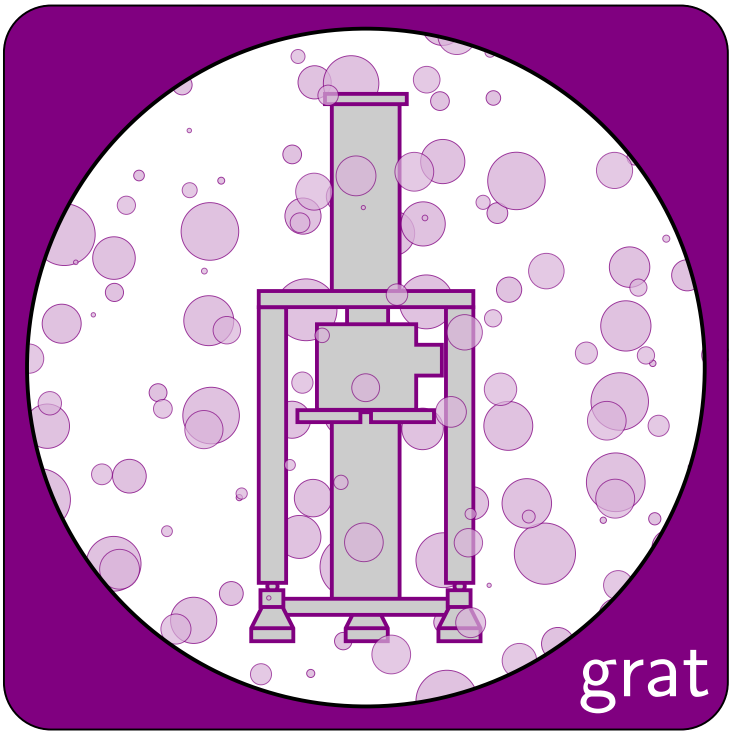

grat

Marcin Rajner
Politechnika Warszawska | Warsaw University of Technology

Contents

1 grat overview 1

1.1 Purpose . . . 1

2 License 3

2.1 Usage . . . 10

3 External resources 11

4 polygon_check 13

5 Todo List 15

6 Data Type Index 17

6.1 Data Types List . . . 17

7 File Index 19

7.1 File List . . . 19

8 Data Type Documentation 21

8.1 mod_cmdline::admitance_info Type Reference . . . 21

8.1.1 Detailed Description . . . 21

8.2 mod_constants::atmosphere_data Type Reference . . . 21

8.2.1 Detailed Description . . . 22

8.3 mod_constants::celestial_object_data Type Reference . . . 22

8.3.1 Detailed Description . . . 22

8.4 mod_cmdline::cmd_line_arg Type Reference . . . 22

8.4.1 Detailed Description . . . 23

8.5 mod_date::dateandmjd Type Reference . . . 23

8.5.1 Detailed Description . . . 23

8.6 mod_constants::density_info Type Reference . . . 23

8.6.1 Detailed Description . . . 23

8.7 mod_constants::earth_data Type Reference . . . 24

8.7.1 Detailed Description . . . 24

8.8 mod_constants::earth_density Type Reference . . . 24

8.8.1 Detailed Description 24
8.9 mod_constants::earth_gravity Type Reference 25
8.9.1 Detailed Description 25
8.10 mod_cmdline::field_info Type Reference 25
8.10.1 Detailed Description 25
8.11 mod_data::file Type Reference 25
8.11.1 Detailed Description 26
8.12 mod_constants::gravity_data Type Reference 26
8.12.1 Detailed Description 26
8.13 mod_green::green_common_info Type Reference 26
8.13.1 Detailed Description 27
8.14 mod_green::green_functions Type Reference 27
8.14.1 Detailed Description 27
8.15 mod_cmdline::green_index Type Reference 27
8.15.1 Detailed Description 27
8.16 mod_cmdline::index_info Type Reference 28
8.16.1 Detailed Description 28
8.17 mod_cmdline::info_info Type Reference 29
8.17.1 Detailed Description 29
8.18 mod_data::level_info Type Reference 29
8.18.1 Detailed Description 29
8.19 mod_site::lp_info Type Reference 30
8.19.1 Detailed Description 30
8.20 mod_3d Module Reference 30
8.20.1 Detailed Description 30
8.21 mod_admit Module Reference 30
8.21.1 Detailed Description 30
8.21.2 Member Function/Subroutine Documentation 31
8.21.2.1 parse_admit 31
8.22 mod_aggf Module Reference 31
8.22.1 Detailed Description 31
8.22.2 Member Function/Subroutine Documentation 31
8.22.2.1 aggf 31
8.22.2.2 aggfd 32
8.22.2.3 bouger 32
8.22.2.4 gn_thin_layer 32
8.22.2.5 simple_def 33
8.23 mod_atmosphere Module Reference 33
8.23.1 Detailed Description 33
8.23.2 Member Function/Subroutine Documentation 33

8.23.2.1 geop2geom 33
8.23.2.2 standard_density 34
8.23.2.3 standard_gravity 34
8.23.2.4 standard_pressure 34
8.23.2.5 standard_temperature 34
8.24 mod_cmdline Module Reference 35
8.24.1 Detailed Description 36
8.24.2 Member Function/Subroutine Documentation 36
8.24.2.1 collect_args 36
8.24.2.2 get_command_cleaned 36
8.25 mod_constants Module Reference 37
8.25.1 Detailed Description 38
8.26 mod_data Module Reference 38
8.26.1 Detailed Description 39
8.26.2 Member Function/Subroutine Documentation 40
8.26.2.1 bilinear 40
8.26.2.2 check 40
8.26.2.3 get_dimension 40
8.26.2.4 get_scale_and_offset 40
8.26.2.5 get_value 41
8.26.2.6 nctime2date 41
8.26.2.7 parse_model 41
8.27 mod_date Module Reference 42
8.27.1 Detailed Description 42
8.27.2 Member Function/Subroutine Documentation 42
8.27.2.1 parse_date 42
8.27.2.2 string2date 43
8.28 mod_green Module Reference 43
8.28.1 Detailed Description 44
8.28.2 Member Function/Subroutine Documentation 44
8.28.2.1 convolve 44
8.28.2.2 parse_green 44
8.28.2.3 printmoreverbose 44
8.29 mod_mjd Module Reference 45
8.29.1 Detailed Description 45
8.29.2 Member Function/Subroutine Documentation 45
8.29.2.1 invmjd 45
8.29.2.2 jd 45
8.29.2.3 mjd 46
8.30 mod_normalization Module Reference 46

8.30.1 Detailed Description 46
8.31 mod_parser Module Reference 46
8.31.1 Detailed Description 47
8.31.2 Member Function/Subroutine Documentation 47
8.31.2.1 dataname 47
8.31.2.2 intro 47
8.31.2.3 parse_info 48
8.31.2.4 parse_moreverbose 48
8.31.2.5 print_version 48
8.32 mod_polygon Module Reference 49
8.32.1 Detailed Description 49
8.32.2 Member Function/Subroutine Documentation 50
8.32.2.1 chkgon 50
8.32.2.2 ncross 50
8.32.2.3 parse_polygon 51
8.32.2.4 read_polygon 51
8.33 mod_printing Module Reference 51
8.33.1 Detailed Description 52
8.34 mod_site Module Reference 52
8.34.1 Detailed Description 53
8.34.2 Member Function/Subroutine Documentation 53
8.34.2.1 read_site_file 53
8.35 mod_spherical Module Reference 53
8.35.1 Detailed Description 53
8.35.2 Member Function/Subroutine Documentation 54
8.35.2.1 spher_area 54
8.35.2.2 spher_trig 54
8.35.2.3 spher_trig_inverse 55
8.36 mod_utilities Module Reference 55
8.36.1 Detailed Description 56
8.36.2 Member Function/Subroutine Documentation 56
8.36.2.1 count_records_to_read 56
8.36.2.2 d2r 56
8.36.2.3 file_exists 57
8.36.2.4 is_numeric 57
8.36.2.5 ispline 57
8.36.2.6 ntokens 58
8.36.2.7 r2d 58
8.36.2.8 size_ntimes_denser 58
8.36.2.9 spline 58

8.36.2.10 spline_interpolation 59
8.37 mod_cmdline::model_index Type Reference 59
8.37.1 Detailed Description 59
8.38 mod_site::more_site_heights Type Reference 59
8.38.1 Detailed Description 59
8.39 mod_cmdline::moreverbose_index Type Reference 60
8.39.1 Detailed Description 60
8.40 mod_cmdline::moreverbose_info Type Reference 60
8.40.1 Detailed Description 60
8.41 mod_printing::output_info Type Reference 60
8.41.1 Detailed Description 61
8.42 mod_cmdline::poly_index Type Reference 61
8.42.1 Detailed Description 61
8.43 mod_polygon::polygon_data Type Reference 61
8.43.1 Detailed Description 61
8.44 mod_polygon::polygon_info Type Reference 62
8.44.1 Detailed Description 62
8.45 mod_constants::pressure_data Type Reference 62
8.45.1 Detailed Description 62
8.46 mod_printing::printing_info Type Reference 63
8.46.1 Detailed Description 63
8.47 mod_cmdline::range Type Reference 63
8.47.1 Detailed Description 63
8.48 mod_site::site_info Type Reference 64
8.48.1 Detailed Description 64
8.49 mod_cmdline::subfield_info Type Reference 64
8.49.1 Detailed Description 64
8.50 mod_constants::temperature_data Type Reference 65
8.50.1 Detailed Description 65
8.51 mod_cmdline::transfer_sp_info Type Reference 65
8.51.1 Detailed Description 65
8.52 mod_cmdline::warnings_info Type Reference 65
8.52.1 Detailed Description 65

9 File Documentation 67

9.1 grat/doc/figures/interpolation_ilustration.sh File Reference 67
9.1.1 Detailed Description 67
9.2 interpolation_ilustration.sh 67
9.3 grat/src/grat.f90 File Reference 67
9.3.1 Detailed Description 67

9.4 grat.f90 68

9.5 grat/src/mod_admit.f90 File Reference 71

9.5.1 Detailed Description 71

9.6 mod_admit.f90 71

9.7 grat/src/mod_aggf.f90 File Reference 73

9.7.1 Detailed Description 74

9.8 mod_aggf.f90 74

9.9 grat/src/mod_cmdline.f90 File Reference 77

9.9.1 Detailed Description 78

9.10 mod_cmdline.f90 78

9.11 grat/src/mod_green.f90 File Reference 81

9.11.1 Detailed Description 81

9.12 mod_green.f90 81

9.13 grat/src/mod_normalization.f90 File Reference 99

9.13.1 Detailed Description 99

9.14 mod_normalization.f90 99

9.15 grat/src/mod_polygon.f90 File Reference 99

9.15.1 Detailed Description 99

9.16 mod_polygon.f90 100

9.17 grat/src/real_vs_standard.f90 File Reference 104

9.17.1 Detailed Description 104

9.18 real_vs_standard.f90 104

9.19 grat/src/value_check.f90 File Reference 105

9.19.1 Detailed Description 105

9.20 value_check.f90 105

10 Example Documentation **109**

10.1 example_aggf.f90 109

10.2 grat_usage.sh 116

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

pre-alpha

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 unnecesarry 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 **?**

Copyright

Chapter 2

License

```
                    GNU GENERAL PUBLIC LICENSE
                       Version 3, 29 June 2007

 Copyright (C) 2007 Free Software Foundation, Inc. <http://fsf.org/>
 Everyone is permitted to copy and distribute verbatim copies
 of this license document, but changing it is not allowed.

                            Preamble

  The GNU General Public License is a free, copyleft license for
software and other kinds of works.

  The licenses for most software and other practical works are designed
to take away your freedom to share and change the works.  By contrast,
the GNU General Public License is intended to guarantee your freedom to
share and change all versions of a program--to make sure it remains free
software for all its users.  We, the Free Software Foundation, use the
GNU General Public License for most of our software; it applies also to
any other work released this way by its authors.  You can apply it to
your programs, too.

  When we speak of free software, we are referring to freedom, not
price.  Our General Public Licenses are designed to make sure that you
have the freedom to distribute copies of free software (and charge for
them if you wish), that you receive source code or can get it if you
want it, that you can change the software or use pieces of it in new
free programs, and that you know you can do these things.

  To protect your rights, we need to prevent others from denying you
these rights or asking you to surrender the rights.  Therefore, you have
certain responsibilities if you distribute copies of the software, or if
you modify it: responsibilities to respect the freedom of others.

  For example, if you distribute copies of such a program, whether
gratis or for a fee, you must pass on to the recipients the same
freedoms that you received.  You must make sure that they, too, receive
or can get the source code.  And you must show them these terms so they
know their rights.

  Developers that use the GNU GPL protect your rights with two steps:
(1) assert copyright on the software, and (2) offer you this License
giving you legal permission to copy, distribute and/or modify it.

  For the developers' and authors' protection, the GPL clearly explains
that there is no warranty for this free software.  For both users' and
authors' sake, the GPL requires that modified versions be marked as
changed, so that their problems will not be attributed erroneously to
authors of previous versions.

  Some devices are designed to deny users access to install or run
modified versions of the software inside them, although the manufacturer
can do so.  This is fundamentally incompatible with the aim of
protecting users' freedom to change the software.  The systematic
pattern of such abuse occurs in the area of products for individuals to
use, which is precisely where it is most unacceptable.  Therefore, we
have designed this version of the GPL to prohibit the practice for those
products.  If such problems arise substantially in other domains, we
stand ready to extend this provision to those domains in future versions
of the GPL, as needed to protect the freedom of users.

  Finally, every program is threatened constantly by software patents.
States should not allow patents to restrict development and use of
software on general-purpose computers, but in those that do, we wish to
```

avoid the special danger that patents applied to a free program could make it effectively proprietary. To prevent this, the GPL assures that patents cannot be used to render the program non-free.

The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS

0. Definitions.

"This License" refers to version 3 of the GNU General Public License.

"Copyright" also means copyright-like laws that apply to other kinds of works, such as semiconductor masks.

"The Program" refers to any copyrightable work licensed under this License. Each licensee is addressed as "you". "Licensees" and "recipients" may be individuals or organizations.

To "modify" a work means to copy from or adapt all or part of the work in a fashion requiring copyright permission, other than the making of an exact copy. The resulting work is called a "modified version" of the earlier work or a work "based on" the earlier work.

A "covered work" means either the unmodified Program or a work based on the Program.

To "propagate" a work means to do anything with it that, without permission, would make you directly or secondarily liable for infringement under applicable copyright law, except executing it on a computer or modifying a private copy. Propagation includes copying, distribution (with or without modification), making available to the public, and in some countries other activities as well.

To "convey" a work means any kind of propagation that enables other parties to make or receive copies. Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying.

An interactive user interface displays "Appropriate Legal Notices" to the extent that it includes a convenient and prominently visible feature that (1) displays an appropriate copyright notice, and (2) tells the user that there is no warranty for the work (except to the extent that warranties are provided), that licensees may convey the work under this License, and how to view a copy of this License. If the interface presents a list of user commands or options, such as a menu, a prominent item in the list meets this criterion.

1. Source Code.

The "source code" for a work means the preferred form of the work for making modifications to it. "Object code" means any non-source form of a work.

A "Standard Interface" means an interface that either is an official standard defined by a recognized standards body, or, in the case of interfaces specified for a particular programming language, one that is widely used among developers working in that language.

The "System Libraries" of an executable work include anything, other than the work as a whole, that (a) is included in the normal form of packaging a Major Component, but which is not part of that Major Component, and (b) serves only to enable use of the work with that Major Component, or to implement a Standard Interface for which an implementation is available to the public in source code form. A "Major Component", in this context, means a major essential component (kernel, window system, and so on) of the specific operating system (if any) on which the executable work runs, or a compiler used to produce the work, or an object code interpreter used to run it.

The "Corresponding Source" for a work in object code form means all the source code needed to generate, install, and (for an executable work) run the object code and to modify the work, including scripts to control those activities. However, it does not include the work's System Libraries, or general-purpose tools or generally available free programs which are used unmodified in performing those activities but which are not part of the work. For example, Corresponding Source includes interface definition files associated with source files for the work, and the source code for shared libraries and dynamically linked subprograms that the work is specifically designed to require, such as by intimate data communication or control flow between those subprograms and other parts of the work.

The Corresponding Source need not include anything that users can regenerate automatically from other parts of the Corresponding Source.

```
  The Corresponding Source for a work in source code form is that
same work.

  2. Basic Permissions.

  All rights granted under this License are granted for the term of
copyright on the Program, and are irrevocable provided the stated
conditions are met.  This License explicitly affirms your unlimited
permission to run the unmodified Program.  The output from running a
covered work is covered by this License only if the output, given its
content, constitutes a covered work.  This License acknowledges your
rights of fair use or other equivalent, as provided by copyright law.

  You may make, run and propagate covered works that you do not
convey, without conditions so long as your license otherwise remains
in force.  You may convey covered works to others for the sole purpose
of having them make modifications exclusively for you, or provide you
with facilities for running those works, provided that you comply with
the terms of this License in conveying all material for which you do
not control copyright.  Those thus making or running the covered works
for you must do so exclusively on your behalf, under your direction
and control, on terms that prohibit them from making any copies of
your copyrighted material outside their relationship with you.

  Conveying under any other circumstances is permitted solely under
the conditions stated below.  Sublicensing is not allowed; section 10
makes it unnecessary.

  3. Protecting Users' Legal Rights From Anti-Circumvention Law.

  No covered work shall be deemed part of an effective technological
measure under any applicable law fulfilling obligations under article
11 of the WIPO copyright treaty adopted on 20 December 1996, or
similar laws prohibiting or restricting circumvention of such
measures.

  When you convey a covered work, you waive any legal power to forbid
circumvention of technological measures to the extent such circumvention
is effected by exercising rights under this License with respect to
the covered work, and you disclaim any intention to limit operation or
modification of the work as a means of enforcing, against the work's
users, your or third parties' legal rights to forbid circumvention of
technological measures.

  4. Conveying Verbatim Copies.

  You may convey verbatim copies of the Program's source code as you
receive it, in any medium, provided that you conspicuously and
appropriately publish on each copy an appropriate copyright notice;
keep intact all notices stating that this License and any
non-permissive terms added in accord with section 7 apply to the code;
keep intact all notices of the absence of any warranty; and give all
recipients a copy of this License along with the Program.

  You may charge any price or no price for each copy that you convey,
and you may offer support or warranty protection for a fee.

  5. Conveying Modified Source Versions.

  You may convey a work based on the Program, or the modifications to
produce it from the Program, in the form of source code under the
terms of section 4, provided that you also meet all of these conditions:

    a) The work must carry prominent notices stating that you modified
    it, and giving a relevant date.

    b) The work must carry prominent notices stating that it is
    released under this License and any conditions added under section
    7.  This requirement modifies the requirement in section 4 to
    "keep intact all notices".

    c) You must license the entire work, as a whole, under this
    License to anyone who comes into possession of a copy.  This
    License will therefore apply, along with any applicable section 7
    additional terms, to the whole of the work, and all its parts,
    regardless of how they are packaged.  This License gives no
    permission to license the work in any other way, but it does not
    invalidate such permission if you have separately received it.

    d) If the work has interactive user interfaces, each must display
    Appropriate Legal Notices; however, if the Program has interactive
    interfaces that do not display Appropriate Legal Notices, your
    work need not make them do so.

  A compilation of a covered work with other separate and independent
works, which are not by their nature extensions of the covered work,
and which are not combined with it such as to form a larger program,
in or on a volume of a storage or distribution medium, is called an
"aggregate" if the compilation and its resulting copyright are not
used to limit the access or legal rights of the compilation's users
beyond what the individual works permit.  Inclusion of a covered work
in an aggregate does not cause this License to apply to the other
parts of the aggregate.

  6. Conveying Non-Source Forms.

  You may convey a covered work in object code form under the terms
of sections 4 and 5, provided that you also convey the
machine-readable Corresponding Source under the terms of this License,
in one of these ways:

    a) Convey the object code in, or embodied in, a physical product
    (including a physical distribution medium), accompanied by the
    Corresponding Source fixed on a durable physical medium
    customarily used for software interchange.

    b) Convey the object code in, or embodied in, a physical product
    (including a physical distribution medium), accompanied by a
    written offer, valid for at least three years and valid for as
    long as you offer spare parts or customer support for that product
    model, to give anyone who possesses the object code either (1) a
    copy of the Corresponding Source for all the software in the
    product that is covered by this License, on a durable physical
    medium customarily used for software interchange, for a price no
    more than your reasonable cost of physically performing this
    conveying of source, or (2) access to copy the
    Corresponding Source from a network server at no charge.

    c) Convey individual copies of the object code with a copy of the
    written offer to provide the Corresponding Source.  This
    alternative is allowed only occasionally and noncommercially, and
    only if you received the object code with such an offer, in accord
    with subsection 6b.

    d) Convey the object code by offering access from a designated
    place (gratis or for a charge), and offer equivalent access to the
    Corresponding Source in the same way through the same place at no
    further charge.  You need not require recipients to copy the
    Corresponding Source along with the object code.  If the place to
    copy the object code is a network server, the Corresponding Source
    may be on a different server (operated by you or a third party)
    that supports equivalent copying facilities, provided you maintain
    clear directions next to the object code saying where to find the
    Corresponding Source.  Regardless of what server hosts the
    Corresponding Source, you remain obligated to ensure that it is
    available for as long as needed to satisfy these requirements.

    e) Convey the object code using peer-to-peer transmission, provided
    you inform other peers where the object code and Corresponding
    Source of the work are being offered to the general public at no
    charge under subsection 6d.

  A separable portion of the object code, whose source code is excluded
from the Corresponding Source as a System Library, need not be
included in conveying the object code work.

  A "User Product" is either (1) a "consumer product", which means any
tangible personal property which is normally used for personal, family,
or household purposes, or (2) anything designed or sold for incorporation
into a dwelling.  In determining whether a product is a consumer product,
doubtful cases shall be resolved in favor of coverage.  For a particular
product received by a particular user, "normally used" refers to a
typical or common use of that class of product, regardless of the status
of the particular user or of the way in which the particular user
actually uses, or expects or is expected to use, the product.  A product
is a consumer product regardless of whether the product has substantial
commercial, industrial or non-consumer uses, unless such uses represent
the only significant mode of use of the product.

  "Installation Information" for a User Product means any methods,
procedures, authorization keys, or other information required to install
and execute modified versions of a covered work in that User Product from
a modified version of its Corresponding Source.  The information must
suffice to ensure that the continued functioning of the modified object
code is in no case prevented or interfered with solely because
modification has been made.

  If you convey an object code work under this section in, or with, or
specifically for use in, a User Product, and the conveying occurs as
part of a transaction in which the right of possession and use of the
User Product is transferred to the recipient in perpetuity or for a
fixed term (regardless of how the transaction is characterized), the
Corresponding Source conveyed under this section must be accompanied
by the Installation Information.  But this requirement does not apply
```

if neither you nor any third party retains the ability to install modified object code on the User Product (for example, the work has been installed in ROM).

The requirement to provide Installation Information does not include a requirement to continue to provide support service, warranty, or updates for a work that has been modified or installed by the recipient, or for the User Product in which it has been modified or installed. Access to a network may be denied when the modification itself materially and adversely affects the operation of the network or violates the rules and protocols for communication across the network.

Corresponding Source conveyed, and Installation Information provided, in accord with this section must be in a format that is publicly documented (and with an implementation available to the public in source code form), and must require no special password or key for unpacking, reading or copying.

7. Additional Terms.

"Additional permissions" are terms that supplement the terms of this License by making exceptions from one or more of its conditions. Additional permissions that are applicable to the entire Program shall be treated as though they were included in this License, to the extent that they are valid under applicable law. If additional permissions apply only to part of the Program, that part may be used separately under those permissions, but the entire Program remains governed by this License without regard to the additional permissions.

When you convey a copy of a covered work, you may at your option remove any additional permissions from that copy, or from any part of it. (Additional permissions may be written to require their own removal in certain cases when you modify the work.) You may place additional permissions on material, added by you to a covered work, for which you have or can give appropriate copyright permission.

Notwithstanding any other provision of this License, for material you add to a covered work, you may (if authorized by the copyright holders of that material) supplement the terms of this License with terms:

a) Disclaiming warranty or limiting liability differently from the terms of sections 15 and 16 of this License; or

b) Requiring preservation of specified reasonable legal notices or author attributions in that material or in the Appropriate Legal Notices displayed by works containing it; or

c) Prohibiting misrepresentation of the origin of that material, or requiring that modified versions of such material be marked in reasonable ways as different from the original version; or

d) Limiting the use for publicity purposes of names of licensors or authors of the material; or

e) Declining to grant rights under trademark law for use of some trade names, trademarks, or service marks; or

f) Requiring indemnification of licensors and authors of that material by anyone who conveys the material (or modified versions of it) with contractual assumptions of liability to the recipient, for any liability that these contractual assumptions directly impose on those licensors and authors.

All other non-permissive additional terms are considered "further restrictions" within the meaning of section 10. If the Program as you received it, or any part of it, contains a notice stating that it is governed by this License along with a term that is a further restriction, you may remove that term. If a license document contains a further restriction but permits relicensing or conveying under this License, you may add to a covered work material governed by the terms of that license document, provided that the further restriction does not survive such relicensing or conveying.

If you add terms to a covered work in accord with this section, you must place, in the relevant source files, a statement of the additional terms that apply to those files, or a notice indicating where to find the applicable terms.

Additional terms, permissive or non-permissive, may be stated in the form of a separately written license, or stated as exceptions; the above requirements apply either way.

