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
10.10mod_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
11.10mod_green.f90 70
11.11grat/src/real_vs_standard.f90 File Reference 75
11.11.1 Detailed Description 75
11.12real_vs_standard.f90 75
11.13grat/src/value_check.f90 File Reference 76
11.13.1 Detailed Description 77
11.14value_check.f90 77
11.15grat/tmp/compar.sh File Reference 78
11.15.1 Detailed Description 78
11.16compar.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

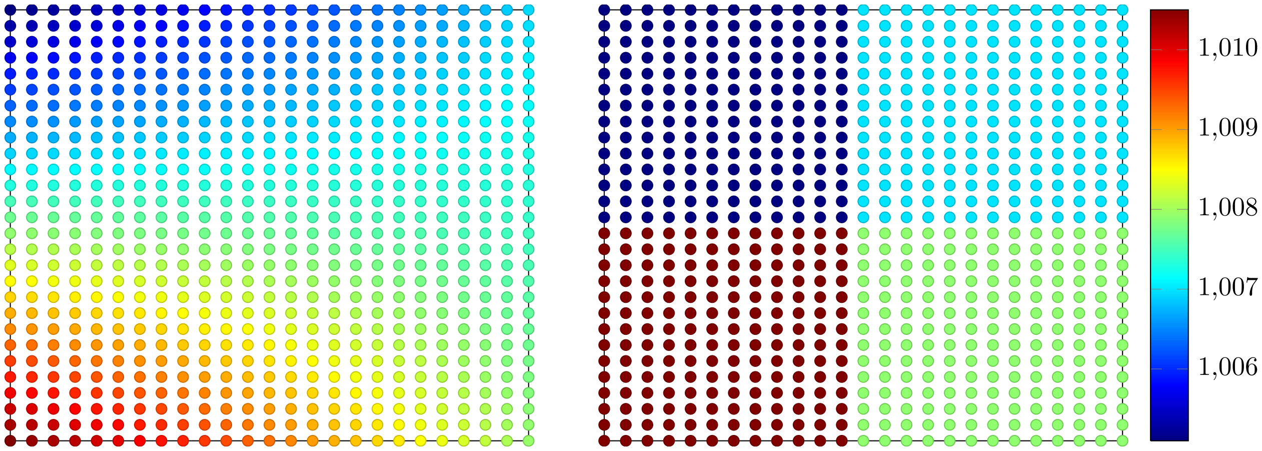
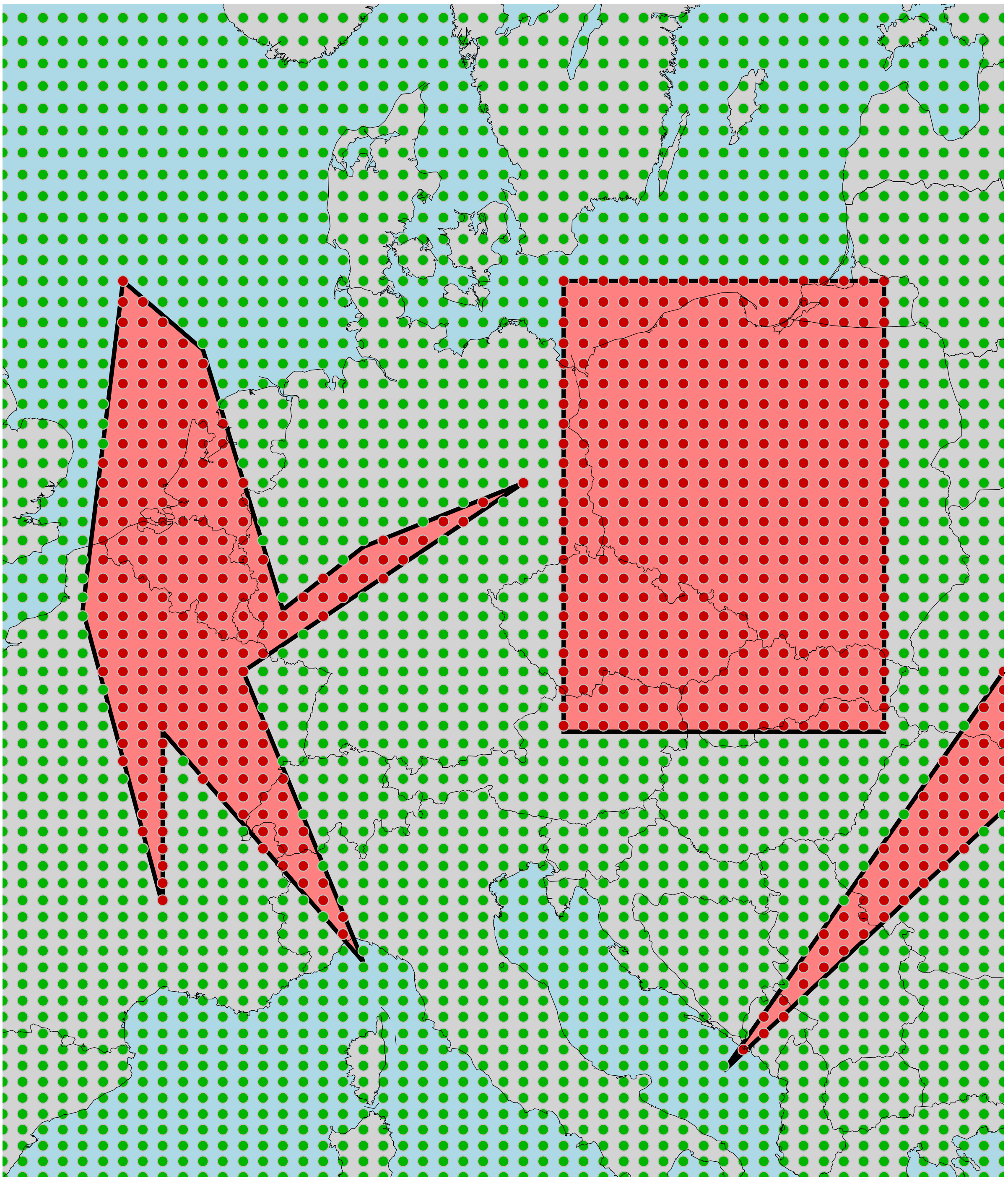
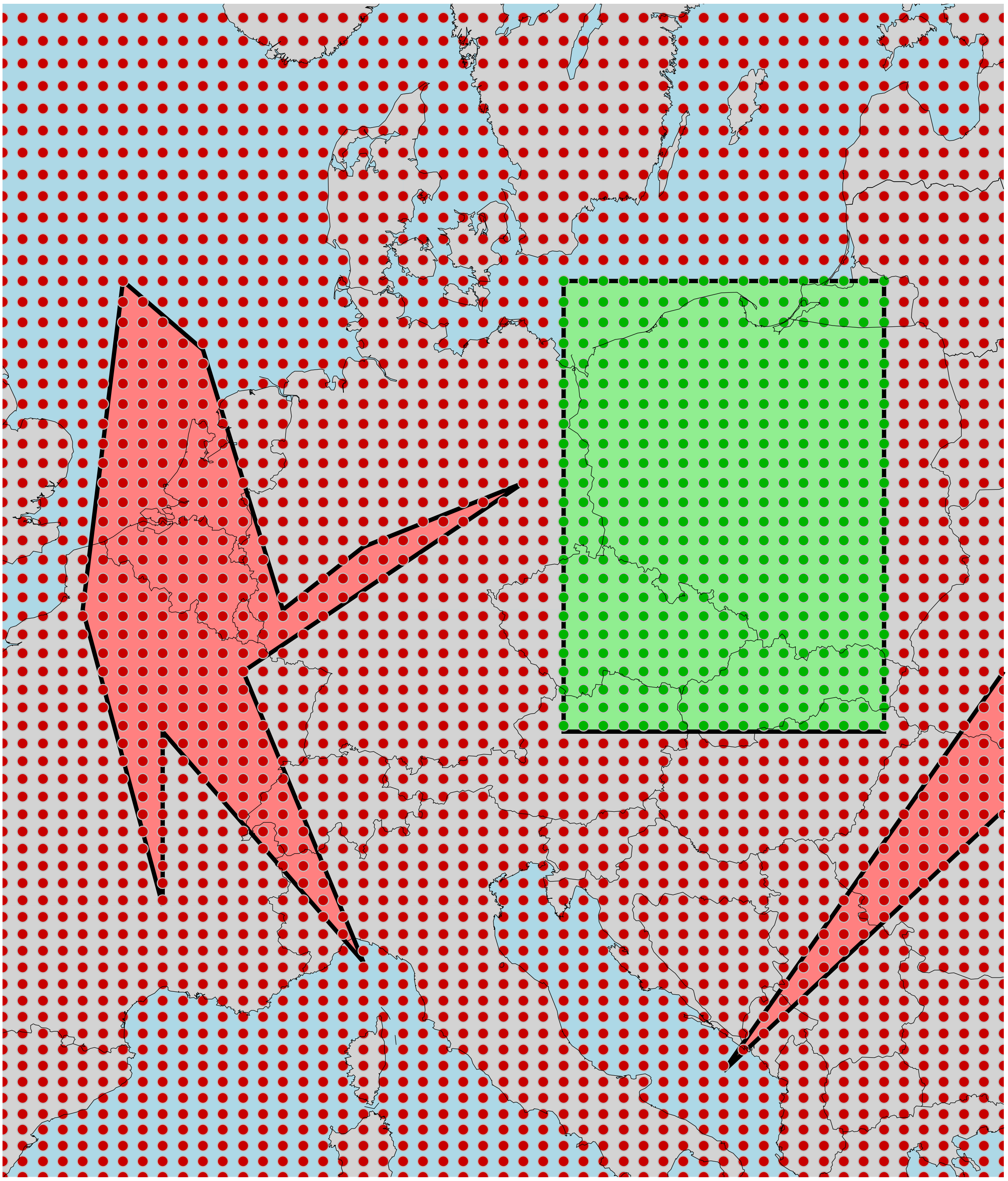
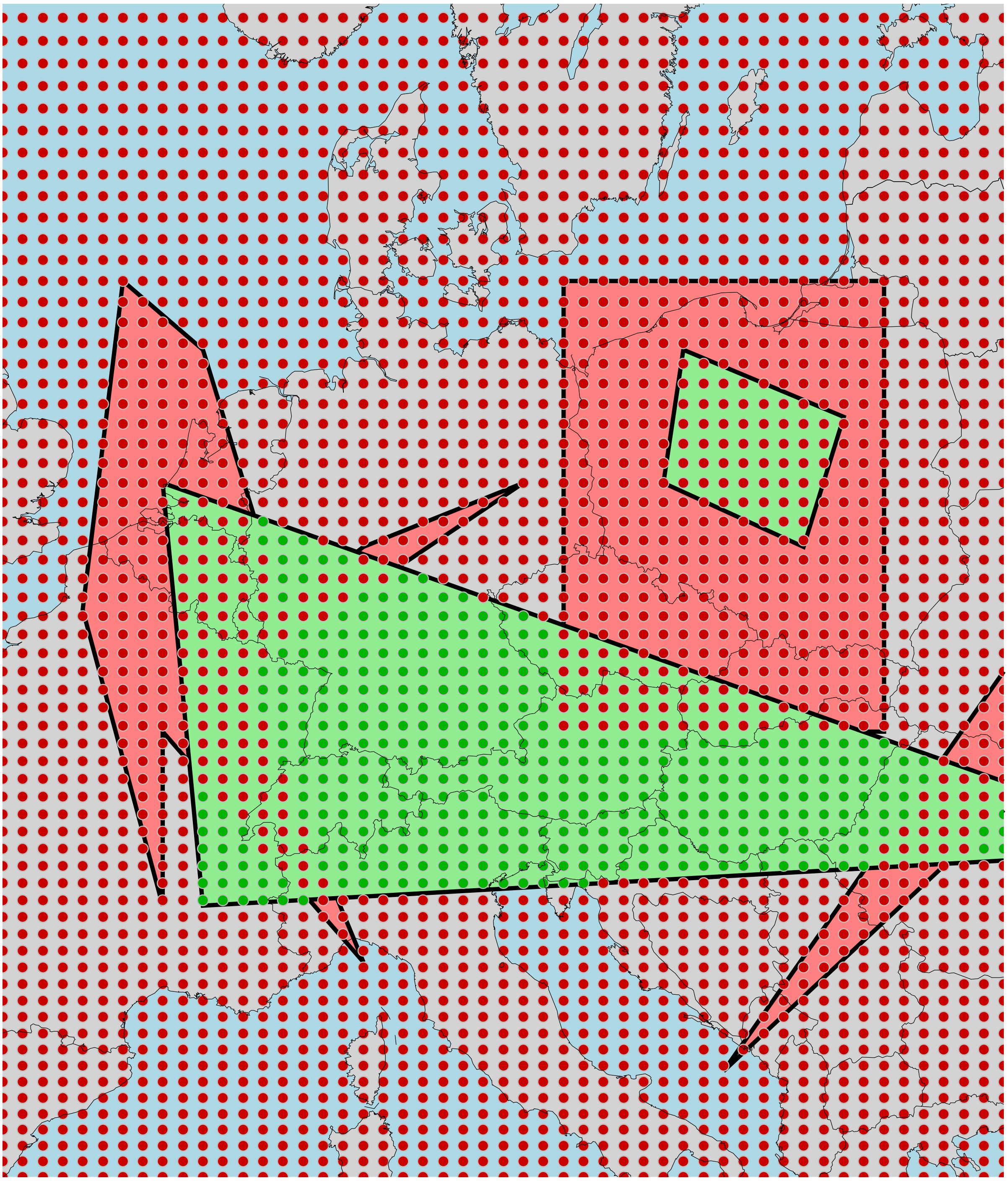


