
00182
00183          results%dh = results%dh + (val(1)/ 100.-ref_p) * &
00184           (green_common(igreen,5)*(site%height/1000.) ) * area * normalize
00185
00186          results%dz = results%dz + (val(1)/ 100.-ref_p) * &
00187           (green_common(igreen,6)*(val(3)/1000.) ) * area * normalize
00188        endif
00189       endif
00190       if (moreverbose%if.and. moreverbose%names(1).eq."g") then
00191         call convolve_moreverbose(site%lat,site%lon , azimuth , dble(360./
      nazimuth) , green_common(igreen,1), green_common(igreen,1))
00192         write (moreverbose%unit, '(">")')
00193       endif
00194     enddo
00195   enddo
00196   if (moreverbose%if.and. moreverbose%names(1).eq."i") then
00197     write (moreverbose%unit, '(a,x,g0)') "Points used in convolution" ,npoints
00198   endif
00199 end subroutine
00200
00201 !!> \todo site height from model
00202 !
00203 !
00204 subroutine convolve_moreverbose (latin , lonin , azimuth , azstep ,  distance ,
       distancestep)
00205   use mod_cmdline , only : moreverbose
00206   use mod_utilities, only: spher_trig
00207
00208   real(dp), intent(in) :: azimuth ,azstep, latin, lonin
00209   real(dp) :: distance, lat , lon , distancestep
00210
00211   call spher_trig( latin , lonin , distance - distancestep/2. , azimuth -
      azstep/2. , lat , lon)
00212   write(moreverbose%unit, '(2f12.6)') , lat , lon
00213   call spher_trig( latin , lonin , distance - distancestep/2. , azimuth +
      azstep/2. , lat , lon)
00214   write(moreverbose%unit, '(2f12.6)') , lat , lon
00215   call spher_trig( latin , lonin , distance + distancestep/2. , azimuth +
      azstep/2. , lat , lon)
00216   write(moreverbose%unit, '(2f12.6)') , lat , lon
00217   call spher_trig( latin , lonin , distance + distancestep/2. , azimuth -
      azstep/2. , lat , lon)
00218   write(moreverbose%unit, '(2f12.6)') , lat , lon
00219 end subroutine
00220
00221
00222 subroutine wczytaj_linie_informacyjne
00223 !!     do i=1,size(linie_informacyjne);        linie_informacyjne(i)%j_l = i;
          enddo
00224 !!      linie_informacyjne%Nj     = (/ 95    , 30    , 95    , 90    , 160    ,
       90       /)
00225 !!      linie_informacyjne%deltal = (/ 0.0011, 0.0205, 0.0550, 1.0500, 10.2500,
       90.5000  /)
00226 !!      linie_informacyjne%deltah = (/ 0.0199, 0.0495, 0.9950, 9.9500, 89.7500,
       179.5000 /)
00227 !!      linie_informacyjne%delta  = (/ 0.0002, 0.0010, 0.0100, 0.1000, 0.5000 ,
       1.0000   /)
00228 !!      linie_informacyjne%fine_l = (/ 'F'   , 'F'   , 'F'   , 'F'   , 'C'    ,
       'C'      /)
00229 end subroutine
00230
```

```
00231 subroutine plot2green(green_file)
00232   character(len=*),intent (in) :: green_file
00233 !!  integer                          :: ile_linii_komentarza , i, j , jj ,
       ile_rekordow , io
00234 !!  logical                          :: czy_komentarz=.true.
00235 !!  character(len=1)                 :: fine
00236 !!  real,dimension(:,:), allocatable :: values
00237 !!  real, dimension(7)               :: values_interpolowane=0.,
       values_interpolowane_integrated=0.
00238 !!  real,dimension(:), allocatable   :: b,c,d
00239 !!  real,dimension(3)                :: G_t
00240 !!  real                             :: dist
00241 !
00242 !!ile_rekordow=0; ile_linii_komentarza=0
00243 !!call wczytaj_linie_informacyjne
00244 !
00245 !
00246 !!  open(1, file=trim(green_file)              , action='read'  , status='old')
00247 !!  open(2, file=trim(green_file)//'.mrp02.dat', action='write')
00248 !
00249 !!  do while(czy_komentarz)
00250 !!    read(1, *) dummy
00251 !!    if( dummy(1:1).eq.'#') then
00252 !!      ile_linii_komentarza=ile_linii_komentarza+1
00253 !!    else
00254 !!      czy_komentarz=.false.
00255 !!    endif
00256 !!  enddo
00257 !!  rewind (1)
00258 !!  do i=1, ile_linii_komentarza
00259 !!    read (1,*) dummy
00260 !!  enddo
00261 !!  do while (io.eq.0)
00262 !!    read(1,*, iostat=io) dummy
00263 !!    ile_rekordow=ile_rekordow+1
00264 !!  enddo
00265 !!  ile_rekordow=ile_rekordow-1
00266 !
00267 !!  allocate(values(ile_rekordow,4))
00268 !!  allocate(b(ile_rekordow))
00269 !!  allocate(c(ile_rekordow))
00270 !!  allocate(d(ile_rekordow))
00271 !
00272 !!  rewind(1)
00273 !!  print *, ile_linii_komentarza,ile_rekordow
00274 !!  do i=1, ile_linii_komentarza
00275 !!    read (1,*) dummy
00276 !!  enddo
00277 !!  do i=1, ile_rekordow
00278 !!    read (1,*) (values(i,j), j=1,4)
00279 !  enddo
00280 !
00281 !
00282 !!  write(2,'(a)'), '# program '
00283 !!  do i=1,size(linie_informacyjne)
00284 !!      write(2, '(i1, i3, 2i4, 3f10.4, 5x, a1)'), linie_informacyjne(i)
00285 !!      write(*, '(i1, i3, 2i4, 3f10.4, 5x, a1)'), linie_informacyjne(i)
00286 !!      do j= 1, linie_informacyjne(i)%Nj
00287 !!         dist =
       linie_informacyjne(i)%deltal+(j-1)*linie_informacyjne(i)%delta
00288 !!!         print * ,dist
00289 !!        do jj=2,4
00290 !!          call spline(values(:,1), values(:,jj) ,b,c,d,ile_rekordow)
00291 !!          values_interpolowane(jj-1)  = ispline(dist , values(:,1),
       values(:,jj), b, c, d, ile_rekordow)
00292 !!          call pointmass2integrated(values_interpolowane(jj-1), dist ,
       linie_informacyjne(i)%delta , K(jj-1), values_interpolowane_integrated(jj-1) )
00293 !!!          print*,ile_rekordow, values(1,1),
       values_interpolowane(jj-1),dist,values_interpolowane_integrated(jj-1)
00294 !!!call exit
00295 !!        enddo
00296 !!        write(2,'(7e13.6)') (values_interpolowane_integrated(jj),jj=1,7)
00297 !!      enddo
00298 !!  enddo
00299 !!  close(1)
00300 !!  close(2)
00301 !!  deallocate(values,b,c,d)
00302 end subroutine
00303
00304 subroutine green2plot(green_file)
00305   character(len=*),intent (in) :: green_file
00306 !!  character(len=1) :: fine
00307 !!  integer :: ngr, j, M, Nj, i, ii, iii, i_plik
00308 !!  real :: deltal, deltah, delta,dist
00309 !!  real, dimension(7) :: val
00310 !!  real,dimension(3) :: G_t
00311 !
```