8. Termination.

You may not propagate or modify a covered work except as expressly provided under this License. Any attempt otherwise to propagate or modify it is void, and will automatically terminate your rights under

```
this License (including any patent licenses granted under the third
paragraph of section 11).

  However, if you cease all violation of this License, then your
license from a particular copyright holder is reinstated (a)
provisionally, unless and until the copyright holder explicitly and
finally terminates your license, and (b) permanently, if the copyright
holder fails to notify you of the violation by some reasonable means
prior to 60 days after the cessation.

  Moreover, your license from a particular copyright holder is
reinstated permanently if the copyright holder notifies you of the
violation by some reasonable means, this is the first time you have
received notice of violation of this License (for any work) from that
copyright holder, and you cure the violation prior to 30 days after
your receipt of the notice.

  Termination of your rights under this section does not terminate the
licenses of parties who have received copies or rights from you under
this License.  If your rights have been terminated and not permanently
reinstated, you do not qualify to receive new licenses for the same
material under section 10.

  9. Acceptance Not Required for Having Copies.

  You are not required to accept this License in order to receive or
run a copy of the Program.  Ancillary propagation of a covered work
occurring solely as a consequence of using peer-to-peer transmission
to receive a copy likewise does not require acceptance.  However,
nothing other than this License grants you permission to propagate or
modify any covered work.  These actions infringe copyright if you do
not accept this License.  Therefore, by modifying or propagating a
covered work, you indicate your acceptance of this License to do so.

  10. Automatic Licensing of Downstream Recipients.

  Each time you convey a covered work, the recipient automatically
receives a license from the original licensors, to run, modify and
propagate that work, subject to this License.  You are not responsible
for enforcing compliance by third parties with this License.

  An "entity transaction" is a transaction transferring control of an
organization, or substantially all assets of one, or subdividing an
organization, or merging organizations.  If propagation of a covered
work results from an entity transaction, each party to that
transaction who receives a copy of the work also receives whatever
licenses to the work the party's predecessor in interest had or could
give under the previous paragraph, plus a right to possession of the
Corresponding Source of the work from the predecessor in interest, if
the predecessor has it or can get it with reasonable efforts.

  You may not impose any further restrictions on the exercise of the
rights granted or affirmed under this License.  For example, you may
not impose a license fee, royalty, or other charge for exercise of
rights granted under this License, and you may not initiate litigation
(including a cross-claim or counterclaim in a lawsuit) alleging that
any patent claim is infringed by making, using, selling, offering for
sale, or importing the Program or any portion of it.

  11. Patents.

  A "contributor" is a copyright holder who authorizes use under this
License of the Program or a work on which the Program is based.  The
work thus licensed is called the contributor's "contributor version".

  A contributor's "essential patent claims" are all patent claims
owned or controlled by the contributor, whether already acquired or
hereafter acquired, that would be infringed by some manner, permitted
by this License, of making, using, or selling its contributor version,
but do not include claims that would be infringed only as a
consequence of further modification of the contributor version.  For
purposes of this definition, "control" includes the right to grant
patent sublicenses in a manner consistent with the requirements of
this License.

  Each contributor grants you a non-exclusive, worldwide, royalty-free
patent license under the contributor's essential patent claims, to
make, use, sell, offer for sale, import and otherwise run, modify and
propagate the contents of its contributor version.

  In the following three paragraphs, a "patent license" is any express
agreement or commitment, however denominated, not to enforce a patent
(such as an express permission to practice a patent or covenant not to
sue for patent infringement).  To "grant" such a patent license to a
party means to make such an agreement or commitment not to enforce a
patent against the party.
```

```
  If you convey a covered work, knowingly relying on a patent license,
and the Corresponding Source of the work is not available for anyone
to copy, free of charge and under the terms of this License, through a
publicly available network server or other readily accessible means,
then you must either (1) cause the Corresponding Source to be so
available, or (2) arrange to deprive yourself of the benefit of the
patent license for this particular work, or (3) arrange, in a manner
consistent with the requirements of this License, to extend the patent
license to downstream recipients.  "Knowingly relying" means you have
actual knowledge that, but for the patent license, your conveying the
covered work in a country, or your recipient's use of the covered work
in a country, would infringe one or more identifiable patents in that
country that you have reason to believe are valid.

  If, pursuant to or in connection with a single transaction or
arrangement, you convey, or propagate by procuring conveyance of, a
covered work, and grant a patent license to some of the parties
receiving the covered work authorizing them to use, propagate, modify
or convey a specific copy of the covered work, then the patent license
you grant is automatically extended to all recipients of the covered
work and works based on it.

  A patent license is "discriminatory" if it does not include within
the scope of its coverage, prohibits the exercise of, or is
conditioned on the non-exercise of one or more of the rights that are
specifically granted under this License.  You may not convey a covered
work if you are a party to an arrangement with a third party that is
in the business of distributing software, under which you make payment
to the third party based on the extent of your activity of conveying
the work, and under which the third party grants, to any of the
parties who would receive the covered work from you, a discriminatory
patent license (a) in connection with copies of the covered work
conveyed by you (or copies made from those copies), or (b) primarily
for and in connection with specific products or compilations that
contain the covered work, unless you entered into that arrangement,
or that patent license was granted, prior to 28 March 2007.

  Nothing in this License shall be construed as excluding or limiting
any implied license or other defenses to infringement that may
otherwise be available to you under applicable patent law.

  12. No Surrender of Others' Freedom.

  If conditions are imposed on you (whether by court order, agreement or
otherwise) that contradict the conditions of this License, they do not
excuse you from the conditions of this License.  If you cannot convey a
covered work so as to satisfy simultaneously your obligations under this
License and any other pertinent obligations, then as a consequence you may
not convey it at all.  For example, if you agree to terms that obligate you
to collect a royalty for further conveying from those to whom you convey
the Program, the only way you could satisfy both those terms and this
License would be to refrain entirely from conveying the Program.

  13. Use with the GNU Affero General Public License.

  Notwithstanding any other provision of this License, you have
permission to link or combine any covered work with a work licensed
under version 3 of the GNU Affero General Public License into a single
combined work, and to convey the resulting work.  The terms of this
License will continue to apply to the part which is the covered work,
but the special requirements of the GNU Affero General Public License,
section 13, concerning interaction through a network will apply to the
combination as such.

  14. Revised Versions of this License.

  The Free Software Foundation may publish revised and/or new versions of
the GNU General Public License from time to time.  Such new versions will
be similar in spirit to the present version, but may differ in detail to
address new problems or concerns.

  Each version is given a distinguishing version number.  If the
Program specifies that a certain numbered version of the GNU General
Public License "or any later version" applies to it, you have the
option of following the terms and conditions either of that numbered
version or of any later version published by the Free Software
Foundation.  If the Program does not specify a version number of the
GNU General Public License, you may choose any version ever published
by the Free Software Foundation.

  If the Program specifies that a proxy can decide which future
versions of the GNU General Public License can be used, that proxy's
public statement of acceptance of a version permanently authorizes you
to choose that version for the Program.

  Later license versions may give you additional or different
permissions.  However, no additional obligations are imposed on any
```

```
author or copyright holder as a result of your choosing to follow a
later version.

  15. Disclaimer of Warranty.

  THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY
APPLICABLE LAW.  EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT
HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY
OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO,
THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
PURPOSE.  THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM
IS WITH YOU.  SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF
ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

  16. Limitation of Liability.

  IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING
WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS
THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY
GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE
USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF
DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD
PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS),
EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF
SUCH DAMAGES.

  17. Interpretation of Sections 15 and 16.

  If the disclaimer of warranty and limitation of liability provided
above cannot be given local legal effect according to their terms,
reviewing courts shall apply local law that most closely approximates
an absolute waiver of all civil liability in connection with the
Program, unless a warranty or assumption of liability accompanies a
copy of the Program in return for a fee.
```

2.1 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

Chapter 3

External resources

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

Chapter 4

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 5

Todo List

Subprogram mod_aggf::gn_thin_layer (psi)

explanaition ??

Subprogram mod_mjd::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

Chapter 6

Data Type Index

6.1 Data Types List

Here are the data types with brief descriptions:

mod_cmdline::admitance_info 21
mod_constants::atmosphere_data 21
mod_constants::celestial_object_data 22
mod_cmdline::cmd_line_arg 22
mod_date::dateandmjd 23
mod_constants::density_info 23
mod_constants::earth_data 24
mod_constants::earth_density 24
mod_constants::earth_gravity 25
mod_cmdline::field_info 25
mod_data::file 25
mod_constants::gravity_data 26
mod_green::green_common_info 26
mod_green::green_functions 27
mod_cmdline::green_index 27
mod_cmdline::index_info 28
mod_cmdline::info_info 29
mod_data::level_info 29
mod_site::lp_info 30
mod_3d 30
mod_admit 30
mod_aggf 31
mod_atmosphere 33
mod_cmdline 35
mod_constants
Define constant values 37
mod_data
This modele gives routines to read, and write data 38
mod_date 42
mod_green 43
mod_mjd 45
mod_normalization 46
mod_parser 46
mod_polygon 49
mod_printing 51
mod_site 52
mod_spherical 53
mod_utilities 55

mod_cmdline::model_index 59
mod_site::more_site_heights 59
mod_cmdline::moreverbose_index 60
mod_cmdline::moreverbose_info 60
mod_printing::output_info 60
mod_cmdline::poly_index 61
mod_polygon::polygon_data 61
mod_polygon::polygon_info 62
mod_constants::pressure_data 62
mod_printing::printing_info 63
mod_cmdline::range 63
mod_site::site_info 64
mod_cmdline::subfield_info 64
mod_constants::temperature_data 65
mod_cmdline::transfer_sp_info 65
mod_cmdline::warnings_info 65

Chapter 7

File Index

7.1 File List

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

grat/dat/**grat.hlp** . . . . . ??
grat/dat/**help.hlp** . . . . . ??
grat/dat/**polygon_check.hlp** . . . . . ??
grat/dat/**value_check.hlp** . . . . . ??
grat/doc/**grat.hlp** . . . . . ??
grat/doc/**LICENSE** . . . . . ??
grat/doc/**polygon_check.hlp** . . . . . ??
grat/doc/**value_check.hlp** . . . . . ??
grat/doc/figures/**convolution_scheme.sh** . . . . . ??
grat/doc/figures/interpolation_ilustration.sh 67
grat/doc/figures/**polygon_ilustration.sh** . . . . . ??
grat/examples/**barometric_formula.f90** . . . . . ??
grat/examples/**bug.sh** . . . . . ??
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/grat.f90 68
grat/src/**mod_3d.f90** . . . . . ??
grat/src/mod_admit.f90 71
grat/src/mod_aggf.f90
This module contains utitlities for computing Atmospheric Gravity Green Functions 74
grat/src/**mod_atmosphere.f90** . . . . . ??
grat/src/mod_cmdline.f90
This module gather cmd line arguments 78
grat/src/**mod_constants.f90** . . . . . ??
grat/src/**mod_data.f90** . . . . . ??
grat/src/**mod_date.f90** . . . . . ??
grat/src/mod_green.f90 81
grat/src/**mod_mjd.f90** . . . . . ??
grat/src/mod_normalization.f90 99
grat/src/**mod_parser.f90** . . . . . ??
grat/src/mod_polygon.f90
Some routines to deal with inclusion or exclusion of polygons 100
grat/src/**mod_printing.f90** . . . . . ??
grat/src/**mod_site.f90** . . . . . ??
grat/src/**mod_spherical.f90** . . . . . ??

grat/src/**mod_utilities.f90** . . . . **??**
grat/src/**polygon_check.f90** . . . . **??**
grat/src/value_check.f90 105

Chapter 8

Data Type Documentation

8.1 mod_cmdline::admitance_info Type Reference

Public Attributes

- logical **if**
- real(dp) **value** = -0.3

8.1.1 Detailed Description

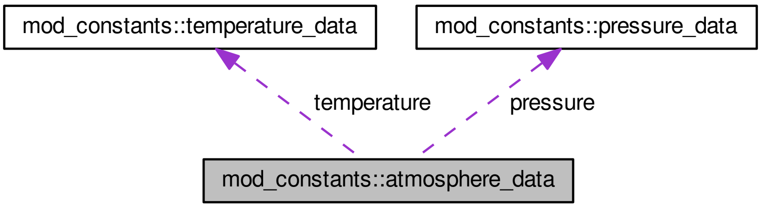
Definition at line 123 of file mod_cmdline.f90.

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

- grat/src/mod_cmdline.f90

8.2 mod_constants::atmosphere_data Type Reference

Collaboration diagram for mod_constants::atmosphere_data:


mod_constants::temperature_data
mod_constants::pressure_data
temperature
pressure
mod_constants::atmosphere_data

Public Attributes

- type(pressure_data) **pressure**
- type(temperature_data) **temperature**

8.2.1 Detailed Description

Definition at line 41 of file mod_constants.f90.

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

- grat/src/mod_constants.f90

8.3 mod_constants::celestial_object_data Type Reference

Public Attributes

- real(dp) **mass**
- real(dp) **distance**

8.3.1 Detailed Description

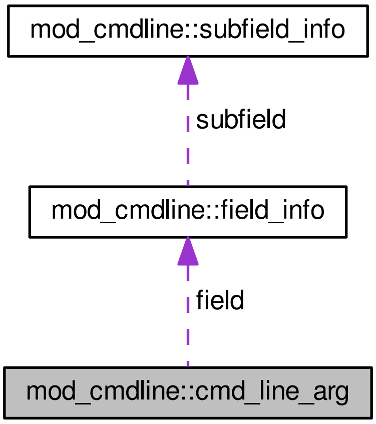
Definition at line 93 of file mod_constants.f90.

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

- grat/src/mod_constants.f90

8.4 mod_cmdline::cmd_line_arg Type Reference

Collaboration diagram for mod_cmdline::cmd_line_arg:



Public Attributes

- character(2) **switch**
- type(field_info), dimension(:), allocatable **field**
- character(len=455) **full**

8.4.1 Detailed Description

Definition at line 24 of file mod_cmdline.f90.

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

- grat/src/mod_cmdline.f90

8.5 mod_date::dateandmjd Type Reference

Public Attributes

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

8.5.1 Detailed Description

Definition at line 10 of file mod_date.f90.

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

- grat/src/mod_date.f90

8.6 mod_constants::density_info Type Reference

Public Attributes

- real(dp) **water**

8.6.1 Detailed Description

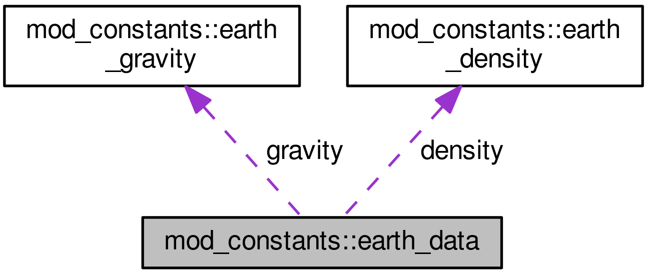
Definition at line 110 of file mod_constants.f90.

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

- grat/src/mod_constants.f90

8.7 mod_constants::earth_data Type Reference

Collaboration diagram for mod_constants::earth_data:



Public Attributes

- real(dp) **mass**
- real(dp) **radius**
- real(dp) **gm**
- type(earth_gravity) **gravity**
- type(earth_density) **density**

8.7.1 Detailed Description

Definition at line 68 of file mod_constants.f90.

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

- grat/src/mod_constants.f90

8.8 mod_constants::earth_density Type Reference

Public Attributes

- real(dp) **crust**
- real(dp) **mean**

8.8.1 Detailed Description

Definition at line 63 of file mod_constants.f90.

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

- grat/src/mod_constants.f90

8.9 mod_constants::earth_gravity Type Reference

Public Attributes

- real(dp) **mean**

8.9.1 Detailed Description

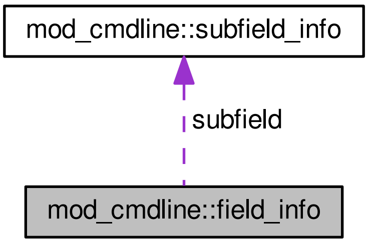
Definition at line 59 of file mod_constants.f90.

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

- grat/src/mod_constants.f90

8.10 mod_cmdline::field_info Type Reference

Collaboration diagram for mod_cmdline::field_info:



Public Attributes

- character(len=355) **full**
- type(subfield_info), dimension(:), allocatable **subfield**

8.10.1 Detailed Description

Definition at line 19 of file mod_cmdline.f90.

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

- grat/src/mod_cmdline.f90

8.11 mod_data::file Type Reference

Public Attributes

- character(90) **name**

- character(len=50), dimension(5) **names** = ["z"
- character(len=100), dimension(5) **datanames** = " "
- character(len=15) **dataname**
- logical **if** = .false.
- real(dp), dimension(:), allocatable **lat**
- real(dp), dimension(:), allocatable **lon**
- real(dp), dimension(:), allocatable **time**
- integer, 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**
- integer **ncid**
- logical **huge** = .false.
- logical **autoload** = .false.
- logical **exist** = .false.
- character(10) **autoloadname**

8.11.1 Detailed Description

Definition at line 17 of file mod_data.f90.

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

- grat/src/mod_data.f90

8.12 mod_constants::gravity_data Type Reference

Public Attributes

- real(dp) **constant**

8.12.1 Detailed Description

Definition at line 22 of file mod_constants.f90.

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

- grat/src/mod_constants.f90

8.13 mod_green::green_common_info Type Reference

Public Attributes

- real(dp), dimension(:), allocatable **distance**
- real(dp), dimension(:), allocatable **start**
- real(dp), dimension(:), allocatable **stop**

- real(dp), dimension(:,:),
  allocatable **data**
- character(len=25), dimension(:),
  allocatable **dataname**
- logical, dimension(:), allocatable **elastic**

8.13.1 Detailed Description

Definition at line 21 of file mod_green.f90.

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

- grat/src/mod_green.f90

8.14 mod_green::green_functions Type Reference

Public Attributes

- character(len=255) **name**
- character(len=25) **dataname**
- integer, dimension(2) **column**
- character(10), dimension(2) **columndataname**
- real(dp), dimension(:), allocatable **distance**
- real(dp), dimension(:), allocatable **data**

8.14.1 Detailed Description

Definition at line 9 of file mod_green.f90.

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

- grat/src/mod_green.f90

8.15 mod_cmdline::green_index Type Reference

Public Attributes

- integer(2) **gn** = 0
- integer(2) **ge** = 0
- integer(2) **gegdt** = 0
- integer(2) **gr** = 0
- integer(2) **ghn** = 0
- integer(2) **ghe** = 0
- integer(2) **gg** = 0

8.15.1 Detailed Description

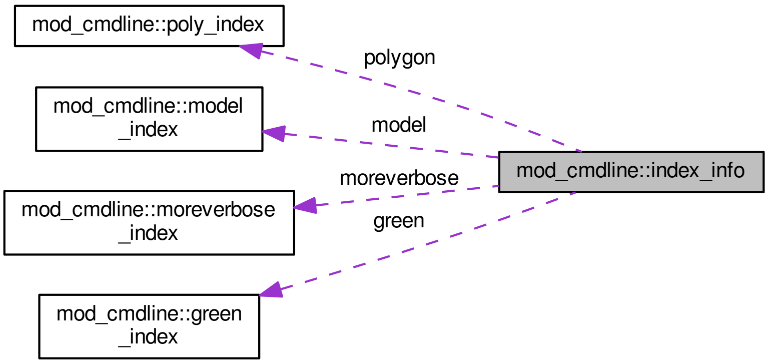
Definition at line 98 of file mod_cmdline.f90.

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

- grat/src/mod_cmdline.f90

8.16 mod_cmdline::index_info Type Reference

Collaboration diagram for mod_cmdline::index_info:



Public Attributes

- type(model_index) **model**
- type(moreverbose_index) **moreverbose**
- type(green_index) **green**
- type(poly_index) **polygon**

8.16.1 Detailed Description

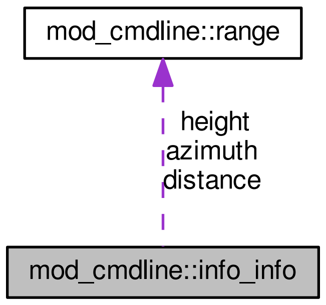
Definition at line 115 of file mod_cmdline.f90.

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

- grat/src/mod_cmdline.f90

8.17 mod_cmdline::info_info Type Reference

Collaboration diagram for mod_cmdline::info_info:



Public Attributes

- type(range) **distance**
- type(range) **azimuth**
- type(range) **height**
- character(1) **interpolation**

8.17.1 Detailed Description

Definition at line 55 of file mod_cmdline.f90.

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

- grat/src/mod_cmdline.f90

8.18 mod_data::level_info Type Reference

Public Attributes

- integer, dimension(:), allocatable **level**
- real(dp), dimension(:), allocatable **height**
- real(dp), dimension(:), allocatable **temperature**
- real(dp), dimension(:), allocatable **humidity**
- logical **all** = .false.

8.18.1 Detailed Description

Definition at line 54 of file mod_data.f90.

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

- grat/src/mod_data.f90

8.19 mod_site::lp_info Type Reference

Public Attributes

- real(dp), dimension(:,:), allocatable **date**
- real(dp), dimension(:), allocatable **data**
- logical **if** = .false.

8.19.1 Detailed Description

Definition at line 15 of file mod_site.f90.

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

- grat/src/mod_site.f90

8.20 mod_3d Module Reference

Public Member Functions

- real(dp) function geometry (psi, h, z, method)

  all values in radians
- real(dp) function potential (psi1, psi2, dazimuth, h, z1, z2)

  all values in radians
- real(dp) function cylinder (psi1, psi2, dazimuth, h, z1, z2)

  all values in radians second improved version of cylinder, includes curvature of the earth
- real(dp) function point_mass_a (theta_s, lambda_s, height_s, theta, lambda, height)

  all values in radians see formula Neumeyer et al., 2004 p. 442-443 this formula is identical as geometry in this module but is uses the

  ### geographical coordinates

8.20.1 Detailed Description

Definition at line 1 of file mod_3d.f90.

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

- grat/src/mod_3d.f90

8.21 mod_admit Module Reference

Public Member Functions

- real(dp) function **admit** (site_, date)
- subroutine parse_admit (cmd_line_entry)

8.21.1 Detailed Description

Definition at line 2 of file mod_admit.f90.

8.21.2 Member Function/Subroutine Documentation

8.21.2.1 subroutine mod_admit::parse_admit (type (cmd_line_arg) *cmd_line_entry*)

Date

2013.10.15

Author

Marcin Rajner

Definition at line 139 of file mod_admit.f90.

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

- grat/src/mod_admit.f90

8.22 mod_aggf Module Reference

Public Member Functions

- real(dp) function aggfd (psi, delta, dz, method, aggfdh, aggfdz, aggfdt, predefined, fels_type, rough)

  Compute first derivative of AGGF with respect to temperature for specific angular distance (psi)
- real(dp) function aggf (psi, zmin, zmax, dz, t_zero, h, first_derivative_h, first_derivative_z, fels_type, method, predefined, rough)

  This function computes the value of atmospheric gravity green functions (AGGF) on the basis of spherical distance (psi)
- real(dp) function gn_thin_layer (psi)

  Compute AGGF GN for thin layer.
- real(dp) function bouger (h, R)

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

  Bouger plate computation.

8.22.1 Detailed Description

Definition at line 9 of file mod_aggf.f90.

8.22.2 Member Function/Subroutine Documentation

8.22.2.1 real(dp) function mod_aggf::aggf (real(dp), intent(in) *psi,* real(dp), intent(in), optional *zmin,* real(dp), intent(in), optional *zmax,* real(dp), intent(in), optional *dz,* 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,* character (len=∗), intent(in), optional *method,* logical, intent(in), optional *predefined,* logical, intent(in), optional *rough*)

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

Author

Marcin Rajner

Date

2013.07.15

Warning

psi in radians h in meter t_zero is actually delta_t so if t_zero=10 (t_zero=288.15+10)

Definition at line 111 of file mod_aggf.f90.

8.22.2.2 real(dp) function mod_aggf::aggfd (real(dp), intent(in) *psi,* real(dp), intent(in), optional *delta,* real(dp), intent(in), optional *dz,* character (len=∗), intent(in), optional *method,* logical, intent(in), optional *aggfdh,* logical, intent(in), optional *aggfdz,* logical, intent(in), optional *aggfdt,* logical, intent(in), optional *predefined,* character (len=∗), intent(in), optional *fels_type,* logical, intent(in), optional *rough*)

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

optional argument define (-dt;-dt) range See equation 19 in **?**

Author

M. Rajner

Date

2013-03-19

Warning

psi in radians

Definition at line 24 of file mod_aggf.f90.

8.22.2.3 real(dp) function mod_aggf::bouger (real(dp), intent(in) *h,* real(dp), intent(in), optional *R*)

Bouger plate computation.

Parameters

in	*r*	height of point above the cylinder

Definition at line 274 of file mod_aggf.f90.

8.22.2.4 real(dp) function 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 **?**

Author

M. Rajner

Date

2013-03-19

Warning

psi in radian

Todo explanaition ??

Definition at line 261 of file mod_aggf.f90.

8.22.2.5 real(dp) function mod_aggf::simple_def (real(dp) *R*)

Bouger plate computation.

see eq. page 288 **?**

Date

2013-03-18

Author

M. Rajner

Definition at line 295 of file mod_aggf.f90.

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

- grat/src/mod_aggf.f90

8.23 mod_atmosphere Module Reference

Public Member Functions

- real(dp) function standard_density (height, temperature, fels_type, method)

  Compute air density for given altitude for standard atmosphere.

- real(dp) function standard_gravity (height)

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

- real(dp) function standard_pressure (height, p_zero, temperature, h_zero, method, dz, fels_type, use_standard_temperature, nan_as_zero)

  Computes pressure [Pa] for specific height.

- real(dp) function standard_temperature (height, fels_type, t_zero)

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

- real(dp) function geop2geom (geopotential_height, inverse)

  Compute geometric height from geopotential heights.

- real(dp) function virtual_temperature (t, sh)

  Compute virtual temperature using temperature and specific humidity.

8.23.1 Detailed Description

Definition at line 1 of file mod_atmosphere.f90.

8.23.2 Member Function/Subroutine Documentation

8.23.2.1 real(dp) function mod_atmosphere::geop2geom (real (dp) *geopotential_height,* logical, intent(in), optional *inverse*)

Compute geometric height from geopotential heights.

Author

M. Rajner

Date

2013-03-19

Definition at line 244 of file mod_atmosphere.f90.

8.23.2.2 real(dp) function mod_atmosphere::standard_density (real(dp), intent(in) *height,* real(dp), intent(in), optional *temperature,* character(len=22), optional *fels_type,* character(len=∗), optional *method*)

Compute air density for given altitude for standard atmosphere.

using formulae 12 in **?**

Date

2013-03-18

Author

M. Rajner height in meter

Definition at line 13 of file mod_atmosphere.f90.

8.23.2.3 real(dp) function mod_atmosphere::standard_gravity (real(dp), intent(in) *height*)

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

see **?** height in meters

Definition at line 38 of file mod_atmosphere.f90.

8.23.2.4 real(dp) function mod_atmosphere::standard_pressure (real(dp), intent(in) *height,* real(dp), intent(in), optional *p_zero,* real(dp), intent(in), optional *temperature,* real(dp), intent(in), optional *h_zero,* character(∗), intent(in), optional *method,* real(dp), intent(in), optional *dz,* character(∗), intent(in), optional *fels_type,* logical, intent(in), optional *use_standard_temperature,* logical, intent(in), optional *nan_as_zero*)

Computes pressure [Pa] for specific height.

See **?** or **?** for details. Uses formulae 5 from **?**.

Warning

pressure in Pa, height in meters

Definition at line 54 of file mod_atmosphere.f90.

8.23.2.5 real(dp) function mod_atmosphere::standard_temperature (real(dp), intent(in) *height*, character (len=∗), intent(in), optional *fels_type*, real(dp), intent(in), optional *t_zero*)

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 **?**

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

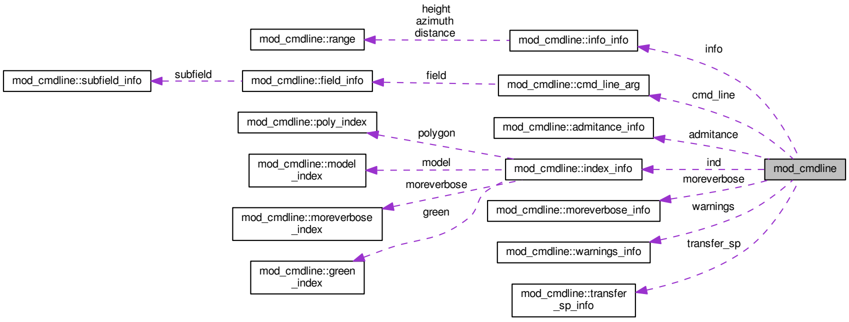
Definition at line 166 of file mod_atmosphere.f90.