Figure 3.1: example







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

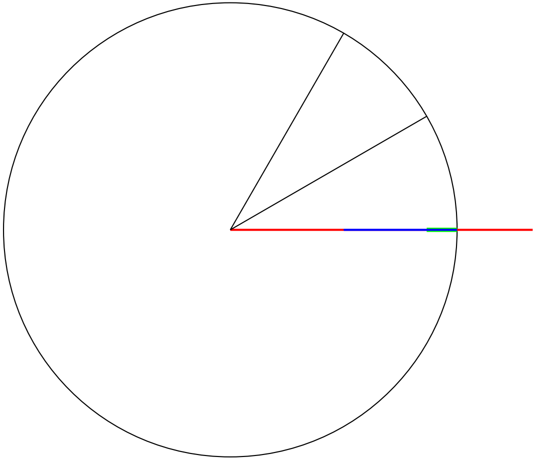
M. Rajner

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
```

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

```

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

```
! ===============================================================================
!> \file
!! \mainpage grat overview
!! \section 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\n
!! Politechnika Warszawska | Warsaw University of Technology
!!
!! \warning This program is written in Fortran90 standard but uses some featerus
!! of 2003 specification (e.g., \c 'newunit='). It was also written
!! for <tt>Intel Fortran Compiler</tt> hence some commands can be unavailable
!! for other compilers (e.g., \c <integer_parameter> for \c IO statements. This should be
!! easily modifiable according to your output needs.
!! Also you need to have \c iso_fortran_env module available to guess the number
!! of output_unit for your compiler.
!! When you don't want a \c log_file and you don't switch \c verbose all
!! unneceserry information whitch are normally collected goes to \c /dev/null
!! file. This is *nix system default trash. For other system or file system
!! organization, please change this value in \c mod_cmdline module.
!!
!! \attention
!! \c grat and value_check needs a \c netCDF library \cite netcdf
!!
!! \section Usage
!! After sucsesfull compiling make sure the executables are in your search path
!!
!! There is main program \c grat and some utilities program. For the options see
!! the appropriate help:
!!  - \link grat-h grat\endlink
!!  - \link value_check-h value_check\endlink
!!  - \link polygon_check-h polygon_check\endlink
!!
!! \page grat-h grat
!!    \include grat.hlp

!> \page ilustration
!! \image latex /home/mrajner/src/grat/doc/interpolation_ilustration.pdf "example"
!! \image latex /home/mrajner/src/grat/doc/mapa1
!! \image latex /home/mrajner/src/grat/doc/mapa2
!! \image latex /home/mrajner/src/grat/doc/mapa3
!!
!! \image html /home/mrajner/src/grat/doc/interpolation_ilustration.png "interpolation example" width=\textwidth
!! \image html /home/mrajner/src/grat/doc/mapa1.png
!! \image html /home/mrajner/src/grat/doc/mapa2.png
!! \image html /home/mrajner/src/grat/doc/mapa3.png

!> \page intro_sec External resources
!!   - <a href="https://code.google.com/p/grat">project page</a> (git repository)
!!   - \htmlonly <a href="../latex/refman.pdf">[pdf]</a> version of this manual\endhtmlonly
!!   \latexonly \href{https://grat.googlecode.com/git/doc/html/index.html}{html} version of this manual\endlatexonly
!! \TODO give source for grant presentation
!!   - <a href="">[pdf]</a> command line options (in Polish)

!> \example example_aggf.f90
!! \example grat_usage.sh
! ==============================================================================
program grat

  use mod_constants , only : dp
  use mod_cmdline   , only : cpu_start , cpu_finish,  intro , print_settings , &
    polygons , model , refpres, form_separator , log ,dates , sites, output, &
    moreverbose, form_60 , form_61, green ,denser
  use mod_green     , only : results ,convolve
  use mod_polygon   , only : read_polygon
  use mod_data      , only : read_netCDF , get_variable

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

  ! program starts here with time stamp
  call cpu_time(cpu_start)

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

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

  ! read polygons
  do i =1 , 2
   call read_polygon(polygons(i))
  enddo

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

  ! todo refpres in get_cmd-line
  if (refpres%if) then
    refpres%name="/home/mrajner/src/grat/data/refpres/vienna_p0.grd"
    call read_netcdf(refpres)
  endif


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

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

    write(log%unit, form_separator)
    write(log%unit, form_60) "Results:"
    if (output%if.and.(output%name /= "")) write(log%unit, form_61) "written into file:" , trim(output%name)
    do i = 1 , size(sites)
      write(output%unit, '(2f15.5f)', advance ="no") sites(i)%lat ,sites(i)%lon
      iii=iii+1
      call convolve(sites(i) , green , results(iii), denserdist = denser(1) , denseraz = denser(2))
      write (output%unit,'(15f13.5)') , results(iii)%e ,results(iii)%n  , results(iii)%dt , results(iii)%dh, results(iii)%dz
    enddo
  enddo


  if (moreverbose%if .and. moreverbose%names(1).eq."s") then
    print '(15f13.5)', &
      results( maxloc( results%e  )  ) %e  - results( minloc( results%e  ) ) %e ,  &
      results( maxloc( results%n  )  ) %n  - results( minloc( results%n  ) ) %n ,  &
      results( maxloc( results%dh )  ) %dh - results( minloc( results%dh ) ) %dh ,  &
      results( maxloc( results%dz )  ) %dz - results( minloc( results%dz ) ) %dz ,  &
      results( maxloc( results%dt )  ) %dt - results( minloc( results%dt ) ) %dt
  endif


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

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

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

  use mod_constants
  implicit none
  private

  public:: size_ntimes_denser, read_tabulated_green , standard_pressure, &
         standard_temperature,               bouger, simple_def, &
             standard_density,     standard_gravity ,       compute_aggf, &
               compute_aggfdt,        gn_thin_layer, geop2geom

contains

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

```
!!
!! optional argument define (-dt;-dt) range
!! See equation 19 in \cite Huang05
!! Same simple method is applied for aggf(gn) if \c aggf optional parameter
!! is set to \c .true.
!! \warning Please do not use \c aggf=.true. this option was added only
!! for testing some numerical routines
!! \author M. Rajner
!! \date 2013-03-19
! ================================================================================
subroutine compute_aggfdt ( psi , aggfdt , delta_ , aggf )
  real(dp) , intent (in) :: psi
  real(dp) , intent (in) , optional :: delta_
  logical , intent (in) , optional :: aggf
  real(dp) , intent (out) :: aggfdt
  real(dp) :: deltat , aux , h_

  deltat = 10. !< Default value
  if (present( delta_) )  deltat = delta_
  if (present( aggf ) .and. aggf ) then
    h_ = 0.001 ! default if we compute dggfdh using this routine
      if (present( delta_) )  h_ = deltat
    call compute_aggf( psi , aux , h = + h_ )
    aggfdt = aux
    call compute_aggf( psi , aux , h= -h_ )
    aggfdt = aggfdt - aux
    aggfdt = aggfdt / ( 2. * h_ )
  else
    call compute_aggf( psi , aux , t_zero = t0 + deltat )
    aggfdt = aux
    call compute_aggf( psi , aux , t_zero = t0 - deltat )
    aggfdt = aggfdt - aux
    aggfdt = aggfdt / ( 2. * deltat)
  endif
end subroutine

! ================================================================================
!> Wczytuje tablice danych AGGF
!! \li merriam \cite Merriam92
!! \li huang   \cite Huang05
!! \li rajner  \cite Rajnerdr
!!
!! This is just quick solution for \c example_aggf program
!! in \c grat see the more general routine \c parse_green()
! ================================================================================
subroutine read_tabulated_green ( table , author )
  use mod_utilities, only: skip_header , count_records_to_read
  real(dp), intent (inout),dimension(:,:), allocatable :: table
  character ( len = * ) , intent (in)                  :: author
  integer                                              :: i , j
  integer                                              :: rows , columns , file_unit
  character (len=255)                                  :: file_name

  if ( author .eq. "huang" ) then
    rows    = 80
    columns = 5
    file_name = ’../dat/huang_green.dat’
  elseif( author .eq. "rajner" ) then
    rows    = 85
    columns = 5
    file_name = ’../dat/rajner_green.dat’
  elseif( author .eq. "merriam" ) then
    rows      = 85
    columns   = 6
    file_name = ’../dat/merriam_green.dat’
  elseif( author .eq. "farrell" ) then
    file_name = ’/home/mrajner/src/gotic2/data/grn1.data’
    call count_records_to_read(file_name, rows = rows, columns = columns)
  else
    write ( * , * ) ’cannot find specified tables, using merriam instead’
  endif

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

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

  call skip_header(file_unit)

