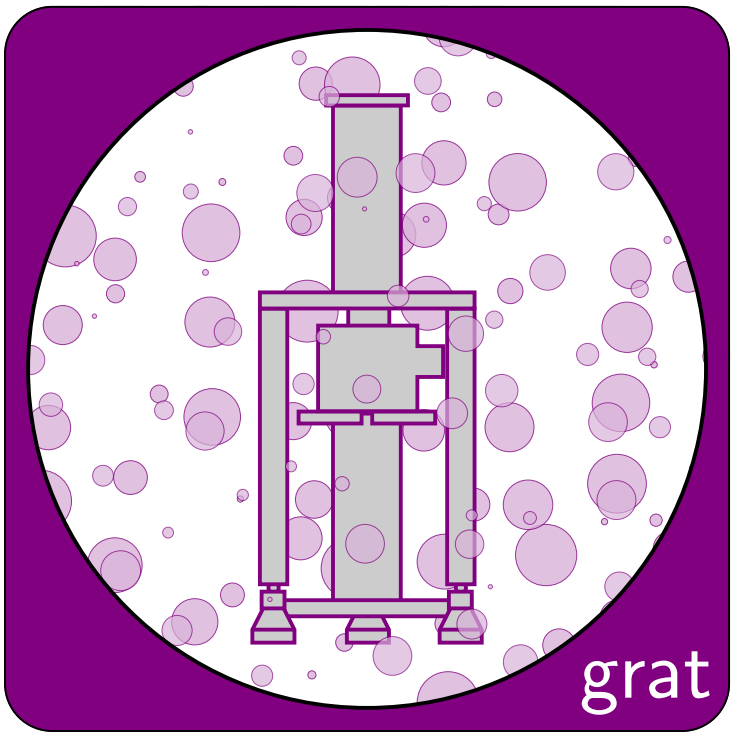

grat

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

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

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

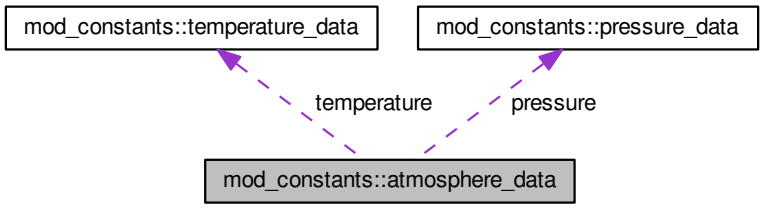
Definition at line 123 of file mod_cmdline.f90.

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:



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

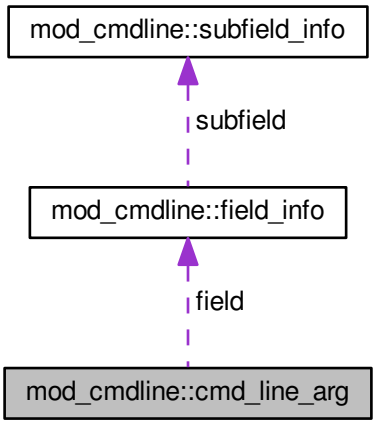
Definition at line 93 of file mod_constants.f90.

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

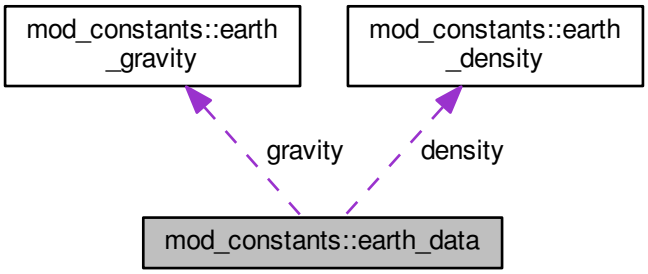
Definition at line 110 of file mod_constants.f90.

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

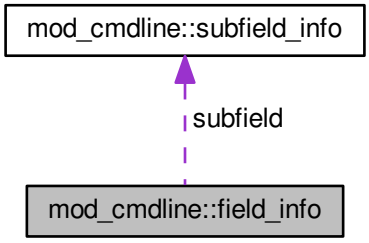
Definition at line 59 of file mod_constants.f90.

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

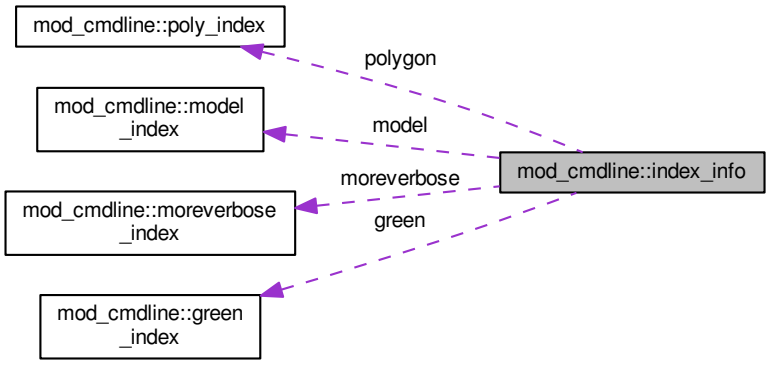
Definition at line 98 of file mod_cmdline.f90.

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

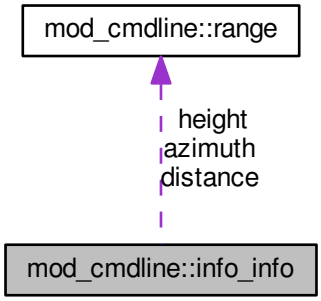
Definition at line 115 of file mod_cmdline.f90.

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

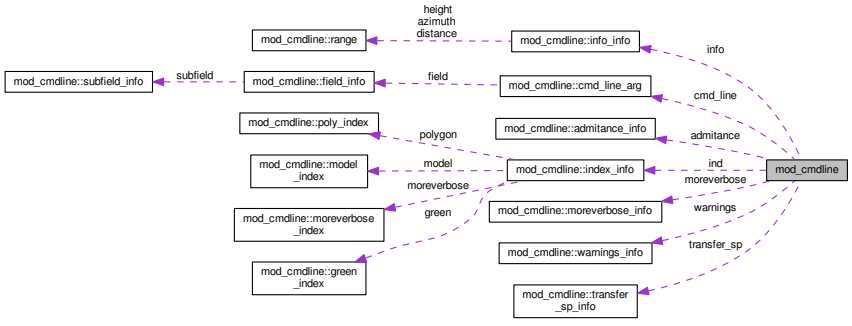
Definition at line 166 of file mod_atmosphere.f90.

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.

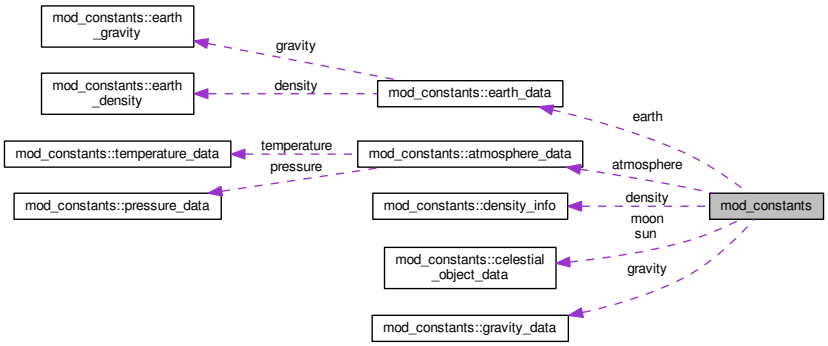
The documentation for this module was generated from the following file:

- grat/src/mod_cmdline.f90

8.25 mod_constants Module Reference

Define constant values.

Collaboration diagram for mod_constants:



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.