```
00312 !! print *,trim(green_file)
00313 !
00314 !!  open(1,file=trim(green_file),action='read',status='old')
00315 !!  open(2,file=trim(green_file)//'.dat_i',action='write')
00316 !!  open(3,file=trim(green_file)//'.dat',action='write')
00317 !
00318 !!  read (1,'(a70)') header
00319 !!  write(2,*),'# '//trim(header)
00320 !!  write(3,*),'# '//trim(header)
00321 !!  write(2,*),'# Przerobione z pliku w formacie spotl - mrajner'
00322 !!  write(3,*),'# Przerobione z pliku w formacie spotl - mrajner'
00323 !
00324 !
00325 !!  read (1,'(i1,i3,2i4)') ngr,j,M,Nj
00326 !!  rewind(1)
00327 !!  read (1,*) header
00328 !
00329 !
00330 !!  do i=1,M
00331 !!    read (1,'(i1,i3,2i4,3f10.4,5x,a1)') ngr,j,M,Nj,deltal, deltah,delta,fine
00332 !!    do ii=1,Nj
00333 !!      read (1,'(<ngr>e13.6)'), (val(iii),iii=1,7)
00334 !!      dist=deltal+(ii-1)*delta
00335 !!      write (2, '(f10.5,7e)'),dist,val
00336 !
00337 !!      do iii=1,3  ! dla vert_disp, hor_disp, gravity -- jest taka
00338 !!      call integrated2pointmass(val(iii),dist , delta, K(iii),  G_t(iii))
00339 !!      enddo
00340 !!      write (3,'(100(e20.11))') dist,(G_t(iii),iii=1,3)
00341 !!    enddo
00342 !!  enddo
00343 !
00344 end subroutine
00345
00346
00347 ! ==============================================================================
00348 !>
00349 !! chapter 4.1 of spotl manual \cite Agnew12
00350 !!
00351 !! \date 2013-03-15
00352 !! \author M. Rajner
00353 ! ==============================================================================
00354 !subroutine integrated2pointmass(G_integrated,dist, delta, K, G_t)
00355 !  use mod_utilities, only: d2r
00356 !  real(dp), intent (in) :: G_integrated, dist,delta
00357 !  integer , intent(in) :: K
00358 !  real(dp), intent(out) :: G_t
00359 !  real :: G_prim_t
00360
00361 !  G_prim_t = G_integrated  / ( 4 *  cos( d2r(dist) / 2. ) * sin( d2r(delta)
       /4. ) )
00362 !  G_t = G_prim_t * ( ( 10.**K * a ) / ( a**2 * ( 2 *  sin (d2r(dist) /2  )/
       d2r(dist)  ) ) )
00363   !/ ( 10.**K * a * d2r(dist) )
00364 !end subroutine
00365
00366 !subroutine pointmass2integrated(G_t,dist, delta, K, G_integrated)
00367 !!  ! rozdział 4.1 spotlman
00368 !!  implicit none
00369 !!  real, intent (in) :: G_t, dist,delta
00370 !!  integer , intent(in) :: K
00371 !!  real, intent(out) :: G_integrated
00372 !!  real :: G_prim_t
00373 !
00374 !!  G_prim_t = G_t / ( ( 10.**K * a ) / ( a**2 * ( 2 *  sin (d2r(dist) /2  ) /
       d2r(dist)  ) ) )
00375 !!  G_integrated = G_prim_t  * ( 4 *  cos( d2r(dist) / 2. ) * sin( d2r(delta)
       /4. ) )
00376 !!end subroutine
00377 !
00378 !subroutine ignewt_(del,stp,grav_new)
00379 ! width stp (ie, the interval [del-stp/2,del+stp/2],
00380 ! del and stp both being in radians
00381 !  the height correction is included in the green functions,
00382 ! the station height in meters being passed as ht in the common block
00383 ! stloc
00384 !
00385 !!  $$$$$$calls only system routines
00386 !
00387 !implicit none
00388 !!      real ::  eps,eps1,eps2,s,gt,c1
00389 !!      real :: del,stp,g,g2,em ,plc
00390 !!      real , intent(out) :: grav_new
00391 !
00392 !!         eps = wysokosc_stacji/a
00393 !!         eps1=1.+eps
00394 !!         eps2=eps*eps
```

```
00395 !!         g2 = gn/(eps1*eps1)
00396 !!        g = 9.7803327*(1+.005279*ct*ct) - 3.08e-6*wysokosc_stacji
00397 !
00398 !!   em = gn/g
00399 !!   plc = 4*a*em
00400 !!      if(eps.ne.0) then
00401 !!        s=sin(d2r(del+stp/2)/2.d0)
00402 !!        gt=(2.d0*eps1*s**2-eps)/sqrt(4*eps1*s**2+eps2)
00403 !!        s=sin(d2r(del-stp/2)/2.d0)
00404 !!        grav_new=gt-(2.d0*eps1*s**2-eps)/sqrt(4*eps1*s**2+eps2)
00405 !!        grav_new=-g2*grav_new
00406 !!      endif
00407 !!      if(eps.eq.0) then
00408 !!        grav_new=-g2*(sin(( d2r ( del+stp/2 ) )/2.d0)-sin(( d2r(del-stp/2)
      )/2.d0))
00409 !!      endif
00410 !!      return
00411 !!      end subroutine
00412 !
00413 !
00414 !
00415 subroutine getgrf(num,ntot,ngr,fingrd)
00416       character*80 grname
00417       character*1 fingrd
00418       integer llu,ngr,ntot,num
00419          llu =  71
00420          open(unit=llu,file='~/src/spotl/green/gr.gbaver.wef.p01.ce',status=
     'old',action='read')
00421 !
      open(unit=llu,file='~/src/spotl/working/tmpgr',status='old',access='sequential',form="formatted")
00422 !
      open(unit=llu,file='~/dr/merriam/green.dat_zmienione_kolumny.mrp02.dat',status='old',access='sequential',form="formatte
00423 !   read(llu,'(a)') grname
00424 !      endif
00425 !      read(llu,102) ngreen,num,ntot,ngr,beg,end,spc,fingrd
00426 !      fingrd='L' ! IB , tmp
00427 ! 102  format(i1,i3,2i4,3f10.4,5x,a)
00428 !      read(llu,104) ((grfn(ii,j),j=1,7),ii=1,ngr)
00429 ! 104  format(7e13.6)
00430 !      rin(num) = beg
00431 !      rout(num) = end
00432 !      rsz(num) = spc
00433 !     statgr(num) = fingrd
00434 !      nring=ntot
00435 end subroutine
00436
00437 end module
```

11.11 grat/src/real_vs_standard.f90 File Reference

This program.

Functions/Subroutines

- program **real_vs_standard**

11.11.1 Detailed Description

This program.

Todo put description

Definition in file real_vs_standard.f90.

11.12 real_vs_standard.f90

```
00001 !
      ================================================================================
00002 !> \file
00003 !! This program
```