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

```
end subroutine


! ==============================================================================
!> This subroutine computes the value of atmospheric gravity green functions
!! (AGGF) on the basis of spherical distance (psi)
! ==============================================================================
subroutine compute_aggf (psi , aggf_val , hmin , hmax , dh , if_normalization, &
                      t_zero , h ,  first_derivative_h , first_derivative_z , fels_type )
  implicit none
  real(dp), intent(in)          :: psi       !< spherical distance from site [degree]
  real(dp), intent(in),optional :: hmin ,  & !< minimum height, starting point [km]      (default=0)
                               hmax ,  & !< maximum height. eding point    [km]      (default=60)
                               dh ,    & !< integration step               [km]      (default=0.0001 -> 10 cm)
                               t_zero, & !< temperature at the surface     [K]       (default=288.15=t0)
                               h         !< station height                 [km]      (default=0)
  logical, intent(in), optional :: if_normalization , first_derivative_h , first_derivative_z
  character (len=*) , intent(in), optional  :: fels_type
  real(dp), intent(out)         :: aggf_val
  real(dp)                      :: r , z , psir , da , dz , rho , h_min , h_max , h_station , j_aux

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

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


  psir = psi * pi / 180.

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


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

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

    !> first derivative (respective to station height)
    !> micro Gal height / km
    if ( present( first_derivative_h) .and. first_derivative_h ) then

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

      !> direct derivative of equation 20 \cite Huang05
      j_aux = (2.* (r0 ) - 2 * (r0 +z )*cos(psir)) / (2. * r)
      j_aux =  -r - 3 * j_aux * ((r0+z)*cos(psir) - r0)
      aggf_val =  aggf_val +  rho * (  j_aux  /  r**4  ) * dz
    else
      !> first derivative (respective to column height)
      !! according to equation 26 in \cite Huang05
      !! micro Gal / hPa / km
      if ( present( first_derivative_z) .and. first_derivative_z ) then
        if (z.eq.h_min) then
            aggf_val = aggf_val  &
              + rho*( ((r0 + z)*cos(psir) - ( r0 + h_station ) ) / ( r**3 ) )
        endif
      else
        !> aggf GN
        !! micro Gal / hPa
        aggf_val = aggf_val  &
         + rho * ( ( (r0 + z ) * cos( psir ) - ( r0 + h_station ) ) / ( r**3 ) ) * dz
      endif
    endif
  enddo

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

  !> if you put the optional parameter \c if_normalization=.false.
  !! this block will be skipped
  !! by default the normalization is applied according to \cite Merriam92
  if ( (.not.present(if_normalization)) .or. (if_normalization)) then
    aggf_val= psir * aggf_val * 1e5  / p0
  endif

end subroutine

! ==============================================================================
!> Compute air density for given altitude for standard atmosphere
!!
!! using formulae 12 in \cite Huang05
!! \date 2013-03-18
!! \author M. Rajner
! ==============================================================================
subroutine standard_density ( height , rho , t_zero ,fels_type )
  real(dp) , intent(in)  ::  height !< height [km]
  real(dp) , intent(in), optional  :: t_zero !< if this parameter is given
  character(len = 22) , optional :: fels_type
  ! surface temperature is set to this value,
  ! otherwise the T0 for standard atmosphere is used
  real(dp) , intent(out) :: rho
  real(dp)  :: p ,t

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

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

! ==============================================================================
!> \brief Computes pressure [hPa] for specific height
!!
!! See \cite US1976 or  \cite Huang05 for details.
!! Uses formulae 5 from \cite Huang05.
!! Simplified method if optional argument if_simplificated = .true.
! ==============================================================================
subroutine standard_pressure (height, pressure , &
          p_zero , t_zero , h_zero,  if_simplificated ,fels_type , inverted)
  implicit none
  real(dp) , intent(in)            :: height
  real(dp) , intent(in) , optional :: t_zero , p_zero , h_zero
  character(len = 22) , optional :: fels_type
  logical         , intent(in) , optional :: if_simplificated
  logical         , intent(in) , optional :: inverted
  real(dp), intent(out) :: pressure
  real(dp) ::  lambda , sfc_height , sfc_temperature , sfc_gravity , alpha , sfc_pressure

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

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

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

  lambda = r_air * sfc_temperature / sfc_gravity

  if (present(if_simplificated) .and. if_simplificated ) then
    ! use simplified formulae
    alpha = -6.5
    pressure = sfc_pressure  &
      * ( 1 + alpha / sfc_temperature * (height-sfc_height)) &
      ** ( -sfc_gravity / (r_air * alpha / 1000. ) )
  else
```

```
    ! use precise formulae
    pressure = sfc_pressure * exp( -1000. * (height -sfc_height) / lambda )
  endif
  if (present(inverted).and.inverted) then
    pressure = sfc_pressure  / ( exp( -1000. * (height-sfc_height) / lambda ) )
  endif


  !todo incorporate this

!  Zdunkowski and Bott
!  p(z) = p0 (T0-gamm z )/T0


end subroutine

! ============================================================================
!> \brief Compute gravity acceleration of the Earth
!! for the specific height using formula
!!
!! see \cite US1976
! ============================================================================
subroutine standard_gravity ( height , g )
  implicit none
  real(dp), intent(in)  :: height
  real(dp), intent(out) :: g

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


! ============================================================================
!> Compute geometric height from geopotential heights
!!
!! \author M. Rajner
!! \date 2013-03-19
! ============================================================================
real(dp) function geop2geom (geopotential_height)
  real (dp) :: geopotential_height

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


! ============================================================================
!> Iterative computation of surface temp. from given height using bisection
!! method
! ============================================================================
subroutine surface_temperature (height , temperature1 , &
  temperature2, fels_type , tolerance)
  real(dp) , intent(in)  :: height , temperature1
  real(dp) , intent(out) :: temperature2
  real(dp) :: temp(3) , temp_ (3) , tolerance_ = 0.1
  character (len=*) , intent(in), optional  :: fels_type
  real(dp) , intent(in), optional  :: tolerance
  integer :: i

  if (present(tolerance)) tolerance_ = tolerance

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

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

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

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

    if ( (temperature1 - temp_(1) ) * (temperature1 - temp_(2) ) .lt.0 ) then
      temp(3) = temp(2)
    elseif( (temperature1 - temp_(3) ) * (temperature1 - temp_(2) ) .lt.0 ) then
      temp(1) = temp(2)
    else
      stop "surface_temp"
    endif
  enddo
end subroutine
```

```
! ==============================================================================
!> \brief Compute standard temperature [K] for specific height [km]
!!
!! if t_zero is specified use this as surface temperature
!! otherwise use T0.
!! A set of predifined temperature profiles ca be set using
!! optional argument \argument fels_type
!! \cite Fels86
!! \li US standard atmosphere (default)
!! \li tropical
!! \li subtropical_summer
!! \li subtropical_winter
!! \li subarctic_summer
!! \li subarctic_winter
! ==============================================================================
subroutine standard_temperature ( height , temperature , t_zero , fels_type )
  real(dp) , intent(in)  :: height
  real(dp) , intent(out) :: temperature
  real(dp) , intent(in), optional  :: t_zero
  character (len=*) , intent(in), optional  :: fels_type
  real(dp) :: aux , cn , t
  integer :: i,indeks
  real(dp) , dimension (10) :: z,c,d

  ! Read into memory the parameters of temparature height profiles
  ! for standard atmosphere
  z = (/11.0 , 20.1 , 32.1 , 47.4 , 51.4 , 71.7 , 85.7, 100.0, 200.0, 300.0/)
  c = (/-6.5,   0.0,   1.0,   2.75,  0.0,  -2.75, -1.97,  0.0,   0.0,   0.0/)
  d = (/ 0.3,   1.0,   1.0,   1.0,   1.0,   1.0,   1.0,   1.0,   1.0,   1.0/)
  t = t0

  if ( present(fels_type)) then
    if (fels_type .eq. "US1976" ) then
    elseif(fels_type .eq. "tropical" ) then
      z=(/ 2.0  , 3.0, 16.5 , 21.5 , 45.0 , 51.0, 70.0 , 100.0 , 200.0 , 300.0 /)
      c=(/-6.0 , -4.0, -6.7  , 4.0  , 2.2  , 1.0, -2.8  , -0.27  , 0.0   , 0.0 /)
      d=(/ 0.5  , 0.5 , 0.3  , 0.5  , 1.0  , 1.0 , 1.0   , 1.0   , 1.0   , 1.0 /)
      t=300.0
    elseif(fels_type .eq. "subtropical_summer" ) then
      z = (/ 1.5  , 6.5  , 13.0 , 18.0 , 26.0 , 36.0  , 48.0 ,  50.0 ,  70.0 , 100.0  /)
      c = (/-4.0 , -6.0  , -6.5  , 0.0  , 1.2  , 2.2  ,  2.5  ,  0.0  ,-3.0 ,-0.025/)
      d = (/ 0.5  , 1.0   , 0.5  , 0.5  , 1.0  , 1.0   , 2.5   , 0.5   , 1.0 , 1.0  /)
      t = 294.0
    elseif(fels_type .eq. "subtropical_winter" ) then
      z = (/ 3.0  ,10.0  , 19.0 , 25.0 , 32.0 , 44.5  , 50.0 ,  71.0 ,  98.0 , 200.0 /)
      c = (/-3.5 , -6.0  , -0.5  , 0.0  , 0.4  , 3.2  ,  1.6  , -1.8  , 0.7 , 0.0 /)
      d = (/ 0.5  , 0.5   , 1.0  , 1.0  , 1.0  , 1.0   , 1.0   , 1.0   , 1.0 , 1.0 /)
      t = 272.2
    elseif(fels_type .eq. "subarctic_summer" ) then
      z = (/  4.7 , 10.0 , 23.0 , 31.8 , 44.0 , 50.2  , 69.2 , 100.0 , 200.0 , 300.0 /)
      c = (/ -5.3 , -7.0 ,  0.0 ,  1.4 ,  3.0 , 0.7  , -3.3  , -0.2   , 0.0 , 0.0 /)
      d = (/  0.5 ,  0.3 ,  1.0 ,  1.0 ,  2.0 , 1.0   , 1.5   , 1.0   , 1.0 , 1.0 /)
      t = 287.0
    elseif(fels_type .eq. "subarctic_winter" ) then
      z = (/  1.0 ,  3.2 ,  8.5 , 15.5 , 25.0 , 30.0 , 35.0 , 50.0 , 70.0 , 100.0 /)
      c = (/  3.0 , -3.2 , -6.8 ,  0.0 , -0.6 ,  1.0 ,  1.2 ,  2.5 , -0.7 ,  -1.2 /)
      d = (/  0.4 ,  1.5 ,  0.3 ,  0.5 ,  1.0 ,  1.0 ,  1.0 ,  1.0 ,  1.0 ,   1.0 /)
      t = 257.1
    else
      print * ,
"unknown fels_type argument: &        using US standard atmosphere 1976 instead"
    endif
  endif

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

  do i=1,10
```

```
    if (height.le.z(i)) then
      indeks=i
      exit
    endif
  enddo

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

! ==============================================================================
!> Compute AGGF GN for thin layer
!!
!! Simple function added to provide complete module
!! but this should not be used for atmosphere layer
!! See eq p. 491 in \cite Merriam92
!! \author M. Rajner
!! \date 2013-03-19
! ==============================================================================
real function gn_thin_layer (psi)
  use mod_utilities, only : d2r
  real(dp) , intent(in) :: psi
  real(dp) :: psir

  psir = d2r(psi)
  gn_thin_layer = 1.627 * psir / sin( psir / 2. )
end function


! ==============================================================================
!> \brief returns numbers of arguments for n times denser size
!!
!! i.e. * * * *  -->  * . . * . . * . . * (3 times denser)
! ==============================================================================
function size_ntimes_denser (size_original, ndenser)
  integer :: size_ntimes_denser
  integer, intent(in) :: size_original , ndenser
  size_ntimes_denser= (size_original - 1 ) * (ndenser +1 ) + 1
end function

! ==============================================================================
!> \brief Bouger plate computation
!!
! ==============================================================================
  real(dp) function bouger ( R_opt )
  real(dp), optional :: r_opt !< height of point above the cylinder
  real(dp) :: aux
  real(dp) :: r
  real(dp) ::  h = 8.84 ! scale height of standard atmosphere

  aux = 1

  if (present( r_opt ) ) then
    r = r_opt
    aux = h  + r  - sqrt(  r**2  + (h/2. ) ** 2 )
    bouger = 2 * pi * g  * aux
  else
    aux = h
    bouger = 2 * pi * g * aux
    return
  endif
end function
! ==============================================================================
!> Bouger plate computation
!!
!! see eq. page 288 \cite Warburton77
!! \date 2013-03-18
!! \author M. Rajner
! ==============================================================================
function simple_def (R)
  real(dp) :: r ,delta
  real(dp) :: simple_def

  delta = 0.22e-11 * r
  simple_def = g0 / r0 * delta * ( 2. - 3./2. * rho_crust / rho_earth &
    -3./4. * rho_crust / rho_earth * sqrt(2* (1. )) ) * 1000
end function

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

```
! ==============================================================================
!> \file
!! \brief This module sets the initial values for parameters
!! reads from command line and gives help
!! it allows to specify commands with or without spaces therefore it is
!! convienient to use with auto completion of names
! ==============================================================================
module mod_cmdline

  use mod_constants, only: dp
  use iso_fortran_env

  implicit none

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

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

  type polygon_info
    integer :: unit
    character(:), allocatable  :: name
    type(polygon_data) , dimension (:) , allocatable :: polygons
    logical :: if
    ! global setting (+|-) which override this in polygon file
    character(1):: pm
  end type

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

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

  real(dp) :: cpu_start , cpu_finish
  type(dateandmjd) , allocatable,dimension (:) :: dates


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

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

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

  !----------------------------------------------------
  ! various
  !----------------------------------------------------
  integer :: fileunit_tmp               !< unit of scratch file
  integer,dimension(8):: execution_date !< To give time stamp of execution
  character (len = 2) :: method = "2D"  !< computation method

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

  type file
    character(:), allocatable :: name

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

    !choose with -F filename@XX:pres...
    character(len=40) :: dataname

    integer :: unit = output_unit

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

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

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

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

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

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

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

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

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

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

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


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

  logical :: if_verbose  = .false.
  logical :: inverted_barometer  = .true.