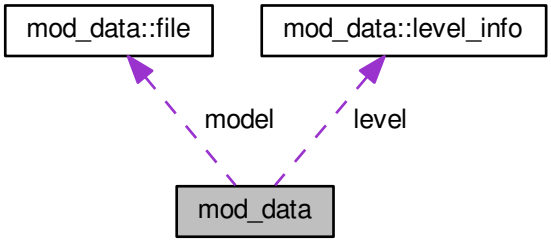
The documentation for this module was generated from the following file:

- 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 geoscienses. Moreover it is self-describing and machine independent. It also allows for reading and writing small subset of data therefore very efficient for large datafiles (this case) **?**

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.

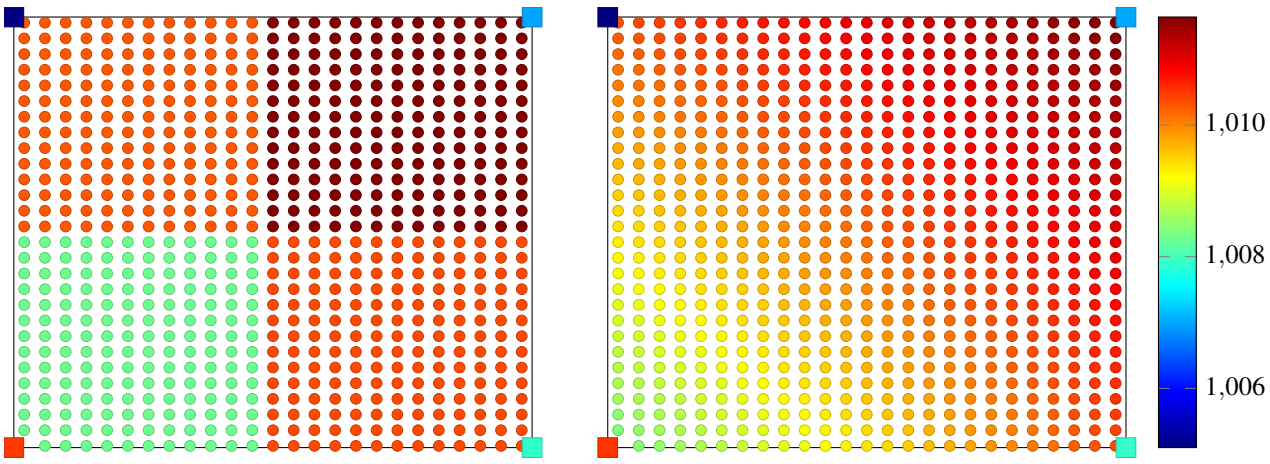
from **?** see http://www.unidata.ucar.edu/software/netcdf/docs/BestPractices.html

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

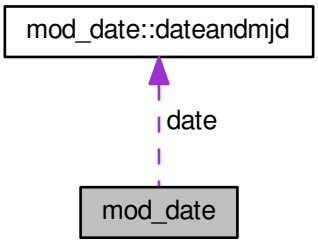
Definition at line 71 of file mod_data.f90.

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

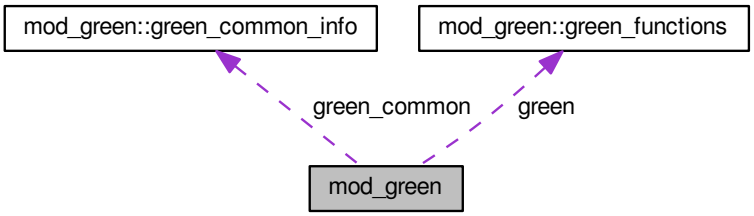
Definition at line 247 of file mod_date.f90.

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

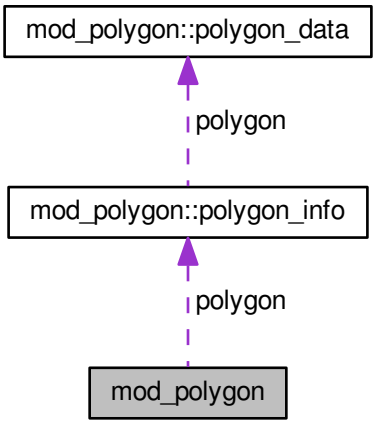
Definition at line 704 of file mod_parser.f90.

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

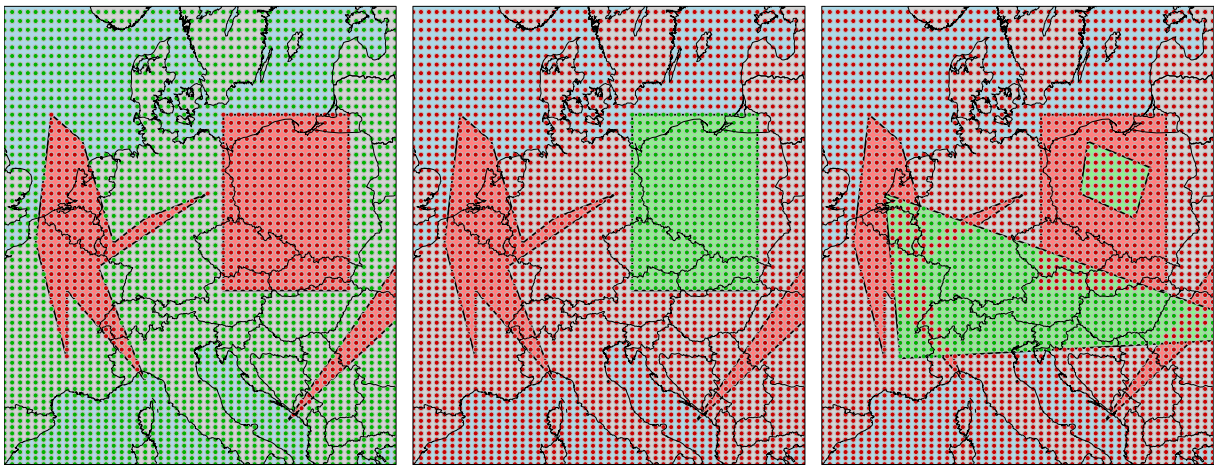


Figure 8.1: capt

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 **?**

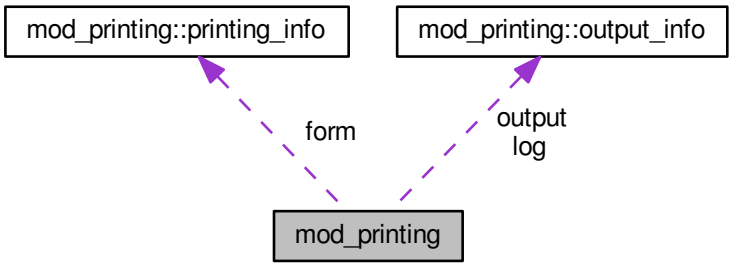
Definition at line 80 of file mod_polygon.f90.

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

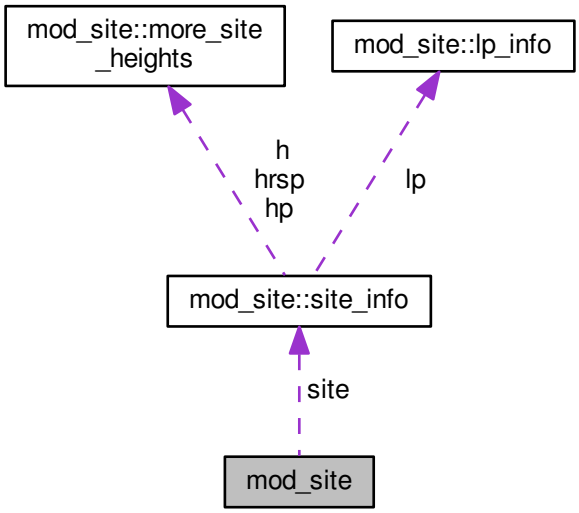
Definition at line 1 of file mod_printing.f90.