```
00004 !!
00005 !! \todo put description
00006 !
      ================================================================================
00007 program real_vs_standard
00008   use mod_constants, only :dp
00009   use mod_cmdline,   only :intro,cpu_start, cpu_finish,
      print_settings, model , &
00010     dates , sites , output , log , form_separator, green , denser
00011   use mod_green,     only : results, convolve
00012   use mod_data,      only : read_netCDF , get_variable , get_value
00013   use mod_aggf,      only : geop2geom
00014
00015   implicit none
00016   real(dp) :: x , y , z , lat ,lon ,val(0:100) !tmp variables
00017   integer :: i , j, ii, iii
00018
00019   !> program starts here with time stamp
00020   call cpu_time(cpu_start)
00021
00022   ! gather cmd line option decide where to put output
00023   ! todo specific for current program
00024   call intro("rat")
00025
00026   ! print header to log: version, date and summary of command line options
00027   call print_settings("rat")
00028
00029    !read models into memory
00030   do i =1 , size(model)
00031     if (model(i)%if) call read_netcdf( model(i) )
00032   enddo
00033
00034
00035   allocate (results(size(sites)*max(size(dates),1)))
00036   iii=0
00037   do j = 1 , max(size (dates),1)
00038     if(size(dates).gt.0)  write(output%unit, '(i4,5(i2.2))', advance ="no")
      dates(j)%date
00039
00040     do ii = 1 , min(2,size(model))
00041       if (model(ii)%if) call get_variable( model(ii) , date = dates(j)%date)
00042     enddo
00043
00044
00045
00046 !\todo
00047     do i = 1 , size(sites)
00048       write(output%unit, '(2f15.5f)', advance ="no") sites(i)%lat ,sites(i)%lon
00049       iii=iii+1
00050 !      call convolve (sites(i) , green , results(iii), denserdist = denser(1) ,
       denseraz = denser(2))
00051       write (output%unit,'(15f13.5)') , results(iii)%e ,results(iii)%n  ,
      results(iii)%dt , results(iii)%dh, results(iii)%dz
00052     enddo
00053   enddo
00054
00055
00056   call cpu_time(cpu_finish)
00057   write(log%unit, '(/,"Execution time:",1x,f16.9," seconds")') cpu_finish -
      cpu_start
00058   write(log%unit, form_separator)
00059
00060   print * , model(1)%level
00061   print *
00062   lat =00
00063   lon = 00
00064   call get_value(model(1),lat,lon, val(0))
00065
00066   do i =1, size(model(2)%level)
00067     call get_value(model(2),lat,lon, val(i), level = i, method=1)
00068   enddo
00069   print  '(2f10.2)', lat , lon , (val(i),geop2geom(val(i)/1000)*1000.,
       i=0,size(model(2)%level))
00070
00071
00072 end program
```

11.13 grat/src/value_check.f90 File Reference

Functions/Subroutines

- program **value_check**

11.13.1 Detailed Description

Definition in file value_check.f90.

11.14 value_check.f90

```
00001 ! ================================================================================
00002 !>  \file
00003 !! \mainpage
00004 !! \brief ...put...
00005 !! \page value_check-h value_check
00006 !!    \include value_check.hlp
00007 !! \date 2013-01-09
00008 !! \author M. Rajner
00009 !!
00010 !! \date 2013-03-19 added -P (if point is excluded all values are zero)
00011 ! ================================================================================
00012
00013 program value_check
00014   use mod_cmdline  , only: output      , sites , model , dates , &
00015     print_settings , intro,nmodels , polygons,
      form_separator , log
00016   use mod_data     , only: get_variable, get_value,read_netCDF
00017   use mod_constants, only: dp
00018   use mod_polygon  , only: read_polygon, chkgon
00019 !  use ieee_arithmetic
00020
00021   implicit none
00022   real (dp) , allocatable , dimension(:) :: val
00023   integer :: i,ii ,j ,start , imodel, iok
00024
00025   call intro(program_calling = "value_check")
00026   call print_settings(program_calling = "value_check")
00027
00028   do i = 1 , size(model)
00029     if (model(i)%if) call read_netcdf(model(i))
00030   enddo
00031
00032   ! check of exclusion or inclusion in polygon file
00033   ! for every site
00034   call read_polygon(polygons(1))
00035
00036   write(log%unit, form_separator)
00037   allocate (val(nmodels(model)))
00038
00039   start =0
00040   if (size(dates).gt.0) then
00041     start=1
00042     ! print header
00043     write (output%unit , '(a15,x,a14)' , advance = "no" ) "#mjd" , "date"
00044   endif
00045
00046   ! print header
00047   write (output%unit , '(30a15)', advance ="no"  ) "lat" , "lon"
00048   do i = 1 ,size(model)
00049   if (model(i)%if .or. model(i)%if_constant_value ) write (output%unit ,
      '(a15)',advance='no'  ) , trim( model(i)%dataname )
00050   enddo
00051   write (output%unit , *)
00052
00053   do j = start , size (dates)
00054     do i = 1 , size(model)
00055       if (model(i)%if) then
00056         ! only read from multidate files for specific date
00057         ! for 'static' data files get_variable was performed
00058         ! during read_netCDF
00059         if (size(model(i)%date).gt.1) then
00060           call get_variable( model(i) , date = dates(j)%date)
00061         endif
00062       endif
00063     enddo
00064
00065     do i = 1 , size(sites)
00066       ! add time stamp if -D option was specified
00067       if (j.gt.0) then
00068         write (output%unit , '(f15.3,x,i4.4,5(i2.2))' , advance = "no" ) dates(
      j)%mjd , dates(j)%date
00069       endif
00070
00071
00072       ! if this point should not be used (polygon) leave as zero
00073       ! get polygons
```

```
00074       if (polygons(1)%if) then
00075         call chkgon( sites(i)%lon , sites(i)%lat , polygons(1) , iok)
00076       else
00077         iok=1
00078       endif
00079
00080       imodel = 0
00081       do ii = 1 , size (model)
00082         if (model(ii)%if .or. model(ii)%if_constant_value) then
00083           imodel = imodel + 1
00084           if (model(ii)%if) then
00085             if (iok.eq.1) then
00086               call get_value(model(ii), sites(i)%lat, sites(i)%lon, val(imodel)
      , method=model(ii)%interpolation)
00087             else
00088               val(imodel) = 0
00089             endif
00090           elseif(model(ii)%if_constant_value) then
00091             val(imodel) = model(ii)%constant_value
00092           endif
00093         endif
00094       enddo
00095
00096       write (output%unit ,   '(30f15.4)') , sites(i)%lat, sites(i)%lon, val
00097
00098     enddo
00099
00100   enddo
00101
00102 end program
```

11.15 grat/tmp/compar.sh File Reference

11.15.1 Detailed Description

Definition in file compar.sh.

11.16 compar.sh