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

  !----------------------------------------------------
  ! For preety printing
  !----------------------------------------------------
  character(len=255), parameter ::  &
    form_header    = '(60("#"))' , &
    form_separator = '("#",59("-"))' , &
    form_inheader  = '(("#"),1x,a56,1x,("#"))' , &
    form_60        = "(a,100(1x,g0))",          &
    form_61        = "(2x,a,100(1x,g0))",    &
    form_62        = "(4x,a,100(1x,g0))",       &
    form_63        = "(6x,100(x,g0))",       &
    form_64        = "(4x,4x,a,4x,a)"


!  private
!  public :: nmodels

contains
! ============================================================================
!> 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)
! ============================================================================
subroutine intro (program_calling, accepted_switch )
  integer :: i, j
  character(len=255) :: dummy, dummy2,arg
  character(len=*), intent(in) :: program_calling
  type(cmd_line) :: cmd_line_entry
  character(len=*) , intent (in), optional :: accepted_switch

  if(iargc().eq.0) then
    write(output_unit , '(a)' ) , 'Short description: ./'//program_calling//' -h'
    call exit
  else
    open(newunit=fileunit_tmp,status='scratch')
    write (fileunit_tmp,form_61) "command invoked"
    call get_command(dummy)
    write (fileunit_tmp,form_62) trim(dummy)
    do i = 1 , iargc()
      call get_command_argument(i,dummy)
      ! allows specification like '-F file' and '-Ffile'
      ! but  if -[0,9] it is treated as number belonging to switch (-S -2)
      ! but  if -[\s,:] do not start next command line option
      call get_command_argument(i+1,dummy2)
      if (check_if_switch_or_minus(dummy)) then
        arg = trim(dummy)
      else
        arg=trim(arg)//trim(dummy)
      endif
      if(check_if_switch_or_minus(dummy2).or.i.eq.iargc()) then
         call mod_cmdline_entry(arg, cmd_line_entry , program_calling = program_calling)
      endif
    enddo

    call if_minimum_args( program_calling = program_calling )

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

! =============================================================================
!> Check if - 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
! =============================================================================
function check_if_switch_or_minus(dummy)
  use mod_utilities, only: is_numeric
  logical:: check_if_switch_or_minus
  character(*) :: dummy

  check_if_switch_or_minus = .false.
  if (dummy(1:1).eq."-") check_if_switch_or_minus = .true.
  if (dummy(2:2).eq." ") check_if_switch_or_minus = .false.
  if (dummy(2:2).eq.",") check_if_switch_or_minus = .false.
  if (dummy(2:2).eq.":") check_if_switch_or_minus = .false.
  if (is_numeric(dummy(2:2))) check_if_switch_or_minus = .false.
end function

! =============================================================================
!> 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 ....)
! =============================================================================
subroutine if_minimum_args (program_calling)
  character (*) , intent(in) :: program_calling
  type(cmd_line) :: cmd_line_entry
  character(len=100) :: dummy

  ! all programs
!  if (size(sites) .eq. 0) then
!    write(error_unit, * ) "ERROR:", program_calling, " -- no sites!"
!    call exit
!  endif

  if (program_calling.eq."grat" ) then
    ! for grat set default for Green functions if not given in command line
    ! options
    if (.not.allocated(green)) then
      dummy="-G,,,"
      call mod_cmdline_entry(dummy,cmd_line_entry, program_calling="grat")
    endif

    if (size(model) .eq. 0) then
      write(error_unit, * ) "ERROR:", program_calling, " -- model file not specified!"
      call exit
    endif
  elseif(program_calling.eq."polygon_check" ) then
  endif
end subroutine

! ==============================================================================
!> This function is true if switch is used by calling program or false if it
!! is not
! ==============================================================================
logical function if_switch_program (program_calling , switch )
  character(len= *), intent (in) :: program_calling
  character(len= *), intent (in) :: switch
  character, dimension(:) , allocatable :: accepted_switch
  integer :: i

  ! default
  if_switch_program=.false.

  ! depending on program calling decide if switch is permitted
  if (program_calling.eq."grat") then
    allocate( accepted_switch(17) )
    accepted_switch = [ &
      "V", "f", "S", "B", "L", "G", "P", "p", &
      "o", "F", "I", "D", "L", "v", "h", "R", "Q" &
    ]
  elseif(program_calling.eq."polygon_check") then
    allocate( accepted_switch(13) )
    accepted_switch = [ "V" , "f" , "A", "B" , "L" , "P" , "o", "S" , &
      "h" , "v" , "I" , "i" , "R" ]
  elseif(program_calling.eq."value_check") then
    allocate( accepted_switch(11) )
    accepted_switch = [ "V" , "F" , "o", "S" , "h" , "v" , "I" , "D" , "L" , "P" , "R" ]
  else
    if_switch_program=.true.
    return
  endif

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

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

  ! all the command line option are stored in tmp file and later its decide
  ! if it is written to STDOUT , log_file or nowwhere
    select case (cmd_line_entry%switch)
    case (’-h’)
      call print_help(program_calling)
      call exit
    case (’-v’)
      call print_version(program_calling)
      call exit()
    case (’-V’)
      if_verbose = .true.
      write(fileunit_tmp, form_62) ’verbose mode’ ,trim(log%name)
      if (len(trim(cmd_line_entry%field(1))).gt.0) then
        log%if = .true.
        log%name = trim(cmd_line_entry%field(1))
        write(fileunit_tmp, form_62) ’the log file was set:’ ,log%name
      endif
    case (’-S’,’-R’)
      ! check if format is proper for site
      ! i,e. -Sname,B,L[,H]
      if (.not. allocated(sites)) then
        if (is_numeric(cmd_line_entry%field(2)) &
        .and.is_numeric(cmd_line_entry%field(3)) &
        .and.index(cmd_line_entry%field(1), "/" ).eq.0 &
        .and.(.not.cmd_line_entry%field(1).eq. "Rg" ) &
        ) then
            allocate (sites(1))
            sites(1)%name = trim(cmd_line_entry%field(1))
            read ( cmd_line_entry%field(2) , * ) sites(1)%lat
            if (abs(sites(1)%lat).gt.90.) &
              sites(1)%lat = sign(90.,sites(1)%lat)
            read ( cmd_line_entry%field(3) , * ) sites(1)%lon
            if (sites(1)%lon.ge.360.) sites(1)%lon = mod(sites(1)%lon,360.)
            if (is_numeric(cmd_line_entry%field(4) ) ) then
              read ( cmd_line_entry%field(4) , * ) sites(1)%height
            endif
            write(fileunit_tmp, form_62) 'the site was set (BLH):' , &
              sites(1)%name, real(sites(1)%lat) , &
              real(sites(1)%lon) , real(sites(1)%height)
        else
          ! or read sites from file
          if (file_exists(cmd_line_entry%field(1) ) ) then
            write(fileunit_tmp, form_62) 'the site file was set:' , &
              cmd_line_entry%field(1)
            call read_site_file(cmd_line_entry%field(1))
          elseif(index(cmd_line_entry%field(1), "/" ).ne.0 &
              .or.cmd_line_entry%field(1).eq."Rg")  then
            call parse_gmt_like_boundaries( cmd_line_entry )
          else
            call print_warning( "site" , fileunit_tmp)
          endif
        endif
      else
        call print_warning( "repeated" , fileunit_tmp)
      endif
    case ("-I")
      !> \todo add maximum minimum distances for integration
      write( fileunit_tmp , form_62 , advance="no" ) "interpolation method was set:"
      do i = 1 , cmd_line_entry%fields
        if (is_numeric(cmd_line_entry%field(i))) then
          read ( cmd_line_entry%field(i) , * ) model(i)%interpolation
          write(fileunit_tmp , '(a10,x,$)' ) interpolation_names(model(i)%interpolation)
          if (model(i)%interpolation.gt.size(interpolation_names)) then
            model(i)%interpolation=1
          endif
        endif
      enddo
      write(fileunit_tmp , *)
    case ("-L")
      !> \todo make it mulitichoice: -Lfile:s,file2:b ...
      write (fileunit_tmp , form_62) "printing additional information"
!      allocate(moreverbose(cmd_line_entry%fields))
!      print *,size(moreverbose),"XXXX"
!      do i = 1, cmd_line_entry%fields
!        call get_model_info (moreverbose (i) , cmd_line_entry , i )
!        write (fileunit_tmp , form_62) "file: ", moreverbose(i)%name
!      enddo
!      write (fileunit_tmp , form_62) "what: ", moreverbose%names(1)
!      if (len(moreverbose%name).gt.0 .and. moreverbose%name.ne."") then
!        open (newunit = moreverbose%unit , file = moreverbose%name , action = "write" )
!      endif
    case ("-B")
      if (cmd_line_entry%field(1).eq."N" ) inverted_barometer = .false.
    case ("-Q")
      if (cmd_line_entry%field(1).eq."+" ) refpres%if = .true.
      write (fileunit_tmp , form_62) "Reference pressure was set."
    case ('-D')
      call parse_dates( cmd_line_entry )
    case ('-F')
      allocate(model(cmd_line_entry%fields))
      do i = 1, cmd_line_entry%fields
        call get_model_info(model(i) , cmd_line_entry , i )
      enddo
    case ("-G")
      !> \todo when no given take defaults
      call parse_green(cmd_line_entry)
    case ('-M')
      !> \todo rozbudować
      method = cmd_line_entry%field(1)
      write(fileunit_tmp, form_62), 'method was set: ' , method
    case ('-o')
      output%if=.true.
      output%name=cmd_line_entry%field(1)
      write(fileunit_tmp, form_62), 'output file was set: ' , output%name
      if (len(output%name).gt.0.and. output%name.ne."") then
        open (newunit = output%unit , file = output%name , action = "write"
 )
      endif
    case (’-P’)
      do i = 1, cmd_line_entry%fields
        ! prevent from multiple -P
        if (polygons(i)%if) then
          call print_warning("repeated", fileunit_tmp)
          return
        endif
        polygons(i)%name=cmd_line_entry%field(i)
        if (file_exists((polygons(i)%name))) then
          write(fileunit_tmp, form_62), ’polygon file was set: ’ , polygons(i)
%name
          polygons(i)%if=.true.
          if (allocated(cmd_line_entry%fieldnames)) then
            polygons(i)%pm = trim(cmd_line_entry%fieldnames(i)%names(1))
          endif
        else
          write(fileunit_tmp, form_62), ’file do not exist. Polygon file was
 IGNORED’
        endif
      enddo
    case default
      write(fileunit_tmp,form_62), "unknown argument: IGNORING"
    end select
end subroutine

! ==============================================================================
!> This subroutine parse -G option -- Greens function.
!!
!! This subroutines takes the -G argument specified as follows:
!!   -G
!!
!! \author M. Rajner
!! \date 2013-03-06
! ==============================================================================
subroutine parse_green ( cmd_line_entry)
  use mod_utilities, only: file_exists, is_numeric, skip_header
  type (cmd_line)  :: cmd_line_entry
  character (60) :: filename
  integer :: i , iunit , io_status , lines ,  ii
  integer :: fields(2)= [1,2]
  real (dp) , allocatable , dimension(:) :: tmp

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

    do i = 1 , cmd_line_entry%fields

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

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

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

      allocate(tmp(max(fields(1),fields(2))))
      lines = 0
      open ( newunit =iunit,file=filename,action="read")
      do
        call skip_header(iunit)
        read (iunit , * , iostat = io_status)
        if (io_status == iostat_end) exit
        lines = lines + 1
      enddo
      allocate (green(i)%distance(lines))
      allocate (green(i)%data(lines))
      rewind(iunit)
      lines = 0
      do
        call skip_header(iunit)
        lines = lines + 1
        read (iunit , * , iostat = io_status) tmp
        if (io_status == iostat_end) exit
        green(i)%distance(lines) = tmp(fields(1))
        green(i)%data(lines)     = tmp(fields(2))
      enddo
      deallocate(tmp)
      close(iunit)

      ! file specific
      if (cmd_line_entry%field(i).eq."merriam" .and. i.eq.4) then
        green(i)%data = green(i)%data * (-1.)
      endif
      if (cmd_line_entry%field(i).eq."huang" .and. (i.eq.3.or.i.eq.4)) then
        green(i)%data = green(i)%data * 1000.
      endif
      write(fileunit_tmp , form_63) trim(green_names(i)), &
        trim(cmd_line_entry%field(i)),":", fields
    enddo
end subroutine