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

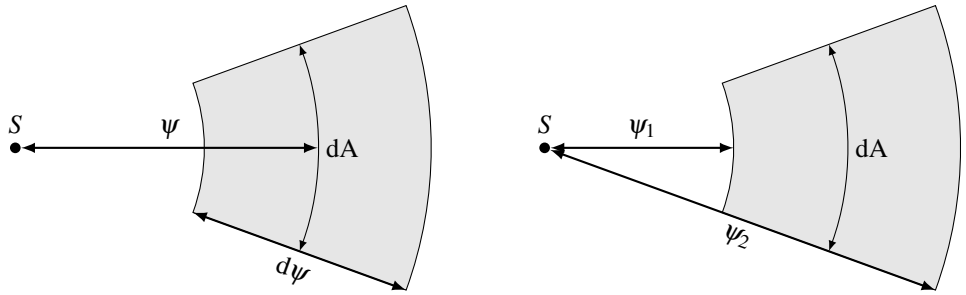
8.35.2.1 real(dp) function mod_spherical::spher_area (real(dp), intent(in) *distance,* real(dp), intent(in) *ddistance,* real(dp), intent(in) *azstp,* real(dp), intent(in), optional *radius,* logical, intent(in), optional *alternative_method*)

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

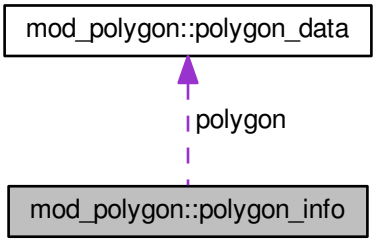
Definition at line 17 of file mod_polygon.f90.

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

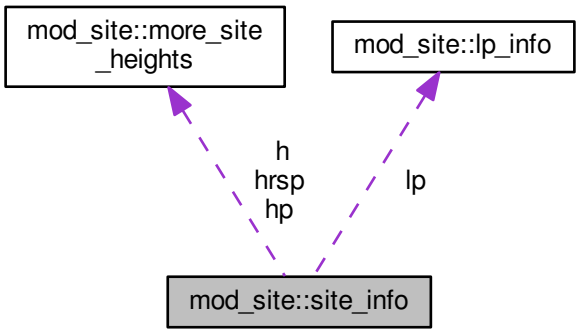
Definition at line 47 of file mod_cmdline.f90.

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

```
00001 #!/bin/bash -
00002 #
      ================================================================================
00003 #          FILE: interpolation_ilustration.sh
00004 #         USAGE: ./interpolation_ilustration.sh
00005 #   DESCRIPTION:
00006 #       OPTIONS: ---
00007 #        AUTHOR: mrajner
00008 #       CREATED: 05.12.2012 10:38:30 CET
00009 #      REVISION:  ---
00010 #
      ================================================================================
00011
00012 ## \file
00013 set -o nounset                              # Treat unset variables as an error
00014 for co in n 1
00015 do
00016   value_check                                                          \
00017     -F /home/mrajner/dat/ncep_reanalysis/pres.sfc.2011.nc@SP:pres \
00018     -S 2.51/4.99/0.05/2.45:0.091:0.1 -I ${co} @ I                      \
00019     -V                                                                 \
00020     -o interp${co}1.dat                                                \
00021     -L interp1.dat@1
00022 done
00023 perl -n -i -e 'print if $. <= 4' interp1.dat
00024
```