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

- grat/src/mod_atmosphere.f90

8.24 mod_cmdline Module Reference

Collaboration diagram for mod_cmdline:



Data Types

- type admitance_info
- type cmd_line_arg
- type field_info
- type green_index
- type index_info
- type info_info
- type model_index
- type moreverbose_index
- type moreverbose_info
- type poly_index
- type range

- type subfield_info
- type transfer_sp_info
- type warnings_info

Public Member Functions

- subroutine collect_args (dummy)

  This routine collect command line arguments to one matrix depending on given switches and separators.
- subroutine get_command_cleaned (dummy)

  This subroutine removes unnecesary blank spaces from cmdline entry.

Public Attributes

- type(cmd_line_arg), dimension(:), allocatable **cmd_line**
- type(moreverbose_info), dimension(:), allocatable **moreverbose**
- type(info_info), dimension(:), allocatable **info**
- logical **inverted_barometer** = .true.
- logical **non_inverted_barometer** = .false.
- logical **ocean_conserve_mass** = .false.
- logical **inverted_landsea_mask** = .false.
- logical **optimize** = .false.
- logical **quiet** = .false.
- integer **quiet_step** = 50
- type(transfer_sp_info) **transfer_sp**
- type(warnings_info) **warnings**
- type(index_info) **ind**
- type(admitance_info) **admitance**
- logical, dimension(3) **method**
- logical, dimension(3) **method3d**
- logical **method3d_compute_reference** = .false.
- real **method3d_refinment_distance** = 0.1
- logical **dryrun**
- logical **result_total** = .false.
- logical **result_component** = .true.

8.24.1 Detailed Description

Definition at line 7 of file mod_cmdline.f90.

8.24.2 Member Function/Subroutine Documentation

8.24.2.1 subroutine mod_cmdline::collect_args (character(∗) *dummy*)

This routine collect command line arguments to one matrix depending on given switches and separators.

Date

2013.05.21

Author

Marcin Rajner

Definition at line 148 of file mod_cmdline.f90.

8.24.2.2 subroutine mod_cmdline::get_command_cleaned (character(∗), intent(out) *dummy*)

This subroutine removes unnecesary blank spaces from cmdline entry.

Marcin Rajner

Date

2013-05-13 allows specification like '-F file' and '-Ffile' but if -[0,9] it is treated as number belonging to switch (-S -2) but if -[,:] do not start next command line option

Definition at line 207 of file mod_cmdline.f90.

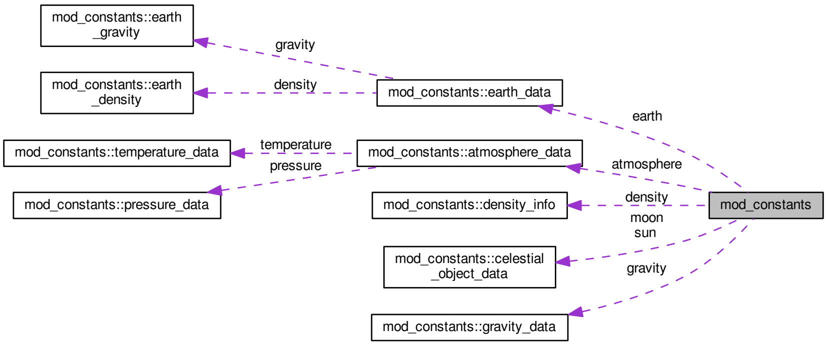
The documentation for this module was generated from the following file:

- grat/src/mod_cmdline.f90

8.25 mod_constants Module Reference

Define constant values.

Collaboration diagram for mod_constants:


mod_constants::earth_gravity
gravity
mod_constants::earth_density
density
mod_constants::earth_data
earth
mod_constants::temperature_data
temperature
mod_constants::atmosphere_data
pressure
atmosphere
mod_constants::pressure_data
mod_constants::density_info
density
mod_constants
moon
sun
mod_constants::celestial_object_data
gravity
mod_constants::gravity_data

Data Types

- type atmosphere_data
- type celestial_object_data
- type density_info
- type earth_data
- type earth_density
- type earth_gravity
- type gravity_data
- type pressure_data
- type temperature_data

Public Attributes

- integer, parameter **dp** = selected_real_kind(15)
- integer, parameter **sp** = selected_real_kind(6)
- real(dp), parameter **r_air** = 287.05
- real(dp), parameter **pi** = 4.∗atan(dble(1.))
- real(dp), parameter **t_zero** = -273.15
- type(gravity_data), parameter **gravity** = gravity_data(constant = 6.674e-11_dp)
- type(atmosphere_data), parameter **atmosphere** = atmosphere_data (pressure = pressure_data (standard = 101325._dp), temperature = temperature_data (standard = 288.15_dp))
- type(earth_data), parameter **earth** = earth_data (mass = 5.97219e24_dp, radius = 6371000., gm = 398600.-4419, gravity = earth_gravity(mean = 9.80665), density = earth_density(crust = 2670., mean = 5500.))
- type(celestial_object_data),
  parameter **moon** = celestial_object_data (distance = 384000000._dp, mass = 7.35e22_dp)
- type(celestial_object_data),
  parameter **sun** = celestial_object_data (distance = 149600000000._dp, mass = 1.99e30_dp)
- type(density_info), parameter **density** = density_info (water = 1000._dp)

8.25.1 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.

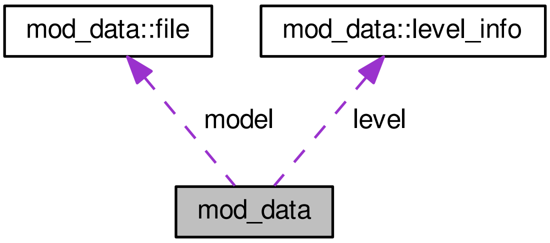
The documentation for this module was generated from the following file:

- grat/src/mod_constants.f90

8.26 mod_data Module Reference

This modele gives routines to read, and write data.

Collaboration diagram for mod_data:



Data Types

- type file
- type level_info

Public Member Functions

- subroutine parse_model (cmd_line_entry)

  This subroutine parse model information from command line entry.
- subroutine **model_aliases** (model, dryrun, year, month)
- real(dp) function **variable_modifier** (val, modifier, verbose, list_only)
- subroutine read_netcdf (model, print, force)

  Read netCDF file into memory.
- subroutine get_dimension (model, i, print)

  Get dimension, allocate memory and fill with values.
- subroutine nctime2date (model, print)

  Change time in netcdf to dates.
- integer function get_time_index (model, date)

  get time index
- integer function get_level_index (model, level, sucess)

  get level index
- subroutine **nc_info** (model)
- subroutine get_variable (model, date, print, level)

  Get variable from netCDF file for specified variables.
- subroutine get_scale_and_offset (ncid, varname, scale_factor, add_offset, status)

  Unpack variable.
- subroutine check (status, success)

  Check the return code from netCDF manipulation.
- subroutine get_value (model, lat, lon, val, level, method, date)

  Returns the value from model file.
- real(dp) function bilinear (x, y, aux)

  Performs bilinear interpolation.
- subroutine conserve_mass (model, landseamask, date, inverted_landsea_mask)

  If inverted barometer is set then averaga all pressure above the oceans.
- subroutine total_mass (model, date)

Mean pressure all over the model area.
- subroutine **parse_level** (cmd_line_entry)
- subroutine **customfile_value** (what, sp, t, hp, sh, gp, vsh, vt, level, val, rho)

Public Attributes

- type(file), dimension(:), allocatable **model**
- logical **all_huge** = .false.
- type(level_info) **level**

8.26.1 Detailed Description

This modele gives routines to read, and write data.

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

Author

M. Rajner

Date

2013-03-04

Definition at line 12 of file mod_data.f90.

8.26.2 Member Function/Subroutine Documentation

8.26.2.1 real(dp) function mod_data::bilinear (real(dp) *x,* real(dp) *y,* real(dp), dimension(4,3) *aux*)

Performs bilinear interpolation.

Author

Marcin Rajner

Date

2013-05-07

Definition at line 1030 of file mod_data.f90.

8.26.2.2 subroutine mod_data::check (integer, intent(in) *status,* logical, intent(out), optional *success*)

Check the return code from netCDF manipulation.

Author

From netcdf website **?**

Date

2013-03-04

Definition at line 862 of file mod_data.f90.

8.26.2.3 subroutine mod_data::get_dimension (type(file) *model*, integer, intent(in) *i*, logical, optional *print*)

Get dimension, allocate memory and fill with values.

Author

Marcin Rajner

Date

2013.05.24

Definition at line 515 of file mod_data.f90.

8.26.2.4 subroutine mod_data::get_scale_and_offset (integer, intent(in) *ncid*, character(∗), intent(in) *varname*, real(dp), intent(out) *scale_factor*, real(dp), intent(out) *add_offset*, integer, intent(out) *status*)

Unpack variable.

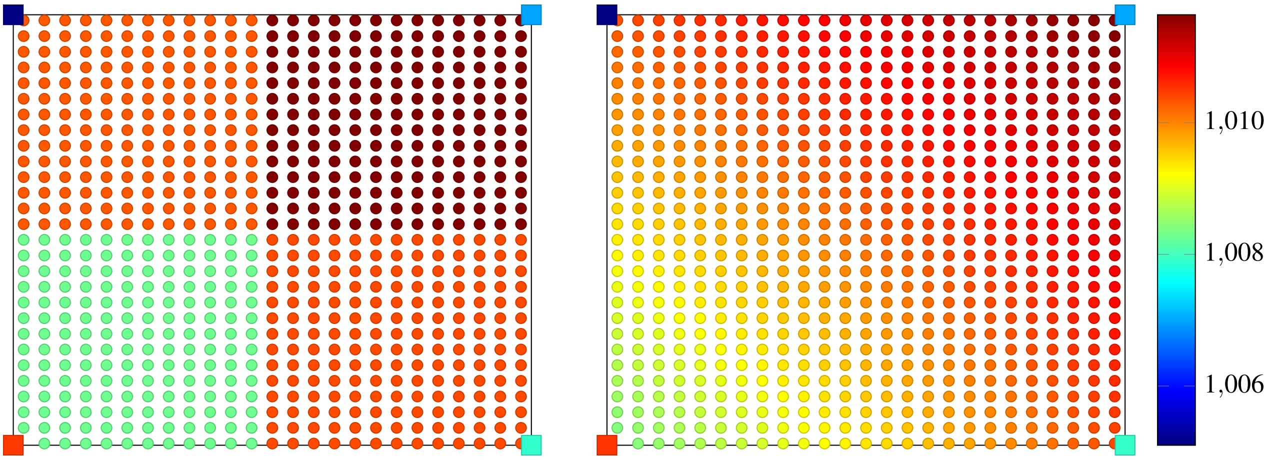
from **?** see `http://www.unidata.ucar.edu/software/netcdf/docs/BestPractices.html`

Definition at line 839 of file mod_data.f90.

8.26.2.5 subroutine mod_data::get_value (type(file), intent(in) *model*, real(dp) *lat*, real(dp) *lon*, real(dp), intent(out) *val*, integer, intent(in), optional *level*, character(1), intent(in), optional *method*, integer, dimension(6), intent(in), optional *date*)

Returns the value from model file.

The ilustration explain optional `method` argument



lat and lon in decimal degree

Definition at line 899 of file mod_data.f90.

8.26.2.6 subroutine mod_data::nctime2date (type (file) *model*, logical, optional *print*)

Change time in netcdf to dates.

Author

M. Rajner

Date

2013-03-04

Definition at line 600 of file mod_data.f90.

8.26.2.7 subroutine mod_data::parse_model (type(cmd_line_arg) *cmd_line_entry*)

This subroutine parse model information from command line entry.

Author

M. Rajner

Date

2013.05.20

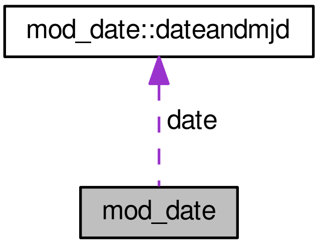
Definition at line 71 of file mod_data.f90.

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

- grat/src/mod_data.f90

8.27 mod_date Module Reference

Collaboration diagram for mod_date:



Data Types

- type dateandmjd

Public Member Functions

- subroutine parse_date (cmd_line_entry)

  Parse date given as 20110503020103 to yy mm dd hh mm ss and mjd.
- subroutine **more_dates** (number, start_index)
- subroutine string2date (string, date, success)

  Convert dates given as string to integer (6 elements)

Public Attributes

- real(dp) **cpu_start**
- real(dp) **cpu_finish**
- type(dateandmjd), dimension(:), allocatable **date**

8.27.1 Detailed Description

Definition at line 1 of file mod_date.f90.

8.27.2 Member Function/Subroutine Documentation

8.27.2.1 subroutine mod_date::parse_date (type(cmd_line_arg) *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 23 of file mod_date.f90.

8.27.2.2 subroutine mod_date::string2date (character (∗), intent(in) *string,* integer, dimension(6), intent(out) *date,* logical, optional *success*)

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

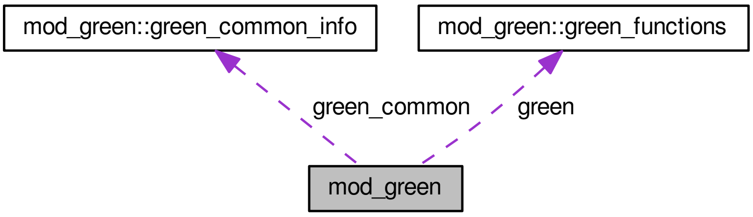
Definition at line 247 of file mod_date.f90.

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

- grat/src/mod_date.f90

8.28 mod_green Module Reference

Collaboration diagram for mod_green:



Data Types

- type green_common_info
- type green_functions

Public Member Functions

- subroutine parse_green (cmd_line_entry)

  This subroutine parse -G option – Greens function.
- subroutine read_green (green, print)

  This subroutine read green file.
- subroutine green_unification ()

  Unification:
- subroutine convolve (site, date)

  Perform convolution.
- subroutine printmoreverbose (latin, lonin, azimuth, azstep, distancestart, distancestop)

  returns lat and lon of spherical trapezoid
- real(dp) function **green_newtonian** (psi, h, z, method)

Public Attributes

- type(green_functions),
  dimension(:), allocatable **green**
- real(dp), dimension(:), allocatable **result**
- type(green_common_info),
  dimension(:), allocatable **green_common**
- integer **gnc_looseness** = 1

8.28.1 Detailed Description

Definition at line 2 of file mod_green.f90.

8.28.2 Member Function/Subroutine Documentation

8.28.2.1 subroutine mod_green::convolve (type(site_info), intent(in) *site*, type(dateandmjd), intent(in), optional *date*)

Perform convolution.

Date

2013-03-15

Author

M. Rajner

Definition at line 450 of file mod_green.f90.

8.28.2.2 subroutine mod_green::parse_green (type (cmd_line_arg), optional *cmd_line_entry*)

This subroutine parse -G option – Greens function.

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

Author

M. Rajner

Date

2013-03-06

Definition at line 42 of file mod_green.f90.

8.28.2.3 subroutine mod_green::printmoreverbose (real(dp), intent(in) *latin,* real(dp), intent(in) *lonin,* real(dp), intent(in) *azimuth,* real(dp), intent(in) *azstep,* real(dp) *distancestart,* real(dp) *distancestop*)

returns lat and lon of spherical trapezoid

Date

2013.07.03

Author

Marcin Rajner

Definition at line 1420 of file mod_green.f90.

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

- grat/src/mod_green.f90

8.29 mod_mjd Module Reference

Public Member Functions

- subroutine invmjd (mjd, date)

  Compute date from given Julian Day.
- real(dp) function jd (year, month, day, hh, mm, ss)

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

  MJD from date.

8.29.1 Detailed Description

Author

M. Rajner

Date

2013.06.27

Definition at line 5 of file mod_mjd.f90.

8.29.2 Member Function/Subroutine Documentation

8.29.2.1 subroutine mod_mjd::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 16 of file mod_mjd.f90.

8.29.2.2 real(dp) function mod_mjd::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 55 of file mod_mjd.f90.

8.29.2.3 real(dp) function mod_mjd::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 76 of file mod_mjd.f90.

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

- grat/src/mod_mjd.f90

8.30 mod_normalization Module Reference

Public Member Functions

- real(dp) function **green_normalization** (method, psi)

8.30.1 Detailed Description

Definition at line 4 of file mod_normalization.f90.

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

- grat/src/mod_normalization.f90

8.31 mod_parser Module Reference

Public Member Functions

- subroutine parse_option (cmd_line_entry, accepted_switches)

  This subroutine counts the command line arguments and parse appropriately.
- subroutine intro (program_calling, accepted_switches, cmdlineargs, version)

  This subroutine counts the command line arguments.
- subroutine **check_arguments** (program_calling)
- logical function if_accepted_switch (switch, accepted_switches)

  This function is true if switch is used by calling program or false if it is not.
- subroutine parse_moreverbose (cmd_line_entry)

  This subroutine parse -L option.
- subroutine parse_info (cmd_line_entry)

  This subroutine parse -I option.
- subroutine **info_defaults** (info)
- subroutine print_version (program_calling, version)

  Print version of program depending on program calling.
- subroutine **print_help** (program_calling, accepted_switches)
- character(len=40) function dataname (abbreviation)

  Attach full dataname by abbreviation.
- subroutine get_index ()

  This soubroutine stores indexes of specific dataname for data, green functions, polygon etc.

8.31.1 Detailed Description

Definition at line 1 of file mod_parser.f90.

8.31.2 Member Function/Subroutine Documentation

8.31.2.1 character(len=40) function mod_parser::dataname (character(len=2), intent(in) *abbreviation*)

Attach full dataname by abbreviation.

Date

2013-03-21

Author

M. Rajner

Definition at line 801 of file mod_parser.f90.

8.31.2.2 subroutine mod_parser::intro (character(len=∗), intent(in) *program_calling,* character(len=∗), intent(in), optional *accepted_switches,* logical, intent(in), optional *cmdlineargs,* character(∗), intent(in), optional *version*)

This subroutine counts the command line arguments.

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

optional accepted_switches: if given check if cmdlineargs are accepted, if not ignore them optional cmdlineargs: if .false. [default] run program anyway. if .true. stop program if no cmdline argumenst was given.

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 285 of file mod_parser.f90.

8.31.2.3 subroutine mod_parser::parse_info (type (cmd_line_arg), intent(in), optional *cmd_line_entry*)

This subroutine parse -I option.

Author

M. Rajner

Date

2013-05-17

Definition at line 573 of file mod_parser.f90.

8.31.2.4 subroutine mod_parser::parse_moreverbose (type (cmd_line_arg) *cmd_line_entry*)

This subroutine parse -L option.

Author

M. Rajner

Date

2013.05.24

Definition at line 527 of file mod_parser.f90.

8.31.2.5 subroutine mod_parser::print_version (character(∗) *program_calling,* character(∗), optional *version*)

Print version of program depending on program calling.

Author

M. Rajner

Date

2013-03-06

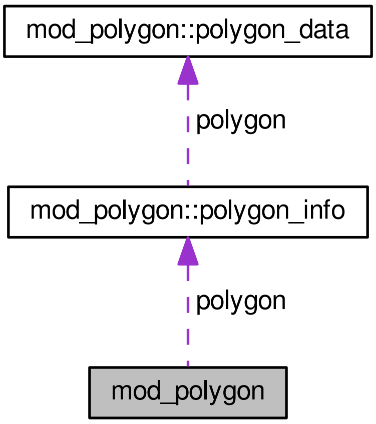
Definition at line 704 of file mod_parser.f90.

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

- grat/src/mod_parser.f90

8.32 mod_polygon Module Reference

Collaboration diagram for mod_polygon:



Data Types

- type polygon_data
- type polygon_info

Public Member Functions

- subroutine parse_polygon (cmd_line_entry)

  This subroutine parse polygon information from command line entry.
- subroutine read_polygon (polygon)

  Reads polygon data.

- subroutine chkgon (rlong, rlat, polygon, iok)

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

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

Public Attributes

- type(polygon_info), dimension(:),
  allocatable **polygon**

8.32.1 Detailed Description

Definition at line 10 of file mod_polygon.f90.

8.32.2 Member Function/Subroutine Documentation

8.32.2.1 subroutine mod_polygon::chkgon (real(dp), intent(in) *rlong,* real(dp), intent(in) *rlat,* type(polygon_info), intent(in) *polygon,* integer(2), intent(out) *iok*)

Check if point is in closed polygon.

From spotl **?** 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 **?**
adopted by Marcin Rajner

Date

2013-03-04

The ilustration explain exclusion idea

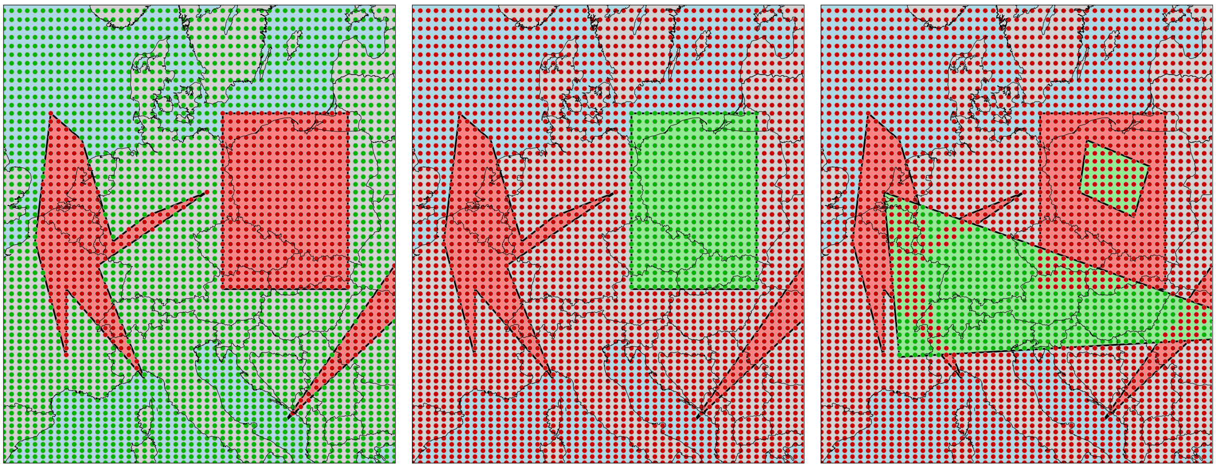


Figure 8.1: capt

Definition at line 164 of file mod_polygon.f90.

8.32.2.2 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 **?** slightly modified

Definition at line 276 of file mod_polygon.f90.

8.32.2.3 subroutine mod_polygon::parse_polygon (type(cmd_line_arg), intent(in) *cmd_line_entry*)

This subroutine parse polygon information from command line entry.

Author

M. Rajner

Date

2013.05.20

Definition at line 40 of file mod_polygon.f90.

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

Reads polygon data.

inspired by spotl **?**

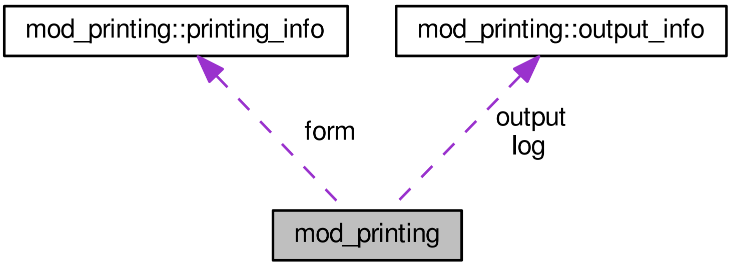
Definition at line 80 of file mod_polygon.f90.

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

- grat/src/mod_polygon.f90

8.33 mod_printing Module Reference

Collaboration diagram for mod_printing:



Data Types

- type output_info
- type printing_info

Public Member Functions

- subroutine **print_warning** (warn, unit, more, error, program_calling)
- subroutine **progress** (j, time, every)
- character(200) function **basename** (file)

Public Attributes

- character(len=255), parameter **form_header** = '(72("#"))'
- character(len=255), parameter **form_separator** = '("#",71("-"))'
- character(len=255), parameter **form_inheader** = '(("#"),1x,a68,1x,("#"))'
- character(len=255), parameter **form_inheader_n** = '(("#"),1x,a55,1x,i2.2,"(",i8,")",x,("#"))'
- 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** = "(8x,100(x,g0))"
- type(printing_info) **form**
- type(output_info) **log**
- type(output_info) **output**

8.33.1 Detailed Description

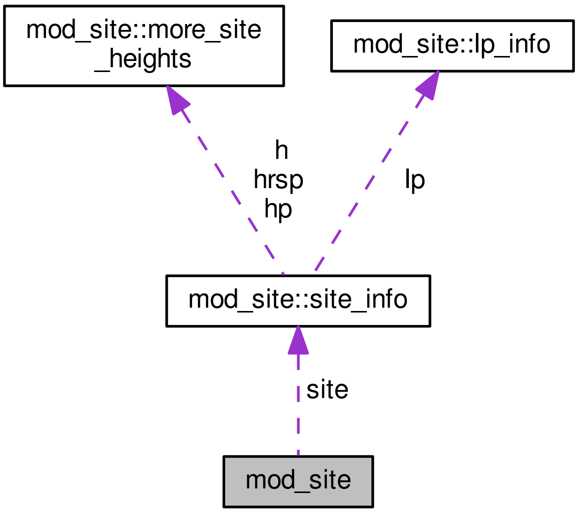
Definition at line 1 of file mod_printing.f90.

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

- grat/src/mod_printing.f90

8.34 mod_site Module Reference

Collaboration diagram for mod_site:



Data Types

- type lp_info
- type more_site_heights
- type site_info

Public Member Functions

- subroutine **parse_site** (cmd_line_entry)
- subroutine **print_site_summary** (site_parsing)
- subroutine **parse_gmt_like_boundaries** (field)
- subroutine **more_sites** (number, start_index)
- subroutine read_site_file (file_name)

  Read site list from file.
- subroutine **gather_site_model_info** ()
- subroutine **read_local_pressure** (file)

Public Attributes

- type(site_info), dimension(:),
  allocatable **site**
- logical **site_height_from_model** = .false.
- real(dp) **local_pressure_distance** = 0.25

8.34.1 Detailed Description

Definition at line 1 of file mod_site.f90.

8.34.2 Member Function/Subroutine Documentation

8.34.2.1 subroutine mod_site::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 351 of file mod_site.f90.

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

- grat/src/mod_site.f90

8.35 mod_spherical Module Reference

Public Member Functions

- real(dp) function spher_area (distance, ddistance, azstp, radius, alternative_method)

  Calculate area of spherical segment.
- subroutine spher_trig (latin, lonin, distance, azimuth, latout, lonout, domain)

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

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

8.35.1 Detailed Description

Definition at line 1 of file mod_spherical.f90.