```
00001 #!/bin/bash -
00002 ## \file
00003 #
      ================================================================================
00004 #          FILE: compar.sh
00005 #         USAGE: ./compar.sh
00006 #   DESCRIPTION:
00007 #       OPTIONS: ---
00008 #        AUTHOR: mrajner
00009 #       CREATED: 13.12.2012 21:15:45 CET
00010 #      REVISION:  ---
00011 #
      ================================================================================
00012
00013 set -o nounset                              # Treat unset variables as an error
00014
00015   WEN="/home/mrajner/pub/2012_wenecja/dane"
00016   SFC="/home/mrajner/src/grat/data/ncep_reanalysis/pres.sfc.2011.nc:pres"
00017   TMP="../data/ncep_reanalysis/air.sig995.2011.nc:air:lon:lat:level:time"
00018   LND="../data/landsea/test.grd:z:x:y"
00019   HGT="../data/topo/ETOPO2v2g_f4.nc:z:x:y"
00020 #  LND="../data/landsea/test_.grd:z:x:y"
00021 #  POL= ../polygon/tmp.poly
00022
00023
00024   numer=354
00025   I=1
00026
00027   TAB=($(sed -ne 2p ${WEN}/szereg_${numer}.txt))
00028   L=$(echo ${TAB[4]}|tr "," " ")
00029   B=$(echo ${TAB[3]}|sed 's/,//')
00030
00031   echo $B $L
00032 #../bin/grat -V -Stmp,${B},${L}  -F${SFC},${TMP},${HGT},${LND}   -Ghuang,huang,
      huang,huang,,1:1     -D20110218,2012  -o${numer}_${I}_5  -I1
00033   grat -V -L:G -Stmp,${B},${L}  -F${SFC},${TMP},${HGT},${LND}   -G,rajner,,,,1:
      1    -D2011,2012  -o${numer}_${I}_3  -I1
00034 #../bin/grat -V -Stmp,${B},${L}  -F${SFC},${TMP},,${LND} -Bi  -G,,,,,1:1
```

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

Chapter 12

Example Documentation

12.1 example_aggf.f90

```
00001 ! ========================================================================
00002 !! This program shows some example of using AGGF module
00003 !!
00004 !! \author Marcin Rajner
00005 !! \date 20121108
00006 ! ========================================================================
00007 program example_aggf
00008   implicit none
00009
00010 !  print *, "...standard1976 ()"
00011 !  call standard1976 ()
00012
00013 !  print *, "...aggf_resp_hmax ()"
00014 !  call aggf_resp_hmax ()
00015
00016 !  print *, "...aggf_resp_dz ()"
00017 !  call aggf_resp_dz ()
00018
00019 !  print *, "...aggf_resp_t ()"
00020 !  call aggf_resp_t ()
00021
00022 !  print *, "...aggf_resp_h ()"
00023 !  call aggf_resp_h ()
00024
00025 !  print *, "...aggfdt_resp_dt ()"
00026 !  call aggfdt_resp_dt ()
00027
00028 !  print *, "...compare_fels_profiles ()"
00029 !  call compare_fels_profiles ()
00030
00031 !  print *, "...compute_tabulated_green_functions ()"
00032 !  call compute_tabulated_green_functions ()
00033
00034 !  print *, "...aggf_thin_layer ()"
00035 !   call aggf_thin_layer ()
00036
00037 !  print *, "...aggf_resp_fels_profiles ()"
00038 !  call aggf_resp_fels_profiles ()
00039
00040 !  print *, "...compare_tabulated_green_functions ()"
00041 !  call compare_tabulated_green_functions ()
00042
00043 !  print *, "...simple_atmospheric_model()"
00044 !  call simple_atmospheric_model()
00045
00046 contains
00047
00048 ! ========================================================================
00049 !> Reproduces data to Fig.~3 in \cite Warburton77
00050 !!
00051 !! \date 2013-03-18
00052 !! \author M. Rajner
00053 !!
00054 ! ========================================================================
00055 subroutine simple_atmospheric_model ()
00056   use mod_constants, only:dp
00057   use mod_aggf, only:simple_def, bouger
00058
00059   real(dp) :: r ! km
```

```
00060   integer :: iunit
00061
00062   open (newunit=iunit,file="/home/mrajner/dr/rysunki/simple_approach.dat" ,&
00063     action = "write")
00064     do r = 0. , 25*8
00065     write ( iunit ,  * ) , r , bouger( r_opt= r) * 1e8, & !conversion to
      microGal
00066       simple_def(r) * 1e8
00067   enddo
00068
00069 end subroutine
00070
00071 ! ==========================================================================
00072 !> Compare tabulated green functions from different authors
00073 !!
00074 !! \date 2013-03-18
00075 !! \author M. Rajner
00076 ! ==========================================================================
00077 subroutine compare_tabulated_green_functions ()
00078   use mod_constants, only : dp
00079   use mod_aggf, only:size_ntimes_denser,read_tabulated_green
00080   use mod_utilities, only : spline_interpolation
00081
00082   integer :: i , j , file_unit , ii , iii
00083   real(dp), dimension(:,:), allocatable :: table , results
00084   real(dp), dimension(:,:), allocatable :: parameters
00085   real(dp), dimension(:), allocatable :: x1, y1 ,x2 , y2 , x, y ,
      x_interpolated, y_interpolated
00086   integer :: how_many_denser
00087   character(len=255), dimension(3) :: authors
00088   integer , dimension(3) :: columns
00089
00090   authors=["rajner", "merriam" , "huang"]
00091   ! selected columns for comparison in appropriate tables
00092   columns=[2 , 2, 2]
00093
00094   how_many_denser=0
00095
00096   ! reference author
00097   call read_tabulated_green(table , author = authors(1) )
00098   allocate (results(size_ntimes_denser(size(table(:,1)),
      how_many_denser) , 0 : size(authors) ))
00099
00100   ! fill abscissa in column 0
00101   ii = 1
00102   do i = 1 ,  size (table(:,1) ) - 1
00103     do j = 0 , how_many_denser
00104         results(ii,0) = table(i,1 ) + j * (table(i+1, 1) -table(i,1) ) / (
      how_many_denser + 1 )
00105         ii=ii+1
00106     enddo
00107   enddo
00108   ! and the last element
00109   results( size (results(:,0) )  , 0) =  table( size(table(:,1)) ,1 )
00110
00111   ! take it as main for all series
00112   allocate(x_interpolated( size ( results(:,0))))
00113   x_interpolated = results(:,0)
00114
00115   open (newunit = file_unit , file = "../examples/compare_aggf.dat", action=
      "write")
00116
00117   ! for every author
00118   do i= 1, size(authors)
00119     print * , trim( authors( i ) )
00120     call read_tabulated_green(table , author = authors(i) )
00121     allocate(x( size (table(:,1))))
00122     allocate(y( size (table(:,2))))
00123     x = table(:,1)
00124     y = table(:, columns(i))
00125     call spline_interpolation( x , y , x_interpolated, y_interpolated )
00126     if (i.gt.1) then
00127       y_interpolated = ( y_interpolated - results(:,1) ) / results(:,1)  * 100.
00128     endif
00129
00130     results(:, i ) = y_interpolated
00131     deallocate(x,y)
00132   enddo
00133
00134   write (file_unit , '(<size(results(1,:))>f20.5)' ) ( results(i , :) , i = 1 ,
       size(results( :,1)) )
00135   close(file_unit)
00136 end subroutine
00137
00138 ! ==========================================================================
00139 !> Compute AGGF and derivatives
00140 !!
```