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

```
00001 !> \file
00002 !! \mainpage grat overview
00003 !! \section Purpose
00004 !! This program was created to make computation of atmospheric gravity
00005 !! correction easier. Still developing. Consider visiting later...
00006 !!
00007 !! \version pre-alpha
00008 !! \date 2013-01-12
00009 !! \author Marcin Rajner\n
00010 !! Politechnika Warszawska | Warsaw University of Technology
00011 !!
00012 !! \warning This program is written in Fortran90 standard but uses some
       featerus
00013 !! of 2003 specification (e.g., \c 'newunit='). It was also written
00014 !! for <tt>Intel Fortran Compiler</tt> hence some commands can be unavailable
00015 !! for other compilers (e.g., \c <integer_parameter> for \c IO statements. This
       should be
00016 !! easily modifiable according to your output needs.
00017 !! Also you need to have \c iso_fortran_env module available to guess the
       number
00018 !! of output_unit for your compiler.
00019 !! When you don't want a \c log_file and you don't switch \c verbose all
00020 !! unnecesarry information whitch are normally collected goes to \c /dev/null
00021 !! file. This is *nix system default trash. For other system or file system
00022 !! organization, please change this value in \c mod_cmdline module.
00023 !!
00024 !! \attention
00025 !! \c grat and value_check needs a \c netCDF library \cite netcdf
00026 !> \copyright
00027 !! Copyright 2013 by Marcin Rajner\n
00028 !! This program is free software: you can redistribute it and/or modify
00029 !! it under the terms of the GNU General Public License as published by
00030 !! the Free Software Foundation, either version 3 of the License, or
00031 !! (at your option) any later version.
00032 !! \n\n
00033 !! This program is distributed in the hope that it will be useful,
00034 !! but WITHOUT ANY WARRANTY; without even the implied warranty of
00035 !! MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
00036 !! GNU General Public License for more details.
00037 !! \n\n
00038 !! You should have received a copy of the GNU General Public License
00039 !! along with this program.
00040 !! If not, see <http://www.gnu.org/licenses/>.
00041 !! \page License
00042 !! \include LICENSE
00043 !!
00044 !! \section Usage
00045 !! After sucsesfull compiling make sure the executables are in your search path
00046 !!
00047 !! There is main program \c grat and some utilities program. For the options
       see
00048
00049 !> \page intro_sec External resources
00050 !!   - <a href="https://code.google.com/p/grat">project page</a> (git
       repository)
00051 !!   - \htmlonly <a href="../latex/refman.pdf">[pdf]</a> version of this
       manual\endhtmlonly
00052 !!   \latexonly
       \href{https://grat.googlecode.com/git/doc/html/index.html}{html} version of this manual\endlatexonly
00053 !! \TODO give source for grant presentation
00054 !!   - <a href="">[pdf]</a> command line options (in Polish)
00055 !! \example example_aggf.f90
00056 !! \example grat_usage.sh
00057 !
       ================================================================================
00058 program grat
00059   ! use omp_lib parallel computation not yet enabled
00060   use mod_parser, only: intro
00061   use mod_data
00062   use mod_date
00063   use mod_green, only: convolve, green
00064   use mod_site, only: print_site_summary, site
00065   use mod_cmdline
00066   use mod_admit, only: admit
00067   use mod_utilities, only: Bubble_Sort
00068
00069   implicit none
00070   real(dp) :: cpu(2)
00071   integer  :: isite, i, idate, start, iprogress = 0
00072   logical  :: first_waning = .true.
00073
00074   ! program starts here with time stamp
00075   call cpu_time(cpu(1))
00076
```

```
00077   ! gather cmd line option decide where to put output
00078   call intro(                                          &
00079     program_calling   = "grat",                         &
00080     version           = "pre-alpha",                    &
00081     accepted_switches = "VSBLGPqoFIDLvhRrMOAHUwJQ&!n", &
00082     cmdlineargs       = .true.                          &
00083     )
00084
00085   start = 0
00086
00087   if (dryrun) then
00088     call print_site_summary(site_parsing=.true.)
00089     call exit(0)
00090   endif
00091
00092   if (size(date).gt.0) then
00093     if(output%header) then
00094       write (output%unit, '(a12,x,a14,x)', advance = "no" ) "mjd", "date"
00095     endif
00096     start = 1
00097   endif
00098
00099   if(output%header) then
00100     write (output%unit, '(a8,3(x,a9$))') "name", "lat", "lon", "h"
00101   endif
00102
00103   if(output%header) then
00104
00105     if (method(1)) then
00106       write (output%unit,'(a13)', advance='no'), "G1D"
00107     endif
00108
00109     if (method(2).or.method(3)) then
00110       if (result_component) then
00111         do i = 1, size(green)
00112           if (green(i)%dataname.eq."GE") then
00113             if (inverted_barometer) then
00114               write (output%unit,'(a13$)'), trim(green(i)%dataname)//"_IB"
00115             else
00116               write (output%unit,'(a13$)'), trim(green(i)%dataname)//"_NIB"
00117             endif
00118           else
00119             write (output%unit,'(a13$)'), trim(green(i)%dataname)
00120           endif
00121         enddo
00122         if (inverted_barometer.and.non_inverted_barometer) then
00123           write (output%unit,'(a13$)'), "GE_NIB"
00124         endif
00125       endif
00126
00127       if (result_total) then
00128         if (method(2)) then
00129           write (output%unit,'(a13)',advance='no'), "G2D_t"
00130         endif
00131         if (method(3)) then
00132           write (output%unit,'(a13)',advance='no'), "G3D_t"
00133         endif
00134       endif
00135     endif
00136   endif
00137
00138   if(output%header) then
00139     write (output%unit, *)
00140   endif
00141
00142   ! read only once Land-sea, reference surface pressure
00143   if (ind%model%ls.ne.0) then
00144     call get_variable(model(ind%model%ls))
00145   endif
00146   if (ind%model%rsp.ne.0) then
00147     call get_variable(model(ind%model%rsp))
00148   endif
00149   if (ind%model%hrsp.ne.0) then
00150     call get_variable(model(ind%model%hrsp))
00151   endif
00152
00153   if (inverted_landsea_mask.and.ind%model%ls.ne.0) then
00154     model(ind%model%ls)%data = int(abs(model(ind%model%ls)%data-1))
00155   endif
00156
00157
00158   do idate=start, size (date)
00159     if (idate.ge.1) then
00160       if(.not.(output%nan).and.modulo(date(idate)%date(4),6).ne.0) then
00161         if (first_waning) call print_warning  &
00162           ("hours not matching model dates (0,6,12,18) are rejecting and not
       shown in output")
```

```
00163         first_waning=.false.
00164         cycle
00165       endif
00166     endif
00167
00168     do i = 1, size(model)
00169       if(model(i)%if) then
00170
00171         select case (model(i)%dataname)
00172         case ("SP", "T", "GP", "VT", "VSH")
00173           if (model(i)%autoload                                    &
00174             .and.                                                  &
00175             .not.(                                                 &
00176             model(i)%autoloadname.eq."ERA"                         &
00177             .and.(any(model(i)%dataname.eq.["GP","VT","VSH"])))) &
00178             then
00179
00180             if ( &
00181               (idate.eq.1 &
00182               .or. .not. date(idate)%date(1).eq.date(idate-1)%date(1) &
00183               )) then
00184
00185               call model_aliases(model(i), year=date(idate)%date(1))
00186             endif
00187
00188           else if (model(i)%autoload) then
00189             if (                                                   &
00190               (idate.eq.1                                          &
00191               .or. .not.(                                          &
00192               date(idate)%date(1).eq.date(idate-1)%date(1)         &
00193               .and.date(idate)%date(2).eq.date(idate-1)%date(2)) &
00194               )                                                    &
00195               ) then
00196
00197               call model_aliases( &
00198                 model(i), year=date(idate)%date(1), month=date(idate)%date(2))
00199             endif
00200           endif
00201
00202           if (size(date).eq.0.and.model(i)%exist) then
00203             stop "temporary"
00204             call get_variable(model(i))
00205           elseif(model(i)%exist) then
00206             call get_variable(model(i), date = date(idate)%date)
00207           endif
00208
00209         end select
00210       endif
00211     enddo
00212
00213     if (any(.not.model%exist).and..not.output%nan) cycle
00214
00215     if (level%all.and..not.allocated(level%level)) then
00216       allocate(level%level(size(model(ind%model%gp)%level)))
00217       level%level=model(ind%model%gp)%level
00218     endif
00219
00220     ! sort levels for 3D method
00221     call bubble_sort(level%level)
00222
00223     ! if ocean mass should be conserved (-O C)
00224     if (ocean_conserve_mass) then
00225       if (ind%model%sp.ne.0 .and. ind%model%ls.ne.0) then
00226         if(size(date).eq.0) then
00227           call conserve_mass(model(ind%model%sp), model(ind%model%ls), &
00228             inverted_landsea_mask = inverted_landsea_mask)
00229         else
00230           call conserve_mass(model(ind%model%sp), model(ind%model%ls), &
00231             date=date(idate)%date, &
00232             inverted_landsea_mask = inverted_landsea_mask)
00233         endif
00234       endif
00235     endif
00236
00237     ! calculate total mass if asked for
00238     if (ind%moreverbose%t.ne.0) then
00239       if (size(date).eq.0) then
00240         call total_mass(model(ind%model%sp))
00241       else
00242         call total_mass(model(ind%model%sp), date=date(idate)%date)
00243       endif
00244     endif
00245
00246
00247     do isite = 1, size(site)
00248       iprogress = iprogress + 1
00249
```

```
00250       if (idate.gt.0) then
00251         write(output%unit, '(f12.3,x,i4.4,5(i2.2),x)', advance="no") &
00252           date(idate)%mjd, date(idate)%date
00253       endif
00254
00255       write (output%unit, '(a8,2(x,f9.4),x,f9.3,$)' ), &
00256         site(isite)%name,                              &
00257         site(isite)%lat,                               &
00258         site(isite)%lon,                               &
00259         site(isite)%height
00260
00261       if (method(1)) then
00262         write (output%unit, "("// output%form // '$)'), &
00263           admit(                                        &
00264           site(isite),                                 &
00265           date=date(idate)%date                        &
00266           )
00267       endif
00268
00269       if (method(2).or.method(3)) then
00270         ! perform convolution
00271         call convolve(site(isite), date = date(idate))
00272       endif
00273
00274       write(output%unit,*)
00275
00276       if (output%unit.ne.output_unit.and..not.(quiet.and.quiet_step.eq.0)) then
00277         open(unit=output_unit, carriagecontrol='fortran')
00278         call cpu_time(cpu(2))
00279         call progress(                        &
00280           100*iprogress/(max(size(date),1) &
00281           *max(size(site),1)),             &
00282           cpu(2)-cpu(1),                   &
00283           every=quiet_step                 &
00284           )
00285       endif
00286     enddo
00287   enddo
00288
00289   ! execution time-stamp
00290   call cpu_time(cpu(2))
00291   if (output%unit.ne.output_unit.and..not.(quiet.and.quiet_step.eq.0)) then
00292     call progress(100*iprogress/(max(size(date),1)*max(size(site),1)), cpu(2)-
      cpu(1), every=1)
00293     close(output_unit)
00294   endif
00295   write(log%unit, '("Execution time:",1x,f10.4," seconds")') cpu(2)-cpu(1)
00296   if (output%time) write(output%unit, '("Execution time:",1x,f10.4," seconds")'
      ) cpu(2)-cpu(1)
00297   write(log%unit, form_separator)
00298 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

```
00001 !> \file
00002 module mod_admit
00003   use mod_constants, only: dp
00004
00005   implicit none
00006
00007 contains
00008 ! ================================================================================
00009 ! ================================================================================
00010 real(dp) function admit(site_, date)
```