8.35.2 Member Function/Subroutine Documentation

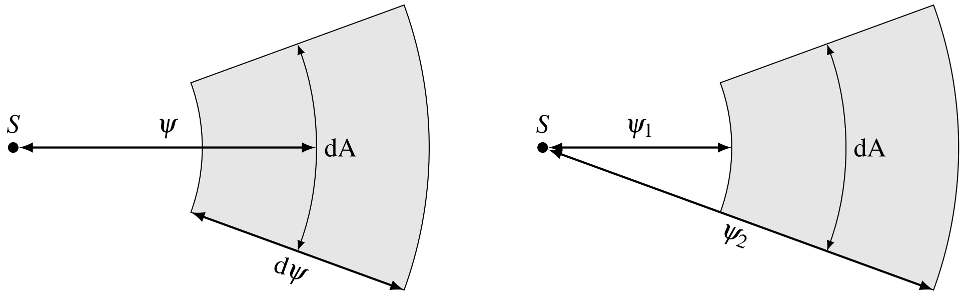
8.35.2.1 real(dp) function mod_spherical::spher_area (real(dp), intent(in) *distance,* real(dp), intent(in) *ddistance,* real(dp), intent(in) *azstp,* real(dp), intent(in), optional *radius,* logical, intent(in), optional *alternative_method*)

Calculate area of spherical segment.

Computes spherical area on unit (default if optional argument `radius` is not given) sphere given by:

- method 1 (`alternative_method` not given or `alternative_method` .false.)
  - distance from station, segment size in spher distance and angle
- method 2 (`alternative_method` .true.)
  - distance from station start, distance from station end

The ilustration explain optional `method` argument



Warning

All input angles in radians, output area on unit sphere or in square units of given (optionally) `radius`.

Definition at line 27 of file mod_spherical.f90.

8.35.2.2 subroutine mod_spherical::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*, logical, intent(in), optional *domain*)

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 **?**

Date

2012

Author

M. Rajner - modification

Date

2013-03-06

Warning

all values in radians

Definition at line 54 of file mod_spherical.f90.

8.35.2.3 subroutine mod_spherical::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.

All arguments in radians

Definition at line 90 of file mod_spherical.f90.

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

- grat/src/mod_spherical.f90

8.36 mod_utilities Module Reference

Public Member Functions

- subroutine spline_interpolation (x, y, n, x_interpolated, y_interpolated, n2, method)

  For given vectors x1, y1 and x2, y2 it gives x2 interpolated for x1.
- subroutine spline (x, y, b, c, d, n)

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

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

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

  This routine skips the lines with comment chars (default '#') from opened files (unit) to read.
- logical function is_numeric (string)

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

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

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

  radian -> degree
- subroutine count_records_to_read (file_name, rows, columns, comment_char)

  Count rows and (or) columns of file.
- integer function size_ntimes_denser (size_original, ndenser)

  returns numbers of arguments for n times denser size
- integer function count_separator (dummy, separator)

  Counts occurence of character (separator, default comma) in string.
- integer function **datanameunit** (dataname, datanames, count)
- real(dp) function **mmwater2pascal** (mmwater, inverted)
- real(dp) function, dimension(:),
  allocatable **linspace** (xmin, xmax, n)
- real(dp) function, dimension(:),
  allocatable **logspace** (xmin, xmax, n)
- subroutine **uniq_name_unit** (prefix, suffix, digits, start, unit, filename)
- real function **mean** (vec, i, nan)
- real function **stdev** (vec, i, nan)
- integer function **countsubstring** (s1, s2)
- subroutine **bubble_sort** (a)

8.36.1 Detailed Description

Definition at line 1 of file mod_utilities.f90.

8.36.2 Member Function/Subroutine Documentation

8.36.2.1 subroutine 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 370 of file mod_utilities.f90.

8.36.2.2 real(dp) function 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 342 of file mod_utilities.f90.

8.36.2.3 logical function mod_utilities::file_exists (character(len=∗), intent(in) *string,* logical, intent(in), optional *double_check,* logical, intent(in), optional *verbose*)

Check if file exists.

Logical function checking if given file exists.

Author

M. Rajner (based on www)

Date

2013-03-04

Definition at line 294 of file mod_utilities.f90.

8.36.2.4 logical function mod_utilities::is_numeric (character(len=∗), intent(in) *string*)

Check if argument is numeric.

Author

Taken from www

Date

2013-03-19
2013.07.16 added exception e.g /home/...

Definition at line 269 of file mod_utilities.f90.

8.36.2.5 real (dp) function 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 142 of file mod_utilities.f90.

8.36.2.6 integer function mod_utilities::ntokens (character, dimension(∗), intent(in) *line,* character(1), intent(in), optional *separator*)

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

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

or other optional separator added Marcin Rajner 2013.10.08

Definition at line 202 of file mod_utilities.f90.

8.36.2.7 real(dp) function 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 355 of file mod_utilities.f90.

8.36.2.8 integer function mod_utilities::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 405 of file mod_utilities.f90.

8.36.2.9 subroutine 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 51 of file mod_utilities.f90.

8.36.2.10 subroutine mod_utilities::spline_interpolation (real(dp), dimension(n), intent(in) *x,* real(dp), dimension(n), intent(in) *y,* integer, intent(in) *n,* real(dp), dimension(n2), intent(in) *x_interpolated,* real(dp), dimension(n2), intent(out) *y_interpolated,* integer, intent(in) *n2,* character(∗), optional *method*)

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

uses `ispline` and `spline` subroutines

Definition at line 13 of file mod_utilities.f90.

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

- grat/src/mod_utilities.f90

8.37 mod_cmdline::model_index Type Reference

Public Attributes

- integer(2) **sp**

- integer(2) **t**
- integer(2) **rsp**
- integer(2) **ewt**
- integer(2) **h**
- integer(2) **ls**
- integer(2) **hp**
- integer(2) **hrsp**
- integer(2) **gp**
- integer(2) **vt**
- integer(2) **vsh**

8.37.1 Detailed Description

Definition at line 89 of file mod_cmdline.f90.

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

- grat/src/mod_cmdline.f90

8.38 mod_site::more_site_heights Type Reference

Public Attributes

- real(dp) **val**
- logical **if** = .false.

8.38.1 Detailed Description

Definition at line 10 of file mod_site.f90.

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

- grat/src/mod_site.f90

8.39 mod_cmdline::moreverbose_index Type Reference

Public Attributes

- integer(2) **p**
- integer(2) **g**
- integer(2) **t**
- integer(2) **a**
- integer(2) **d**
- integer(2) **l**
- integer(2) **n**
- integer(2) **r**
- integer(2) **s**
- integer(2) **o**
- integer(2) **b**
- integer(2) **j**
- integer(2) **v**

8.39.1 Detailed Description

Definition at line 95 of file mod_cmdline.f90.

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

- grat/src/mod_cmdline.f90

8.40 mod_cmdline::moreverbose_info Type Reference

Public Attributes

- character(60) **name**
- character(30) **dataname**
- logical **sparse** = .false.
- logical **first_call** = .true.
- integer **unit**
- logical **noclobber** = .false.

8.40.1 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

8.41 mod_printing::output_info Type Reference

Public Attributes

- integer **unit** = output_unit
- character(255) **name**
- logical **if**
- logical **header**
- logical **tee**
- logical **noclobber** = .false.
- logical **full** = .false.
- logical **sparse** = .false.
- logical **height** = .false.
- logical **level** = .false.
- logical **time** = .false.
- logical **rho** = .false.
- logical **gp2h** = .false.
- logical **prune** = .false.
- logical **nan** = .false.
- character(10) **form** = "en13.3"

8.41.1 Detailed Description

Definition at line 35 of file mod_printing.f90.

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

- grat/src/mod_printing.f90

8.42 mod_cmdline::poly_index Type Reference

Public Attributes

- integer(2) **e**
- integer(2) **n**

8.42.1 Detailed Description

Definition at line 92 of file mod_cmdline.f90.

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

- grat/src/mod_cmdline.f90

8.43 mod_polygon::polygon_data Type Reference

Public Attributes

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

8.43.1 Detailed Description

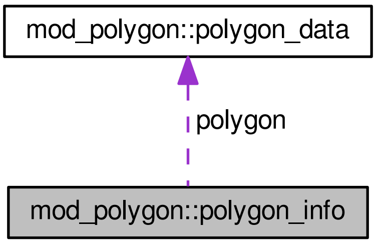
Definition at line 17 of file mod_polygon.f90.

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

- grat/src/mod_polygon.f90

8.44 mod_polygon::polygon_info Type Reference

Collaboration diagram for mod_polygon::polygon_info:



Public Attributes

- integer **unit**
- character(:), allocatable **name**
- character(len=25) **dataname**
- type(polygon_data), dimension(:), allocatable **polygon**
- logical **if**
- character(1) **pm**

8.44.1 Detailed Description

Definition at line 22 of file mod_polygon.f90.

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

- grat/src/mod_polygon.f90

8.45 mod_constants::pressure_data Type Reference

Public Attributes

- real(dp) **standard**

8.45.1 Detailed Description

Definition at line 33 of file mod_constants.f90.

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

- grat/src/mod_constants.f90

8.46 mod_printing::printing_info Type Reference

Public Attributes

- character(60) **a**
- character(60) **i0** = "(a,100(1x,g0))"
- character(60) **i1** = "(2x,a,100(1x,g0))"
- character(60) **i2** = "(4x,a,100(1x,g0))"
- character(60) **i3** = "(6x,a,100(1x,g0))"
- character(60) **i4** = "(8x,a,100(1x,g0))"
- character(60) **i5** = "(10x,a,100(1x,g0))"
- character(60) **t1** = "2x"
- character(60) **t2** = "4x"
- character(60) **t3** = "6x"
- character(60) **separator** = '("#",71("-"))'

8.46.1 Detailed Description

Definition at line 19 of file mod_printing.f90.

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

- grat/src/mod_printing.f90

8.47 mod_cmdline::range Type Reference

Public Attributes

- real(dp) **start**
- real(dp) **stop**
- real(dp) **step**
- integer **denser**
- real(dp) **stop_3d**

8.47.1 Detailed Description

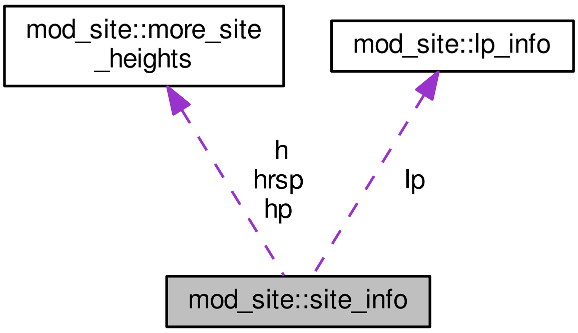
Definition at line 47 of file mod_cmdline.f90.

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

- grat/src/mod_cmdline.f90

8.48 mod_site::site_info Type Reference

Collaboration diagram for mod_site::site_info:



Public Attributes

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

- real(dp) **height**
- type(more_site_heights) **hp**
- type(more_site_heights) **h**
- type(more_site_heights) **hrsp**
- logical **use_local_pressure** = .false.
- type(lp_info) **lp**

8.48.1 Detailed Description

Definition at line 21 of file mod_site.f90.

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

- grat/src/mod_site.f90

8.49 mod_cmdline::subfield_info Type Reference

Public Attributes

- character(len=100) **name**
- character(len=100) **dataname**

8.49.1 Detailed Description

Definition at line 15 of file mod_cmdline.f90.

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

- grat/src/mod_cmdline.f90

8.50 mod_constants::temperature_data Type Reference

Public Attributes

- real(dp) **standard**

8.50.1 Detailed Description

Definition at line 37 of file mod_constants.f90.

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

- grat/src/mod_constants.f90

8.51 mod_cmdline::transfer_sp_info Type Reference

Public Attributes

- logical **if** = .false.
- character(20) **method** = "standard"

8.51.1 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

8.52 mod_cmdline::warnings_info Type Reference

Public Attributes

- logical **if** = .true.
- logical **strict** = .false.
- logical **time** = .false.

8.52.1 Detailed Description

Definition at line 81 of file mod_cmdline.f90.

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

- grat/src/mod_cmdline.f90

Chapter 9

File Documentation

9.1 grat/doc/figures/interpolation_ilustration.sh File Reference

9.1.1 Detailed Description

Definition in file interpolation_ilustration.sh.

9.2 interpolation_ilustration.sh

```
#!/bin/bash -
#================================================================================
#          FILE: interpolation_ilustration.sh
#         USAGE: ./interpolation_ilustration.sh
#   DESCRIPTION:
#       OPTIONS: ---
#        AUTHOR: mrajner
#       CREATED: 05.12.2012 10:38:30 CET
#      REVISION:  ---
#================================================================================

## \file
set -o nounset                              # Treat unset variables as an error
for co in n l
do
  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 ${co} @ I                 \
    -V                                                            \
    -o interp${co}1.dat                                           \
    -L interp1.dat@l
done
perl -n -i -e 'print if $. <= 4' interp1.dat

```

9.3 grat/src/grat.f90 File Reference

Functions/Subroutines

- program **grat**

9.3.1 Detailed Description

Definition in file grat.f90.

9.4 grat.f90

```fortran
!> \file
!! \mainpage grat overview
!! \section Purpose
!! This program was created to make computation of atmospheric gravity
!! correction easier. Still developing. Consider visiting later...
!!
!! \version pre-alpha
!! \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
!! unnecesarry 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
!> \copyright
!! Copyright 2013 by Marcin Rajner\n
!! This program is free software: you can redistribute it and/or modify
!! it under the terms of the GNU General Public License as published by
!! the Free Software Foundation, either version 3 of the License, or
!! (at your option) any later version.
!! \n\n
!! This program is distributed in the hope that it will be useful,
!! but WITHOUT ANY WARRANTY; without even the implied warranty of
!! MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
!! GNU General Public License for more details.
!! \n\n
!! You should have received a copy of the GNU General Public License
!! along with this program.
!! If not, see <http://www.gnu.org/licenses/>.
!! \page License
!! \include LICENSE
!!
!! \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

!> \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 omp_lib parallel computation not yet enabled
  use mod_parser, only: intro
  use mod_data
  use mod_date
  use mod_green, only: convolve, green
  use mod_site, only: print_site_summary, site
  use mod_cmdline
  use mod_admit, only: admit
  use mod_utilities, only: Bubble_Sort

  implicit none
  real(dp) :: cpu(2)
  integer  :: isite, i, idate, start, iprogress = 0
  logical  :: first_waning = .true.

  ! program starts here with time stamp
  call cpu_time(cpu(1))

  ! gather cmd line option decide where to put output
  call intro(                                         &
    program_calling   = "grat",                        &
    version           = "pre-alpha",                   &
    accepted_switches = "VSBLGPqoFIDLvhRrMOAHUwJQ&!n", &
    cmdlineargs       = .true.                         &
    )

  start = 0

  if (dryrun) then
    call print_site_summary(site_parsing=.true.)
    call exit(0)
  endif

  if (size(date).gt.0) then
    if(output%header) then
      write (output%unit, '(a12,x,a14,x)', advance = "no" ) "mjd", "date"
    endif
    start = 1
  endif

  if(output%header) then
    write (output%unit, '(a8,3(x,a9$))') "name", "lat", "lon", "h"
  endif

  if(output%header) then

    if (method(1)) then
      write (output%unit,'(a13)', advance='no'), "G1D"
    endif

    if (method(2).or.method(3)) then
      if (result_component) then
        do i = 1, size(green)
          if (green(i)%dataname.eq."GE") then
            if (inverted_barometer) then
              write (output%unit,'(a13$)'), trim(green(i)%dataname)//"_IB"
            else
              write (output%unit,'(a13$)'), trim(green(i)%dataname)//"_NIB"
            endif
          else
            write (output%unit,'(a13$)'), trim(green(i)%dataname)
          endif
        enddo
        if (inverted_barometer.and.non_inverted_barometer) then
          write (output%unit,'(a13$)'), "GE_NIB"
        endif
      endif

      if (result_total) then
        if (method(2)) then
          write (output%unit,'(a13)',advance='no'), "G2D_t"
        endif
        if (method(3)) then
          write (output%unit,'(a13)',advance='no'), "G3D_t"
        endif
      endif
    endif
  endif

  if(output%header) then
    write (output%unit, *)
  endif

  ! read only once Land-sea, reference surface pressure
  if (ind%model%ls.ne.0) then
    call get_variable(model(ind%model%ls))
  endif
  if (ind%model%rsp.ne.0) then
    call get_variable(model(ind%model%rsp))
  endif
  if (ind%model%hrsp.ne.0) then
    call get_variable(model(ind%model%hrsp))
  endif

  if (inverted_landsea_mask.and.ind%model%ls.ne.0) then
    model(ind%model%ls)%data = int(abs(model(ind%model%ls)%data-1))
  endif


  do idate=start, size (date)
    if (idate.ge.1) then
      if(.not.(output%nan).and.modulo(date(idate)%date(4),6).ne.0) then
        if (first_waning) call print_warning  &
          ("hours not matching model dates (0,6,12,18) are rejecting and not shown in output")
        first_waning=.false.
        cycle
      endif
    endif

    do i = 1, size(model)
      if(model(i)%if) then

        select case (model(i)%dataname)
        case ("SP", "T", "GP", "VT", "VSH")
          if (model(i)%autoload                                   &
            .and.                                               &
            .not.(                                              &
            model(i)%autoloadname.eq."ERA"                      &
            .and.(any(model(i)%dataname.eq.["GP","VT","VSH"]))) &
            then

            if ( &
              (idate.eq.1 &
              .or. .not. date(idate)%date(1).eq.date(idate-1)%date(1) &
              )) then

              call model_aliases(model(i), year=date(idate)%date(1))
            endif

          else if (model(i)%autoload) then
            if (                                                  &
              (idate.eq.1                                         &
              .or. .not.(                                         &
              date(idate)%date(1).eq.date(idate-1)%date(1)        &
              .and.date(idate)%date(2).eq.date(idate-1)%date(2)) &
              )                                                   &
              ) then

              call model_aliases( &
                model(i), year=date(idate)%date(1), month=date(idate)%date(2))
            endif
          endif

          if (size(date).eq.0.and.model(i)%exist) then
            stop "temporary"
            call get_variable(model(i))
          elseif(model(i)%exist) then
            call get_variable(model(i), date = date(idate)%date)
          endif

        end select
      endif
    enddo

    if (any(.not.model%exist).and..not.output%nan) cycle

    if (level%all.and..not.allocated(level%level)) then
      allocate(level%level(size(model(ind%model%gp)%level)))
      level%level=model(ind%model%gp)%level
    endif

    ! sort levels for 3D method
    call bubble_sort(level%level)

    ! if ocean mass should be conserved (-O C)
    if (ocean_conserve_mass) then
      if (ind%model%sp.ne.0 .and. ind%model%ls.ne.0) then
        if(size(date).eq.0) then
          call conserve_mass(model(ind%model%sp), model(ind%model%ls), &
            inverted_landsea_mask = inverted_landsea_mask)
        else
          call conserve_mass(model(ind%model%sp), model(ind%model%ls), &
            date=date(idate)%date, &
            inverted_landsea_mask = inverted_landsea_mask)
        endif
      endif
    endif

    ! calculate total mass if asked for
    if (ind%moreverbose%t.ne.0) then
      if (size(date).eq.0) then
        call total_mass(model(ind%model%sp))
      else
        call total_mass(model(ind%model%sp), date=date(idate)%date)
      endif
    endif


    do isite = 1, size(site)
      iprogress = iprogress + 1

```

```
      if (idate.gt.0) then
        write(output%unit, '(f12.3,x,i4.4,5(i2.2),x)', advance="no") &
          date(idate)%mjd, date(idate)%date
      endif

      write (output%unit, '(a8,2(x,f9.4),x,f9.3,$)' ), &
        site(isite)%name,                            &
        site(isite)%lat,                             &
        site(isite)%lon,                             &
        site(isite)%height

      if (method(1)) then
        write (output%unit, "("// output%form // '$)'), &
          admit(                                       &
          site(isite),                               &
          date=date(idate)%date                      &
          )
      endif

      if (method(2).or.method(3)) then
        ! perform convolution
        call convolve(site(isite), date = date(idate))
      endif

      write(output%unit,*)

      if (output%unit.ne.output_unit.and..not.(quiet.and.quiet_step.eq.0)) then
        open(unit=output_unit, carriagecontrol='fortran')
        call cpu_time(cpu(2))
        call progress(                        &
          100*iprogress/(max(size(date),1) &
          *max(size(site),1)),             &
          cpu(2)-cpu(1),                   &
          every=quiet_step                 &
          )
      endif
    enddo
  enddo

  ! execution time-stamp
  call cpu_time(cpu(2))
  if (output%unit.ne.output_unit.and..not.(quiet.and.quiet_step.eq.0)) then
    call progress(100*iprogress/(max(size(date),1)*max(size(site),1)), cpu(2)-cpu(1), every=1)
    close(output_unit)
  endif
  write(log%unit, '("Execution time:",1x,f10.4," seconds")') cpu(2)-cpu(1)
  if (output%time) write(output%unit, '("Execution time:",1x,f10.4," seconds")') cpu(2)-cpu(1)
  write(log%unit, form_separator)
end program
```

9.5 grat/src/mod_admit.f90 File Reference

Data Types

- module mod_admit

9.5.1 Detailed Description

Definition in file mod_admit.f90.

9.6 mod_admit.f90

```
!> \file
module mod_admit
  use mod_constants, only: dp

  implicit none

contains
! =============================================================================
! =============================================================================
real(dp) function admit(site_, date)
```

```
use mod_cmdline, only: ind, info, admitance
use mod_data, only: get_value, model
use mod_utilities, only: r2d
use mod_atmosphere, only: standard_pressure
use mod_site
use mod_cmdline, only: transfer_sp

real(dp) :: val, rsp, t !, hrsp
type(site_info) :: site_
integer, optional :: date(6)
integer :: i
logical, save :: first_warning=.true.


if (site_%lp%if) then
  val=0
  do i=1,size(site_%lp%date)
    if(all(site_%lp%date(i,1:6).eq.date(1:6))) then
      val=site_%lp%data(i)
      exit
    endif
    if(i.eq.size(site_%lp%date)) then
      if(first_warning) call print_warning("date not found in @LP")
      val=sqrt(-1.)
    endif

  enddo
else
  ! get SP
  if (ind%model%sp.ne.0                           &
    .and.(model(ind%model%sp)%if                &
    .or. model(ind%model%sp)%if_constant_value) &
    ) then
    call get_value(                    &
      model=model(ind%model%sp),       &
      lat=site_%lat,                  &
      lon=site_%lon,                  &
      val=val,                        &
      level=1,                        &
      method = info(1)%interpolation, &
      date=date                       &
      )
  else
    call print_warning("@SP is required with -M1D", error=.true.)
  endif
endif


! get RSP
if (ind%model%rsp.ne.0) then
  call get_value(                   &
    model=model(ind%model%rsp),    &
    lat=site_%lat,                 &
    lon=site_%lon,                 &
    val=rsp,                       &
    level=1,                       &
    method = info(1)%interpolation &
    )
endif

if (transfer_sp%if) then
  if (ind%model%h.eq.0 ) then
    if (first_warning) call print_warning("transfer on topo but no @H")
  endif

  ! get T
  if (ind%model%t.ne.0) then
    call get_value(                  &
      model=model(ind%model%t),      &
      lat=site_%lat,                &
      lon=site_%lon,                &
      val=t,                        &
      level=1,                      &
      method=info(1)%interpolation, &
      date=date                     &
      )
  endif

  ! transfer SP
  if (site_%hp%if.and..not.isnan(val)) then
    val = standard_pressure(                &
      height=site_%height,                 &
      h_zero=site_%hp%val,                 &
      p_zero=val,                          &
      method=transfer_sp%method,           &
      temperature=t,                       &
      use_standard_temperature             &
```

```
        = ind%model%t.eq.0,                 &
        nan_as_zero=.false.)
    endif

    ! if (ind%model%hrsp.ne.0 .and.ind%model%rsp.ne.0)  then
    ! call get_value (                 &
    ! model=model(ind%model%hrsp),   &
    ! lat=site_%lat,                 &
    ! lon=site_%lon,                 &
    ! val=hrsp,                      &
    ! level=1,                       &
    ! method = info(1)%interpolation &
    ! )

    ! rsp = standard_pressure(     &
    ! height=site_%height,       &
    ! h_zero=hrsp,               &
    ! p_zero=rsp,                &
    ! method=transfer_sp%method, &
    ! temperature=t,             &
    ! use_standard_temperature   &
    ! = ind%model%t.eq.0,        &
    ! nan_as_zero=.false.)

    ! elseif(ind%model%hrsp.ne.0) then
    ! if (first_warning) call print_warning("@RSP not found but @HRSP and -U given")
    ! elseif(ind%model%rsp.ne.0) then
    ! if (first_warning) call print_warning("@HRSP not found but @RSP and -U given")
    ! end if
  endif

  if (ind%model%rsp.ne.0) val = val-rsp
  admit = admitance%value*1.e-2 * val

  if (first_warning) first_warning=.false.
end function

! =============================================================================
!> \date 2013.10.15
!! \author Marcin Rajner
! =============================================================================
subroutine parse_admit(cmd_line_entry)
  use mod_cmdline
  use mod_printing
  type (cmd_line_arg) :: cmd_line_entry
  if (cmd_line_entry%field(1)%subfield(1)%name.ne."") then
    read(cmd_line_entry%field(1)%subfield(1)%name, *) admitance%value
  endif
  if (.not.log%sparse) &
    write(log%unit, '('//form%t2//',a,x,f6.2,x,a)') "admitance:", admitance%value, "uGal/hPa"

  ! not sure what trying to achive
  ! if (size(cmd_line_entry%field(1)%subfield).gt.1 &
  ! .and.cmd_line_entry%field(1)%subfield(2)%name.ne." ") then
  ! admitance%level=cmd_line_entry%field(1)%subfield(2)%name
  ! else
  ! admitance%level="none"
  ! endif
  ! write(log%unit, form%i2) "level:", admitance%level
end subroutine
end module
```

9.7 grat/src/mod_aggf.f90 File Reference

This module contains utitlities for computing Atmospheric Gravity Green Functions.

Data Types

- module mod_aggf

9.7.1 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.