```
00141 !! \author M. Rajner
00142 !! \date 2013-03-18
00143 ! ==========================================================================
00144 subroutine compute_tabulated_green_functions ()
00145   use mod_constants, only:dp
00146   use mod_aggf, only: read_tabulated_green , compute_aggf,
      compute_aggfdt
00147   integer :: i , file_unit
00148   real(dp) :: val_aggf , val_aggfdt ,val_aggfdh, val_aggfdz
00149   real(dp), dimension(:,:), allocatable :: table , results
00150
00151   ! Get the spherical distances from Merriam92
00152   call read_tabulated_green( table , author = "merriam")
00153
00154   open  ( newunit = file_unit, &
00155           file    = '../dat/rajner_green.dat', &
00156           action  = 'write' &
00157         )
00158
00159   ! print header
00160   write ( file_unit,*) '# This is set of AGGF computed using module ', &
00161   'aggf from grat software'
00162   write ( file_unit,*) '# Normalization according to Merriam92'
00163   write ( file_unit,*) '# Marcin Rajner'
00164   write ( file_unit,*) '# For detail see www.geo.republika.pl'
00165   write ( file_unit,'(10(a23))')  '#psi[deg]', &
00166     'GN[microGal/hPa]'       , 'GN/dT[microGal/hPa/K]' , &
00167     'GN/dh[microGal/hPa/km]' , 'GN/dz[microGal/hPa/km]'
00168
00169   do i= 1, size(table(:,1))
00170     call compute_aggf( table(i,1) , val_aggf   )
00171     call compute_aggfdt( table(i,1) , val_aggfdt )
00172     call compute_aggf( table(i,1) , val_aggfdh , first_derivative_h
      =.true. )
00173     call compute_aggf( table(i,1) , val_aggfdz , first_derivative_z
      =.true. )
00174     write ( file_unit, '(10(e23.5))' ) &
00175       table(i,1) , val_aggf , val_aggfdt , val_aggfdh, val_aggfdz
00176   enddo
00177   close(file_unit)
00178 end subroutine
00179
00180 ! ==========================================================================
00181 !> Compare different vertical temperature profiles impact on AGGF
00182 ! ==========================================================================
00183 subroutine aggf_resp_fels_profiles ()
00184   use mod_constants, only: dp
00185   use mod_aggf, only : read_tabulated_green , compute_aggf
00186   character (len=255) ,dimension (6) :: fels_types
00187   real (dp) :: val_aggf
00188   integer :: i , j, file_unit
00189   real(dp), dimension(:,:), allocatable :: table
00190
00191   ! All possible optional arguments for standard_temperature
00192   fels_types = (/ "US1976"            , "tropical",    &
00193                   "subtropical_summer" , "subtropical_winter" , &
00194                   "subarctic_summer"   , "subarctic_winter"    /)
00195
00196   open  ( newunit = file_unit, &
00197           file    = '../examples/aggf_resp_fels_profiles.dat' , &
00198           action  = 'write' &
00199         )
00200
00201   call read_tabulated_green(table, "merriam")
00202
00203   ! print header
00204   write ( file_unit , '(100(a20))' ) &
00205     'psi', ( trim( fels_types(i) ) , i = 1 , size (fels_types) )
00206
00207   ! print results
00208   do i = 1 , size (table(:,1))
00209     write (file_unit, '(f20.6$)') table(i,1)
00210     do j = 1 , size(fels_types)
00211       call compute_aggf(table(i,1), val_aggf ,fels_type=fels_types(
      j))
00212       write (file_unit, '(f20.6$)') val_aggf
00213     enddo
00214     write(file_unit, *)
00215   enddo
00216   close(file_unit)
00217 end subroutine
00218
00219
00220 ! ==========================================================================
00221 !> Compare different vertical temperature profiles
00222 !!
00223 !! Using tables and formula from \cite Fels86
```

```
00224 !! \author M. Rajner
00225 !! \date 2013-03-19
00226 ! ============================================================================
00227 subroutine compare_fels_profiles ()
00228   use mod_constants, only: dp
00229   use mod_aggf, only : standard_temperature
00230   character (len=255) ,dimension (6) :: fels_types
00231   real (dp) :: height , temperature
00232   integer :: i , file_unit , i_height
00233
00234   ! All possible optional arguments for standard_temperature
00235   fels_types = (/ "US1976"            , "tropical",   &
00236                   "subtropical_summer" , "subtropical_winter" , &
00237                   "subarctic_summer"   , "subarctic_winter"    /)
00238
00239   open  ( newunit = file_unit, &
00240           file    = '../examples/compare_fels_profiles.dat' , &
00241           action  = 'write' &
00242         )
00243
00244   ! Print header
00245   write ( file_unit , '(100(a20))' ) &
00246     'height', ( trim( fels_types(i) ) , i = 1 , size (fels_types) )
00247
00248   ! Print results
00249   do i_height = 0 , 70 , 1
00250     height=dble(i_height)
00251     write ( file_unit , '(f20.3$)' ) , height
00252     do i = 1 , size (fels_types)
00253       call standard_temperature(height, temperature,
      fels_type=fels_types(i))
00254       write ( file_unit , '(f20.3$)' ),  temperature
00255     enddo
00256     write ( file_unit , * )
00257   enddo
00258   close(file_unit)
00259 end subroutine
00260
00261 ! ============================================================================
00262 !> Computes AGGF for different site height (h)
00263 ! ============================================================================
00264 subroutine aggf_resp_h ()
00265   use mod_constants, only : dp
00266   use mod_aggf , only : read_tabulated_green , compute_aggf
00267   real(dp), dimension(:,:), allocatable :: table , results
00268   integer :: i, j, file_unit , ii
00269   real(dp) :: val_aggf
00270
00271   ! Get the spherical distances from Merriam92
00272   call read_tabulated_green( table , author = "merriam")
00273
00274   ! Specify the output table and put station height in first row
00275   allocate ( results( 0 : size (table(:,1)) , 7 ) )
00276   results(0,1) = 1./0     ! Infinity in first header
00277   results(0,3) = 0.0      !   0 m
00278   results(0,3) = 0.001    !   1 m
00279   results(0,4) = 0.01     !  10 m
00280   results(0,5) = 0.1      ! 100 m
00281   results(0,6) = 1.       !   1 km
00282   results(0,7) = 10.      !  10 km
00283
00284   ! write results to file
00285   open (                                  &
00286     newunit = file_unit,                  &
00287     file    = '../examples/aggf_resp_h.dat',  &
00288     action  = 'write'                     &
00289     )
00290
00291   write (file_unit, '(8(F20.8))' ) results(0, :)
00292   do i =1 , size (table(:,1))
00293     ! denser sampling
00294     do ii = 0,8
00295       results( i , 1 )  = table(i,1) + ii * (table(i+1,1) - table(i,1)) / 9.
00296       ! only compute for small spherical distances
00297       if (results(i, 1) .gt. 0.2 ) exit
00298       write (file_unit, '(F20.7,$)') , results(i,1)
00299       do j =  2 , size(results(1,: ) )
00300         call compute_aggf(results(i,1) , val_aggf, dh=dble(0.0001),
      h =results(0,j))
00301         results(i,j) = val_aggf
00302         write (file_unit,'(f20.7,1x,$)') results(i,j)
00303       enddo
00304       write (file_unit,*)
00305     enddo
00306   enddo
00307   close (file_unit)
00308 end subroutine
```