```
00011   use mod_cmdline, only: ind, info, admitance
00012   use mod_data, only: get_value, model
00013   use mod_utilities, only: r2d
00014   use mod_atmosphere, only: standard_pressure
00015   use mod_site
00016   use mod_cmdline, only: transfer_sp
00017
00018   real(dp) :: val, rsp, t !, hrsp
00019   type(site_info) :: site_
00020   integer, optional :: date(6)
00021   integer :: i
00022   logical, save :: first_warning=.true.
00023
00024
00025   if (site_%lp%if) then
00026     val=0
00027     do i=1,size(site_%lp%date)
00028       if(all(site_%lp%date(i,1:6).eq.date(1:6))) then
00029         val=site_%lp%data(i)
00030         exit
00031       endif
00032       if(i.eq.size(site_%lp%date)) then
00033         if(first_warning) call print_warning("date not found in @LP")
00034         val=sqrt(-1.)
00035       endif
00036
00037     enddo
00038   else
00039     ! get SP
00040     if (ind%model%sp.ne.0                          &
00041       .and.(model(ind%model%sp)%if                &
00042       .or. model(ind%model%sp)%if_constant_value) &
00043       ) then
00044       call get_value(                    &
00045         model=model(ind%model%sp),       &
00046         lat=site_%lat,                   &
00047         lon=site_%lon,                   &
00048         val=val,                         &
00049         level=1,                         &
00050         method = info(1)%interpolation, &
00051         date=date                        &
00052         )
00053     else
00054       call print_warning("@SP is required with -M1D", error=.true.)
00055     endif
00056   endif
00057
00058
00059   ! get RSP
00060   if (ind%model%rsp.ne.0) then
00061     call get_value(                   &
00062       model=model(ind%model%rsp),     &
00063       lat=site_%lat,                  &
00064       lon=site_%lon,                  &
00065       val=rsp,                        &
00066       level=1,                        &
00067       method = info(1)%interpolation &
00068       )
00069   endif
00070
00071   if (transfer_sp%if) then
00072     if (ind%model%h.eq.0 ) then
00073       if (first_warning) call print_warning("transfer on topo but no @H")
00074     endif
00075
00076     ! get T
00077     if (ind%model%t.ne.0) then
00078       call get_value(                    &
00079         model=model(ind%model%t),      &
00080         lat=site_%lat,                 &
00081         lon=site_%lon,                 &
00082         val=t,                         &
00083         level=1,                       &
00084         method=info(1)%interpolation, &
00085         date=date                      &
00086         )
00087     endif
00088
00089     ! transfer SP
00090     if (site_%hp%if.and..not.isnan(val)) then
00091       val = standard_pressure(                &
00092         height=site_%height,                 &
00093         h_zero=site_%hp%val,                 &
00094         p_zero=val,                          &
00095         method=transfer_sp%method,           &
00096         temperature=t,                       &
00097         use_standard_temperature             &
```