9.8 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
  implicit none

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
!! \author M. Rajner
!! \date 2013-03-19
!! \warning psi in radians
! ===============================================================================
function aggfd ( &
    psi,         &
    delta,       &
    dz,          &
    method,      &
    aggfdh,      &
    aggfdz,      &
    aggfdt,      &
    predefined,  &
    fels_type,   &
    rough)

  use mod_constants, only: atmosphere, dp

  real(dp), intent (in) :: psi
  real(dp), intent (in), optional :: delta
  real(dp), intent (in), optional :: dz
  logical, intent (in), optional :: aggfdh, aggfdz, aggfdt, predefined, rough
  real(dp) :: aggfd
  real(dp) :: delta_
  character (len=*), intent(in), optional  :: method, fels_type

  delta_ = 10. ! Default value
  if (present(delta))  delta_ = delta

  if(present(aggfdh).and.aggfdh) then
    aggfd = (                &
      + aggf(psi,           &
      h=+delta_,             &
      dz=dz,                 &
      method=method,         &
      predefined=predefined, &
      fels_type=fels_type,   &
      rough=rough)           &
      - aggf(psi,           &
      h=-delta_,             &
      dz=dz,                 &
      method=method,         &
      predefined=predefined, &
      fels_type=fels_type,   &
      rough=rough))          &
      / ( 2. * delta_)
  else if(present(aggfdz).and.aggfdz) then
    aggfd = (                &
      + aggf(psi,           &
      zmin = +delta_,        &
      dz=dz,                 &
      method = method,       &
      predefined=predefined, &
      fels_type=fels_type,   &
      rough=rough)           &
      - aggf(psi,           &
      zmin = -delta_,        &
      dz=dz,                 &
      method = method,       &
      predefined=predefined, &
      fels_type=fels_type,   &
      rough=rough))          &
      / ( 2. * delta_)
  else if(present(aggfdt).and.aggfdt) then
    aggfd = (                &
      + aggf(psi,           &
      t_zero = +delta_,      &
      dz=dz,                 &
      method = method,       &
      predefined=predefined, &
      fels_type=fels_type,   &
      rough=rough)           &
      - aggf(psi,           &
      t_zero = -delta_,      &
      dz=dz,                 &
      method = method,       &
      predefined=predefined, &
      fels_type=fels_type,   &
      rough=rough))          &
      / ( 2. * delta_)
  endif
end function

!
 ==============================================================================
!> This function computes the value of atmospheric gravity green functions
!! (AGGF) on the basis of spherical distance (psi)
!! \author Marcin Rajner
!! \date 2013.07.15
!! \warning psi in radians h in meter
!! t_zero is actually delta_t so if t_zero=10 (t_zero=288.15+10)
!
 ==============================================================================
function aggf (          &
    psi,                 &
    zmin, &
    zmax, &
    dz,     &
    t_zero,              &
    h,                   &
    first_derivative_h,  &
    first_derivative_z,  &
    fels_type,           &
    method,              &
    predefined,          &
    rough)

  use mod_constants, only: dp, pi, earth, gravity, atmosphere,
R_air
  use mod_utilities, only: d2r
  use mod_atmosphere
  use mod_normalization, only : green_normalization

  real(dp), intent(in)          :: psi ! spherical distance from site [rad]
  real(dp), intent(in),optional :: &
    zmin,                          &   ! minimum height, starting point [m]
 (default = 0)
    zmax,                          &   ! maximum height, ending point   [m]
 (default = 60000)
    dz,                            &   ! integration step               [m]
 (default = 0.1 -> 10 cm)
    t_zero,                        &   ! temperature at the surface     [K]
 (default = 15°C i.e., 288.15=t0)
    h                                  ! station height                 [m]
 (default = 0)
  logical, intent(in), optional :: &
    first_derivative_h, first_derivative_z, predefined, rough
  character (len=*), intent(in), optional  :: fels_type, method
  character (len=20) :: old_method
  real(dp) :: aggf
  real(dp) :: zmin_, zmax_, dz_, h_
  real(dp) :: j_aux
  real(dp) :: rho, l, deltat

  real(dp), dimension(:), allocatable, save :: heights, pressures
  integer :: i

  zmin_ = 0.
  zmax_ = 60000.
  dz_   = 0.1
  h_    = 0.

  aggf=0.

  if (present(zmin)) zmin_ = zmin
  if (present(zmax)) zmax_ = zmax
  if (present(  dz))   dz_ = dz
  if (present(   h))    h_ = h
  if (present(t_zero)) deltat=t_zero

  if(allocated(heights)) then
    if ( &
      ((zmin_ +dz_/2).ne.heights(1)) &
      .or.abs((zmax_-dz_/2)-heights(size(heights))).gt.zmax_/1e6 &
      .or.nint((zmax_-zmin_)/dz_).ne.size(heights) &
      .or. (present(predefined)) &
      .or. method.ne.old_method &
      .or. present(t_zero) &
      ) then
      deallocate(heights)
      deallocate(pressures)
    endif
  endif

  if (.not.allocated(heights))  then
    allocate(heights(nint((zmax_-zmin_)/dz_)))
    allocate(pressures(size(heights)))
    do i = 1, size(heights)
      heights(i) = zmin_ &
        + dz_/2  &
        + (i-1) * dz_
    enddo

    if (present(rough).and.rough) then
      ! do not use rough! it is only for testing
      do i = 1, size(heights)
        pressures(i) = standard_pressure( &
          heights(i),                      &
          method=method,                   &
          dz=dz,                           &
          use_standard_temperature=.true.  &
          )
      enddo
    else
      pressures(1) = standard_pressure(     &
        heights(1),                         &
        method = method,                    &
        h_zero = zmin_,                     &
        dz = dz,                            &
        fels_type=fels_type,                &
        use_standard_temperature=.true.,    &
        temperature = standard_temperature( &
        zmin_, fels_type=fels_type)+deltat  &
        )
      do i = 2, size(heights)
        pressures(i) = standard_pressure(                  &
          heights(i),                                      &
          p_zero = pressures(i-1),                         &
          h_zero = heights(i-1),                           &
          method = method,                                 &
          dz = dz,                                         &
          fels_type=fels_type,                             &
          use_standard_temperature=.true.,                 &
          temperature = standard_temperature(heights(i-1), &
          fels_type=fels_type)+deltat                      &
          )
      enddo
    endif
  endif
  old_method=method

  do i = 1, size(heights)
    l = ((earth%radius + heights(i))**2 + (earth%radius + h_)**2 &
      - 2.*(earth%radius + h_)*(earth%radius+heights(i))*cos(psi))**(0.5)
    rho = pressures(i)/ r_air / (deltat+standard_temperature(heights(i), fels_type=fels_type))
    if (present(first_derivative_h) .and. first_derivative_h) then
      ! first derivative (respective to station height)
      ! micro Gal height / m
      ! see equation 22, 23 in \cite Huang05
      j_aux =  ((earth%radius + heights(i) )**2)*(1.-3.*((cos(psi))**2)) -2.*(earth%radius + h_)**2  &
        + 4.*(earth%radius+h_)*(earth%radius+heights(i))*cos(psi)
      aggf =  aggf + rho * (  j_aux  /  l**5  ) * dz_

    else if (present(first_derivative_z) .and. first_derivative_z) then
      ! first derivative (respective to column height)
      ! according to equation 26 in \cite Huang05
      ! micro Gal / hPa / m
      if (i.gt.1) exit
      aggf = rho *( ((earth%radius + heights(i))*cos(psi)-(earth%radius + h_)) / (l**3))
    else
      ! GN microGal/hPa
      aggf = aggf &
        -rho*((earth%radius +heights(i))*cos(psi) - (earth%radius + h_)) / (l**3.)  * dz_
    endif
  enddo
  aggf = aggf/atmosphere%pressure%standard*gravity%constant*green_normalization("m", psi=psi)
end function

! ===============================================================================
!> 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
!! \warning psi in radian
!! \todo explanaition ??
! ===============================================================================
function gn_thin_layer (psi)
  use mod_constants, only: dp
  real(dp), intent(in) :: psi
  real(dp) :: gn_thin_layer

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


! ===============================================================================
!> \brief Bouger plate computation
!!
! ===============================================================================
real(dp) function bouger (h, R )
  use mod_constants, only: dp, gravity, pi
  real(dp), intent(in), optional :: r !< height of point above the cylinder
  real(dp), intent(in) ::  h

  if (present( r ) ) then
    bouger = h + r - sqrt(r**2+h**2)
  else
    bouger = h
  endif
  bouger = 2 * pi * gravity%constant * bouger
  return
end function

! ===============================================================================
!> Bouger plate computation
!!
!! see eq. page 288 \cite Warburton77
!! \date 2013-03-18
!! \author M. Rajner
! ===============================================================================
function simple_def (R)
  use mod_constants, only: dp, earth
  real(dp) :: r, delta
  real(dp) :: simple_def

  delta = 0.22e-11 * r
  simple_def = earth%gravity%mean / earth%radius *1000 * &
    delta * ( 2. - 3./2. * earth%density%crust / earth%density%mean &
    -3./4. * earth%density%crust / earth%density%mean * sqrt(2* (1. )) &
    ) * 1000
end function

end module
```

9.9 grat/src/mod_cmdline.f90 File Reference

This module gather cmd line arguments.

Data Types

- module mod_cmdline
- type mod_cmdline::subfield_info
- type mod_cmdline::field_info
- type mod_cmdline::cmd_line_arg
- type mod_cmdline::moreverbose_info
- type mod_cmdline::range
- type mod_cmdline::info_info
- type mod_cmdline::transfer_sp_info
- type mod_cmdline::warnings_info
- type mod_cmdline::model_index
- type mod_cmdline::poly_index
- type mod_cmdline::moreverbose_index
- type mod_cmdline::green_index
- type mod_cmdline::index_info
- type mod_cmdline::admitance_info

9.9.1 Detailed Description

This module gather cmd line arguments. 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.

9.10 mod_cmdline.f90

```
!> \file
!! \brief This module gather cmd line arguments
!!
!! 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

    implicit none

    !----------------------------------------------------
    ! command line entry
    !----------------------------------------------------
    type subfield_info
      character (len=100) :: name
      character (len=100) :: dataname
    end type
    type field_info
      character (len=355) :: full
      type(subfield_info), allocatable, &
          dimension(:) :: subfield
    end type
    type cmd_line_arg
      character(2) :: switch
      type (field_info), allocatable, &
          dimension(:) :: field
      character (len=455) :: full
    end type
    type(cmd_line_arg), allocatable, dimension(:) :: cmd_line

    private :: check_if_switch_or_minus

    type moreverbose_info
      character(60) :: name
      character(30):: dataname
      logical  :: sparse=.false.
      logical :: first_call = .true.
      integer :: unit
      logical :: noclobber = .false.
    end type
    type(moreverbose_info), allocatable, dimension(:) :: moreverbose

    !----------------------------------------------------
    ! info
    !----------------------------------------------------
    type range
      real(dp):: start
      real(dp):: stop
      real(dp):: step
      integer :: denser
      real(dp):: stop_3d
      ! logical :: stop_3d_if
    end type
    type info_info
      type (range):: distance,azimuth, height
      character (1) :: interpolation
    end type
    type(info_info), dimension(:), allocatable:: info

    !----------------------------------------------------
    ! general settings
    !----------------------------------------------------
    logical :: &
        inverted_barometer     = .true.  , &
        non_inverted_barometer = .false. , &
        ocean_conserve_mass    = .false. , &
        inverted_landsea_mask  = .false. , &
        optimize               = .false. , &
        quiet                  = .false.
    integer :: quiet_step=50

    type transfer_sp_info
      logical :: if = .false.
      ! by default with 2D method pressure is transfered
      ! on topography (@H)
      character(20) :: method="standard"
    end type
    type(transfer_sp_info) transfer_sp

    type warnings_info
      logical :: &
          if = .true., &
          strict=.false., &
          time=.false.
    end type
    type(warnings_info) warnings

    type model_index
      integer(2) :: sp, t, rsp, ewt, h, ls, hp, hrsp, gp, vt, vsh
    end type
    type poly_index
      integer(2) :: e, n
    end type
    type moreverbose_index
      integer(2) :: p, g, t, a, d, l, n, r, s, o, b, j, v
    end type
    type green_index
      integer(2) :: &
          gn          = 0,  & ! green newtonian   - with SP  in Pa
          ge          = 0,  & ! green elastic     - with SP  in Pa
          gegdt       = 0,  & ! green elastic     - first derivative of gravity part respect to temp (see Guo et al., 2004)
          gr          = 0,  & ! green radial      - with EWT in mm
          ghn         = 0,  & ! green horizontal  - with EWT in mm
          ghe         = 0,  & ! green horizontal  - with EWT in mm
          gg          = 0,  & ! green gravimetric - with SP  in Pa
          ! (like elastic but uses green not normalized according to Merriam)
      gndt        = 0,  & ! first derivative respect to temperature
          gndh        = 0,  & ! first derivative respect to station height
          gndz        = 0,  & ! first derivative respect to column height
          gndz2       = 0,  & ! second derivative respect to column height
          gnc         = 0,  & ! compute aggf every time
          g3d
    end type
    type index_info
      type (model_index)       :: model
      type (moreverbose_index) :: moreverbose
      type (green_index) :: green
      type (poly_index) :: polygon
    end type
    type(index_info) :: ind

    type admitance_info
      logical :: if
      real(dp):: value = -0.3
    end type
    type(admitance_info) :: admitance

    logical :: method(3)

    ! point mass - method3d(1)=.true.
    ! potential  - method3d(2)=.true.
    ! cylinder   - method3d(3)=.true.
    logical :: method3d(3)
    logical :: method3d_compute_reference  = .false.
    real    :: method3d_refinment_distance = 0.1
    logical :: dryrun

    logical :: result_total=.false., result_component=.true.
contains
  ! ==============================================================================
  !> This routine collect command line arguments to one matrix depending on
  !! given switches and separators
  !!
  !! \date 2013.05.21
  !! \author Marcin Rajner
  ! ==============================================================================
  subroutine collect_args (dummy)
      use mod_utilities, only: ntokens, count_separator
      character(*) :: dummy
      character(455) :: dummy_aux, dummy_aux2
      integer :: i, j, n
      integer :: indeks_space, indeks_comma, indeks_at, indeks_colon

      allocate(cmd_line(ntokens(dummy)))
      do i=1, ntokens(dummy)
        indeks_space = index(dummy," ")
        cmd_line(i)%full= dummy(1:indeks_space-1)
        cmd_line(i)%switch=cmd_line(i)%full(1:2)
        allocate(cmd_line(i)%field (count_separator(cmd_line(i)%full,",") + 1))

        dummy_aux = cmd_line(i)%full(3:)
        do j=1,size(cmd_line(i)%field)
          indeks_comma=index(dummy_aux,",")
          if (indeks_comma.gt.0) then
            cmd_line(i)%field(j)%full=dummy_aux(1:indeks_comma-1)
          else
            cmd_line(i)%field(j)%full=dummy_aux
          endif

          allocate(cmd_line(i)%field(j)%subfield &
              (count_separator(cmd_line(i)%field(j)%full,":") + 1))
          dummy_aux2 = cmd_line(i)%field(j)%full
          do n = 1, count_separator(cmd_line(i)%field(j)%full,":")+1
            indeks_colon=index(dummy_aux2,":")
            if (indeks_colon.gt.0) then
              cmd_line(i)%field(j)%subfield(n)%name=dummy_aux2(1:indeks_colon-1)
            else
              cmd_line(i)%field(j)%subfield(n)%name=dummy_aux2
            endif
            dummy_aux2=dummy_aux2(indeks_colon+1:)
            indeks_at=index(cmd_line(i)%field(j)%subfield(n)%name,"@")
            if (indeks_at.gt.0) then
              cmd_line(i)%field(j)%subfield(n)%dataname = &
                  cmd_line(i)%field(j)%subfield(n)%name(indeks_at+1:)
              cmd_line(i)%field(j)%subfield(n)%name = &
                  cmd_line(i)%field(j)%subfield(n)%name(1:indeks_at-1)
            else
              cmd_line(i)%field(j)%subfield(n)%dataname = " "
            endif
          enddo
          dummy_aux=dummy_aux(indeks_comma+1:)
        enddo
        dummy= dummy(indeks_space+1:)
      enddo
  end subroutine

  ! ===============================================================================
  !> This subroutine removes unnecesary blank spaces from cmdline entry
  !!
  !! Marcin Rajner
  !! \date 2013-05-13
  !! 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
  !==============================================================================
  subroutine get_command_cleaned(dummy)
      character(*), intent(out) :: dummy
      character(355) :: a, b, arg
      integer :: i
      dummy=" "
      do i = 1, iargc()
        call get_command_argument(i,a)
        call get_command_argument(i+1,b)
        if (check_if_switch_or_minus(a)) then
          arg = trim(a)
        else
          arg=trim(arg)//trim(a)
        endif
        if(check_if_switch_or_minus(b).or.i.eq.iargc()) then
          if(trim(dummy).eq."") then
            dummy=trim(arg)
          else
            dummy=trim(dummy)//" "//trim(arg)
          endif
        endif
      enddo
  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

end module
```

9.11 grat/src/mod_green.f90 File Reference

Data Types

- module mod_green
- type mod_green::green_functions
- type mod_green::green_common_info

9.11.1 Detailed Description

Definition in file mod_green.f90.

9.12 mod_green.f90

```
!> \file
module mod_green
  use mod_constants, only: dp

  implicit none
  !----------------------------------------------------
  ! Greens function
  !----------------------------------------------------
  type green_functions
    character (len=255) :: name
    character (len=25) :: dataname
    integer, dimension(2) :: column
    character(10), dimension(2) :: columndataname
    real(dp), allocatable,dimension(:) :: distance
    real(dp), allocatable,dimension(:) :: data
  end type
  type(green_functions), allocatable, dimension(:) :: green

  real(dp), allocatable, dimension(:) :: result

  type green_common_info
    real(dp), allocatable, dimension(:) :: distance
    real(dp), allocatable, dimension(:) :: start
    real(dp), allocatable, dimension(:) :: stop
    real(dp), allocatable, dimension(:,:) :: data
    character (len=25), allocatable, dimension(:) :: dataname
    logical, allocatable, dimension(:) :: elastic
  end type
  type(green_common_info), allocatable, dimension(:) :: green_common

  integer :: gnc_looseness=1

contains
! ============================================================================
!> 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
  use mod_cmdline
  use mod_printing
  type (cmd_line_arg), optional  :: cmd_line_entry
  integer :: i, ii

  if (allocated(green)) then
    call print_warning("repeated")
    return
  endif

  if (method(3)) then
    if (present(cmd_line_entry)) then
      allocate (green(size(cmd_line_entry%field)+1))
    else
      allocate (green(1))
    endif
    ind%green%g3d=ubound(green,1)
    green(ind%green%g3d)%name="merriam"
    green(ind%green%g3d)%column=[1, 2]
    green(ind%green%g3d)%dataname="G3D"
    call read_green(green(ind%green%g3d))
  else
    allocate (green(size(cmd_line_entry%field)))
  endif

  if (present(cmd_line_entry)) then
    do i = 1, size(cmd_line_entry%field)

      if (.not.log%sparse) &
        write(log%unit, form%i2) trim(basename(trim(cmd_line_entry%field(i)%full)))

      green(i)%name = cmd_line_entry%field(i)%subfield(1)%name

      if (i.gt.1.and.cmd_line_entry%field(i)%subfield(1)%name.eq."") then
        green(i)%name = green(i-1)%name
      endif

      if (any(green%dataname.eq.cmd_line_entry%field(i)%subfield(1)%dataname )) then
        call print_warning("repeated dataname for Green")
        continue
      else
        green(i)%dataname = cmd_line_entry%field(i)%subfield(1)%dataname
      endif

      do ii=1, 2
        green(i)%column(ii) = green(i-1)%column(ii)
        green(i)%columndataname(ii) = green(i-1)%columndataname(ii)
        if(is_numeric(cmd_line_entry%field(i)%subfield(ii+1)%name ) ) then
          read(cmd_line_entry%field(i)%subfield(ii+1)%name, *) green(i)%column(ii)
          green(i)%columndataname(ii) = cmd_line_entry%field(i)%subfield(ii+1)%dataname
        endif

      enddo
      if (green(i)%dataname.eq."GNc") then
        if(is_numeric(cmd_line_entry%field(i)%subfield(2)%name)) then
          read(cmd_line_entry%field(i)%subfield(2)%name, *) gnc_looseness
          if (gnc_looseness.lt.1) then
            call print_warning("gnc_looseness < 1", error=.true.)
          endif
        endif
      endif

      call read_green(green(i))

    enddo
  endif

  ! check completness
  ! if ( &
  ! ! any(green%name.eq."/home/mrajner/src/grat/dat/merriam_green.dat" &
  ! ! .and. green%dataname.eq."GNdz" ) &
  ! ! .neqv. &
  ! any(green%name.eq."/home/mrajner/src/grat/dat/merriam_green.dat" &
  ! .and. green%dataname.eq."GNdz2" ) &
  ! ) call print_warning("-G: merriam@GNdz should go with merriam @GNdz2")
end subroutine

! ============================================================================
!> This subroutine read  green file
! ============================================================================
subroutine read_green (green, print)
  use mod_utilities, only: file_exists, skip_header, r2d, d2r
  use iso_fortran_env
  use mod_printing
  use mod_constants, only: earth, pi
  use mod_normalization, only: green_normalization

  integer :: lines, fileunit, io_status, i
  real (dp), allocatable, dimension(:) :: tmp
  type(green_functions) :: green
  logical, optional :: print

  ! change the paths accordingly
  if (.not.file_exists(green%name)       &
    .and. (.not. green%name.eq."merriam" &
    .and.   .not. green%name.eq."huang"   &
    .and.   .not. green%name.eq."rajner"  &
    ! this will be feature added for hydrosphere loading later...
  ! .and.   .not. green%name.eq."GB" &
  )) then
    green%name="merriam"
  endif

  select case (green%name)
  case ("merriam", "compute", "/home/mrajner/src/grat/dat/merriam_green.dat")
    green%name="/home/mrajner/src/grat/dat/merriam_green.dat"
    select case (green%dataname)
    case("GN")
      green%column=[1, 2]
    case("GNdt")
      green%column=[1, 3]
    case("GNdz")
      green%column=[1, 4]
    case("GNdz2")
      green%column=[1, 5]
    case("GE")
      green%column=[1, 6]
    case("GNc")
      green%column=[1, 2]
    case("G3D")
      green%column=[1, 2]
    case default
      call print_warning( &
        "green type not found", &
        more=trim(green%dataname), &
        error=.true.)
    endselect

  case ("huang", "/home/mrajner/src/grat/dat/huang_green.dat" )
```

```
    green%name="/home/mrajner/src/grat/dat/huang_green.dat"
    select case (green%dataname)
    case("GN")
      green%column=[1, 2]
    case("GNdt")
      green%column=[1, 3]
    case("GNdh")
      green%column=[1, 4]
    case("GNdz")
      green%column=[1, 5]
    case default
      call print_warning( &
        trim(green%dataname) //" not found in " &
        // trim(green%name), error=.true.)
    endselect

  case ("rajner", "/home/mrajner/src/grat/dat/rajner_green.dat")
    green%name="/home/mrajner/src/grat/dat/rajner_green.dat"
    select case (green%dataname)
    case("GN")
      green%column=[1, 2]
    case("GNdt")
      green%column=[1, 3]
    case("GNdh")
      green%column=[1, 4]
    case("GNdz")
      green%column=[1, 5]
    case default
      call print_warning( &
        trim(green%dataname) //" not found in " &
        // trim(green%name), error=.true.)
      call print_warning(green%dataname //"not found in " // green%name, &
        error=.true.)
    endselect
  endselect

  if(green%column(1).ne.0 .and. green%column(2).ne.0) then
    allocate(tmp(max(green%column(1), green%column(2))))
    lines = 0
    open (newunit =fileunit, file=green%name, action="read", status="old")
    do
      call skip_header(fileunit)
      read (fileunit, *, iostat = io_status) tmp
      if (io_status == iostat_end) exit
      lines = lines + 1
    enddo

    allocate (green%distance(lines))
    allocate (green%data(lines))
    rewind(fileunit)
    lines = 0
    do
      call skip_header(fileunit)
      lines = lines + 1
      read (fileunit, *, iostat = io_status) tmp
      if (io_status == iostat_end) then
        close(fileunit)
        exit
      endif
      green%distance(lines) = tmp(green%column(1))
      green%data(lines)     = tmp(green%column(2))
    enddo
    deallocate(tmp)
  endif

  ! file specific
  if (green%name.eq."/home/mrajner/src/grat/dat/merriam_green.dat") then
    select case(green%dataname)
    case("GNdz")
      green%data = green%data * 1.e-3
    endselect
  endif

  if (.not.present(print)) then
    if (.not.log%sparse) &
      write(log%unit, form%i3) &
      trim(basename(trim(green%name))), trim(green%dataname), &
      "columns:", green%column, &
      "lines:", size(green%distance)

      if (green%dataname.eq."GNc") then
      write(log%unit, form%i3) "gnc loosenes" , gnc_looseness
    endif
  endif

  if (green%columndataname(1).eq."R") then
    green%distance=(/ (r2d(green%distance(i)), i=1, size(green%distance)) /)
    write(log%unit, form_63) "conversion: radians --> to degrees"
  endif
  if (green%columndataname(2).eq."a2f") then
    green%data=green%data  / (earth%radius)*1e12 * earth%gravity%mean
    write(log%unit, form_63) "conversion: aplo --> to farrell"
  endif
  if (green%columndataname(2).eq."f2m") then
    green%data= &
      -green%data * green_normalization("f2m")
    write(log%unit, form_63) "conversion: farrell --> to merriam"
  endif
end subroutine

! =============================================================================
!> Unification:
! =============================================================================
subroutine green_unification ()
  use mod_utilities, only: size_ntimes_denser, spline_interpolation, d2r
  use mod_cmdline,   only: info, moreverbose, ind, method3d_compute_reference
  use mod_printing
  use mod_site, only: site
  use mod_aggf, only: aggf

  type(green_functions) :: tmpgreen
  integer :: i, iinfo, imin, imax, j, ii
  integer, allocatable, dimension(:):: which_green, tmp

  allocate (green_common(size(info)))
  allocate (which_green(size(info)))
  allocate (tmp(size(green)))

  do iinfo=1, size(info)

    if (info(iinfo)%distance%step.eq.0) then
      do i=1, size(green)
        tmp(i) = count(                                          &
          green(i)%distance.le.info(iinfo)%distance%stop       &
          .and.green(i)%distance.ge.info(iinfo)%distance%start &
          )
      enddo

      which_green(iinfo) = maxloc(tmp, 1)

      imin=minloc( &
        abs(green(which_green(iinfo))%distance - info(iinfo)%distance%start), 1)-1
      imax=minloc( &
        abs(green(which_green(iinfo))%distance - info(iinfo)%distance%stop), 1)+1

      if (imin.lt.1) imin = 1
      if (imax.gt.size(green(which_green(iinfo))%distance)) then
        imax = size(green(which_green(iinfo))%distance)
      endif

      if (info(iinfo)%distance%denser.ge.0) then
        allocate(tmpgreen%distance(                                   &
          size_ntimes_denser(imax-imin+1, info(iinfo)%distance%denser) &
          ))

        do ii = 1, imax-imin
          do j = 1, info(iinfo)%distance%denser
            tmpgreen%distance((ii-1)*info(iinfo)%distance%denser+j) = &
              green(which_green(iinfo))%distance(imin+ii-1)           &
              +(j-1)*(green(which_green(iinfo))%distance(imin+ii)     &
              -green(which_green(iinfo))%distance(imin+ii-1))         &
              /info(iinfo)%distance%denser
          enddo
        enddo

      else
        ! if @DD is negative make distance sparse
        allocate(tmpgreen%distance((imax-imin)/-info(iinfo)%distance%denser &
          +1+min(1,modulo(imax-imin,-info(iinfo)%distance%denser))))
        ii=0
        do j=1,imax-imin+1
          if (j.eq.imax-imin+1.or.modulo(j-1,info(iinfo)%distance%denser).eq.0) then
            ii=ii+1
            tmpgreen%distance(ii)=green(which_green(iinfo))%distance(j)
          endif
        enddo
      endif

      tmpgreen%distance(size(tmpgreen%distance)) = &
```

```
        green(which_green(iinfo))%distance(imax)

      imin = count(tmpgreen%distance.le.info(iinfo)%distance%start)
      imax = size(tmpgreen%distance) - &
        count(tmpgreen%distance.ge.info(iinfo)%distance%stop ) + 1

      allocate(green_common(iinfo)%distance(imax-imin+1))
      green_common(iinfo)%distance =        &
        tmpgreen%distance(imin:imax)
      green_common(iinfo)%distance(1) =     &
        (3/4.*info(iinfo)%distance%start+  &
        green_common(iinfo)%distance(2)/4)
      green_common(iinfo)%distance(size(green_common(iinfo)%distance)) =       &
        (3/4.*info(iinfo)%distance%stop+                                       &
        green_common(iinfo)%distance(size(green_common(iinfo)%distance)-1)/4)

      allocate(green_common(iinfo)%start(size(green_common(iinfo)%distance)))
      allocate(green_common(iinfo)%stop(size(green_common(iinfo)%distance)))

      green_common(iinfo)%start=(green_common(iinfo)%distance)

      do i =1, size(green_common(iinfo)%distance)

        green_common(iinfo)%start(i)=(green_common(iinfo)%distance(i) + &
          green_common(iinfo)%distance(i-1) ) / 2.