```
00309
00310 ! ========================================================================
00311 !> This computes AGGF for different surface temperature
00312 !!
00313 !! \author M. Rajner
00314 !! \date 2013-03-18
00315 ! ========================================================================
00316 subroutine aggf_resp_t ()
00317   use mod_constants, only : dp , T0
00318   use mod_aggf, only : read_tabulated_green , compute_aggf
00319   real(dp), dimension(:,:), allocatable :: table , results
00320   integer :: i, j , file_unit
00321   real(dp) :: val_aggf
00322
00323   ! read spherical distances from Merriam
00324   call read_tabulated_green( table , "merriam" )
00325
00326   ! Header in first row with surface temperature [K]
00327   allocate ( results(0 : size (table(:,1)) , 4 ) )
00328   results(0,1) = 1./0
00329   results(0,2) = t0 +   0.
00330   results(0,3) = t0 +  15.0
00331   results(0,4) = t0 + -45.0
00332   do i =1 , size (table(:,1))
00333     results( i , 1 )  = table(i,1)
00334     do j =  2 , 4
00335     call compute_aggf( results(i , 1 ) , val_aggf, dh = dble(0.0000
      1), t_zero = results(0, j) )
00336     results(i,j) = val_aggf
00337     enddo
00338   enddo
00339
00340   ! Print results to file
00341   open ( newunit = file_unit , &
00342          file    = '../examples/aggf_resp_t.dat' , &
00343          action  = 'write')
00344   write (file_unit , '(4F20.5)' ) &
00345     ( (results(i,j) , j=1,4) , i = 0, size ( table(:,1) ) )
00346   close (file_unit)
00347 end subroutine
00348
00349 ! ========================================================================
00350 !> \brief This computes AGGFDT for different dT
00351 ! ========================================================================
00352 subroutine aggfdt_resp_dt ()
00353   use mod_constants, only : dp
00354   use mod_aggf , only : read_tabulated_green, compute_aggfdt
00355   real(dp), dimension(:,:), allocatable :: table , results
00356   integer :: i, j , file_unit
00357   real(dp) :: val_aggf
00358
00359   ! read spherical distances from Merriam
00360   call read_tabulated_green( table , "merriam" )
00361
00362   ! Header in first row with surface temperature [K]
00363   allocate ( results(0 : size (table(:,1)) , 6 ) )
00364   results(0,1) = 1./0
00365   results(0,2) = 1.
00366   results(0,3) = 5.
00367   results(0,4) = 10.
00368   results(0,5) = 20.
00369   results(0,6) = 50.
00370   do i =1 , size (table(:,1))
00371     results( i , 1 )  = table(i,1)
00372     do j =  2 , 6
00373       call compute_aggfdt( results(i , 1 ) , val_aggf, results(0,
       j) )
00374       results(i,j) = val_aggf
00375     enddo
00376   enddo
00377
00378   ! Print results to file
00379   open ( newunit = file_unit , &
00380          file    = '../examples/aggfdt_resp_dt.dat' , &
00381          action  = 'write')
00382   write (file_unit , '(6F20.5)' ) &
00383     ( (results(i,j) , j=1,6) , i = 0, size ( table(:,1) ) )
00384   close (file_unit)
00385 end subroutine
00386
00387 ! ========================================================================
00388 !> \brief This computes AGGF for different height integration step
00389 ! ========================================================================
00390 subroutine aggf_resp_dz ()
00391   use mod_constants, only : dp
00392   use mod_aggf , only : read_tabulated_green, compute_aggf
00393   real(dp), dimension(:,:), allocatable :: table , results
```

```
00394   integer :: file_unit , i , j
00395   real(dp) :: val_aggf
00396
00397   open ( newunit = file_unit, &
00398          file    = '../examples/aggf_resp_dz.dat', &
00399          action='write')
00400
00401   ! read spherical distances from Merriam
00402   call read_tabulated_green(table, "merriam")
00403
00404   ! Differences in AGGF(dz) only for small spherical distances
00405   allocate ( results( 0 : 29 , 0: 5 ) )
00406   results = 0.
00407
00408   ! Header in first row [ infty and selected dz follow on ]
00409   results(0,0) = 1./0
00410   results(0,1:5)=(/ 0.0001, 0.001, 0.01, 0.1, 1./)
00411
00412   do i = 1 , size ( results(:,1) ) - 1
00413     results(i,0) = table(i , 1 )
00414     do j = 1 , size (results(1,:) ) - 1
00415     call compute_aggf( results(i,0) , val_aggf , dh = results(0,j)
      )
00416     results(i, j) =  val_aggf
00417     enddo
00418
00419     ! compute relative errors from column 2 for all dz with respect to column 1
00420     results(i,2:) = abs((results(i,2:) - results(i,1)) / results(i,1) * 100 )
00421   enddo
00422
00423   ! write result to file
00424   write ( file_unit , '(<size(results(1,:))>f14.6)' ) &
00425     ((results(i,j), j=0,size(results(1,:)) - 1), i=0,size(results(:,1)) - 1)
00426   close(file_unit)
00427 end subroutine
00428
00429 ! ============================================================================
00430 !> \brief This computes standard atmosphere parameters
00431 !!
00432 !! It computes temperature, gravity, pressure, pressure (simplified formula)
00433 !! density for given height
00434 ! ============================================================================
00435 subroutine standard1976  !()
00436   use mod_constants, only : dp
00437   use mod_aggf, only : standard_temperature, standard_pressure , &
00438     standard_gravity , standard_density
00439   real(dp) :: height , temperature , gravity , pressure , pressure2 , density
00440   integer :: file_unit
00441
00442   open ( newunit = file_unit , &
00443          file    = '../examples/standard1976.dat', &
00444          action  = 'write' )
00445   ! print header
00446   write ( file_unit , '(6(a12))' ) &
00447     'height[km]', 'T[K]' , 'g[m/s2]' , 'p[hPa]', 'p_simp[hPa]' , 'rho[kg/m3]'
00448   do height=0.,98.
00449     call standard_temperature( height , temperature )
00450     call standard_gravity( height , gravity )
00451     call standard_pressure( height , pressure )
00452     call standard_pressure( height , pressure2 ,
      if_simplificated = .true. )
00453     call standard_density( height , density )
00454     ! print results to file
00455     write( file_unit,'(5f12.5, e12.3)'), &
00456     height,temperature , gravity , pressure , pressure2 , density
00457   enddo
00458   close( file_unit )
00459 end subroutine
00460
00461 ! ============================================================================
00462 !> \brief This computes relative values of AGGF for different atmosphere
00463 !! height integration
00464 ! ============================================================================
00465 subroutine aggf_resp_hmax ()
00466   use mod_constants, only : dp
00467   use mod_aggf, only : compute_aggf
00468   real (dp) , dimension (10) :: psi
00469   real (dp) , dimension (:)   , allocatable :: heights
00470   real (dp) , dimension (:,:) , allocatable :: results
00471   integer :: file_unit , i , j
00472   real(dp) :: val_aggf
00473
00474   ! selected spherical distances
00475   psi=(/0.000001, 0.000005,0.00001, 1,  2, 3 , 5, 10 , 90 ,  180 /)
00476
00477   ! get heights (for nice graph) - call auxiliary subroutine
00478   call aux_heights( heights )
```