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

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


! =============================================================================
!> This subroutine fills the fields of command line entry for every input arg
!!
!! \author M. Rajner
!! \date 2013-03-21
! =============================================================================
subroutine mod_cmdline_entry (dummy , cmd_line_entry ,
 program_calling )
  character(*) :: dummy
  character(:), allocatable :: dummy2
  type (cmd_line),intent(out) :: cmd_line_entry
  character(1) :: separator=","
  character(len=*) , intent(in) , optional :: program_calling
  integer :: i , j , ii , jj

```

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

  dummy=dummy(3:)
  cmd_line_entry%fields = count_separator(dummy) + 1
  allocate(cmd_line_entry%field (cmd_line_entry%fields) )

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

    ! separate field and fieldnames
    if ( index(cmd_line_entry%field(i),":").ne.0 ) then
      dummy2 = trim(cmd_line_entry%field(i))//":"
      allocate ( cmd_line_entry%fieldnames(i)%names(count_separator(dummy2,":") - 1 ))
      do ii = 1, size(cmd_line_entry%fieldnames(i)%names)+1
        jj = index(dummy2, ":")
        if (ii.eq.1) then
          cmd_line_entry%field(i) = dummy2(1:jj-1)
        else
          cmd_line_entry%fieldnames(i)%names(ii-1) = dummy2(1:jj-1)
        endif
        dummy2 = dummy2(jj+1:)
      enddo
    endif
  enddo
  call parse_option(cmd_line_entry , program_calling = program_calling)
end subroutine
!
! ==============================================================================
!> This subroutine fills the model info
! ==============================================================================
subroutine get_model_info (model , cmd_line_entry , field)
  use mod_utilities, only : file_exists, is_numeric
  type(cmd_line),intent(in):: cmd_line_entry
  type(file),intent(inout):: model
  integer :: field , i , indeks

  ! split name and dataname (separated by @ - optional)
  model%name = trim(cmd_line_entry%field(field))
  model%dataname = "NN"
  indeks = index(cmd_line_entry%field(field),'@')
  if (indeks.gt.0) then
    model%name = trim(cmd_line_entry%field(field)(1:indeks-1))
    model%dataname = trim(cmd_line_entry%field(field)(indeks+1:))
  endif
  if (model%name.eq."") return
  write (fileunit_tmp , form_62) , trim(dataname(model%dataname)), &
    "("//trim(model%dataname)//")"
  if ( file_exists(model%name) ) then
    do i =1 , size (model%names)
      if (size(cmd_line_entry%fieldnames).gt.0) then
        if (i.le.size (cmd_line_entry%fieldnames(field)%names) &
          .and. cmd_line_entry%fieldnames(field)%names(i).ne."" &
          ) then
          model%names(i) = cmd_line_entry%fieldnames(field)%names(i)
        endif
      endif
      write(fileunit_tmp, form_63, advance="no") , trim(model%names(i))
    enddo
    model%if=.true.
    write(fileunit_tmp, form_63)
  elseif(is_numeric(model%name)) then
    model%if_constant_value=.true.
    read (model%name , * ) model%constant_value
    write(fileunit_tmp, form_63), 'constant value was set: ' , model%constant_value
    model%if_constant_value=.true.
  else
    write (fileunit_tmp , form_63 ) "no (correct) model in field: ", field
  endif
end subroutine
```

```
! ==============================================================================
!> P
! ==============================================================================
subroutine parse_gmt_like_boundaries ( cmd_line_entry )
  use mod_constants, only : dp ,dp
  use mod_utilities, only : is_numeric
  real(dp) :: limits (4) , resolution (2) =[1,1]
  real(dp) :: range_lon , range_lat , lat , lon
  character(10) :: dummy
  integer :: i , ii
  type (cmd_line) , intent (in) :: cmd_line_entry
  character(:) ,allocatable :: text
  integer :: n_lon , n_lat

  text = cmd_line_entry%field(1)

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

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

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

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

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

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

end subroutine

! ==============================================================================
!> Read site list from file
!!
!! checks for arguments and put it into array \c sites
! ==============================================================================
subroutine read_site_file ( file_name )
  use mod_utilities, only: is_numeric, ntokens
```

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



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

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

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

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

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

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


! =============================================================================
!> Parse date given as 20110503020103  to yy mm dd hh mm ss and mjd
!!
!! \warning decimal seconds are not allowed
```

```
! ==============================================================================
subroutine parse_dates (cmd_line_entry )
  use mod_utilities, only: is_numeric,mjd,invmjd
  type(cmd_line) cmd_line_entry
  integer , dimension(6) :: start , stop , swap
  real (dp) :: step =6. ! step in hours
  integer :: i

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

  ! allow that stop is previous than start and list in reverse order
  ! chage the sign of step in dates if necessery
  if(mjd(stop).lt.mjd(start).and. step.gt.0) step = -step
  ! or if step is negative
  if(mjd(stop).gt.mjd(start).and. step.lt.0) then
    swap=start
    start=stop
    stop=swap
  endif

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


! ==============================================================================
!> 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
! ==============================================================================
subroutine string2date ( string , date )
  use mod_utilities, only: is_numeric
  integer , dimension(6) ,intent(out):: date
  character (*) , intent(in) :: string
  integer :: start_char , end_char , j

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

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

end subroutine


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

! ==============================================================================
!> Print version of program depending on program calling
!!
!! \author M. Rajner
!! \date 2013-03-06
! ==============================================================================
subroutine print_version (program_calling)
  character(*) :: program_calling
  integer :: version_unit , io_stat
  character(40) :: version

  ! from the file storing version number
  open(newunit=version_unit, file =
’/home/mrajner/src/grat/dat/version.txt’, &
    action = ’read’ , status = ’old’)
  do
    read (version_unit , ’(a)’ , iostat = io_stat ) version
    if (io_stat == iostat_end) exit
    if (version(1:2) == ’ ’//program_calling(1:1)) exit
  enddo
  write(log%unit , form_header )
  write(log%unit,form_inheader ) , trim(program_calling)
  write(log%unit,form_inheader ) , trim(version(3:))
  write(log%unit,form_inheader ) , ’’
  write(log%unit,form_inheader ) , ’Marcin Rajner’
  write(log%unit,form_inheader ) , ’Warsaw University of Technology’
  write(log%unit , form_header )
end subroutine

! ==============================================================================
!> Print settings
! ==============================================================================
subroutine print_settings (program_calling)
  logical :: exists
  character (len=255):: dummy
  integer :: io_status , j
  character(*), intent(in), optional :: program_calling

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

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

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

    !----------------------------------------------------
    ! Site summary
    !----------------------------------------------------
    write(log%unit, form_separator)
    write(log%unit, form_60 ) "Processing:", size(sites), "site(s)"

    if (size(sites).le.15) then
      write(log%unit, ’(2x,a,t16,3a15)’) "Name" , "lat [deg]" , "lon [deg]" ,"H
 [m]"
      do j = 1,size(sites)
        write(log%unit, ’(2x,a,t16,3f15.4)’) &
          sites(j)%name, sites(j)%lat, sites(j)%lon , sites(j)%height
      enddo
    endif

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

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

! =============================================================================
! =============================================================================
subroutine print_help (program_calling)
  character(*) :: program_calling
  integer :: help_unit , io_stat
  character(500)::line
  character(255)::syntax
  logical:: if_print_line = .false., if_optional=.true.

  if_print_line=.false.

  ! change this path according to your settings
  open(newunit=help_unit, file="~/src/grat/dat/help.hlp", action="read", status="old")

  write (log%unit ,"(a)" , advance="no" ) program_calling
  ! first loop - print only syntax with squre brackets if parameter is optional
  do
    read (help_unit , '(a)', iostat=io_stat) line
    if ((io_stat==iostat_end .or. line(1:1) == "-") .and. if_print_line ) then
      if (if_optional) write(log%unit, '(a)' , advance="no") " ["
      if (if_optional) write(log%unit, '(a)' , advance="no") trim(syntax)
      if (if_optional) write(log%unit, '(a)' , advance="no") "]"
    endif
    if (io_stat==iostat_end) then
      write(log%unit, *) " "
      if_print_line = .false.
      exit
    endif
    if(line(1:1)=="-") then
      if(if_switch_program(program_calling , line(1:2) )) then
        if_print_line = .true.
      else
        if(line(1:1)=="-") if_print_line=.false.
      endif
    endif

    if (line(5:13) == "optional " .and. (line(2:2) == program_calling(1:1) .or. line(2:2)=="")) then
      if_optional=.true.
    elseif(line(5:13) == "mandatory") then
      if_optional=.false.
    endif
    if (line(2:2)=="s") then
      syntax = trim(adjustl(line(3:)))
    endif
  enddo
  rewind(help_unit)

  write(log%unit , form_60) , 'Summary of available options for program '//program_calling
  ! second loop - print informations
  do
    read (help_unit , '(a)', iostat=io_stat) line
    if (io_stat==iostat_end) exit

    if(line(1:1)=="-") then
      if(if_switch_program(program_calling , line(1:2) )) then
        if_print_line = .true.
        write (log%unit , form_61 ) trim(line)
      else
        if(line(1:1)=="-") if_print_line=.false.
      endif
    elseif(line(2:2)==program_calling(1:1) .or. line(2:2)=="s") then
      if (if_print_line) then
        write (log%unit , form_61 ) "  "//trim(line(3:))
      endif
    elseif(line(2:2)=="") then
      if (if_print_line) write (log%unit , form_61 ) trim(line)
    endif
  enddo
  close(help_unit)

end subroutine

subroutine print_warning (  warn , unit)
  character (len=*)  :: warn
  integer , optional :: unit
  integer :: def_unit

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

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


! ==============================================================================
!> Counts number of properly specified models
!!
!! \date 2013-03-15
!! \author M. Rajner
! ==============================================================================
integer function nmodels (model)
  type(file) , allocatable, dimension (:) :: model
  integer :: i

  nmodels = 0
  do i = 1 , size (model)
    if (model(i)%if) nmodels =nmodels + 1
    if (model(i)%if_constant_value) nmodels =nmodels + 1
  enddo
end function

! ==============================================================================
!> Attach full dataname by abbreviation
!!
!! \date 2013-03-21
!! \author M. Rajner
! ==============================================================================
function dataname(abbreviation)
  character(len=40) :: dataname
  character(len=2) :: abbreviation

  dataname="unknown"
  if (abbreviation.eq."LS") dataname = "Land-sea mask"
  if (abbreviation.eq."SP") dataname = "Surface pressure"
end function