        green_common(iinfo)%stop(i)=(green_common(iinfo)%distance(i)  + &
          green_common(iinfo)%distance(i+1) ) / 2.

      enddo

      green_common(iinfo)%start(1)= info(iinfo)%distance%start
      green_common(iinfo)%stop(size(green_common(iinfo)%stop)) = &
        info(iinfo)%distance%stop
      deallocate(tmpgreen%distance)

      !@DS =/ 0
    else
      allocate(green_common(iinfo)%distance( &
        ceiling( &
        (info(iinfo)%distance%stop - info(iinfo)%distance%start) &
        /info(iinfo)%distance%step) &
        ))
      allocate(green_common(iinfo)%start(size(green_common(iinfo)%distance)))
      allocate(green_common(iinfo)%stop(size(green_common(iinfo)%distance)))

      green_common(iinfo)%start =                  &
        [                                          &
        (info(iinfo)%distance%start +              &
        (i-1)*info(iinfo)%distance%step,           &
        i=1, size(green_common(iinfo)%distance))  &
        ]

      green_common(iinfo)%stop = green_common(iinfo)%start(2:)
      green_common(iinfo)%stop(ubound(green_common(iinfo)%stop)) = &
        info(iinfo)%distance%stop
      green_common(iinfo)%distance = &
        (green_common(iinfo)%stop + green_common(iinfo)%start)/2
    endif

    allocate(green_common(iinfo)%data(size(green_common(iinfo)%distance), size(green)))
    allocate(green_common(iinfo)%dataname(size(green)))

    do i = 1,  size(green_common(iinfo)%data, 2)
      call  spline_interpolation(           &
        green(i)%distance,                  &
        green(i)%data,                      &
        size(green(i)%distance),            &
        green_common(iinfo)%distance,       &
        green_common(iinfo)%data(:, i),      &
        size(green_common(iinfo)%distance) &
        )
      where( &
          green_common(iinfo)%distance.gt.green(i)%distance(size(green(i)%distance)) &
          .or.green_common(iinfo)%distance.lt.green(i)%distance(1) &
          )
        green_common(iinfo)%data(:, i)=0
      end where

      green_common(iinfo)%dataname(i) = green(i)%dataname

      if(green_common(iinfo)%dataname(i) == "G3D") then
        if (method3d_compute_reference) then
```

```
          do ii=1,size(green_common(iinfo)%data(:,i))
            green_common(iinfo)%data(ii,i) =                  &
              aggf(                                           &
              psi    = d2r(green_common(iinfo)%distance(ii)), &
              dz     = info(iinfo)%height%step,               &
              zmin   = info(iinfo)%height%start,              &
              zmax   = info(iinfo)%height%stop,               &
              method = "standard"                             &
              )
          enddo
        endif
      endif

    enddo
  enddo

end subroutine


! =============================================================================
!> Perform convolution
!!
!! \date 2013-03-15
!! \author M. Rajner
! =============================================================================
subroutine convolve(site, date)
  use mod_constants
  use iso_fortran_env
  use mod_site, &
    only : site_info, local_pressure_distance
  use mod_cmdline
  use mod_utilities, &
    only: d2r, r2d, datanameunit, mmwater2pascal, countsubstring
  use mod_spherical
  use mod_data
  use mod_date, only : dateandmjd
  use mod_polygon
  use mod_printing
  use mod_normalization, only: green_normalization
  use mod_aggf, only: aggf
  use mod_atmosphere, only: &
    standard_pressure, standard_temperature, virtual_temperature
  use mod_3d

  type(site_info),  intent(in) :: site
  type(dateandmjd), intent(in), optional :: date

  integer  :: igreen, idist, iazimuth, nazimuth
  real(dp) :: azimuth, dazimuth
  real(dp) :: lat, lon, area, tot_area, tot_area_used
  real(dp) :: val(size(model)), old_val_sp, old_val_rsp
  integer  :: i, j, npoints, iheight, nheight
  integer(2) :: iok(size(polygon))

  real(dp) :: normalize, aux
  real(dp), allocatable, dimension(:) :: azimuths, &
    heights, pressures, temperatures
  logical :: header_p = .true.

  ! real(dp) :: h1,h2, v1,v2, p_int !temporary
  real(dp) :: rsp
  real(dp), dimension(:), allocatable :: result_partial

  logical :: first_reduction
  first_reduction=.true.


  if (transfer_sp%if) then
    if (ind%model%hp.eq.0) call print_warning("no @HP with -U", error=.true.)
    if (ind%model%h .eq.0) call print_warning("no @H  with -U", error=.true.)
  endif

  if(.not.allocated(green_common)) then
    call green_unification()
  endif

  val=0

  if (site%lp%if) then
    do i=1, size(site%lp%date)

      if(all(site%lp%date(i, 1:6).eq.date%date(1:6))) then
        val(ind%model%sp) = site%lp%data(i)
        exit
      endif

      val(ind%model%sp) = sqrt(-1.)
      if(i.eq.size(site%lp%date)) &
        call print_warning("date not found in @LP")
    enddo
  endif

  if (.not. allocated(result)) then
    if (any(green%dataname.eq."GE").and.inverted_barometer &
      .and. non_inverted_barometer) then
      allocate(result(size(green)+1))
    else
      allocate(result(size(green)))
    endif
  endif
  if(.not.allocated(result_partial)) allocate(result_partial(size(result)))

  npoints       = 0
  area          = 0
  tot_area      = 0
  tot_area_used = 0

  result = 0
  rsp    = 0

  do igreen = 1, size(green_common)
    do idist = 1, size(green_common(igreen)%distance)
      if (allocated(azimuths)) deallocate (azimuths)

      if (info(igreen)%azimuth%step.eq.0) then
        nazimuth =                                                     &
          (info(igreen)%azimuth%stop-info(igreen)%azimuth%start)/360 *         &
          max(int(360*sin(d2r(green_common(igreen)%distance(idist)))), 100) * &
          info(igreen)%azimuth%denser
        if (nazimuth.eq.0) nazimuth=1
        dazimuth= (info(igreen)%azimuth%stop-info(igreen)%azimuth%start)/nazimuth
      else
        dazimuth = info(igreen)%azimuth%step
        nazimuth= (info(igreen)%azimuth%stop-info(igreen)%azimuth%start)/dazimuth
      endif

      ! calculate area using spherical formulae
      area = spher_area(                         &
        d2r(green_common(igreen)%start(idist)), &
        d2r(green_common(igreen)%stop(idist)),  &
        d2r(dazimuth),                           &
        radius=earth%radius,                     &
        alternative_method=.true.)

      ! normalization according to Merriam (1992)
      normalize= 1e8 / &
        (green_normalization("m", psi = d2r(green_common(igreen)%distance(idist))))

      allocate(azimuths(nazimuth))
      azimuths = [(info(igreen)%azimuth%start + (i-1) * dazimuth, i= 1, nazimuth)]

      do iazimuth  = 1, nazimuth
        azimuth = azimuths(iazimuth)

        npoints = npoints + 1
        tot_area=tot_area+area

        ! get lat and lon of point
        call spher_trig &
          (d2r(site%lat), d2r(site%lon), &
          d2r(green_common(igreen)%distance(idist)), d2r(azimuth), lat, lon, domain=.true.)

        ! read polygons
        if (ind%polygon%e.ne.0 .or. ind%polygon%n.ne.0) then
          do i =1, size(polygon)
            if (polygon(i)%if) then
              call chkgon(r2d(lon), r2d(lat), polygon(i), iok(i))
            endif
          enddo
        endif

        ! get LS
        if (ind%model%ls.ne.0.and.inverted_barometer) then
          call get_value(                        &
            model(ind%model%ls),                  &
            lat    = r2d(lat),                    &
            lon    = r2d(lon),                    &
            val    = val(ind%model%ls),           &
            level  = 1,                           &
            method = info(igreen)%interpolation, &
            date   = date%date                  &
            )
        endif

        if (iok(1).eq.1 & .and. int(val(ind%model%ls)).eq.1) then
          tot_area_used = tot_area_used +area
        endif

        ! GE, GN, ...
        if (any([&
          ind%green%gn,   ind%green%ge, ind%green%gg, &
          ind%green%gndt, ind%green%gnc, ind%green%gegdt, &
          ind%green%g3d &
          ].ne.0) &
          ) then

          if ( &
            ind%model%sp.ne.0.and.(model(ind%model%sp)%if &
            .or.model(ind%model%sp)%if_constant_value) &
            ) then

            ! get SP
            if (.not.(site%lp%if                      &
              .and.green_common(igreen)%distance(idist) &
              .lt.local_pressure_distance)) then
              call get_value(                                            &
                model(ind%model%sp), r2d(lat), r2d(lon), val(ind%model%sp), &
                level=1,                                                    &
                method = info(igreen)%interpolation,                        &
                date=date%date)
            endif
            old_val_sp=val(ind%model%sp)

            if (.not.isnan(val(ind%model%sp))) then

              ! get RSP if given
              if (ind%model%rsp.ne.0) then
                call get_value(                                             &
                  model(ind%model%rsp), r2d(lat), r2d(lon), val(ind%model%rsp), &
                  level=1, method = info(igreen)%interpolation)
              endif
              old_val_rsp=val(ind%model%rsp)

              if(transfer_sp%if.and..not.all([ind%model%rsp, ind%model%hrsp].ne.0)) then
                call print_warning("@RSP or @HRSP with -U is missing", error=.true.)
              else
                call get_value( &
                  model(ind%model%hrsp), r2d(lat), r2d(lon), val(ind%model%hrsp), &
                  level=1, method = info(igreen)%interpolation)
              endif

              ! get T
              if (ind%model%t.ne.0 &
                .and.( &
                transfer_sp%if &
                .or.any(([ &
                ind%green%gndt, &
                ind%green%gegdt, &
                ind%green%gnc, &
                ind%green%g3d &
                ]).ne.0) &
                ) &
                ) then
                call get_value( &
                  model(ind%model%t), r2d(lat), r2d(lon), val(ind%model%t), &
                  level=1, method=info(igreen)%interpolation, date=date%date)
              endif

              ! get HP
              if (ind%model%hp.ne.0 &
                .and.( &
                transfer_sp%if &
                .or. ind%green%g3d.ne.0 &
                ) &
                ) then
                call get_value( &
                  model(ind%model%hp), r2d(lat), r2d(lon), val(ind%model%hp), &
                  level=1, method = info(igreen)%interpolation)
              endif

              ! get H
              if (ind%model%h.ne.0 &
                .and.(             &
                transfer_sp%if     &
                .or.any(([         &
                ind%green%gndt,    &
                ind%green%gndz,    &
                ind%green%gndz2,   &
                ind%green%gndh,    &
                ind%green%gnc,     &
                ind%green%g3d      &
                ]).ne.0)           &
                )                  &
                ) then

                if (optimize.and.green_common(igreen)%distance(idist).gt.3) then
                  val(ind%model%h)=val(ind%model%hp)
                else
                  call get_value( &
                    model(ind%model%h), r2d(lat), r2d(lon), val(ind%model%h), &
                    level=1, method = info(igreen)%interpolation)
                endif
              endif

              if (ind%model%sp.ne.0) then
                ! transfer SP if necessary on terrain
                if (transfer_sp%if &
                  .and.any([      &
                  ind%green%ge,    &
                  ind%green%gnc,   &
                  ind%green%g3d,   &
                  ind%green%gegdt, &
                  ind%green%gg     &
                  ].ne.0)          &
                  ) then

                  val(ind%model%sp) = standard_pressure(           &
                    height                   = val(ind%model%h),   &
                    h_zero                   = val(ind%model%hp),  &
                    p_zero                   = old_val_sp,         &
                    method                   = transfer_sp%method, &
                    temperature              = val(ind%model%t),   &
                    use_standard_temperature = ind%model%t.eq.0,   &
                    nan_as_zero              = .false.)

                  if(all([ind%model%rsp, ind%model%hrsp].ne.0)) then
                    val(ind%model%rsp) = standard_pressure(           &
                      height                   = val(ind%model%h),    &
                      h_zero                   = val(ind%model%hrsp), &
                      p_zero                   = old_val_rsp,         &
                      method                   = transfer_sp%method,  &
                      temperature              = val(ind%model%t),    &
                      use_standard_temperature = ind%model%t.eq.0,    &
                      nan_as_zero              = .false.)
                  endif
                endif

                if (ind%model%rsp.ne.0) then
                  if (.not.                                                               &
                    (site%lp%if                                                           &
                    .and.green_common(igreen)%distance(idist).lt.local_pressure_distance &
                    .and..not.first_reduction                                             &
                    )                                                                     &
                    ) then

                    val(ind%model%sp) = val(ind%model%sp) - val(ind%model%rsp)

                    if (first_reduction) first_reduction=.false.

                  endif
                endif

                ! if the cell is not over sea and inverted barometer assumption was not set
                ! and is not excluded by polygon
                if ((ind%polygon%e.ne.0.and.iok(ind%polygon%e).ne.0).or.(ind%polygon%e.eq.0)) then
                  !IB or NIB
                  if (.not.(ind%model%ls.ne.0.and.inverted_barometer.and.int(val(ind%model%ls)).eq.0)) then
```

```
                    ! GE
                    if (ind%green%ge.ne.0) then
                      result(ind%green%ge) = result(ind%green%ge)        &
                        + val(ind%model%sp)                              &
                        * green_common(igreen)%data(idist, ind%green%ge) &
                        * area * normalize
                    endif

                    ! GEGdt pressure part from Guo 2004
                    if (ind%green%gegdt.ne.0) then
                      result(ind%green%gegdt) = result(ind%green%gegdt) +   &
                        val(ind%model%sp) *                                 &
                        val(ind%model%t) * 1e-4 *                           &
                        green_common(igreen)%data(idist, ind%green%gegdt) * &
                        area * normalize
                    endif

                    ! GG
                    if (ind%green%gg.ne.0) then
                      aux = mmwater2pascal(val(ind%model%sp), inverted=.true.)
&
                        * area/ (d2r(green_common(igreen)%distance(idist)) *
&
                        earth%radius*1e18)

                      result(ind%green%gg) = result(ind%green%gg) +      &
                        green_common(igreen)%data(idist, ind%green%gg) * &
                        aux * 1e8 ! m s-2 -> microGal
                    endif
                  endif

                  ! ! GE NIB if both IB and NIB wanted
                  if (inverted_barometer.and.non_inverted_barometer) then
                    if (ind%green%ge.ne.0) then
                      result(ubound(result)) = result(ubound(result)) +
&
                        val(ind%model%sp) *                              &
                        green_common(igreen)%data(idist, ind%green%ge) * &
                        area * normalize
                    endif
                  endif
                endif

                if (                                                     &
                  (ind%polygon%n.ne.0.and.iok(ind%polygon%n).ne.0)     &
                  .or.(ind%polygon%n.eq.0)                             &
                  ) then

                  !3D
                  if (method(3)) then

                    ! if distance%stop_3d was set restrict computation of 3D to
 this distance
                    if(green_common(igreen)%distance(idist).lt.info(igreen)
%distance%stop_3d) then

                      if (ind%model%rsp.eq.0) then
                        call print_warning("3D but no RSP", error=.true.)
                      endif
                      if (ind%model%hrsp.eq.0) then
                        call print_warning("3D but no HRSP", error=.true.)
                      endif

                      if (allocated(heights))      deallocate(heights)
                      if (allocated(pressures))    deallocate(pressures)
                      if (allocated(temperatures)) deallocate(temperatures)

                      if( &
                        info(igreen)%height%stop <= max(info(igreen)%height
%start,val(ind%model%h)) &
                        ) then
                        cycle
                      endif

                      nheight=                                            &
                        ceiling((info(igreen)%height%stop                 &
                        -max(info(igreen)%height%start,val(ind%model%h))) &
                        /info(igreen)%height%step)

                      allocate(heights(nheight))
                      allocate(pressures(nheight))
                      allocate(temperatures(nheight))

                      do iheight=1, nheight
                        heights(iheight)=max(info(igreen)%height%start, val(ind
%model%h)) &
                          +(iheight-0.5)*info(igreen)%height%step
                      enddo

                      if (.not.allocated(level%height))      allocate (level%height(size(level%level)))
                      if (.not.allocated(level%temperature)) allocate (level%temperature(size(level%level)))
                      if (.not.allocated(level%humidity))    allocate (level%humidity(size(level%level)))

                      do i=1,size(level%level)
                        call get_value(                                           &
                          model(ind%model%gp), r2d(lat), r2d(lon), level%height(i),                    &
                          level=level%level(i), method = info(igreen)%interpolation, date=date%date)

                        if (ind%model%vt.ne.0) then
                          call get_value(                           &
                            model(ind%model%vt), r2d(lat), r2d(lon), &
                            val    = level%temperature(i),           &
                            level  = level%level(i),                 &
                            method = info(igreen)%interpolation,     &
                            date   = date%date                       &
                            )
                        endif

                        if (ind%model%vsh.ne.0) then
                          call get_value(                             &
                            model(ind%model%vsh), r2d(lat), r2d(lon), &
                            val    = level%humidity(i),               &
                            level  = level%level(i),                  &
                            method = info(igreen)%interpolation,      &
                            date   = date%date                        &
                            )

                          if (.not.isnan(level%humidity(i))) then
                            level%temperature(i)= &
                              virtual_temperature(level%temperature(i),level%humidity(i))
                          endif

                        endif
                      enddo

                      i=1
                      do while(level%height(i).lt.heights(1).and.i.ne.size(level%level))
                        i=i+1
                      end do

                      do iheight=1, nheight

                        if (iheight.eq.1) then
                          ! h1=val(ind%model%h)
                          ! v1=val(ind%model%sp)+val(ind%model%rsp)
                          ! h2=level%height(i)
                          ! v2=1.e2*dble(level%level(i))

                          temperatures(iheight)= &
                            level%temperature(i)-6.5e-3*(val(ind%model%h)-val(ind%model%hp))

                          if (.not.isnan(level%humidity(1))) then
                            val(ind%model%t) = &
                              virtual_temperature(val(ind%model%t), level%humidity(1))
                          endif

                          pressures(iheight) = standard_pressure(        &
                            heights(iheight),                            &
                            p_zero=val(ind%model%sp)+val(ind%model%rsp), &
                            h_zero=val(ind%model%h), &
                            method="standard",                           &
                            use_standard_temperature=.true.,             &
                            temperature=val(ind%model%t)                 &
                            )

                        else
                          do while(level%height(i+1).lt.heights(iheight).and. i.ne.size(level%level))
                            i=i+1
                          end do

                          ! temperature linear interpolation
                          if(i.lt.size(level%level)) then
                            temperatures(iheight)= &
```

```
                              level%temperature(i) &
                              + (level%temperature(i+1)-level%temperature(i)) &
                              /(level%height(i+1)-level%height(i))*(heights(iheight)-level%height(i))
                          else
                            temperatures(iheight)= &
                              level%temperature(i)
                          endif

                          if(heights(iheight-1).lt.level%height(i).and.(heights(iheight).gt.level%height(i))) then
                            ! h1=level%height(i)
                            ! v1=1.e2*dble(level%level(i))
                            ! h2=level%height(i+1)
                            ! v2=1.e2*dble(level%level(i+1))

                            pressures(iheight) =                                                            &
                              standard_pressure(                                                            &
                              height                   = heights(iheight),                                  &
                              p_zero                   = 1.e2*dble(level%level(i)), &
                              h_zero                   = level%height(i),                                   &
                              method                   = "standard",                                        &
                              use_standard_temperature = .true.,                                            &
                              temperature              = temperatures(iheight),                             &
                              nan_as_zero              = .true.                                             &
                              )

                          else

                            pressures(iheight)=                                                            &
                              standard_pressure(                                                           &
                              height                   = heights(iheight),                                 &
                              p_zero                   = pressures(iheight-1),                             &
                              h_zero                   = heights(iheight-1),                               &
                              method                   = "standard",                                       &
                              use_standard_temperature = .true.,                                           &
                              temperature              = temperatures(iheight),                            &
                              nan_as_zero              = .true.                                            &
                              )
                          endif
                        endif

                        ! if (i.lt.size(level%level)) then
                        ! p_int=exp(dlog(v1) +(dlog(v2)-dlog(v1))*(heights(iheight)-h1)/(h2-h1))
                        ! if (p_int.gt.1e29)   p_int=0
                        ! pressures(iheight)=p_int
                        ! endif

                        if (method3d(1).or.green_common(igreen)%distance(idist).gt.method3d_refinment_distance) then
                          result(ind%green%g3d) = result(ind%green%g3d) &
                            + geometry(psi=d2r(green_common(igreen)%distance(idist)), h=site%height, z=heights(iheight)) &
                            * pressures(iheight)/(temperatures(iheight))   &
                            * area * info(igreen)%height%step &
                            *(-gravity%constant)*1e8/r_air

                        else if (method3d(2)) then
                          result(ind%green%g3d) =                                                   &
                            result(ind%green%g3d)                                                   &
                            + potential(                                                            &
                            psi1=d2r(green_common(igreen)%start(idist)),                            &
                            psi2=d2r(green_common(igreen)%stop(idist)),                             &
                            dazimuth=d2r(dazimuth),                                                 &
                            h=site%height,                                                          &
                            z1= heights(iheight)-info(igreen)%height%step/2,                        &
                            z2= heights(iheight)+info(igreen)%height%step/2                         &
                            )                                                                       &
                            * pressures(iheight)/(temperatures(iheight))     &
                            *(-gravity%constant)*1e8/r_air
                          if (isnan(result(ind%green%g3d)))  then
                            ! small distances can cause numerical problems
                            result(ind%green%g3d)=0
                          endif

                        else if (method3d(3)) then
                          result(ind%green%g3d) =                            &
                            result(ind%green%g3d)                            &
                            + cylinder(                                      &
                            psi1=d2r(green_common(igreen)%start(idist)),     &
                            psi2=d2r(green_common(igreen)%stop(idist)),      &
                            dazimuth=d2r(dazimuth),                          &
                            h=site%height,                                   &
                            z1= heights(iheight)-info(igreen)%height%step/2, &
                            z2= heights(iheight)+info(igreen)%height%step/2  &
                            )                                                &
                            * pressures(iheight)/(temperatures(iheight))     &
                            *(-gravity%constant)*1e8/r_air
                        endif


                      enddo
                    endif
                  endif

                  !C before GN GNdt etc because it needs SP on H not on site
                  if(ind%green%gnc.ne.0) then
                    if (            &
                      any([        &
                      ind%model%sp, &
                      ind%model%hp, &
                      ind%model%h,  &
                      ind%model%t   &
                      ].eq.0))      &
                      call print_warning("with @GNc you need to give @T @HP @H"
, error=.true.)

                    result(ind%green%gnc) = result(ind%green%gnc)        &
                      + val(ind%model%sp)                                &
                      * aggf(                                            &
                      d2r(green_common(igreen)%distance(idist)),         &
                      zmin=val(ind%model%h),                             &
                      t_zero=val(ind%model%t),                           &
                      h=site%height,                                     &
                      dz= gnc_looseness*10.                              &
                      *merge(10._dp,                                     &
                      merge(0.1_dp,1._dp,                                &
                      green_common(igreen)%distance(idist).le.1e-5_dp ), &
                      green_common(igreen)%distance(idist).ge.1e-2_dp ), &
                      method="standard",                                 &
                      predefined=.true.)                                 &
                      * area * normalize

                    if (.not.quiet) then
                      open(unit=output_unit, carriagecontrol=’fortran’)
                      call progress(
      &
                        100*igreen*idist
    &
                        /(size(green_common(igreen)%distance)*size(green_common
)), &
                        every=1 &
                        )
                    endif
                  endif

                  ! transfer SP if necessary on site level
                  if (transfer_sp%if &
                    .and.any([ &
                    ind%green%gn, &
                    ind%green%gndt, &
                    ind%green%gndz, &
                    ind%green%gndz2, &
                    ind%green%gndh &
                    ].ne.0) &
                    ) then
                    val(ind%model%sp) = standard_pressure( &
                      height=site%height,                  &
                      h_zero=val(ind%model%hp),            &
                      p_zero=old_val_sp,                   &
                      method=transfer_sp%method,           &
                      temperature=val(ind%model%t),        &
                      use_standard_temperature             &
                      = ind%model%t.eq.0,                  &
                      nan_as_zero=.false.)

                    if(all([ind%model%rsp, ind%model%hrsp].ne.0)) then
                      val(ind%model%rsp) = standard_pressure(          &
                        height=site%height,                            &
                        h_zero=val(ind%model%hrsp),                    &
                        p_zero=old_val_rsp,                            &
                        method=transfer_sp%method,                     &
                        temperature=val(ind%model%t),                  &
                        use_standard_temperature                       &
                        = ind%model%t.eq.0,                            &
                        nan_as_zero=.false.)
                    endif
                    if(ind%model%rsp.ne.0) val(ind%model%sp) = val(ind%model%sp) - val(ind%model%rsp)
                  endif

                  ! GN
                  if (ind%green%gn.ne.0) then
                    result_partial(ind%green%gn) = &
                      val(ind%model%sp) *                              &
                      green_common(igreen)%data(idist, ind%green%gn) * &
                      area * normalize
                    result(ind%green%gn) = &
                      result(ind%green%gn) + result_partial(ind%green%gn)
                  endif

                  ! GNdt
                  if (ind%green%gndt.ne.0) then
                    if (any(                                                &
                      [ind%model%sp, ind%model%t, ind%model%rsp           &
                      ].eq.0)) &
                      call print_warning("not enough data model for GNdt", &
                      error=.true.)
                    result_partial(ind%green%gndt) =       &
                      val(ind%model%sp)                                    &
                      * green_common(igreen)%data(idist, ind%green%gndt)   &
                      * (val(ind%model%t)-atmosphere%temperature%standard) &
                      *  area * normalize
                    result(ind%green%gndt) = result(ind%green%gndt) +       &
                      result_partial(ind%green%gndt)
                  endif