```
00098         = ind%model%t.eq.0,                &
00099         nan_as_zero=.false.)
00100     endif
00101
00102     ! if (ind%model%hrsp.ne.0 .and.ind%model%rsp.ne.0)  then
00103     ! call get_value (                 &
00104     ! model=model(ind%model%hrsp),   &
00105     ! lat=site_%lat,                 &
00106     ! lon=site_%lon,                 &
00107     ! val=hrsp,                      &
00108     ! level=1,                       &
00109     ! method = info(1)%interpolation &
00110     ! )
00111
00112     ! rsp = standard_pressure(     &
00113     ! height=site_%height,       &
00114     ! h_zero=hrsp,               &
00115     ! p_zero=rsp,                &
00116     ! method=transfer_sp%method, &
00117     ! temperature=t,             &
00118     ! use_standard_temperature   &
00119     ! = ind%model%t.eq.0,        &
00120     ! nan_as_zero=.false.)
00121
00122     ! elseif(ind%model%hrsp.ne.0) then
00123     ! if (first_warning) call print_warning("@RSP not found but @HRSP and -U
       given")
00124     ! elseif(ind%model%rsp.ne.0) then
00125     ! if (first_warning) call print_warning("@HRSP not found but @RSP and -U
       given")
00126     ! end if
00127   endif
00128
00129   if (ind%model%rsp.ne.0) val = val-rsp
00130   admit = admitance%value*1.e-2 * val
00131
00132   if (first_warning) first_warning=.false.
00133 end function
00134
00135 ! ========================================================================
00136 !> \date 2013.10.15
00137 !! \author Marcin Rajner
00138 ! ========================================================================
00139 subroutine parse_admit(cmd_line_entry)
00140   use mod_cmdline
00141   use mod_printing
00142   type (cmd_line_arg) :: cmd_line_entry
00143   if (cmd_line_entry%field(1)%subfield(1)%name.ne."") then
00144     read(cmd_line_entry%field(1)%subfield(1)%name, *) admitance%value
00145   endif
00146   if (.not.log%sparse) &
00147     write(log%unit, '(''//form%t2//'',a,x,f6.2,x,a)') "admitance:", admitance
      %value, "uGal/hPa"
00148
00149   ! not sure what trying to achive
00150   ! if (size(cmd_line_entry%field(1)%subfield).gt.1 &
00151   ! .and.cmd_line_entry%field(1)%subfield(2)%name.ne." ") then
00152   ! admitance%level=cmd_line_entry%field(1)%subfield(2)%name
00153   ! else
00154   ! admitance%level="none"
00155   ! endif
00156   ! write(log%unit, form%i2) "level:", admitance%level
00157 end subroutine
00158 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

```
00001 !
      ================================================================================
00002 !> \file
00003 !! \brief This module contains utitlities for computing
00004 !! Atmospheric Gravity Green Functions
00005 !!
00006 !! In this module there are several subroutines for computing
00007 !! AGGF and standard atmosphere parameters
00008 !
      ================================================================================
00009 module mod_aggf
00010   implicit none
00011
00012 contains
00013
00014 !
      ================================================================================
00015 !> Compute first derivative of AGGF with respect to temperature
00016 !! for specific angular distance (psi)
00017 !!
00018 !! optional argument define (-dt;-dt) range
00019 !! See equation 19 in \cite Huang05
00020 !! \author M. Rajner
00021 !! \date 2013-03-19
00022 !! \warning psi in radians
00023 !
      ================================================================================
00024 function aggfd ( &
00025     psi,         &
00026     delta,       &
00027     dz,          &
00028     method,      &
00029     aggfdh,      &
00030     aggfdz,      &
00031     aggfdt,      &
00032     predefined,  &
00033     fels_type,   &
00034     rough)
00035
00036   use mod_constants, only: atmosphere, dp
00037
00038   real(dp), intent (in) :: psi
00039   real(dp), intent (in), optional :: delta
00040   real(dp), intent (in), optional :: dz
00041   logical, intent (in), optional :: aggfdh, aggfdz, aggfdt, predefined, rough
00042   real(dp) :: aggfd
00043   real(dp) :: delta_
00044   character (len=*), intent(in), optional  :: method, fels_type
00045
00046   delta_ = 10. ! Default value
00047   if (present(delta))  delta_ = delta
00048
00049   if(present(aggfdh).and.aggfdh) then
00050     aggfd = (                &
00051       + aggf(psi,           &
00052       h=+delta_,             &
00053       dz=dz,                 &
00054       method=method,         &
00055       predefined=predefined, &
00056       fels_type=fels_type,   &
00057       rough=rough)           &
00058       - aggf(psi,           &
00059       h=-delta_,             &
00060       dz=dz,                 &
00061       method=method,         &
00062       predefined=predefined, &
00063       fels_type=fels_type,   &
00064       rough=rough))          &
00065       / ( 2. * delta_)
00066   else if(present(aggfdz).and.aggfdz) then
00067     aggfd = (                &
00068       + aggf(psi,           &
```

```
00069       zmin = +delta_,         &
00070       dz=dz,                  &
00071       method = method,        &
00072       predefined=predefined, &
00073       fels_type=fels_type,    &
00074       rough=rough)            &
00075       - aggf(psi,            &
00076       zmin = -delta_,         &
00077       dz=dz,                  &
00078       method = method,        &
00079       predefined=predefined, &
00080       fels_type=fels_type,    &
00081       rough=rough))           &
00082       / ( 2. * delta_)
00083   else if(present(aggfdt).and.aggfdt) then
00084     aggfd = (                 &
00085       + aggf(psi,            &
00086       t_zero = +delta_,       &
00087       dz=dz,                  &
00088       method = method,        &
00089       predefined=predefined, &
00090       fels_type=fels_type,    &
00091       rough=rough)            &
00092       - aggf(psi,            &
00093       t_zero = -delta_,       &
00094       dz=dz,                  &
00095       method = method,        &
00096       predefined=predefined, &
00097       fels_type=fels_type,    &
00098       rough=rough))           &
00099       / ( 2. * delta_)
00100   endif
00101 end function
00102
00103 !
      ================================================================================
00104 !> This function computes the value of atmospheric gravity green functions
00105 !! (AGGF) on the basis of spherical distance (psi)
00106 !! \author Marcin Rajner
00107 !! \date 2013.07.15
00108 !! \warning psi in radians h in meter
00109 !! t_zero is actually delta_t so if t_zero=10 (t_zero=288.15+10)
00110 !
      ================================================================================
00111 function aggf (         &
00112     psi,                &
00113     zmin, &
00114     zmax, &
00115     dz,     &
00116     t_zero,             &
00117     h,                  &
00118     first_derivative_h, &
00119     first_derivative_z, &
00120     fels_type,          &
00121     method,             &
00122     predefined,         &
00123     rough)
00124
00125   use mod_constants, only: dp, pi, earth, gravity, atmosphere,
      R_air
00126   use mod_utilities, only: d2r
00127   use mod_atmosphere
00128   use mod_normalization, only : green_normalization
00129
00130   real(dp), intent(in)          :: psi ! spherical distance from site [rad]
00131   real(dp), intent(in),optional :: &
00132     zmin,                           &   ! minimum height, starting point [m]
      (default = 0)
00133     zmax,                           &   ! maximum height, ending point   [m]
      (default = 60000)
00134     dz,                             &   ! integration step               [m]
      (default = 0.1 -> 10 cm)
00135     t_zero,                         &   ! temperature at the surface     [K]
      (default = 15°C i.e., 288.15=t0)
00136     h                                   ! station height                 [m]
      (default = 0)
00137   logical, intent(in), optional :: &
00138     first_derivative_h, first_derivative_z, predefined, rough
00139   character (len=*), intent(in), optional  :: fels_type, method
00140   character (len=20) :: old_method
00141   real(dp) :: aggf
00142   real(dp) :: zmin_, zmax_, dz_, h_
00143   real(dp) :: j_aux
00144   real(dp) :: rho, l, deltat
00145
00146   real(dp), dimension(:), allocatable, save :: heights, pressures
00147   integer :: i
```

```
00148
00149   zmin_ = 0.
00150   zmax_ = 60000.
00151   dz_   = 0.1
00152   h_    = 0.
00153
00154   aggf=0.
00155
00156   if (present(zmin)) zmin_ = zmin
00157   if (present(zmax)) zmax_ = zmax
00158   if (present(  dz))   dz_ = dz
00159   if (present(   h))    h_ = h
00160   if (present(t_zero)) deltat=t_zero
00161
00162   if(allocated(heights)) then
00163     if ( &
00164       ((zmin_ +dz_/2).ne.heights(1)) &
00165       .or.abs((zmax_-dz_/2)-heights(size(heights))).gt.zmax_/1e6 &
00166       .or.nint((zmax_-zmin_)/dz_).ne.size(heights) &
00167       .or. (present(predefined)) &
00168       .or. method.ne.old_method &
00169       .or. present(t_zero) &
00170       ) then
00171       deallocate(heights)
00172       deallocate(pressures)
00173     endif
00174   endif
00175
00176   if (.not.allocated(heights))  then
00177     allocate(heights(nint((zmax_-zmin_)/dz_)))
00178     allocate(pressures(size(heights)))
00179     do i = 1, size(heights)
00180       heights(i) = zmin_ &
00181         + dz_/2  &
00182         + (i-1) * dz_
00183     enddo
00184
00185     if (present(rough).and.rough) then
00186       ! do not use rough! it is only for testing
00187       do i = 1, size(heights)
00188         pressures(i) = standard_pressure( &
00189           heights(i),                     &
00190           method=method,                  &
00191           dz=dz,                          &
00192           use_standard_temperature=.true.  &
00193           )
00194       enddo
00195     else
00196       pressures(1) = standard_pressure(     &
00197         heights(1),                         &
00198         method = method,                    &
00199         h_zero = zmin_,                     &
00200         dz = dz,                            &
00201         fels_type=fels_type,                &
00202         use_standard_temperature=.true.,    &
00203         temperature = standard_temperature( &
00204         zmin_, fels_type=fels_type)+deltat  &
00205         )
00206       do i = 2, size(heights)
00207         pressures(i) = standard_pressure(                 &
00208           heights(i),                                     &
00209           p_zero = pressures(i-1),                        &
00210           h_zero = heights(i-1),                          &
00211           method = method,                                &
00212           dz = dz,                                        &
00213           fels_type=fels_type,                            &
00214           use_standard_temperature=.true.,                &
00215           temperature = standard_temperature(heights(i-1), &
00216           fels_type=fels_type)+deltat                     &
00217           )
00218       enddo
00219     endif
00220   endif
00221   old_method=method
00222
00223   do i = 1, size(heights)
00224     l = ((earth%radius + heights(i))**2 + (earth%radius + h_)**2 &
00225       - 2.*(earth%radius + h_)*(earth%radius+heights(i))*cos(psi))**(0.5)
00226     rho = pressures(i)/ r_air / (deltat+standard_temperature(heights(i),
    fels_type=fels_type))
00227     if (present(first_derivative_h) .and. first_derivative_h) then
00228       ! first derivative (respective to station height)
00229       ! micro Gal height / m
00230       ! see equation 22, 23 in \cite Huang05
00231       j_aux =  ((earth%radius + heights(i) )**2)*(1.-3.*((cos(psi))**2)) -2.*(
    earth%radius + h_)**2  &
00232         + 4.*(earth%radius+h_)*(earth%radius+heights(i))*cos(psi)
```

```
00233       aggf =  aggf + rho * (  j_aux  /  l**5  ) * dz_
00234
00235     else if (present(first_derivative_z) .and. first_derivative_z) then
00236       ! first derivative (respective to column height)
00237       ! according to equation 26 in \cite Huang05
00238       ! micro Gal / hPa / m
00239       if (i.gt.1) exit
00240       aggf = rho *( ((earth%radius + heights(i))*cos(psi)-(earth%radius +
      h_)) / (l**3))
00241     else
00242       ! GN microGal/hPa
00243       aggf = aggf &
00244         -rho*((earth%radius +heights(i))*cos(psi) - (earth%radius + h_)) / (l**
      3.)  * dz_
00245     endif
00246   enddo
00247   aggf = aggf/atmosphere%pressure%standard*gravity%constant*
      green_normalization("m", psi=psi)
00248 end function
00249
00250 !
       ================================================================================
00251 !> Compute AGGF GN for thin layer
00252 !!
00253 !! Simple function added to provide complete module
00254 !! but this should not be used for atmosphere layer
00255 !! See eq p. 491 in \cite Merriam92
00256 !! \author M. Rajner
00257 !! \date 2013-03-19
00258 !! \warning psi in radian
00259 !! \todo explanaition ??
00260 !
       ================================================================================
00261 function gn_thin_layer (psi)
00262   use mod_constants, only: dp
00263   real(dp), intent(in) :: psi
00264   real(dp) :: gn_thin_layer
00265
00266   gn_thin_layer = 1.627 * psi / sin( psi / 2. )
00267 end function
00268
00269
00270 !
       ================================================================================
00271 !> \brief Bouger plate computation
00272 !!
00273 !
       ================================================================================
00274 real(dp) function bouger (h, R )
00275   use mod_constants, only: dp, gravity, pi
00276   real(dp), intent(in), optional :: r !< height of point above the cylinder
00277   real(dp), intent(in) ::  h
00278
00279   if (present( r ) ) then
00280     bouger = h + r - sqrt(r**2+h**2)
00281   else
00282     bouger = h
00283   endif
00284   bouger = 2 * pi * gravity%constant * bouger
00285   return
00286 end function
00287
00288 !
       ================================================================================
00289 !> Bouger plate computation
00290 !!
00291 !! see eq. page 288 \cite Warburton77
00292 !! \date 2013-03-18
00293 !! \author M. Rajner
00294 !
       ================================================================================
00295 function simple_def (R)
00296   use mod_constants, only: dp, earth
00297   real(dp) :: r, delta
00298   real(dp) :: simple_def
00299
00300   delta = 0.22e-11 * r
00301   simple_def = earth%gravity%mean / earth%radius *1000 * &
00302     delta * ( 2. - 3./2. * earth%density%crust / earth%density%mean &
00303     -3./4. * earth%density%crust / earth%density%mean * sqrt(2* (1. )) &
00304     ) * 1000
00305 end function
00306
00307 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