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

```fortran
!> \file
!! module
module mod_green

  use mod_constants, only: dp
  implicit none
!  private
!  public :: results


  real(dp), allocatable , dimension(:,:)  :: green_common
  type result
    real(dp) :: n=0. , dt=0. ,e=0. , dh=0.,dz=0.
  end type
  type (result), allocatable, dimension(:) :: results
!
!
!
contains

! ============================================================================
!> Unification:
! ============================================================================
subroutine green_unification (green , green_common , denser)
  use mod_constants , only : dp
  use mod_cmdline, only: moreverbose, method ,  green_functions
  use mod_aggf, only: size_ntimes_denser
  use mod_utilities, only:spline_interpolation

  type(green_functions), allocatable , dimension(:) , intent(in) :: green
  integer, optional :: denser
  integer :: i , ndenser , j ,ii
  real(dp), allocatable , dimension(:) :: x , y , dist
  real(dp), allocatable , dimension(:,:) , intent(out) :: green_common


  ndenser=0
  if (present(denser)) ndenser = denser

  allocate (x(size_ntimes_denser(size(green(1)%distance), ndenser)-1))
  allocate(dist(size(x)))
  ii=0
  do i = 1 , size(green(1)%distance)-1
    do j = 0 , ndenser
      ii=ii+1
      x(ii) = green(1)%distance (i)  + (j +1./2.) * (green(1)%distance (i+1) - green(1)%distance (i) ) / ( ndenser + 1 )
      dist(ii) = (green(1)%distance (i+1) -green(1)%distance (i) ) / ( ndenser + 1 )
    enddo
  enddo
  ! x(size(x)) = green(1)%distance(size(green(1)%distance))
  allocate(green_common(size(x) , 7))
  green_common(:,1) = x
  green_common(:,2) = dist
  do i = 1 , 5
    if (size(green).ge.i .and. allocated(green(i)%distance)) then
      call spline_interpolation(green(i)%distance , green(i)%data, x , y)
      green_common(:,i+2) = y
    else
      green_common(:,i+2) = 0
    endif
  enddo
  if (moreverbose%if.and. moreverbose%names(1).eq."G") then
    write(moreverbose%unit , '(7F13.6)' ) (green_common(i,:), i =1,ubound(green_common,1))
  endif
end subroutine

! ============================================================================
!> Calculate area of spherical segment
!!
!! Computes spherical area on unit (default) sphere given by
!! distance from station and azimuth angle
!! xxx
!! \image latex /home/mrajner/src/grat/doc/rysunki/spher_area.pdf
!! \image html /home/mrajner/src/grat/doc/rysunki/spher_area.png
! ============================================================================
subroutine spher_area (distance ,ddistance, azstp,  area, method )
  use mod_constants, only: dp, sp
  use mod_utilities, only: d2r, r2d
  real(dp), intent(out) :: area
  real(dp), intent(in)  :: distance,ddistance
  real(dp), intent(in)  :: azstp
  integer , intent(in), optional :: method


  area =  (-cos(d2r(distance+ddistance/2.)) &
           + cos(d2r(distance-ddistance/2.)))*d2r(azstp)

end subroutine

! ==============================================================================
!> Perform convolution
!!
!! \date 2013-03-15
!! \author M. Rajner
! ==============================================================================
subroutine convolve (site ,  green , results, denserdist , denseraz  )
  use mod_constants, only: pi , dp, t0
  use mod_cmdline , only: site_data, green_functions , moreverbose , &
    inverted_barometer , model , polygons , refpres , method
  use mod_utilities, only: d2r , spher_trig
  use mod_data, only: get_value
  use mod_polygon, only: chkgon

  type(site_data) , intent(in) :: site
  type(green_functions), allocatable , dimension(:) :: green
  integer , intent (in) :: denserdist , denseraz
  real(dp) :: latin , lonin
  integer ::  ndenser , igreen  , iazimuth , nazimuth
  real(dp) :: azimuth
  real(dp) :: lat , lon , area
  real(dp) :: val(4) , ref_p
  integer :: i , iok(2) , npoints
  real(dp) :: normalize
  type (result) ,intent(out)  :: results


  if (.not.allocated(green_common))  then
    call green_unification(green , green_common , denser = denserdist-1)
  endif

  npoints=0
  do igreen = 1 ,size(green_common(:,1))
    nazimuth = max(int(360*sin(d2r(green_common(igreen,1)))),100) * denseraz
    do iazimuth  = 1 , nazimuth
      npoints = npoints + 1
      azimuth = (iazimuth - 1) * 360./nazimuth

      ! get lat and lon of point
      call spher_trig( site%lat , site%lon , green_common(igreen,1) , azimuth , lat , lon)

      ! get values of model
      do i = 1 , size(model)
        if(model(i)%if) then
          call get_value(model(i) , lat , lon , val(i) , level=1, method =model(i)%interpolation)
        else
          val(i) = 0.
        endif
      enddo

      if (refpres%if) then
       call get_value(refpres , lat , lon , ref_p , method =1)
     else
       ref_p=0.
     endif

      ! get polygons
      do i = 1 , 2
        if (polygons(i)%if) then
          call chkgon( lon, lat , polygons(i) , iok(i) )
        else
          iok(i)=1
        endif
      end do

      ! calculate area using spherical formulae
      if (val(1).ne.0) then
        call spher_area(green_common(igreen,1) , green_common(igreen,2), dble(360./nazimuth) , area)

        ! force topography to zero over oceans
        if (val(4).eq.0.and.val(3).lt.0) val(3) = 0.

        ! normalization according to Merriam (1992)
        normalize= 1. / ( 2. * pi * ( 1. - cos( d2r(dble(1.)) ) ) * d2r(green_common(igreen,1)) *1.e5  )

       ! elastic part
        ! if the cell is not over sea and inverted barometer assumption was not set
        ! and is not excluded by polygon
        if ((.not.((val(4).eq.0.and.inverted_barometer).or. iok(2).eq.0)).or.size(model).lt.4) then
          results%e = results%e + (val(1) / 100. -ref_p) * green_common(igreen,7) * area * normalize
        endif
  !       print*, results%e , inverted_barometer , .not.((val(4).eq.0.and.inverted_barometer).or. iok(2).eq.0) ,val(4)
  !       stop

        ! newtonian part
        if(.not. iok(1).eq.0) then
         results%n = results%n   + (val(1)/ 100.-ref_p) * green_common(igreen,3) * area * normalize

         if (model(2)%if.and.size(model).ge.2) then
            results%dt = results%dt + (val(1)/ 100.-ref_p) * &
              (green_common(igreen,4)*(val(2)-t0) ) * area * normalize
          endif

         results%dh = results%dh + (val(1)/ 100.-ref_p) * &
          (green_common(igreen,5)*(site%height/1000.) ) * area * normalize

         results%dz = results%dz + (val(1)/ 100.-ref_p) * &
          (green_common(igreen,6)*(val(3)/1000.) ) * area * normalize
       endif
      endif
      if (moreverbose%if.and. moreverbose%names(1).eq."g") then
        call convolve_moreverbose(site%lat,site%lon , azimuth , dble(360./nazimuth) , green_common(igreen,1), green_common(igreen,1))
        write (moreverbose%unit, '(">")')
      endif
    enddo
  enddo
  if (moreverbose%if.and. moreverbose%names(1).eq."i") then
    write (moreverbose%unit, '(a,x,g0)') "Points used in convolution" ,npoints
  endif
end subroutine

!!> \todo site height from model
!
!
subroutine convolve_moreverbose (latin , lonin , azimuth , azstep ,  distance , distancestep)
  use mod_cmdline , only : moreverbose
  use mod_utilities, only: spher_trig

  real(dp), intent(in) :: azimuth ,azstep, latin, lonin
  real(dp) :: distance, lat , lon , distancestep

  call spher_trig( latin , lonin , distance - distancestep/2. , azimuth - azstep/2. , lat , lon)
  write(moreverbose%unit, '(2f12.6)') , lat , lon
  call spher_trig( latin , lonin , distance - distancestep/2. , azimuth + azstep/2. , lat , lon)
  write(moreverbose%unit, '(2f12.6)') , lat , lon
  call spher_trig( latin , lonin , distance + distancestep/2. , azimuth + azstep/2. , lat , lon)
  write(moreverbose%unit, '(2f12.6)') , lat , lon
  call spher_trig( latin , lonin , distance + distancestep/2. , azimuth - azstep/2. , lat , lon)
  write(moreverbose%unit, '(2f12.6)') , lat , lon
end subroutine


subroutine wczytaj_linie_informacyjne
!!     do i=1,size(linie_informacyjne);        linie_informacyjne(i)%j_l = i;   enddo
!!      linie_informacyjne%Nj     = (/ 95    , 30    , 95    , 90    , 160    , 90       /)
!!      linie_informacyjne%deltal = (/ 0.0011, 0.0205, 0.0550, 1.0500, 10.2500, 90.5000  /)
!!      linie_informacyjne%deltah = (/ 0.0199, 0.0495, 0.9950, 9.9500, 89.7500, 179.5000 /)
!!      linie_informacyjne%delta  = (/ 0.0002, 0.0010, 0.0100, 0.1000, 0.5000 , 1.0000   /)
!!      linie_informacyjne%fine_l = (/ 'F'   , 'F'   , 'F'   , 'F'   , 'C'    , 'C'      /)
end subroutine

```

```
subroutine plot2green(green_file)
  character(len=*),intent (in) :: green_file
!!  integer                          :: ile_linii_komentarza , i, j , jj , ile_rekordow , io
!!  logical                          :: czy_komentarz=.true.
!!  character(len=1)                 :: fine
!!  real,dimension(:,:), allocatable :: values
!!  real, dimension(7)               :: values_interpolowane=0., values_interpolowane_integrated=0.
!!  real,dimension(:), allocatable   :: b,c,d
!!  real,dimension(3)                :: G_t
!!  real                             :: dist
!
!!ile_rekordow=0; ile_linii_komentarza=0
!!call wczytaj_linie_informacyjne
!
!
!!  open(1, file=trim(green_file)              , action='read'   , status='old')
!!  open(2, file=trim(green_file)//'.mrp02.dat', action='write')
!
!!  do while(czy_komentarz)
!!    read(1, *) dummy
!!    if( dummy(1:1).eq.'#') then
!!      ile_linii_komentarza=ile_linii_komentarza+1
!!    else
!!      czy_komentarz=.false.
!!    endif
!!  enddo
!!  rewind (1)
!!  do i=1, ile_linii_komentarza
!!    read (1,*) dummy
!!  enddo
!!  do while (io.eq.0)
!!    read(1,*, iostat=io) dummy
!!    ile_rekordow=ile_rekordow+1
!!  enddo
!!  ile_rekordow=ile_rekordow-1
!
!!  allocate(values(ile_rekordow,4))
!!  allocate(b(ile_rekordow))
!!  allocate(c(ile_rekordow))
!!  allocate(d(ile_rekordow))
!
!!  rewind(1)
!!  print *, ile_linii_komentarza,ile_rekordow
!!  do i=1, ile_linii_komentarza
!!    read (1,*) dummy
!!  enddo
!!  do i=1, ile_rekordow
!!    read (1,*)  (values(i,j), j=1,4)
!  enddo
!
!
!!  write(2,'(a)'), '# program '
!!  do i=1,size(linie_informacyjne)
!!      write(2, '(i1, i3, 2i4, 3f10.4, 5x, a1)'), linie_informacyjne(i)
!!      write(*, '(i1, i3, 2i4, 3f10.4, 5x, a1)'), linie_informacyjne(i)
!!      do j= 1, linie_informacyjne(i)%Nj
!!         dist = linie_informacyjne(i)%deltal+(j-1)*linie_informacyjne(i)%delta
!!!         print * ,dist
!!        do jj=2,4
!!          call spline(values(:,1), values(:,jj) ,b,c,d,ile_rekordow)
!!          values_interpolowane(jj-1)  = ispline(dist , values(:,1), values(:,jj), b, c, d, ile_rekordow)
!!          call pointmass2integrated(values_interpolowane(jj-1), dist , linie_informacyjne(i)%delta , K(jj-1), values_interpolowane_integrated(jj-1) )
!!!          print*,ile_rekordow, values(1,1), values_interpolowane(jj-1),dist,values_interpolowane_integrated(jj-1)
!!!call exit
!!        enddo
!!        write(2,'(7e13.6)') (values_interpolowane_integrated(jj),jj=1,7)
!!      enddo
!!  enddo
!!  close(1)
!!  close(2)
!!  deallocate(values,b,c,d)
end subroutine

subroutine green2plot(green_file)
  character(len=*),intent (in) :: green_file
!!  character(len=1) :: fine
!!  integer :: ngr, j, M, Nj, i, ii, iii, i_plik
!!  real :: deltal, deltah, delta,dist
!!  real, dimension(7) :: val
!!  real,dimension(3) :: G_t
!
```

```
!! print *,trim(green_file)
!
!!  open(1,file=trim(green_file),action='read',status='old')
!!  open(2,file=trim(green_file)//'.dat_i',action='write')
!!  open(3,file=trim(green_file)//'.dat',action='write')
!
!!  read (1,'(a70)') header
!!  write(2,*),'# '//trim(header)
!!  write(3,*),'# '//trim(header)
!!  write(2,*),'# Przerobione z pliku w formacie spotl - mrajner'
!!  write(3,*),'# Przerobione z pliku w formacie spotl - mrajner'
!
!
!!  read (1,'(i1,i3,2i4)') ngr,j,M,Nj
!!  rewind(1)
!!  read (1,*) header
!
!
!!  do i=1,M
!!    read (1,'(i1,i3,2i4,3f10.4,5x,a1)') ngr,j,M,Nj,deltal, deltah,delta,fine
!!    do ii=1,Nj
!!      read (1,'(<ngr>e13.6)'), (val(iii),iii=1,7)
!!      dist=deltal+(ii-1)*delta
!!      write (2, '(f10.5,7e)'),dist,val
!
!!      do iii=1,3  ! dla vert_disp, hor_disp, gravity -- jest taka
!!      call integrated2pointmass(val(iii),dist , delta, K(iii),  G_t(iii))
!!      enddo
!!      write (3,'(100(e20.11))') dist,(G_t(iii),iii=1,3)
!!    enddo
!!  enddo
!
end subroutine


! ==============================================================================
!>
!! chapter 4.1 of spotl manual \cite Agnew12
!!
!! \date 2013-03-15
!! \author M. Rajner
! ==============================================================================
!subroutine integrated2pointmass(G_integrated,dist, delta, K, G_t)
!  use mod_utilities, only: d2r
!  real(dp), intent (in) :: G_integrated, dist,delta
!  integer , intent(in) :: K
!  real(dp), intent(out) :: G_t
!  real :: G_prim_t

!  G_prim_t = G_integrated  / ( 4 *  cos( d2r(dist) / 2. ) * sin( d2r(delta)
 /4. ) )
!  G_t = G_prim_t * ( ( 10.**K * a ) / ( a**2 * ( 2 *  sin (d2r(dist) /2  )/
 d2r(dist)  ) ) )
  !/ ( 10.**K * a * d2r(dist) )
!end subroutine

!subroutine pointmass2integrated(G_t,dist, delta, K, G_integrated)
!!  ! rozdział 4.1 spotlman
!!  implicit none
!!  real, intent (in) :: G_t, dist,delta
!!  integer , intent(in) :: K
!!  real, intent(out) :: G_integrated
!!  real :: G_prim_t
!
!!  G_prim_t = G_t / ( ( 10.**K * a ) / ( a**2 * ( 2 *  sin (d2r(dist) /2  ) /
 d2r(dist)  ) ) )
!!  G_integrated = G_prim_t  * ( 4 *  cos( d2r(dist) / 2. ) * sin( d2r(delta)
 /4. ) )
!!end subroutine
!
!subroutine ignewt_(del,stp,grav_new)
! width stp (ie, the interval [del-stp/2,del+stp/2],
! del and stp both being in radians
!  the height correction is included in the green functions,
! the station height in meters being passed as ht in the common block
! stloc
!
!!  $$$$$$calls only system routines
!
!implicit none
!!      real ::  eps,eps1,eps2,s,gt,c1
!!      real :: del,stp,g,g2,em ,plc
!!      real , intent(out) :: grav_new
!
!!         eps = wysokosc_stacji/a
!!         eps1=1.+eps
!!         eps2=eps*eps
!!         g2 = gn/(eps1*eps1)
!!        g = 9.7803327*(1+.005279*ct*ct) - 3.08e-6*wysokosc_stacji
!
!!   em = gn/g
!!   plc = 4*a*em
!!      if(eps.ne.0) then
!!        s=sin(d2r(del+stp/2)/2.d0)
!!        gt=(2.d0*eps1*s**2-eps)/sqrt(4*eps1*s**2+eps2)
!!        s=sin(d2r(del-stp/2)/2.d0)
!!        grav_new=gt-(2.d0*eps1*s**2-eps)/sqrt(4*eps1*s**2+eps2)
!!        grav_new=-g2*grav_new
!!      endif
!!      if(eps.eq.0) then
!!        grav_new=-g2*(sin(( d2r ( del+stp/2 ) )/2.d0)-sin(( d2r(del-stp/2) )/2.d0))
!!      endif
!!      return
!!      end subroutine
!
!
!
subroutine getgrf(num,ntot,ngr,fingrd)
      character*80 grname
      character*1 fingrd
      integer llu,ngr,ntot,num
         llu =  71
         open(unit=llu,file='~/src/spotl/green/gr.gbaver.wef.p01.ce',status='old',action='read')
! open(unit=llu,file='~/src/spotl/working/tmpgr',status='old',access='sequential',form="formatted")
! open(unit=llu,file='~/dr/merriam/green.dat_zmienione_kolumny.mrp02.dat',status='old',access='sequential',form="formatte
!   read(llu,'(a)') grname
!      endif
!      read(llu,102) ngreen,num,ntot,ngr,beg,end,spc,fingrd
!      fingrd='L' ! IB , tmp
! 102  format(i1,i3,2i4,3f10.4,5x,a)
!      read(llu,104) ((grfn(ii,j),j=1,7),ii=1,ngr)
! 104  format(7e13.6)
!      rin(num) = beg
!      rout(num) = end
!      rsz(num) = spc
!     statgr(num) = fingrd
!      nring=ntot
end subroutine

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