                  ! GNdh
                  if (ind%green%gndh.ne.0) then
                    if (any(                                                &
                      [ &
                      ind%model%sp, ind%model%h, ind%model%rsp           &
                      ].eq.0)) &
                      call print_warning("not enough data model for GNdh", &
                      error=.true.)
                    result_partial(ind%green%gndh) = &
                      val(ind%model%sp)                                    &
                      * green_common(igreen)%data(idist, ind%green%gndh)   &
                      * (val(ind%model%h)-site%height) &
                      *  area * normalize
                    result(ind%green%gndh) = result(ind%green%gndh) +       &
                      result_partial(ind%green%gndh)
                  endif

                  ! GNdz
                  if (ind%green%gndz.ne.0) then
                    if (any(                                                &
                      [ &
                      ind%model%sp, ind%model%h, ind%model%rsp           &
                      ].eq.0)) &
                      call print_warning("not enough data model for GNdz", &
                      error=.true.)
                    result_partial(ind%green%gndz) =  +       &
                      val(ind%model%sp)                                    &
                      * green_common(igreen)%data(idist, ind%green%gndz)   &
                      * (val(ind%model%h)-site%height) &
                      *  area * normalize
                    result(ind%green%gndz) = result(ind%green%gndz) +       &
                      result_partial(ind%green%gndz)
                  endif


                  ! GNdz2
                  if (ind%green%gndz2.ne.0) then
                    if (any(                                                &
                      [                                                     &
                      ind%model%sp, ind%model%h, ind%model%rsp             &
                      ].eq.0))                                              &
                      call print_warning("not enough data model for GNdz2", &
                      error=.true.)

                    result_partial(ind%green%gndz2) =
       &
                      val(ind%model%sp) &
                      * green_common(igreen)%data(idist, ind%green%gndz2) &
                      * ( (val(ind%model%h)-site%height) &
                      /(earth%radius * d2r(green_common(igreen)%distance(idist)))**2 &
                      *  area * normalize

                    result(ind%green%gndz2) = result(ind%green%gndz2) &
                      + result_partial(ind%green%gndz2)
                  endif

                  ! reference 2D for 3D method
                  if (ind%green%g3d.ne.0) then

                    if(green_common(igreen)%distance(idist).lt.info(igreen)%distance%stop_3d) then
                      if (info(igreen)%height%start.gt.1) then
                        rsp = rsp                                           &
                          + standard_pressure(                              &
                          height = info(igreen)%height%start,               &
                          h_zero = val(ind%model%hrsp),                     &
                          p_zero = val(ind%model%rsp),                      &
                          method = "standard")                              &
                          * green_common(igreen)%data(idist, ind%green%g3d) &
                          * area * normalize
                      else
                        rsp = rsp                                           &
                          + val(ind%model%rsp)                              &
                          * green_common(igreen)%data(idist, ind%green%g3d) &
                          * area * normalize
                      endif

                    else
                      result(ind%green%g3d) =         &
                        result(ind%green%g3d)         &
                        + sum(result_partial,         &
                        mask=(                        &
                        green%dataname.eq."GN"        &
                        .or.green%dataname.eq."GNdt"  &
                        .or.green%dataname.eq."GNdz"  &
                        .or.green%dataname.eq."GNdz2" &
                        .or.green%dataname.eq."GNdh"  &
                        ))
                    endif

                  endif

                endif
              endif
            else
              result=sqrt(-1.)
            endif
          elseif(ind%model%ewt.eq.0) then
            call print_warning("@SP is required with -M2D -G", error=.true.)
          endif
        endif

        ! surface loads from EWT
        if ( &
          ind%green%gr.ne.0 &
          .or.ind%green%ghn.ne.0 &
          .or.ind%green%ghe.ne.0 &
          ) then
          if ((ind%polygon%e.ne.0.and.iok(ind%polygon%e).ne.0).or.(ind%polygon%e.eq.0)) then
            if (.not.(ind%model%ls.ne.0.and.inverted_barometer.and.int(val(ind%model%ls)).eq.0)) then
              call get_value( &
                model(ind%model%ewt), r2d(lat), r2d(lon), val(ind%model%ewt), &
                level=1, method = info(igreen)%interpolation, date=date%date)
              aux = (val(ind%model%ewt))  * &
                area/d2r(green_common(igreen)%distance(idist)) * &
                1./earth%radius/1e12* 1e3 ! m -> mm
              if (isnan(aux)) aux = 0
              if (ind%green%gr.ne.0) then
```

```
                result(ind%green%gr) = result(ind%green%gr) +       &
                  green_common(igreen)%data(idist, ind%green%gr) &
                  * aux

                if (ind%green%ghn.ne.0) then
                  result(ind%green%ghn) = result(ind%green%ghn) +      &
                    green_common(igreen)%data(idist, ind%green%ghn) * &
                    aux * (-cos(d2r(azimuth)))
                endif
                if (ind%green%ghe.ne.0) then
                  result(ind%green%ghe) = result(ind%green%ghe) +      &
                    green_common(igreen)%data(idist, ind%green%ghe) * &
                    aux * (-sin(d2r(azimuth)))
                endif
              endif
            endif
          endif
        endif

        ! moreverbose point: -L@p
        if(ind%moreverbose%p.ne.0) then
          if (header_p.and. output%header) then
            if(size(green_common).gt.1) &
              write(moreverbose(ind%moreverbose%p)%unit, "(a2, x$)") "i"

            write(moreverbose(ind%moreverbose%p)%unit, &
              '(a8, 8a13, $)')                         &
              "name", "lat", "lon",                  &
              "distance", "azimuth",                 &
              "lat", "lon",                          &
              "area", "totarea"

            if (result_component) then
              write(moreverbose(ind%moreverbose%p)%unit, &
                '(a13, $)')                &
                (trim(green(i)%dataname), &
                i=lbound(green, 1),       &
                ubound(green, 1)          &
                )
            endif

            if (result_total) then
              if (method(2)) then
                write(moreverbose(ind%moreverbose%p)%unit, &
                  '(a13, $)') "G2D_t"
              endif
              if (method(3)) then
                write(moreverbose(ind%moreverbose%p)%unit, &
                  '(a13, $)') "G3D_t"
              endif
            endif

            if (.not.moreverbose(ind%moreverbose%p)%sparse) then
              write(moreverbose(ind%moreverbose%p)%unit,
&
                '(<size(model)>a12)', advance='no' )                         &
                (trim(model(i)%dataname), i=lbound(model, 1), ubound(model, 1))
            endif

            if (size(iok).gt.0) then
              write(moreverbose(ind%moreverbose%p)%unit, &
                '(<size(iok)>(a3, i1))'),               &
                ("ok", i, i =1, ubound(iok, 1))
            else
              write(moreverbose(ind%moreverbose%p)%unit, *)
            endif
            header_p=.false.
          endif
          if (                                              &
            .not.moreverbose(ind%moreverbose%p)%sparse    &
            .or.                                          &
            (moreverbose(ind%moreverbose%p)%sparse        &
            .and.(azimuth==azimuths(ubound(azimuths, 1))) &
            )                                             &
            ) then

            if(size(green_common).gt.1) &
              write(moreverbose(ind%moreverbose%p)%unit, "(i2, x$)") igreen

            write(moreverbose(ind%moreverbose%p)%unit,        &
              '(a8, 6' // output%form //',2 en13.3, $)'),       &
              site%name, site%lat, site%lon,                 &
              green_common(igreen)%distance(idist), azimuth, &
              r2d(lat), r2d(lon), area, tot_area

            if (result_component)                          &
              write(moreverbose(ind%moreverbose%p)%unit, &
              '(' // output%form //'$)'),                &
              (result(i), i =1, size(result))

            if (result_total) then
              if (method(2)) then
                write(moreverbose(ind%moreverbose%p)%unit, &
                  '(' // output%form //'$)'), &
                  sum(result, &
                  mask=( &
                  green%dataname.eq."GN" &
                  .or.green%dataname.eq."GE" &
                  .or.green%dataname.eq."GNdt" &
                  .or.green%dataname.eq."GNdz" &
                  .or.green%dataname.eq."GNdz2" &
                  .or.green%dataname.eq."GNdh" &
                  ))
              endif
              if (method(3)) then
                write(moreverbose(ind%moreverbose%p)%unit, &
                  '(' // output%form //'$)'), &
                  sum(result, &
                  mask=( &
                  green%dataname.eq."G3D" &
                  .or.green%dataname.eq."GE" &
                  ))
              endif

            endif
            if (.not.moreverbose(ind%moreverbose%p)%sparse) then
              do i=1, size(val)
                call get_value(                          &
                  model(i), r2d(lat), r2d(lon), val(i), &
                  level=1,                              &
                  method = info(igreen)%interpolation,  &
                  date=date%date)
              enddo
              write(moreverbose(ind%moreverbose%p)%unit, &
                '(<size(model)>en12.2, $)') val
            endif
            if (size(iok).gt.0) then
              write(moreverbose(ind%moreverbose%p)%unit, &
                '(<size(iok)>(i4))'), iok
            else
              write(moreverbose(ind%moreverbose%p)%unit, * )
            endif
          endif
        endif

        ! moreverbose auxilary to draw: -L@a
        if(ind%moreverbose%a.ne.0) then
          call printmoreverbose(
         &
            d2r(site%lat), d2r(site%lon), d2r(azimuth), d2r(dazimuth), &
            d2r(green_common(igreen)%start(idist)),                     &
            d2r(green_common(igreen)%stop(idist))                      &
            )
        endif
      enddo
    enddo
  enddo


  if (ind%green%g3d.ne.0) &
    result(ind%green%g3d)=result(ind%green%g3d) - rsp

  ! results to output
  if (result_component) write (output%unit, "(" // output%form // '$)') result
  if (result_total) then
    if (method(2)) then
      write(output%unit, &
        '(' // output%form //'$)'), &
        sum(result, &
        mask=( &
        green%dataname.eq."GN" &
        .or.green%dataname.eq."GE" &
        .or.green%dataname.eq."GNdt" &
        .or.green%dataname.eq."GNdz" &
        .or.green%dataname.eq."GNdz2" &
        .or.green%dataname.eq."GNdh" &
        ))
    endif
    if (method(3)) then
      write(output%unit, &
        '(' // output%form //'$)'), &
        sum(result, &
        mask=( &
        green%dataname.eq."G3D" &
```

```
        .or.green%dataname.eq."GE" &
        ))
    endif
  endif

  ! summary: -L@s
  if (ind%moreverbose%s.ne.0) then
    if (output%header) write(moreverbose(ind%moreverbose%s)%unit, '(2a8, 3a12)'
 ) &
      "station", "npoints", "area", "area/R2", "t_area_used"
    write(moreverbose(ind%moreverbose%s)%unit, '(a8, i8, 3en12.2)') &
      site%name, npoints, tot_area, tot_area/earth%radius**2, tot_area_used
  endif

  ! green values : -L@g
  if(ind%moreverbose%g.ne.0) then
    do i = 1, size(green_common)
      if (output%header) &
        write(moreverbose(ind%moreverbose%g)%unit, '(a3,100a14)') &
        "nr", "distance", "start", "stop", "data", "di(j)-di(j-1)"
      do j=1,size(green_common(i)%distance)
        write(moreverbose(ind%moreverbose%g)%unit, '(i3,f14.6, 100f14.7)'), &
          j, green_common(i)%distance(j), &
          green_common(i)%start(j), &
          green_common(i)%stop(j), &
          green_common(i)%data(j,:), &
          green_common(i)%distance(j)-green_common(i)%distance(j-1)
      enddo
    enddo
  endif
end subroutine

! ==============================================================================
!> returns lat and lon of spherical trapezoid
!! \date 2013.07.03
!! \author Marcin Rajner
! ==============================================================================
subroutine printmoreverbose (latin, lonin, azimuth, azstep,
 distancestart, distancestop)
  use mod_spherical, only : spher_trig
  use mod_cmdline,   only : moreverbose, ind
  use mod_utilities, only : r2d

  real(dp), intent(in) :: azimuth, azstep, latin, lonin
  real(dp) ::  lat, lon, distancestart, distancestop

  call spher_trig(latin, lonin, distancestart, azimuth - azstep/2, lat, lon)
  write(moreverbose(ind%moreverbose%a)%unit, '(8f12.6)'), r2d(lat), r2d(lon)
  call spher_trig(latin, lonin, distancestop, azimuth - azstep/2, lat, lon)
  write(moreverbose(ind%moreverbose%a)%unit, '(8f12.6)'), r2d(lat), r2d(lon)
  call spher_trig(latin, lonin, distancestop, azimuth + azstep/2, lat, lon)
  write(moreverbose(ind%moreverbose%a)%unit, '(8f12.6)'), r2d(lat), r2d(lon)
  call spher_trig(latin, lonin, distancestart, azimuth + azstep/2, lat, lon)
  write(moreverbose(ind%moreverbose%a)%unit, '(8f12.6)'), r2d(lat), r2d(lon)
  write(moreverbose(ind%moreverbose%a)%unit, '(">")')
end subroutine

! ==============================================================================
!! \date 2013-07-02
!! \author M. Rajner
!! \warning input spherical distance in radian
!!
!! method:
!!   default see equation in Rajnerdr
!!   spotl   see \cite spotl manual
!!   olssson see \cite olsson2009
!!
 ==============================================================================
function green_newtonian (psi, h, z, method)
  use mod_constants, only: earth, gravity
  use mod_normalization, only: green_normalization
  real(dp) :: green_newtonian
  real(dp), intent (in) :: psi
  real(dp), intent (in), optional :: h
  real(dp), intent (in), optional :: z
  character(*), optional :: method
  real(dp) :: h_, z_, eps, t
  if (present(h)) then
    h_=h
  else
    h_=0.
  endif
  if (present(z)) then
    z_=z
  else
    z_=0.
  endif
  if (present(method) &
    .and. (method.eq."spotl" .or. method.eq."olsson")) then
    if(method.eq."spotl") then
      eps = h_/ earth%radius
      green_newtonian =                                       &
        1. /earth%radius**2                                   &
        *(eps + 2. * (sin(psi/2.))**2 )                       &
        /((4.*(1.+eps)* (sin(psi/2.))**2 + eps**2)**(3./2.)) &
        * gravity%constant                                    &
        * green_normalization("f",psi=psi)
      return
    else if (method.eq."olsson") then
      t = earth%radius/(earth%radius +h_)
      green_newtonian =                       &
        1 / earth%radius**2 * t**2 *          &
        (1. - t * cos(psi) ) /               &
        ( (1-2*t*cos(psi) +t**2 )**(3./2.) ) &
        * gravity%constant                    &
        * green_normalization("f",psi=psi)
      return
    endif
  else
    green_newtonian =                                                   &
      ((earth%radius + h_) - (earth%radius + z_) * cos(psi))         &
      / ((earth%radius + h_)**2 + (earth%radius + z_)**2             &
      -2*(earth%radius + h_)*(earth%radius + z_)*cos(psi))**(3./2.)

    green_newtonian = green_newtonian &
      * gravity%constant / earth%gravity%mean  * green_normalization("m", psi=psi)
    return
  endif
end function
end module
```

9.13 grat/src/mod_normalization.f90 File Reference

Data Types

- module mod_normalization

9.13.1 Detailed Description

Definition in file mod_normalization.f90.

9.14 mod_normalization.f90

```
! ==============================================================================
!> \file
! ==============================================================================
module mod_normalization
  implicit none

contains
! ==============================================================================
! ==============================================================================
function green_normalization(method, psi)
  use mod_constants, only: pi, earth, gravity, dp
  use mod_utilities, only: d2r
  real(dp):: green_normalization
  character(*) :: method
  real(dp), optional :: psi

  if (method.eq."f2m") then
    green_normalization = &
        1e-3 &
        / earth%gravity%mean  * earth%radius * 2 * pi * (1.- cos(d2r(1._dp)))
  else if (method.eq."m") then ! merriam normalization
    green_normalization =  &
        psi * 1e15 * earth%radius**2 * 2 * pi * (1.- cos(d2r(1._dp)))
  else if (method.eq."f") then ! farrell normalization
    green_normalization =  &
```

```
        psi * 1e18 * earth%radius
  endif
end function

end module
```

9.15 grat/src/mod_polygon.f90 File Reference

Some routines to deal with inclusion or exclusion of polygons.

Data Types

- module mod_polygon
- type mod_polygon::polygon_data
- type mod_polygon::polygon_info

9.15.1 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 by command line like in **?**

Definition in file mod_polygon.f90.

9.16 mod_polygon.f90

```
! ===============================================================================
!> \file
!! Some routines to deal with inclusion or exclusion of polygons
!!
!! \author M.Rajner
!! \date 2012-12-20
!! \date 2013-03-19
!!    added overriding of poly use by command line like in \cite spotl
! ===============================================================================
module mod_polygon
  use mod_constants, only : dp

  implicit none
  !-----------------------------------------------------
  ! polygons
  !-----------------------------------------------------
  type polygon_data
    logical :: use
    real(dp), allocatable , dimension (:,:) :: coords
  end type

  type polygon_info
    integer :: unit
    character(:), allocatable  :: name
    character(len=25) :: dataname
    type(polygon_data), dimension (:), allocatable :: polygon
    logical :: if
    ! global setting (+|-) which override this in polygon file
    character(1):: pm
  end type
```

```fortran
  type(polygon_info) , allocatable, dimension (:) :: polygon

contains
! ==============================================================================
!> This subroutine parse polygon information from command line entry
!!
!! \author M. Rajner
!! \date 2013.05.20
! ==============================================================================
subroutine parse_polygon (cmd_line_entry)
  use mod_printing
  use mod_cmdline
  use mod_utilities, only: file_exists
  type(cmd_line_arg),intent(in):: cmd_line_entry
  integer :: i

  if (allocated(polygon)) then
    call print_warning("repeated")
    return
  endif

  allocate(polygon(size(cmd_line_entry%field)))
  do i=1, size(cmd_line_entry%field)
  polygon(i)%name=cmd_line_entry%field(i)%subfield(1)%name
  if(i.gt.1.and.cmd_line_entry%field(i)%subfield(1)%name.eq."") then
    polygon(i)%name= polygon(i-1)%name
  endif
  polygon(i)%dataname=cmd_line_entry%field(i)%subfield(1)%dataname
  write(log%unit, form%i2), 'polygon file:' , polygon(i)%name
  if (file_exists((polygon(i)%name))) then
    polygon(i)%if=.true.
    if(cmd_line_entry%field(i)%subfield(2)%name.eq."+" &
      .or.cmd_line_entry%field(i)%subfield(2)%name.eq."-" ) then
      polygon(i)%pm = cmd_line_entry%field(i)%subfield(2)%name
      write(log%unit, form%i3) , "global override:", polygon(i)%pm
    endif
    call read_polygon(polygon(i))
  else
    stop 'file do not exist. Polygon file PROBLEM'
  endif
enddo

end subroutine
! ==============================================================================
!> Reads polygon data
!!
!! inspired by spotl \cite Agnew97
! ==============================================================================

subroutine read_polygon (polygon)

  use, intrinsic :: iso_fortran_env
  use mod_utilities, only: skip_header
  use mod_printing

  type(polygon_info) :: polygon
  integer :: i , j , number_of_polygons , nvertex
  character (1)  :: pm

  if (polygon%if) then
    ! polygon file
    open (newunit = polygon%unit , action="read", file=polygon%name )

    ! first get the number of polygon
    call skip_header(polygon%unit)
    read (polygon%unit , * ) number_of_polygons
    allocate (polygon%polygon(number_of_polygons))

    ! loop over all polygons in file
    do  i=1, number_of_polygons
      call skip_header(polygon%unit)
      read (polygon%unit, * ) nvertex
      allocate (polygon%polygon(i)%coords(nvertex, 2 ))
      call skip_header(polygon%unit)
      read (polygon%unit, * ) pm
      if (pm.eq."+") polygon%polygon(i)%use=.true.
      if (pm.eq."-") polygon%polygon(i)%use=.false.
      ! override file +|- with global given with command line
      if (polygon%pm.eq."+") polygon%polygon(i)%use=.true.
      if (polygon%pm.eq."-") polygon%polygon(i)%use=.false.
      do j = 1 , nvertex
        call skip_header(polygon%unit)
        ! lon lat , checks while reading
        read (polygon%unit, * ) polygon%polygon(i)%coords(j,1:2)
        if ( polygon%polygon(i)%coords(j,1).lt.-180. &
          .or.polygon%polygon(i)%coords(j,1).gt.360.  &
          .or.polygon%polygon(i)%coords(j,2).lt.-90.  &
          .or.polygon%polygon(i)%coords(j,2).gt. 90. ) then
          write (error_unit , form_63) "Somethings wrong with coords in polygon file"
          polygon%if=.false.
          return
          elseif( polygon%polygon(i)%coords(j,1).lt.0. ) then
          polygon%polygon(i)%coords(j,1) = polygon%polygon(i)%coords(j,1) + 360.
        endif
      enddo
    enddo
    close (polygon%unit)
    ! print summary to log file
    write (log%unit, form_63) "name:", trim(polygon%name)
    write (log%unit, form_63) "number of polygons:" , size (polygon%polygon)
    do i = 1 , size (polygon%polygon)
      if (polygon%pm.eq."+".or.polygon%pm.eq."-") write (log%unit, form_63) &
        "Usage overwritten with command line option", polygon%pm
      write (log%unit, form_63) "use [true/false]:" , &
        polygon%polygon(i)%use
      write (log%unit, form_63) "number of coords:" , &
        size (polygon%polygon(i)%coords(:,1))
    enddo
  endif

end subroutine

! ==============================================================================
!> Check if point is in closed polygon
!!
!! From spotl \cite Agnew97
!! adopted to \c 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 \cite Agnew96
!! \author adopted by Marcin Rajner
!! \date 2013-03-04
!!
!! The ilustration explain exclusion idea\n
!! \image latex /home/mrajner/src/grat/doc/figures/polygon_ilustration.pdf "capt" width=\textwidth
!! \image html /home/mrajner/src/grat/doc/figures/polygon_ilustration.png
! ==============================================================================
subroutine chkgon (rlong , rlat , polygon , iok)
  real(dp),intent (in) :: rlong, rlat
  integer :: i, ianyok
  integer(2) , intent (out) :: iok
  real(dp) :: rlong2
  type(polygon_info) , intent (in) :: polygon

  !  ! Check first if we need to use this soubroutine
  if (size(polygon%polygon).eq.0) then
    iok=0
    return
  endif

  if(rlong.gt.180) rlong2 = rlong - 360.
  ! loop over polygons
  do i=1,size(polygon%polygon)
    ! loop twice for elastic and newtonian
    ! polygon is one we should not be in
    if(.not.polygon%polygon(i)%use) then
      if (  if_inpoly(rlong  ,rlat,polygon%polygon(i)%coords).ne.0 &
        .or.if_inpoly(rlong2 ,rlat,polygon%polygon(i)%coords).ne.0 ) then
      iok=0
      return
    endif
  endif
enddo
ianyok=0
! polygon is one we should be in; test to see if we are, and if so set
! iok to 1 and return
do i=1,size(polygon%polygon)
  if(polygon%polygon(i)%use) then
    ianyok = ianyok+1
    if (  if_inpoly(rlong  ,rlat,polygon%polygon(i)%coords).ne.0 &
      .or.if_inpoly(rlong2 ,rlat,polygon%polygon(i)%coords).ne.0 ) then
    iok=1
    return
  endif
endif
  enddo
  ! not inside any polygon%polygons; set iok to 0 if there are any we should have
  ! been in
  iok = 1
  if(ianyok.gt.0) iok = 0
  return
end subroutine

! ===============================================================================
!! taken from spotl \cite Agnew97
!! \par oryginal comment:
!!  Rewritten by D. Agnew from the version by Godkin and Pulli,
!!  in BSSA, Vol 74, pp 1847-1848 (1984)
!! adopted and slightly modified M. Rajner
!! cords is x, y (lon, lat) 2 dimensional array
! ===============================================================================
integer function if_inpoly(x,y,coords)
  use mod_constants, only: dp, dp
  real(dp) ,allocatable , dimension (:,:) , intent (in) :: coords
  real(dp) , intent (in) :: x , y
  integer :: i , isc
  ! Returns 1 if point at (x,y) is inside polygon whose nv vertices
  ! Returns 0 if point is outside
  ! Returns 2 if point is on edge or vertex

  if_inpoly = 0
  do  i=1, size(coords(:,1))-1
    isc = ncross( &
      coords(i,1)   - x,   &
      coords(i,2)   - y,   &
      coords(i+1,1) - x,   &
      coords(i+1,2) - y )
    !  on edge - know the answer
    if(isc.eq.4) then
      if_inpoly = 2
      return
    endif
    if_inpoly = if_inpoly + isc
  enddo
  ! check final segment
  isc = ncross( &
    coords(size(coords(:,1)) , 1 ) - x , &
    coords(size(coords(:,2)) , 2 ) - y , &
    coords(1 , 1 ) - x , &
    coords(1 , 2 ) - y )
  if(isc.eq.4) then
    if_inpoly = 2
    return
  endif
  if_inpoly = if_inpoly + isc
  if_inpoly = if_inpoly/2
  ! convert to all positive (a departure from the original)
  if_inpoly = iabs(if_inpoly)
  return
end function

! ===============================================================================
!> \brief 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 \cite Agnew97
!! slightly modified
! ===============================================================================
integer function ncross(x1,y1,x2,y2)
  real(dp) , intent(in) :: x1 , y1, x2 , y2
  real(dp) :: c12 , c21

  ! all above (or below) axis
  if(y1*y2.gt.0) then
    ncross = 0
    return
  endif

  c12 = x1*y2
  c21 = x2*y1

  ! through origin
  if(c12.eq.c21.and.x1*x2.le.0.) then
    ncross = 4
    return
  endif

  ! touches +x axis; crosses +x axis; lies entirely on -x axis
  if(   (y1.eq.0.and.x1.gt.0)    &
    .or.(y2.eq.0.and.x2.gt.0) &
    .or.((y1.lt.0).and.(c12.gt.c21)) &
    .or.((y1.gt.0).and.(c12.lt.c21)) &
    .or.(y1.eq.0.and.y2.eq.0.and.x1.lt.0.and.x2.lt.0)) &
    then
    ncross = 0
    return
  endif