```
00001 !> \file
00002 !! \brief This module gather cmd line arguments
00003 !!
00004 !! it allows to specify commands with or without spaces therefore it is
00005 !! convienient to use with auto completion of names
00006 ! ================================================================================
00007 module mod_cmdline
00008     use mod_constants, only: dp
00009
00010     implicit none
00011
00012     !-----------------------------------------------
00013     ! command line entry
00014     !-----------------------------------------------
00015     type subfield_info
00016       character (len=100) :: name
00017       character (len=100) :: dataname
00018     end type
00019     type field_info
00020       character (len=355) :: full
00021       type(subfield_info), allocatable, &
00022           dimension(:) :: subfield
00023     end type
00024     type cmd_line_arg
00025       character(2) :: switch
00026       type (field_info), allocatable, &
00027           dimension(:) :: field
00028       character (len=455) :: full
00029     end type
00030     type(cmd_line_arg), allocatable, dimension(:) :: cmd_line
00031
00032     private :: check_if_switch_or_minus
00033
00034     type moreverbose_info
00035       character(60) :: name
```

```
00036       character(30):: dataname
00037       logical  :: sparse=.false.
00038       logical :: first_call = .true.
00039       integer :: unit
00040       logical :: noclobber = .false.
00041     end type
00042     type(moreverbose_info), allocatable, dimension(:) ::
      moreverbose
00043
00044     !-----------------------------------------------------
00045     ! info
00046     !-----------------------------------------------------
00047     type range
00048       real(dp):: start
00049       real(dp):: stop
00050       real(dp):: step
00051       integer :: denser
00052       real(dp):: stop_3d
00053       ! logical :: stop_3d_if
00054     end type
00055     type info_info
00056       type (range):: distance,azimuth, height
00057       character (1) :: interpolation
00058     end type
00059     type(info_info), dimension(:), allocatable:: info
00060
00061     !-----------------------------------------------------
00062     ! general settings
00063     !-----------------------------------------------------
00064     logical :: &
00065         inverted_barometer     = .true.  , &
00066         non_inverted_barometer = .false. , &
00067         ocean_conserve_mass    = .false. , &
00068         inverted_landsea_mask  = .false. , &
00069         optimize               = .false. , &
00070         quiet                  = .false.
00071     integer :: quiet_step=50
00072
00073     type transfer_sp_info
00074       logical :: if = .false.
00075       ! by default with 2D method pressure is transfered
00076       ! on topography (@H)
00077       character(20) :: method="standard"
00078     end type
00079     type(transfer_sp_info) transfer_sp
00080
00081     type warnings_info
00082       logical :: &
00083           if = .true., &
00084           strict=.false., &
00085           time=.false.
00086     end type
00087     type(warnings_info) warnings
00088
00089     type model_index
00090       integer(2) :: sp, t, rsp, ewt, h, ls, hp, hrsp, gp, vt, vsh
00091     end type
00092     type poly_index
00093       integer(2) :: e, n
00094     end type
00095     type moreverbose_index
00096       integer(2) :: p, g, t, a, d, l, n, r, s, o, b, j, v
00097     end type
00098     type green_index
00099       integer(2) :: &
00100           gn          = 0,  & ! green newtonian   - with SP  in Pa
00101           ge          = 0,  & ! green elastic     - with SP  in Pa
00102           gegdt       = 0,  & ! green elastic     - first derivative of gravity
      part respect to temp (see Guo et al., 2004)
00103           gr          = 0,  & ! green radial      - with EWT in mm
00104           ghn         = 0,  & ! green horizontal  - with EWT in mm
00105           ghe         = 0,  & ! green horizontal  - with EWT in mm
00106           gg          = 0,  & ! green gravimetric - with SP  in Pa
00107           ! (like elastic but uses green not normalized according to Merriam)
00108       gndt        = 0,  & ! first derivative respect to temperature
00109           gndh        = 0,  & ! first derivative respect to station height
00110           gndz        = 0,  & ! first derivative respect to column height
00111           gndz2       = 0,  & ! second derivative respect to column height
00112           gnc         = 0,  & ! compute aggf every time
00113           g3d
00114     end type
00115     type index_info
00116       type (model_index)       :: model
00117       type (moreverbose_index) :: moreverbose
00118       type (green_index) :: green
00119       type (poly_index) :: polygon
00120     end type
```

```
00121     type(index_info) :: ind
00122
00123     type admitance_info
00124       logical :: if
00125       real(dp):: value = -0.3
00126     end type
00127     type(admitance_info) :: admitance
00128
00129     logical :: method(3)
00130
00131     ! point mass - method3d(1)=.true.
00132     ! potential  - method3d(2)=.true.
00133     ! cylinder   - method3d(3)=.true.
00134     logical :: method3d(3)
00135     logical :: method3d_compute_reference  = .false.
00136     real    :: method3d_refinment_distance = 0.1
00137     logical :: dryrun
00138
00139     logical :: result_total=.false., result_component=.true.
00140 contains
00141   !
      ================================================================================
00142   !> This routine collect command line arguments to one matrix depending on
00143   !! given switches and separators
00144   !!
00145   !! \date 2013.05.21
00146   !! \author Marcin Rajner
00147   !
      ================================================================================
00148   subroutine collect_args (dummy)
00149       use mod_utilities, only: ntokens, count_separator
00150       character(*) :: dummy
00151       character(455) :: dummy_aux, dummy_aux2
00152       integer :: i, j, n
00153       integer :: indeks_space, indeks_comma, indeks_at, indeks_colon
00154
00155       allocate(cmd_line(ntokens(dummy)))
00156       do i=1, ntokens(dummy)
00157         indeks_space = index(dummy," ")
00158         cmd_line(i)%full= dummy(1:indeks_space-1)
00159         cmd_line(i)%switch=cmd_line(i)%full(1:2)
00160         allocate(cmd_line(i)%field (count_separator(cmd_line(i)%full,",") + 1))
00161
00162         dummy_aux = cmd_line(i)%full(3:)
00163         do j=1,size(cmd_line(i)%field)
00164           indeks_comma=index(dummy_aux,",")
00165           if (indeks_comma.gt.0) then
00166             cmd_line(i)%field(j)%full=dummy_aux(1:indeks_comma-1)
00167           else
00168             cmd_line(i)%field(j)%full=dummy_aux
00169           endif
00170
00171           allocate(cmd_line(i)%field(j)%subfield &
00172               (count_separator(cmd_line(i)%field(j)%full,":") + 1))
00173           dummy_aux2 = cmd_line(i)%field(j)%full
00174           do n = 1, count_separator(cmd_line(i)%field(j)%full,":")+1
00175             indeks_colon=index(dummy_aux2,":")
00176             if (indeks_colon.gt.0) then
00177               cmd_line(i)%field(j)%subfield(n)%name=dummy_aux2(1:indeks_colon-1
      )
00178             else
00179               cmd_line(i)%field(j)%subfield(n)%name=dummy_aux2
00180             endif
00181             dummy_aux2=dummy_aux2(indeks_colon+1:)
00182             indeks_at=index(cmd_line(i)%field(j)%subfield(n)%name,"@")
00183             if (indeks_at.gt.0) then
00184               cmd_line(i)%field(j)%subfield(n)%dataname = &
00185                   cmd_line(i)%field(j)%subfield(n)%name(indeks_at+1:)
00186               cmd_line(i)%field(j)%subfield(n)%name = &
00187                   cmd_line(i)%field(j)%subfield(n)%name(1:indeks_at-1)
00188             else
00189               cmd_line(i)%field(j)%subfield(n)%dataname = " "
00190             endif
00191           enddo
00192           dummy_aux=dummy_aux(indeks_comma+1:)
00193         enddo
00194         dummy= dummy(indeks_space+1:)
00195       enddo
00196   end subroutine
00197
00198   !
      ================================================================================
00199   !> This subroutine removes unnecesary blank spaces from cmdline entry
00200   !!
00201   !! Marcin Rajner
00202   !! \date 2013-05-13
00203   !! allows specification like '-F file' and '-Ffile'
```