```
! ==============================================================================
!> \file
!! This program
```

```
!!
!! \todo put description
! ==============================================================================
program real_vs_standard
  use mod_constants, only :dp
  use mod_cmdline,   only :intro,cpu_start, cpu_finish, print_settings, model , &
    dates , sites , output , log , form_separator, green , denser
  use mod_green,     only : results, convolve
  use mod_data,      only : read_netCDF , get_variable , get_value
  use mod_aggf,      only : geop2geom

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

  !> program starts here with time stamp
  call cpu_time(cpu_start)

  ! gather cmd line option decide where to put output
  ! todo specific for current program
  call intro("rat")

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

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


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

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



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


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

  print * , model(1)%level
  print *
  lat =00
  lon = 00
  call get_value(model(1),lat,lon, val(0))

  do i =1, size(model(2)%level)
    call get_value(model(2),lat,lon, val(i), level = i, method=1)
  enddo
  print  '(2f10.2)', lat , lon , (val(i),geop2geom(val(i)/1000)*1000., i=0,size(model(2)%level))


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

```fortran
! ================================================================================
!>  \file
!! \mainpage
!! \brief ...put...
!! \page value_check-h value_check
!!    \include value_check.hlp
!! \date 2013-01-09
!! \author M. Rajner
!!
!! \date 2013-03-19 added -P (if point is excluded all values are zero)
! ================================================================================

program value_check
  use mod_cmdline  , only: output      , sites , model , dates , &
    print_settings , intro,nmodels , polygons, form_separator , log
  use mod_data     , only: get_variable, get_value,read_netCDF
  use mod_constants, only: dp
  use mod_polygon  , only: read_polygon, chkgon
!  use ieee_arithmetic

  implicit none
  real (dp) , allocatable , dimension(:) :: val
  integer :: i,ii ,j ,start , imodel, iok

  call intro(program_calling = "value_check")
  call print_settings(program_calling = "value_check")

  do i = 1 , size(model)
    if (model(i)%if) call read_netcdf(model(i))
  enddo

  ! check of exclusion or inclusion in polygon file
  ! for every site
  call read_polygon(polygons(1))

  write(log%unit, form_separator)
  allocate (val(nmodels(model)))

  start =0
  if (size(dates).gt.0) then
    start=1
    ! print header
    write (output%unit , ’(a15,x,a14)’ , advance = "no" ) "#mjd" , "date"
  endif

  ! print header
  write (output%unit , ’(30a15)’, advance ="no"  ) "lat" , "lon"
  do i = 1 ,size(model)
  if (model(i)%if .or. model(i)%if_constant_value ) write (output%unit , ’(a15)’,advance=’no’  ) , trim( model(i)%dataname )
  enddo
  write (output%unit , *)

  do j = start , size (dates)
    do i = 1 , size(model)
      if (model(i)%if) then
        ! only read from multidate files for specific date
        ! for ’static’ data files get_variable was performed
        ! during read_netCDF
        if (size(model(i)%date).gt.1) then
          call get_variable( model(i) , date = dates(j)%date)
        endif
      endif
    enddo

    do i = 1 , size(sites)
      ! add time stamp if -D option was specified
      if (j.gt.0) then
        write (output%unit , ’(f15.3,x,i4.4,5(i2.2))’ , advance = "no" ) dates(j)%mjd , dates(j)%date
      endif


      ! if this point should not be used (polygon) leave as zero
      ! get polygons
      if (polygons(1)%if) then
        call chkgon( sites(i)%lon , sites(i)%lat , polygons(1) , iok)
      else
        iok=1
      endif

      imodel = 0
      do ii = 1 , size (model)
        if (model(ii)%if .or. model(ii)%if_constant_value) then
          imodel = imodel + 1
          if (model(ii)%if) then
            if (iok.eq.1) then
              call get_value(model(ii), sites(i)%lat, sites(i)%lon, val(imodel), method=model(ii)%interpolation)
            else
              val(imodel) = 0
            endif
          elseif(model(ii)%if_constant_value) then
            val(imodel) = model(ii)%constant_value
          endif
        endif
      enddo

      write (output%unit ,   '(30f15.4)') , sites(i)%lat, sites(i)%lon, val

    enddo

  enddo

end program
```

11.15 grat/tmp/compar.sh File Reference

11.15.1 Detailed Description

Definition in file compar.sh.

11.16 compar.sh

```
#!/bin/bash -
## \file
#================================================================================
#          FILE: compar.sh
#         USAGE: ./compar.sh
#   DESCRIPTION:
#       OPTIONS: ---
#        AUTHOR: mrajner
#       CREATED: 13.12.2012 21:15:45 CET
#      REVISION:  ---
#================================================================================

set -o nounset                              # Treat unset variables as an error

  WEN="/home/mrajner/pub/2012_wenecja/dane"
  SFC="/home/mrajner/src/grat/data/ncep_reanalysis/pres.sfc.2011.nc:pres"
  TMP="../data/ncep_reanalysis/air.sig995.2011.nc:air:lon:lat:level:time"
  LND="../data/landsea/test.grd:z:x:y"
  HGT="../data/topo/ETOPO2v2g_f4.nc:z:x:y"
#  LND="../data/landsea/test_.grd:z:x:y"
#  POL= ../polygon/tmp.poly


  numer=354
  I=1

  TAB=($(sed -ne 2p ${WEN}/szereg_${numer}.txt))
  L=$(echo ${TAB[4]}|tr "," " ")
  B=$(echo ${TAB[3]}|sed 's/,//')

  echo $B $L
#../bin/grat -V -Stmp,${B},${L}  -F${SFC},${TMP},${HGT},${LND}   -Ghuang,huang,huang,huang,,1:1     -D20110218,2012  -o${numer}_${I}_5  -I1
  grat -V -L:G -Stmp,${B},${L}  -F${SFC},${TMP},${HGT},${LND}   -G,rajner,,,,1:1    -D2011,2012  -o${numer}_${I}_3  -I1
#../bin/grat -V -Stmp,${B},${L}  -F${SFC},${TMP},,${LND} -Bi  -G,,,,,1:1
```

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

```
! ==============================================================================
!! This program shows some example of using AGGF module
!!
!! \author Marcin Rajner
!! \date 20121108
! ==============================================================================
program example_aggf
  implicit none

!  print *, "...standard1976 ()"
!  call standard1976 ()

!  print *, "...aggf_resp_hmax ()"
!  call aggf_resp_hmax ()

!  print *, "...aggf_resp_dz ()"
!  call aggf_resp_dz ()

!  print *, "...aggf_resp_t ()"
!  call aggf_resp_t ()

!  print *, "...aggf_resp_h ()"
!  call aggf_resp_h ()

!  print *, "...aggfdt_resp_dt ()"
!  call aggfdt_resp_dt ()

!  print *, "...compare_fels_profiles ()"
!  call compare_fels_profiles ()

!  print *, "...compute_tabulated_green_functions ()"
!  call compute_tabulated_green_functions ()

!  print *, "...aggf_thin_layer ()"
!   call aggf_thin_layer ()

!  print *, "...aggf_resp_fels_profiles ()"
!  call aggf_resp_fels_profiles ()

!  print *, "...compare_tabulated_green_functions ()"
!  call compare_tabulated_green_functions ()

!  print *, "...simple_atmospheric_model()"
!  call simple_atmospheric_model()

contains

! ===============================================================================
!> Reproduces data to Fig.~3 in \cite Warburton77
!!
!! \date 2013-03-18
!! \author M. Rajner
!!
! ===============================================================================
subroutine simple_atmospheric_model ()
  use mod_constants, only:dp
  use mod_aggf, only:simple_def, bouger

  real(dp) :: r ! km
  integer :: iunit

  open (newunit=iunit,file="/home/mrajner/dr/rysunki/simple_approach.dat" ,&
    action = "write")
    do r = 0. , 25*8
    write ( iunit ,  * ) , r , bouger( r_opt= r) * 1e8, & !conversion to microGal
      simple_def(r) * 1e8
  enddo

end subroutine

! ==============================================================================
!> Compare tabulated green functions from different authors
!!
!! \date 2013-03-18
!! \author M. Rajner
! ==============================================================================
subroutine compare_tabulated_green_functions ()
  use mod_constants, only : dp
  use mod_aggf, only:size_ntimes_denser,read_tabulated_green
  use mod_utilities, only : spline_interpolation

  integer :: i , j , file_unit , ii , iii
  real(dp), dimension(:,:), allocatable :: table , results
  real(dp), dimension(:,:), allocatable :: parameters
  real(dp), dimension(:), allocatable :: x1, y1 ,x2 , y2 , x, y , x_interpolated, y_interpolated
  integer :: how_many_denser
  character(len=255), dimension(3) :: authors
  integer , dimension(3) :: columns

  authors=["rajner", "merriam" , "huang"]
  ! selected columns for comparison in appropriate tables
  columns=[2 , 2, 2]

  how_many_denser=0

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

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

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

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

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

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

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

! =============================================================================
!> Compute AGGF and derivatives
!!
```

```
!! \author M. Rajner
!! \date 2013-03-18
! ============================================================================
subroutine compute_tabulated_green_functions ()
  use mod_constants, only:dp
  use mod_aggf, only: read_tabulated_green , compute_aggf, compute_aggfdt
  integer :: i , file_unit
  real(dp) :: val_aggf , val_aggfdt ,val_aggfdh, val_aggfdz
  real(dp), dimension(:,:), allocatable :: table , results

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

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

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

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

! ============================================================================
!> Compare different vertical temperature profiles impact on AGGF
! ============================================================================
subroutine aggf_resp_fels_profiles ()
  use mod_constants, only: dp
  use mod_aggf, only : read_tabulated_green , compute_aggf
  character (len=255) ,dimension (6) :: fels_types
  real (dp) :: val_aggf
  integer :: i , j, file_unit
  real(dp), dimension(:,:), allocatable :: table

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

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

  call read_tabulated_green(table, "merriam")

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

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


! ============================================================================
!> Compare different vertical temperature profiles
!!
!! Using tables and formula from \cite Fels86
!! \author M. Rajner
!! \date 2013-03-19
! ============================================================================
subroutine compare_fels_profiles ()
  use mod_constants, only: dp
  use mod_aggf, only : standard_temperature
  character (len=255) ,dimension (6) :: fels_types
  real (dp) :: height , temperature
  integer :: i , file_unit , i_height

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

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

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

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

! ============================================================================
!> Computes AGGF for different site height (h)
! ============================================================================
subroutine aggf_resp_h ()
  use mod_constants, only : dp
  use mod_aggf , only : read_tabulated_green , compute_aggf
  real(dp), dimension(:,:), allocatable :: table , results
  integer :: i, j, file_unit , ii
  real(dp) :: val_aggf

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

  ! Specify the output table and put station height in first row
  allocate ( results( 0 : size (table(:,1)) , 7 ) )
  results(0,1) = 1./0     ! Infinity in first header
  results(0,3) = 0.0      !   0 m
  results(0,3) = 0.001    !   1 m
  results(0,4) = 0.01     !  10 m
  results(0,5) = 0.1      ! 100 m
  results(0,6) = 1.       !   1 km
  results(0,7) = 10.      !  10 km

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

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