  ! cross axis
  if(y1.ne.0.and.y2.ne.0) then
    if(y1.lt.0) ncross = 2
    if(y1.gt.0) ncross = -2
    return
  endif
  ! one end touches -x axis - goes which way?
  if(y1.eq.0) then
    if(y2.lt.0) ncross = -1
    if(y2.gt.0) ncross = 1
  else
    ! y2=0 - ends on x-axis
    if(y1.lt.0) ncross = 1
    if(y1.gt.0) ncross = -1
  endif
  return
end function

end module

!\appendix
! \chapter{Polygon}
!  This examples show how the exclusion of~selected polygons works
!  \begin{figure}[htb]
!    \includegraphics[width=0.5\textwidth]{../mapa1}
!    \caption{If only excluded polygons (red area) are given
!    all points falling in~it will be excluded (red points) all other
!    will be included}
!  \end{figure}
!  \begin{figure}
!    \includegraphics[width=0.5\textwidth]{../mapa2}
!    \caption{If at least one included are are given
!    (green area) than all points which not fall into included area will
!    be excluded}
!  \end{figure}
!  \begin{figure}
!    \includegraphics[width=0.5\textwidth]{../mapa3}
!    \caption{If there is overlap of~polygons the exclusion has higher
!    priority}
!  \end{figure}
!  \chapter{Interpolation}
!  \begin{figure}
!  \input{/home/mrajner/src/grat/doc/interpolation_ilustration.tex}
!  \caption{Interpoloation}
!  \end{figure}
```

9.17 grat/src/value_check.f90 File Reference

Functions/Subroutines

- program **value_check**

9.17.1 Detailed Description

Date

2013-01-09

Author

M. Rajner

Definition in file value_check.f90.

9.18 value_check.f90

```fortran
! ==============================================================================
!> \file
!! \date 2013-01-09
!! \author M. Rajner
! ==============================================================================
program value_check
  use mod_cmdline
  use mod_parser
  use mod_data
  use mod_date
  use mod_site
  use mod_constants,  only: dp, R_air, earth
  use mod_polygon,    only: read_polygon, chkgon, polygon
  use mod_atmosphere, only: standard_pressure, standard_temperature, geop2geom
  use mod_utilities,  only: d2r

  implicit none
  real (dp) , allocatable , dimension(:) :: val
  real (dp)  :: cpu(2), sh
  integer    :: i, ii, j ,start, imodel, iprogress = 0
  integer(2) :: iok
  integer(2) :: ilevel, start_level


  call cpu_time(cpu(1))

  call intro(                                &
    program_calling   = "value_check",        &
    accepted_switches = "VFoShvIDLPRqwHMJ&!", &
    version           = "beta",               &
    cmdlineargs       = .true.                &
    )

  ! for progress bar
  if (output%unit.ne.output_unit.and..not.quiet) open (unit=output_unit, carriagecontrol=’fortran’)

  allocate (val(size(model)))

  start=0
  if (size(date).gt.0) then
    start=1
    ! print header
    if (output%header) then
      if (.not.output%prune) then
        write (output%unit , ’(a10,1x,a14,1x)’ , advance = "no" ) "#mjd", "date"
      endif
    endif
  endif

  ! print header
  if (output%header.and.size(site).gt.0) then
    if (.not.output%prune) then
      write (output%unit, ’(a8,2a10$)’) "name", "lat", "lon"
      if (output%height) then
        write (output%unit, ’(a10$)’) "height"
      endif
    endif
    if (output%level) then
      write (output%unit, ’(a6$)’) "level"
    endif
  endif

```

```
do i = 1, size(model)
  if (output%header) then
    if (model(i)%dataname.eq."custom") then
      write (output%unit,'(a6,"@custom")', advance='no') trim(model(i)%name)
    else
      write (output%unit,'(a13)', advance='no') trim(model(i)%dataname)
    endif
  endif
enddo
if(output%header) write(output%unit, *)

do j = start, size(date)
  do i = 1 , size(model)
    if (model(i)%if) then
      if (model(i)%autoload  &
        .and. &
        .not.( &
        model(i)%autoloadname.eq."ERA" &
        .and.(any(model(i)%dataname.eq.["GP","VT","VSH"])) &
        )) then

        if ( &
          (j.eq.1 &
          .or. .not. date(j)%date(1).eq.date(j-1)%date(1) &
          ) &
          ) then
          call model_aliases(model(i), year=date(j)%date(1))
        endif

      else if (model(i)%autoload) then

        if ( &
          (j.eq.1 &
          .or. .not.( &
          date(j)%date(1).eq.date(j-1)%date(1) &
          .and.date(j)%date(2).eq.date(j-1)%date(2)) &
          ) &
          ) then

          call model_aliases( &
            model(i), year=date(j)%date(1), month=date(j)%date(2))
        endif
      endif

      if (allocated(date).and.model(i)%exist) then
        call get_variable(model(i), date = date(j)%date)
      elseif(model(i)%exist) then
        call get_variable(model(i))
      endif

    endif
  enddo

  ! print only dates if no site given
  if (j.gt.0 .and. size(site).lt.1) then
    if (dryrun) then
      write (output%unit , '(i4.4,5(i2.2),$)') date(j)%date
      if (j.lt.size(date)) write (output%unit , '(", ",$)')
    else
      write (output%unit , '(f10.3,1x,i4.4,5(i2.2))'  ) date(j)%mjd , date(j)%date
    endif
  endif

  if (level%all.and..not.allocated(level%level)) then
    allocate(level%level(size(model(1)%level)))
    level%level=model(1)%level
  endif

  if (size(level%level).lt.1) then
    start_level=0
  else
    start_level=1
  endif

  do ilevel=start_level, size(level%level)
    do i = 1 , size(site)
      iprogress = iprogress + 1

      ! add time stamp if -D option was specified
      if (j.gt.0) then
        if (.not.output%prune) then
          write (output%unit , '(f10.3,1x,i4.4,5(i2.2),1x)' , advance = "no" ) date(j)%mjd , date(j)%date
        endif
      endif

```

```
        ! if this point should not be used (polygon) leave as zero
        if (allocated(polygon).and.polygon(1)%if) then
          call chkgon(site(i)%lon, site(i)%lat, polygon(1), iok)
        else
          iok=1
        endif

        imodel = 0
        do ii = 1 , size (model)
          imodel = imodel + 1
          if (model(ii)%if.or.model(ii)%if_constant_value) then
            if (iok.eq.1) then
              if (j.eq.0) then
                call get_value(model(ii), site(i)%lat, site(i)%lon, val(imodel), &
                  method=info(1)%interpolation, level=level%level(ilevel))
              else
                call get_value(model(ii), site(i)%lat, site(i)%lon, val(imodel), &
                  method=info(1)%interpolation, date=date(j)%date, level=level%level(ilevel))
              endif
            else
            endif
            if (model(ii)%dataname.eq."LS") val(ii)=int(val(ii))

          else if (model(ii)%dataname.eq."custom") then
            if(ilevel.eq.1) sh=val(ind%model%vsh)
            call customfile_value( &
              what  = model(imodel)%name, &
              sp    = val(ind%model%sp), &
              t     = val(ind%model%t), &
              hp    = val(ind%model%hp), &
              sh    = sh, &
              gp    = val(ind%model%gp), &
              vsh   = val(ind%model%vsh), &
              vt    = val(ind%model%vt), &
              level = level%level(ilevel), &
              val   = val(imodel), &
              rho   = any(model%name.eq."RHO") &
              )
          else
            val(imodel) = sqrt(-1.)
          endif
        enddo

        if (.not.output%prune) then
          write (output%unit , '(a8,2f10.4$)') site(i)%name, site(i)%lat, site(i)%lon
          if (output%height) then
            write (output%unit, '(f10.3$)') site(i)%height
          endif
        endif

        if (output%level.and. allocated(level%level)) then
          write (output%unit, '(i6$)') level%level(ilevel)
        elseif(output%level) then
          write (output%unit, '(i6$)') ilevel
        endif

        write (output%unit , "("//output%form//'$)') val

        if (output%unit.ne.output_unit.and..not.quiet) then
          call cpu_time(cpu(2))

          call progress(                                   &
            100*iprogress/(max(size(date),1)               &
            *max(size(site),1)*max(size(level%level),1)), &
            cpu(2)-cpu(1)                                  &
            )
        endif
        if (size(val).gt.0) write (output%unit , *)
      enddo
    enddo
  enddo

  if (ind%moreverbose%d.ne.0) then
    do i=1, size(model)
      do j=1, size(model(i)%time)
        write (moreverbose(ind%moreverbose%d)%unit, '(g0,1x,i4,5i2.2)') &
          model(i)%time(j), model(i)%date(j,:)
      enddo
    enddo
  endif

  if (ind%moreverbose%j.ne.0) then
    do i = 1, size(model)
```

```
      do j = 1, size(model(i)%level)
        write (moreverbose(ind%moreverbose%j)%unit, '(i5)') &
          model(i)%level(j)
      enddo
    enddo
  endif

  call cpu_time(cpu(2))
  if (output%unit.ne.output_unit.and..not.quiet) then
    call progress(                                  &
      100*iprogress/(max(size(date),1)              &
      *max(size(site),1)*max(size(level%level),1)), &
      cpu(2)-cpu(1),                                &
      every=1                                       &
      )
    close(output_unit)
  endif
  write(log%unit, '(/,"Execution time:",1x,f16.9," seconds")') cpu(2)-cpu(1)
  write(log%unit, form_separator)
end program
```

Chapter 10

Example Documentation

10.1 example_aggf.f90

```fortran
! ============================================================================
!! This program shows some example of using AGGF module
!!
!! \author Marcin Rajner
!! \date 20121108
! ============================================================================
program example_aggf
  use mod_atmosphere
  use mod_constants, only: dp
  use mod_utilities
  use mod_printing, only: log
  implicit none
  real(dp) :: cpu(2)


  call cpu_time(cpu(1))
  call standard1976('/home/mrajner/src/grat/examples/standard1976.dat')
  call compare_fels_profiles(
'/home/mrajner/src/grat/examples/compare_fels_profiles.dat')
  call simple_atmospheric_model("/home/mrajner/dr/rysunki/simple_approach.dat")
  call green_newtonian_compute( &
    ["green_newtonian_olsson.dat","green_newtonian_spotl.dat",
"green_newtonian.dat"])
  call admit_niebauer("/home/mrajner/src/grat/examples/admit_niebauer.dat")
  call aggf_thin_layer("/home/mrajner/src/grat/examples/aggf_thin_layer.dat")
  call compute_tabulated_green_functions(
'/home/mrajner/src/grat/dat/rajner_green_full.dat'  , method="full"       , predefined=.false.)
  call compute_tabulated_green_functions(
'/home/mrajner/src/grat/dat/rajner_green_rough.dat' , predefined=.false., rough=.true.)
  call compute_tabulated_green_functions(
'/home/mrajner/src/grat/dat/rajner_green_simple.dat', method="simple"     , predefined=.false.)
  call compute_tabulated_green_functions(
'/home/mrajner/src/grat/dat/rajner_green.dat'       , predefined=.false. )
  call aggf_resp_fels_profiles(
'/home/mrajner/src/grat/examples/aggf_resp_fels_profiles.dat')
  call mass_vs_height('/home/mrajner/src/grat/examples/mass_vs_height.dat')
  call aggf_resp_hmax('/home/mrajner/src/grat/examples/aggf_resp_zmax.dat')
  call aggf_resp_dz('/home/mrajner/src/grat/examples/aggf_resp_dz.dat')
  call aggf_resp_t('/home/mrajner/src/grat/examples/aggf_resp_t.dat')
  call aggf_resp_h('/home/mrajner/src/grat/examples/aggf_resp_h.dat')

  call cpu_time(cpu(2))
  print '("Total time: ",f8.3,x,"[s]")', cpu(2)-cpu(1)

contains
! =============================================================================
!> Mass of atmosphere respect to height
! =============================================================================
subroutine mass_vs_height (filename)
  use, intrinsic:: iso_fortran_env
  use mod_utilities, only: file_exists
  use mod_constants, only : dp, pi, earth, R_air
  use mod_atmosphere
  character(*), intent (in), optional:: filename
  real(dp) :: max_height,dh, percent
  real(dp), allocatable, dimension(:):: mass, height
  integer::i,j,file_unit

  if (present(filename)) then
```

```
    if (file_exists(filename)) return
    open ( &
      newunit = file_unit, &
      file    = filename,  &
      action  = 'write'   &
      )
  else
    file_unit = output_unit
  endif
  write(*,*), "mass_vs_height ---> ",filename

  max_height=50000.
  dh=10

  allocate(height(int(max_height/dh)+1))
  allocate(mass(size(height)))
  do i =1,size(height)
    height(i) = dh*(i-1)
    mass(i) = standard_pressure( &
      height(i), &
      method="standard", &
      use_standard_temperature=.true., &
      nan_as_zero=.true.) &
      / (r_air * standard_temperature(height(i)))
  enddo

  do i =0,50000,1000
    percent=0
    do j = 1, size(height)
      if (height(j).le.dble(i)) percent=percent+mass(j)
    enddo
    percent = percent / sum(mass)  * 100.
    write(file_unit, '(i6,2f19.9,es10.3)' ), i, percent, &
      100-(earth%radius+dble(1))**2 &
      * standard_pressure(dble(i),method="standard", use_standard_temperature=.true.) &
      / standard_gravity(dble(i))&
      /earth%radius**2/standard_pressure(dble(0),method="standard") * standard_gravity(dble(0))*100
  enddo
end subroutine

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

  real(dp) :: r ! km
  integer :: file_unit
  character(*), intent(in), optional:: filename
  real(dp) :: h =9.

  if (present(filename)) then
    if (file_exists(filename)) return
    open ( &
      newunit = file_unit, &
      file    = filename,  &
      action  = 'write'    &
      )
  else
    file_unit = output_unit
  endif

  write(*,*), "simple_atmospheric_model ---> ",filename

  do r = 0., 25*8
    write (file_unit, *) &
      r, &
      -100*bouger(h=h,r=r)/(earth%gravity%mean*h) * 1e8, & !conversion to microGal
      -simple_def(r) * 1e8
  enddo
end subroutine

! ===========================================================================
!> Compute AGGF and derivatives
!!
!! \author M. Rajner
!! \date 2013-03-18
```

```fortran
! ============================================================================
subroutine compute_tabulated_green_functions ( &
    filename, method, dz, &
    predefined,fels_type, rough)
  use mod_constants, only: dp
  use mod_aggf,      only: aggf, aggfd
  use mod_green,      only: green
  use mod_utilities, only: d2r, file_exists
  use mod_atmosphere

  integer :: i, file_unit
  character(*), intent(in) :: filename
  real(dp), optional :: dz
  character(*), optional :: fels_type
  character(*), optional :: method
  logical, optional, intent(in) :: predefined, rough

  if (file_exists(filename)) then
    return
  else
    print '(a,a)', "compute_tabulated_green_functions --> ", trim(filename)
  endif

  call get_green_distances

  open (                 &
    newunit = file_unit, &
    file    = filename,  &
    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/m]', 'GN/dz[microGal/hPa/m]'

  do i= 1, size(green(1)%distance)
    write(file_unit, '(13f15.6)'), &
      green(1)%distance(i), &
      aggf(d2r(green(1)%distance(i)), method=method, dz=dz
          , predefined=predefined, fels_type=fels_type, rough=rough), &
      aggfd(d2r(green(1)%distance(i)), method=method, dz=dz, aggfdt=.true.
            , predefined=predefined, fels_type=fels_type, rough=rough), &
      aggf(d2r(green(1)%distance(i)), method=method, dz=dz,
first_derivative_h=.true., predefined=predefined, fels_type=fels_type, rough=rough), &
      aggf(d2r(green(1)%distance(i)), method=method, dz=dz,
first_derivative_z=.true., predefined=predefined, fels_type=fels_type, rough=rough)
  enddo
  close(file_unit)
end subroutine

! ============================================================================
!> Compare different vertical temperature profiles impact on AGGF
! ============================================================================
subroutine aggf_resp_fels_profiles (filename)
  use mod_constants, only: dp
  use mod_aggf,  only: aggf
  use mod_green, only: green
  character (len=255), dimension (6) :: fels_types
  integer :: i, j, file_unit
  character(*), intent(in), optional :: filename

  if (present(filename)) then
    if (file_exists(filename)) return
    open ( newunit = file_unit, &
      file =filename, &
      action  = 'write' )
  else
    file_unit = output_unit
  endif
  print *, "aggf_resp_fels_profiles -->", filename

  ! Get the spherical distances from Merriam92
  call get_green_distances()

  ! ! All possible optional arguments for standard_temperature
  fels_types = (/ &
    "US1976"             , "tropical",   &
    "subtropical_summer" , "subtropical_winter" , &
    "subarctic_summer"   , "subarctic_winter"     &
    /)
  ! print header
  write (file_unit, ’(100(a20))’) &
    ’psi’, (trim(fels_types(i)), i = 1, size(fels_types))

  ! print results
  do i = 1, size(green(1)%distance)
    write(file_unit, ’(<size(fels_types)+1>f20.5)’), &
      green(1)%distance(i), &
      (aggf( &
      d2r(green(1)%distance(i)), &
      method="standard", &
      fels_type=fels_types(j)), j=1,size(fels_types) &
      )
  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 (filename)
  use iso_fortran_env
  use mod_utilities, only: file_exists
  use mod_constants, only: dp
  use mod_atmosphere, only : standard_temperature
  character (len=255), dimension (6) :: fels_types
  real (dp) :: height
  integer :: i, file_unit, i_height
  character(*), intent (in),optional:: filename

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

  if (present(filename)) then
    if (file_exists(filename)) return
    open ( newunit = file_unit, &
      file =filename, &
      action  = ’write’ )
  else
    file_unit = output_unit
  endif

  print *, "compare_fels_profiles --->", filename

  ! 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$)’), height
    do i = 1, size (fels_types)
      write (file_unit, ’(f20.3$)’),  standard_temperature(height*1000, fels_type=fels_types(i))
    enddo
    write ( file_unit, *)
  enddo
  close(file_unit)
end subroutine

! ============================================================================
!> Computes AGGF for different site height (h)
! ============================================================================
subroutine aggf_resp_h (filename)
  use mod_green, only: green
  use mod_aggf, only: aggf
  real(dp) :: heights(6)
  character(*), intent(in), optional :: filename
  integer :: file_unit, i, ii, j
  real(dp) :: aux

  if (present(filename)) then
    if (file_exists(filename)) return
    open ( newunit = file_unit, &
      file =filename, &
      action  = ’write’ )
  else
    file_unit = output_unit
  endif
```

```fortran
  print *, "aggf_resp_h --->", filename

  call get_green_distances()

  heights=[0.,1.,10.,100.,1000.,10000.]


  write (file_unit, "(a12,6(x,’h’,f0.0))") "distance", heights(1:6)
  do i =1, size (green(1)%distance)
    ! denser sampling
    do ii = 0,8
      aux  = green(1)%distance(i) + ii * (green(1)%distance(i+1) - green(1)%distance(i)) / 9.
      if (aux.gt.0.2 ) exit
      write (file_unit, ’(F12.6$)’), aux
      do j =  1, size(heights)
        write (file_unit,’(f12.4,1x,$)’) aggf(d2r(aux), method="standard", h=heights(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 (filename)
  use mod_green, only: green
  ! use mod_constants, only : dp, atmosphere
  use mod_aggf, only : aggf
  real(dp), dimension(:,:), allocatable :: results
  integer :: i, j
  character(*), intent(in), optional :: filename
  integer :: file_unit
  real(dp) :: temperatures(3)

  if (present(filename)) then
    if (file_exists(filename)) return
    open ( newunit = file_unit, &
      file =filename, &
      action  = ’write’ )
  else
    file_unit = output_unit
  endif
  call get_green_distances()

  allocate(results(size(green(1)%distance), 3))

  temperatures=[0., 15., -45]

  write(file_unit, ’(4a12)’) "distance","T0+0", "T0+15", "T0-45"
  do i = 1, size(green(1)%distance)
    write(file_unit, ’(f12.5$)’) green(1)%distance(i)
    do j=1, size(temperatures)
      write(file_unit, ’(f12.5$)’) &
        aggf(d2r(green(1)%distance(i)), method="standard", t_zero=temperatures(j))
    enddo
    write(file_unit, *)
  enddo
  close (file_unit)
end subroutine

! ============================================================================
!> \brief This computes AGGF for different height integration step
! ============================================================================
subroutine aggf_resp_dz (filename)
  use mod_green
  use mod_aggf, only: aggf
  real(dp), dimension(:,:), allocatable :: results
  real(dp), dimension(:), allocatable :: dzs

  integer :: file_unit, i, j
  character(*), intent (in), optional:: filename

  if (present(filename)) then
    if (file_exists(filename)) return
    open ( newunit = file_unit, &
      file =filename, &
      action  = ’write’ )
  else
    file_unit = output_unit
  endif

  call get_green_distances()

  allocate(dzs(5))
  dzs=(/ 0.01, 0.1, 1., 10., 100./)

  allocate (results(size(green(1)%distance(1:29)),size(dzs)))
  results = 0.

  do i = 1, size (results(:,1))
    do j=1,size(dzs)
      results(i,j)=i+j
      results(i,j)=aggf(d2r(green(1)%distance(i)), &
        method="standard", &
        dz=dzs(j))
    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(file_unit, '(a14,<size(dzs)>f14.4)') "psi_dz", dzs
  write(file_unit, '(f14.5,<size(dzs)>e14.4)') &
    (green(1)%distance(i), results(i,:), i=1,size(results(:,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(filename)
  use, intrinsic :: iso_fortran_env
  use mod_utilities, only: file_exists
  use mod_constants, only : dp, R_air
  use mod_atmosphere, only: &
    standard_temperature, standard_pressure, &
    standard_gravity,     standard_density
  integer :: file_unit
  character(*), intent (in), optional:: filename
  real(dp) :: height

  if (present(filename)) then
    if (file_exists(filename)) return
    open ( newunit = file_unit, &
      file =filename, &
      action  = 'write' )
  else
    file_unit = output_unit
  endif

  print *, "standard atmosphere --->", filename
  ! print header
  write ( file_unit, '(6(a15))' ) &
    'height', 'T', 'g', 'p', 'rho'
  do height=0.,68000., 1000
    ! print results to file
    write( file_unit,'(5f15.5, e12.3)'), &
      height/1000.,                        &
      standard_temperature(height),        &
      standard_gravity(height),            &
      standard_pressure(height, method="standard")/100.,      &  ! --> hPa
      standard_pressure(height, method="standard") &
      /(r_air*standard_temperature(height))
  enddo
  close( file_unit )
end subroutine

! ===========================================================================
!> \brief This computes relative values of AGGF for different atmosphere
!! height integration
! ===========================================================================
subroutine aggf_resp_hmax (filename)
  use mod_utilities, only: file_exists, logspace, d2r
  ! use mod_constants, only : dp
  use mod_aggf, only : aggf
  real (dp), dimension (2) :: psi
  real (dp), dimension (:), allocatable :: heights
  real (dp), dimension (:,:), allocatable :: results
  integer :: file_unit, n, i, j
  character(*), intent (in), optional:: filename

  if (present(filename)) then
    if (file_exists(filename)) return
    open ( newunit = file_unit, &
      file =filename, &
      action  = 'write' )
  else
    file_unit = output_unit
  endif

  print *, "standard atmosphere ---> ", filename
  psi=(/0.0001, 10 /)

  n=90
  allocate(heights(n))

  heights= logspace(real(1e-1,dp), real(60000,dp),n)

  allocate (results(size(heights), size(psi)))
  results=0

  do j=1, size(heights)
    do i = 1, size(psi)
      results(j,i) =aggf(d2r(psi(i)),method="standard", zmax=heights(j))
    enddo
  enddo
  do i = 1, size(psi)
    results(:,i)=results(:,i)/results(size(heights),i) * 100. ! in %
  enddo

  write(file_unit, '(a14,SP,100f14.5)' ),"#heght\psi", (psi(j), j= 1,size(psi))
  do i=1, size (results(:,1))
    write(file_unit, '(100f14.4)' ) heights(i)/1000, (results(i,j), j = 1, size(psi) )
  enddo
  close(file_unit)
end subroutine

! ============================================================================
! ============================================================================
subroutine aggf_thin_layer (filename)
  use, intrinsic:: iso_fortran_env
  use mod_constants, only: dp, pi
  use mod_aggf, only: GN_thin_layer
  use mod_utilities, only: d2r, file_exists
  use mod_green

  integer :: file_unit, i
  character(*), intent (in), optional:: filename

  if (file_exists(filename)) return

  call get_green_distances()

  write(*,*), "aggf_thin_layer ---> ",filename
  if (present(filename)) then
    open (newunit = file_unit, &
      file =filename, &
      action  = 'write' )
  else
    file_unit = output_unit
  endif
  do i = 1, size (green(1)%distance)
    write(file_unit,*) green(1)%distance(i), green(1)%data(i), &
      gn_thin_layer(d2r(green(1)%distance(i)))
  enddo
end subroutine

! =============================================================================
! =============================================================================
subroutine admit_niebauer(filename)
  use mod_constants
  use mod_utilities
  real(dp) :: a
  real(dp) :: theta
  real(dp) :: b, f
  character(*), intent(in) :: filename
  integer::iun

  if (file_exists(filename)) return
  print *, "admit_niebauer ---> ", filename

  open (newunit=iun, file=filename, action = 'write')

  f=earth%radius/9500
  do theta=0.5, 180, 0.01
    b= 2*f*sin(d2r(theta/2))
    a= 2*pi * gravity%constant / earth%gravity%mean* &
      (1 - b/(2*f) -1/b + 2/f)
    write(iun, *), theta, a *1e10
  enddo
```

```
end subroutine

! ==============================================================================
!> compute green newtonian function
! ==============================================================================
subroutine green_newtonian_compute(filenames)
  use mod_utilities, only: file_exists
  use mod_green
  use mod_utilities, only: logspace, d2r
  integer:: iun, n, i, j, k
  real (dp), allocatable, dimension(:) :: psi, h
  character(12), allocatable, dimension(:) :: column_name
  character(*),  optional :: filenames(3)
  character(20) :: method
  character(40) :: prefix

  prefix="/home/mrajner/src/grat/examples/"

  iun = 6

  n = 9 * 50
  allocate(psi(n))
  psi = logspace(real(1e-6,dp), real(180,dp),n)

  allocate(h(11))
  h = [0., 1., 10., 100., 1000., 10000., -1., -10., -100., -1000., -10000.]

  allocate(column_name(size(h)))
  write(column_name, ’(f0.0)’ ) (h(i),i=1,11)

  do k =1,3
    if (file_exists(trim(prefix)//trim(filenames(k)))) cycle
    print *, "green_newtonian_compute ---> ", trim(prefix)//trim(filenames(k))
    open (newunit=iun, file=trim(prefix)//filenames(k), action = ’write’)

    method = filenames(k)(17:index(filenames(k),".")-1)
    write(iun, ’(a12,<size(h)>a12)’) "#psi", ( "h"//trim(column_name(i)), i = 1, 11)
    write(iun, ’(<size(h)+1>en12.2)’), (psi(i), &
      (green_newtonian(d2r(psi(i)), h= h(j), method = method), j=1,size(h)), &
      i=1,size(psi))
    close(iun)
  enddo
end subroutine

! ==============================================================================
! ==============================================================================
subroutine get_green_distances()
  use mod_green
  if (allocated(green)) deallocate(green)
  allocate (green(1))
  green(1)%name="merriam"
  green(1)%column=[1, 2]
  green(1)%dataname="GN"
  call read_green(green(1),print=.false.)
end subroutine
end program
```

10.2 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, 3:3:3 \
    -F /home/mrajner/dat/ncep_reanalysis/pres.sfc.2011.nc@SP:pres \
    , ~/data/wghm/dat/WGHM.nc @ WGHM \
    -G rajner@GN : 1 : 2 \
    -D 201101:1@D -V

    # 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
```