```
00479
00480   open ( newunit = file_unit , &
00481          file    = '../examples/aggf_resp_hmax.dat', &
00482          action  = 'write')
00483
00484   allocate ( results( 0:size(heights)-1 , 1+size(psi) ) )
00485
00486   do j=0 , size (results(:,1))
00487       results( j , 1 ) = heights(j)
00488
00489     do i = 1 , size(psi)
00490       call compute_aggf( psi(i) , val_aggf , hmax = heights(j) )
00491       results(j,i+1) = val_aggf
00492
00493       !> Relative value of aggf depending on integration height
00494       if (j.gt.0) then
00495         results(j,i+1) = results(j,i+1) / results(0,i+1) * 100
00496       endif
00497     enddo
00498   enddo
00499
00500   ! print header
00501   write(file_unit , '(a14,SP,100f14.5)' ),"#wys\psi", (psi(j) , j= 1,size(psi))
00502   ! print results
00503   do i=1, size (results(:,1))-1
00504     write(file_unit, '(100f14.3)' ) (results(i,j), j = 1, size(psi)+1 )
00505   enddo
00506   close(file_unit)
00507 end subroutine
00508
00509 ! ========================================================================
00510 !> Auxiliary subroutine -- height sampling for semilog plot
00511 ! ========================================================================
00512 subroutine aux_heights ( table )
00513   use mod_constants, only : dp
00514   real(dp) , dimension (:), allocatable, intent(inout) :: table
00515   real(dp) , dimension (0:1000) :: heights
00516   real(dp) :: height
00517   integer :: i , count_heights
00518
00519   heights(0) =60
00520   i=0
00521   height=-0.001
00522   do while (height.lt.60)
00523     i=i+1
00524     if (height.lt.0.10) then
00525       height=height+2./1000
00526     elseif(height.lt.1) then
00527       height=height+50./1000
00528     else
00529       height=height+1
00530     endif
00531     heights(i)= height
00532     count_heights=i
00533   enddo
00534   allocate ( table( 0 : count_heights ) )
00535   table(0 : count_heights ) = heights( 0 : count_heights )
00536 end subroutine
00537
00538 subroutine aggf_thin_layer ()
00539   use mod_constants, only : dp
00540   use mod_aggf, only : read_tabulated_green, GN_thin_layer
00541   integer :: file_unit , i
00542   real(dp) , dimension (:,:), allocatable :: table
00543
00544   ! read spherical distances from Merriam
00545   call read_tabulated_green(table, "merriam")
00546   do i = 1 , size (table(:,1))
00547     write(*,*) table(i,1:2) , gn_thin_layer(table(i,1))
00548   enddo
00549 end subroutine
00550
00551 end program
```

12.2 grat_usage.sh

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

```
#
      ================================================================================

set -o nounset                              # Treat unset variables as an error

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

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

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

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

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

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

Appendix A

Polygon

This examples show how the exclusion of selected polygons works

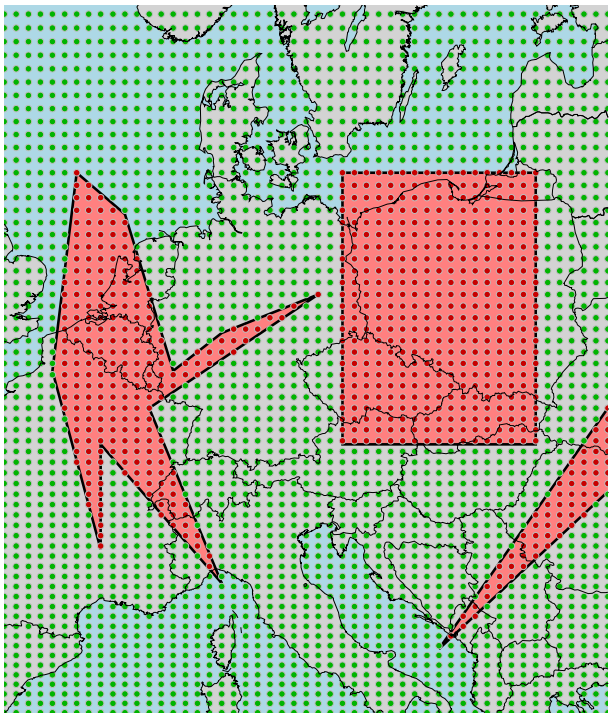


Figure A.1: If only excluded polygons (red area) are given all points falling in it will be excluded (red points) all other will be included

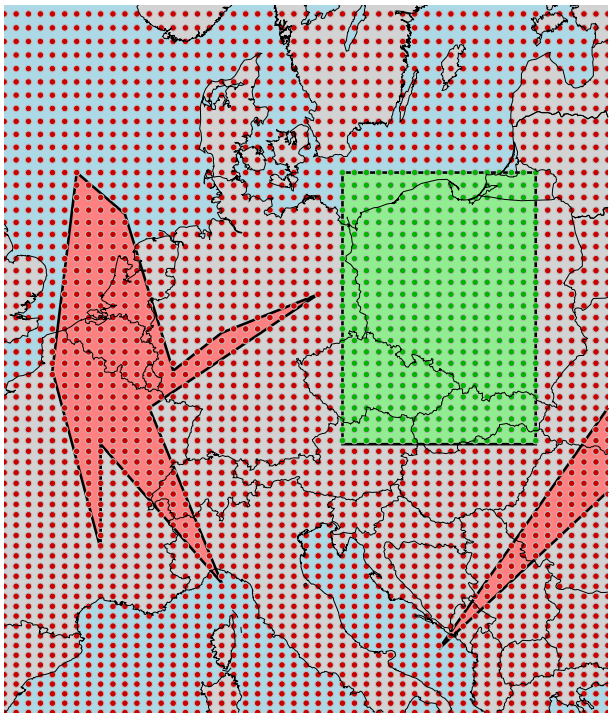


Figure A.2: If at least one included are are given (green area) than all points which not fall into included area will be excluded