```

! ============================================================================
!> This computes AGGF for different surface temperature
!!
!! \author M. Rajner
!! \date 2013-03-18
! ============================================================================
subroutine aggf_resp_t ()
  use mod_constants, only : dp , T0
  use mod_aggf, only : read_tabulated_green , compute_aggf
  real(dp), dimension(:,:), allocatable :: table , results
  integer :: i, j , file_unit
  real(dp) :: val_aggf

  ! read spherical distances from Merriam
  call read_tabulated_green( table , "merriam" )

  ! Header in first row with surface temperature [K]
  allocate ( results(0 : size (table(:,1)) , 4 ) )
  results(0,1) = 1./0
  results(0,2) = t0 +   0.
  results(0,3) = t0 +  15.0
  results(0,4) = t0 + -45.0
  do i =1 , size (table(:,1))
    results( i , 1 )  = table(i,1)
    do j =  2 , 4
    call compute_aggf( results(i , 1 ) , val_aggf, dh = dble(0.0000
1), t_zero = results(0, j) )
    results(i,j) = val_aggf
    enddo
  enddo

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

! ============================================================================
!> \brief This computes AGGFDT for different dT
! ============================================================================
subroutine aggfdt_resp_dt ()
  use mod_constants, only : dp
  use mod_aggf , only : read_tabulated_green, compute_aggfdt
  real(dp), dimension(:,:), allocatable :: table , results
  integer :: i, j , file_unit
  real(dp) :: val_aggf

  ! read spherical distances from Merriam
  call read_tabulated_green( table , "merriam" )

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

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

! ============================================================================
!> \brief This computes AGGF for different height integration step
! ============================================================================
subroutine aggf_resp_dz ()
  use mod_constants, only : dp
  use mod_aggf , only : read_tabulated_green, compute_aggf
  real(dp), dimension(:,:), allocatable :: table , results
  integer :: file_unit , i , j
  real(dp) :: val_aggf

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

  ! read spherical distances from Merriam
  call read_tabulated_green(table, "merriam")

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

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

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

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

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

! ==============================================================================
!> \brief This computes standard atmosphere parameters
!!
!! It computes temperature, gravity, pressure, pressure (simplified formula)
!! density for given height
! ==============================================================================
subroutine standard1976  !()
  use mod_constants, only : dp
  use mod_aggf, only : standard_temperature, standard_pressure , &
    standard_gravity , standard_density
  real(dp) :: height , temperature , gravity , pressure , pressure2 , density
  integer :: file_unit

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

! ==============================================================================
!> \brief This computes relative values of AGGF for different atmosphere
!! height integration
! ==============================================================================
subroutine aggf_resp_hmax ()
  use mod_constants, only : dp
  use mod_aggf, only : compute_aggf
  real (dp) , dimension (10) :: psi
  real (dp) , dimension (:)    , allocatable :: heights
  real (dp) , dimension (:,:) , allocatable :: results
  integer :: file_unit , i , j
  real(dp) :: val_aggf

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

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

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

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

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

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

      !> Relative value of aggf depending on integration height
      if (j.gt.0) then
        results(j,i+1) = results(j,i+1) / results(0,i+1) * 100
      endif
    enddo
  enddo

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

! ============================================================================
!> Auxiliary subroutine -- height sampling for semilog plot
! ============================================================================
subroutine aux_heights ( table )
  use mod_constants, only : dp
  real(dp) , dimension (:), allocatable, intent(inout) :: table
  real(dp) , dimension (0:1000) :: heights
  real(dp) :: height
  integer :: i , count_heights

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

subroutine aggf_thin_layer ()
  use mod_constants, only : dp
  use mod_aggf, only : read_tabulated_green, GN_thin_layer
  integer :: file_unit , i
  real(dp) , dimension (:,:), allocatable :: table

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

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

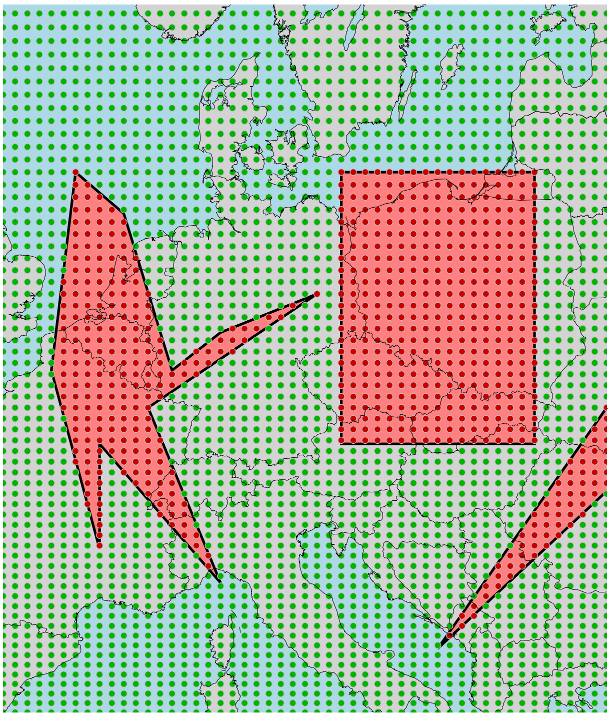


Figure A.1: If only excluded polygons (red area) are given all points falling in it will be excluded (red points) all other will be included

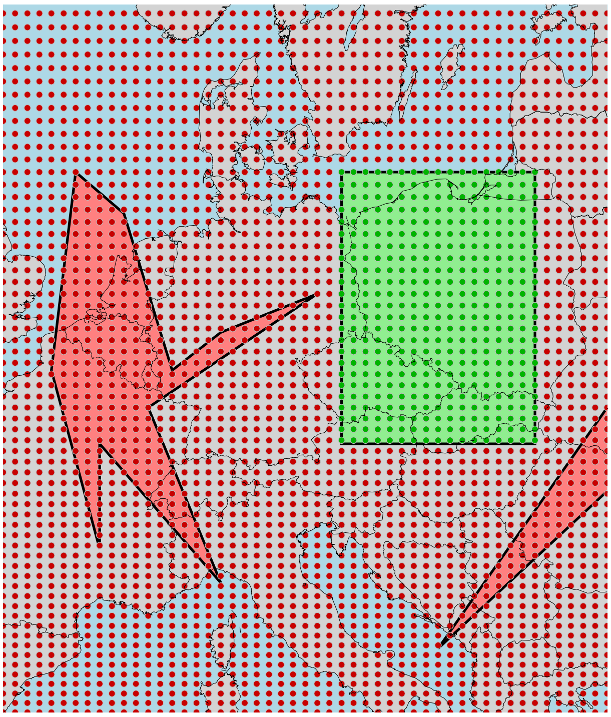


Figure A.2: If at least one included are are given (green area) than all points which not fall into included area will be excluded

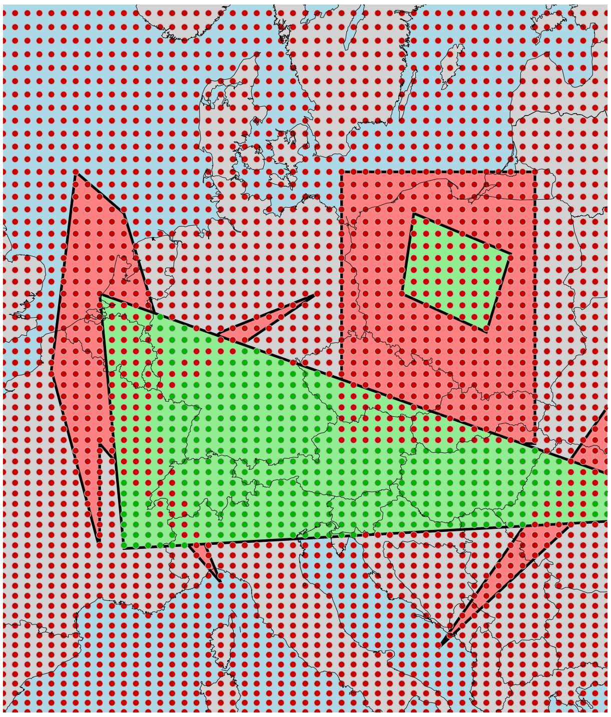


Figure A.3: If there is overlap of polygons the exclusion has higher priority

Appendix B

Interpolation

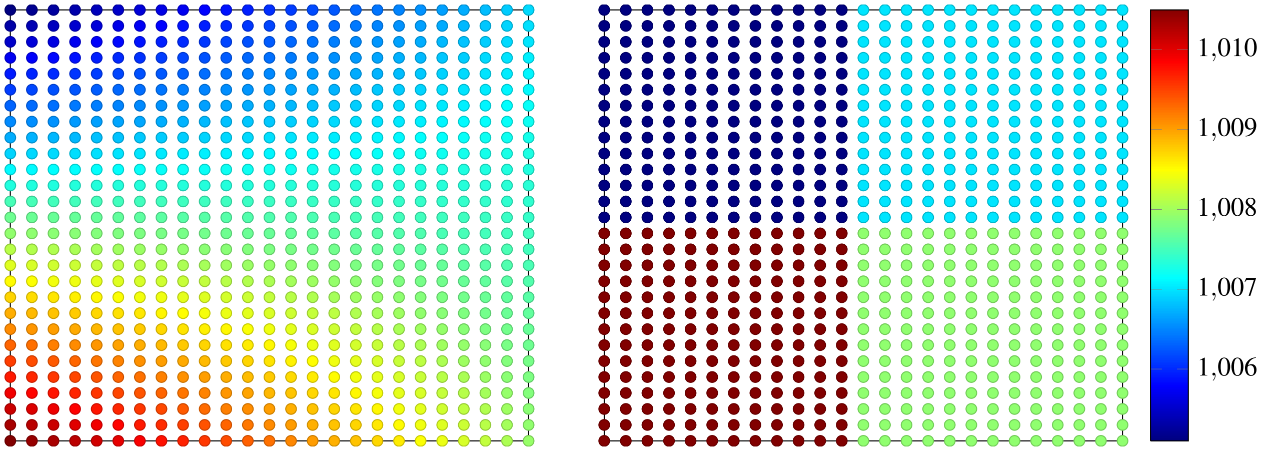


Figure B.1: Interpoloation

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