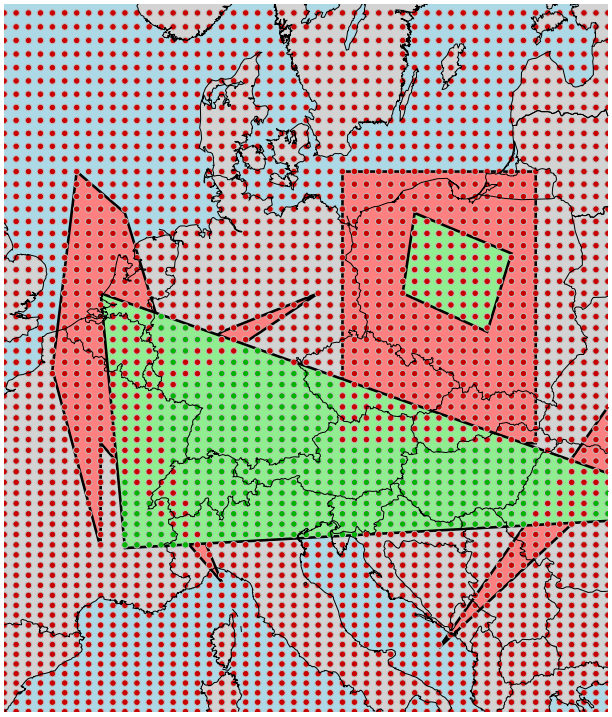


Figure A.3: If there is overlap of polygons the exclusion has higher priority

Appendix B

Interpolation

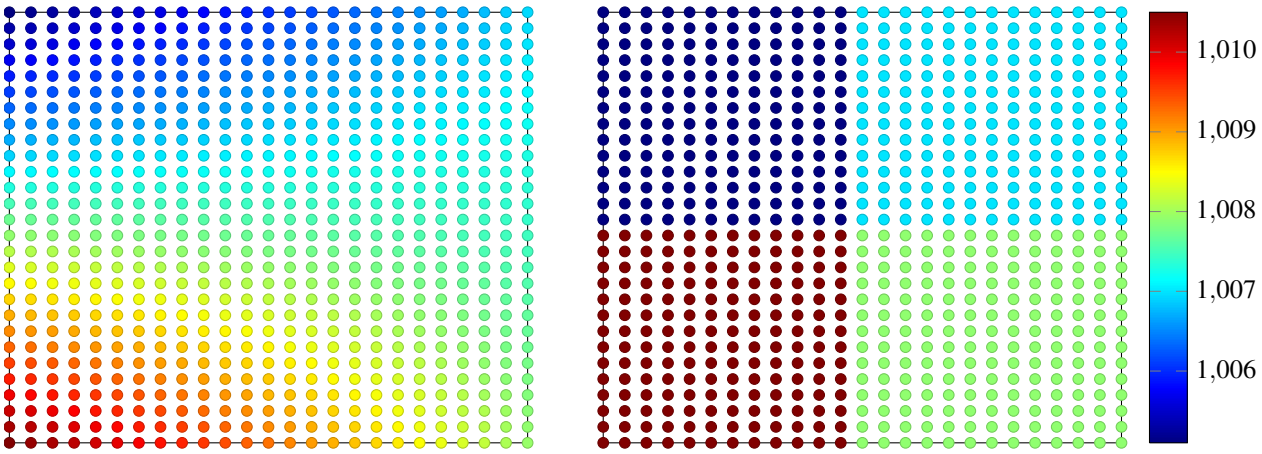


Figure B.1: Interpoloation

Bibliography

netcdf. URL https://www.unidata.ucar.edu/software/netcdf/. 1, 33, 34

D. C. Agnew. Spotl: Some programs for ocean-tide loading. *SIO Ref. Ser.*, 96-8:35 pp., 1996. 37, 41

D. C. Agnew. NLOADF: a program for computing ocean-tide loading. *J. Geophys. Res.*, 102:5109–5110, 1997. 36, 37

COESA Comitee on extension of the Standard Atmosphere. U.S. Standard Atmosphere, 1976. Technical report, 1976. 26

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. 27

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. 24, 25, 26

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. 25, 26

R. J. Warburton and J. M. Goodkind. The influence of barometeic-pressure variations on gravity. *Geophys. J. R. Astron. Society*, 48:281–292, 1977. 25

Index

bouger
    mod_aggf, 22

check
    mod_data, 32
check_if_switch_or_minus
    mod_cmdline, 27
chkgon
    mod_polygon, 34
compute_aggf
    mod_aggf, 22
compute_aggfdt
    mod_aggf, 22
convolve
    mod_green, 33
count_records_to_read
    mod_utilities, 36
count_separator
    mod_cmdline, 27

d2r
    mod_utilities, 36
dataname
    mod_cmdline, 27

file_exists
    mod_utilities, 37

geop2geom
    mod_aggf, 22
getgrf
    mod_green, 33
gn_thin_layer
    mod_aggf, 23
grat/doc/interpolation_ilustration.sh, 43
grat/src/grat.f90, 44
grat/src/mod_aggf.f90, 46
grat/src/mod_cmdline.f90, 53
grat/src/mod_green.f90, 67, 68
grat/src/real_vs_standard.f90, 73
grat/src/value_check.f90, 74, 75
grat/tmp/compar.sh, 76

if_minimum_args
    mod_cmdline, 27
interp
    interpolation_ilustration.sh, 43
interpolation_ilustration.sh
    interp, 43
intro
    mod_cmdline, 28

invmjd
    mod_utilities, 37
is_numeric
    mod_utilities, 37
ispline
    mod_utilities, 37

jd
    mod_utilities, 38

mjd
    mod_utilities, 38
mod_aggf, 21
    bouger, 22
    compute_aggf, 22
    compute_aggfdt, 22
    geop2geom, 22
    gn_thin_layer, 23
    read_tabulated_green, 23
    simple_def, 23
    size_ntimes_denser, 23
    standard_density, 24
    standard_gravity, 24
    standard_pressure, 24
    standard_temperature, 24
mod_cmdline, 25
    check_if_switch_or_minus, 27
    count_separator, 27
    dataname, 27
    if_minimum_args, 27
    intro, 28
    mod_cmdline_entry, 28
    nmodels, 28
    parse_dates, 29
    parse_green, 29
    print_version, 29
    read_site_file, 29
    string2date, 30
mod_cmdline::additional_info, 19
mod_cmdline::cmd_line, 19
mod_cmdline::dateandmjd, 20
mod_cmdline::file, 20
mod_cmdline::green_functions, 21
mod_cmdline::polygon_data, 40
mod_cmdline::polygon_info, 40
mod_cmdline::site_data, 41
mod_cmdline_entry
    mod_cmdline, 28
mod_constants, 30
mod_data, 31

check, 32
nctime2date, 32
unpack_netcdf, 32
mod_green, 32
convolve, 33
getgrf, 33
spher_area, 33
mod_green::result, 41
mod_polygon, 34
chkgon, 34
ncross, 35
read_polygon, 35
mod_utilities, 35
count_records_to_read, 36
d2r, 36
file_exists, 37
invmjd, 37
is_numeric, 37
ispline, 37
jd, 38
mjd, 38
ntokens, 38
r2d, 38
spher_trig, 39
spher_trig_inverse, 39
spline, 39
spline_interpolation, 40

ncross
mod_polygon, 35
nctime2date
mod_data, 32
nmodels
mod_cmdline, 28
ntokens
mod_utilities, 38

parse_dates
mod_cmdline, 29
parse_green
mod_cmdline, 29
print_version
mod_cmdline, 29

r2d
mod_utilities, 38
read_polygon
mod_polygon, 35
read_site_file
mod_cmdline, 29
read_tabulated_green
mod_aggf, 23

simple_def
mod_aggf, 23
size_ntimes_denser
mod_aggf, 23
spher_area
mod_green, 33
spher_trig
mod_utilities, 39
spher_trig_inverse
mod_utilities, 39
spline
mod_utilities, 39
spline_interpolation
mod_utilities, 40
standard_density
mod_aggf, 24
standard_gravity
mod_aggf, 24
standard_pressure
mod_aggf, 24
standard_temperature
mod_aggf, 24
string2date
mod_cmdline, 30

unpack_netcdf
mod_data, 32