```
00204   !! but  if -[0,9] it is treated as number belonging to switch (-S -2)
00205   !! but  if -[\s,:] do not start next command line option
00206   !
      ================================================================================
00207   subroutine get_command_cleaned(dummy)
00208       character(*), intent(out) :: dummy
00209       character(355) :: a, b, arg
00210       integer :: i
00211       dummy=" "
00212       do i = 1, iargc()
00213         call get_command_argument(i,a)
00214         call get_command_argument(i+1,b)
00215         if (check_if_switch_or_minus(a)) then
00216           arg = trim(a)
00217         else
00218           arg=trim(arg)//trim(a)
00219         endif
00220         if(check_if_switch_or_minus(b).or.i.eq.iargc()) then
00221           if(trim(dummy).eq."") then
00222             dummy=trim(arg)
00223           else
00224             dummy=trim(dummy)//" "//trim(arg)
00225           endif
00226         endif
00227       enddo
00228   end subroutine
00229
00230   !
      ================================================================================
00231   !> Check if - starts new option in command line or is just a minus in command
00232   !! line entry
00233   !!
00234   !! if after '-' is space or number or ',' or ':' (field separators) do not
      start
00235   !! next option for command line
00236   !! If switch return .true. otherwise return .false
00237   !!
00238   !! \author M. Rajner
00239   !! \date 2013-03-19
00240   !
      ================================================================================
00241   function check_if_switch_or_minus(dummy)
00242       use mod_utilities, only: is_numeric
00243       logical:: check_if_switch_or_minus
00244       character(*) :: dummy
00245
00246       check_if_switch_or_minus = .false.
00247       if (dummy(1:1).eq."-") check_if_switch_or_minus = .true.
00248       if (dummy(2:2).eq." ") check_if_switch_or_minus = .false.
00249       if (dummy(2:2).eq.",") check_if_switch_or_minus = .false.
00250       if (dummy(2:2).eq.":") check_if_switch_or_minus = .false.
00251       if (is_numeric(dummy(2:2))) check_if_switch_or_minus = .false.
00252   end function
00253
00254 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

```
00001 !> \file
00002 module mod_green
00003   use mod_constants, only: dp
```

```
00004
00005   implicit none
00006   !-----------------------------------------------------
00007   ! Greens function
00008   !-----------------------------------------------------
00009   type green_functions
00010     character (len=255) :: name
00011     character (len=25) :: dataname
00012     integer, dimension(2) :: column
00013     character(10), dimension(2) :: columndataname
00014     real(dp), allocatable,dimension(:) :: distance
00015     real(dp), allocatable,dimension(:) :: data
00016   end type
00017   type(green_functions), allocatable, dimension(:) :: green
00018
00019   real(dp), allocatable, dimension(:) :: result
00020
00021   type green_common_info
00022     real(dp), allocatable, dimension(:) :: distance
00023     real(dp), allocatable, dimension(:) :: start
00024     real(dp), allocatable, dimension(:) :: stop
00025     real(dp), allocatable, dimension(:,:) :: data
00026     character (len=25), allocatable, dimension(:) :: dataname
00027     logical, allocatable, dimension(:) :: elastic
00028   end type
00029   type(green_common_info), allocatable, dimension(:) ::
      green_common
00030
00031   integer :: gnc_looseness=1
00032
00033 contains
00034 ! ========================================================================
00035 !> This subroutine parse -G option -- Greens function.
00036 !!
00037 !! This subroutines takes the -G argument specified as follows:
00038 !!   -G
00039 !! \author M. Rajner
00040 !! \date 2013-03-06
00041 ! ========================================================================
00042 subroutine parse_green (cmd_line_entry)
00043   use mod_utilities, only: file_exists, is_numeric
00044   use mod_cmdline
00045   use mod_printing
00046   type (cmd_line_arg), optional  :: cmd_line_entry
00047   integer :: i, ii
00048
00049   if (allocated(green)) then
00050     call print_warning("repeated")
00051     return
00052   endif
00053
00054   if (method(3)) then
00055     if (present(cmd_line_entry)) then
00056       allocate (green(size(cmd_line_entry%field)+1))
00057     else
00058       allocate (green(1))
00059     endif
00060     ind%green%g3d=ubound(green,1)
00061     green(ind%green%g3d)%name="merriam"
00062     green(ind%green%g3d)%column=[1, 2]
00063     green(ind%green%g3d)%dataname="G3D"
00064     call read_green(green(ind%green%g3d))
00065   else
00066     allocate (green(size(cmd_line_entry%field)))
00067   endif
00068
00069   if (present(cmd_line_entry)) then
00070     do i = 1, size(cmd_line_entry%field)
00071
00072       if (.not.log%sparse) &
00073         write(log%unit, form%i2) trim(basename(trim(cmd_line_entry%field(i)
      %full)))
00074
00075       green(i)%name = cmd_line_entry%field(i)%subfield(1)%name
00076
00077       if (i.gt.1.and.cmd_line_entry%field(i)%subfield(1)%name.eq."") then
00078         green(i)%name = green(i-1)%name
00079       endif
00080
00081       if (any(green%dataname.eq.cmd_line_entry%field(i)%subfield(1)%dataname ))
       then
00082         call print_warning("repeated dataname for Green")
00083         continue
00084       else
00085         green(i)%dataname = cmd_line_entry%field(i)%subfield(1)%dataname
00086       endif
00087
```

```
00088       do ii=1, 2
00089         green(i)%column(ii) = green(i-1)%column(ii)
00090         green(i)%columndataname(ii) = green(i-1)%columndataname(ii)
00091         if(is_numeric(cmd_line_entry%field(i)%subfield(ii+1)%name ) ) then
00092           read(cmd_line_entry%field(i)%subfield(ii+1)%name, *) green(i)%column(
      ii)
00093           green(i)%columndataname(ii) = cmd_line_entry%field(i)%subfield(ii+1)
      %dataname
00094         endif
00095
00096       enddo
00097       if (green(i)%dataname.eq."GNc") then
00098         if(is_numeric(cmd_line_entry%field(i)%subfield(2)%name)) then
00099           read(cmd_line_entry%field(i)%subfield(2)%name, *) gnc_looseness
00100           if (gnc_looseness.lt.1) then
00101             call print_warning("gnc_looseness < 1", error=.true.)
00102           endif
00103         endif
00104       endif
00105
00106       call read_green(green(i))
00107
00108     enddo
00109   endif
00110
00111   ! check completness
00112   ! if ( &
00113   ! ! any(green%name.eq."/home/mrajner/src/grat/dat/merriam_green.dat" &
00114   ! ! .and. green%dataname.eq."GNdz" ) &
00115   ! ! .neqv. &
00116   ! any(green%name.eq."/home/mrajner/src/grat/dat/merriam_green.dat" &
00117   ! .and. green%dataname.eq."GNdz2" ) &
00118   ! ) call print_warning("-G: merriam@GNdz should go with merriam @GNdz2")
00119 end subroutine
00120
00121 ! ========================================================================
00122 !> This subroutine read  green file
00123 ! ========================================================================
00124 subroutine read_green (green, print)
00125   use mod_utilities, only: file_exists, skip_header, r2d, d2r
00126   use iso_fortran_env
00127   use mod_printing
00128   use mod_constants, only: earth, pi
00129   use mod_normalization, only: green_normalization
00130
00131   integer :: lines, fileunit, io_status, i
00132   real (dp), allocatable, dimension(:) :: tmp
00133   type(green_functions) :: green
00134   logical, optional :: print
00135
00136   ! change the paths accordingly
00137   if (.not.file_exists(green%name)       &
00138     .and. (.not. green%name.eq."merriam" &
00139     .and.  .not. green%name.eq."huang"   &
00140     .and.  .not. green%name.eq."rajner"  &
00141     ! this will be feature added for hydrosphere loading later...
00142   ! .and.  .not. green%name.eq."GB" &
00143   )) then
00144     green%name="merriam"
00145   endif
00146
00147   select case (green%name)
00148   case ("merriam", "compute", "/home/mrajner/src/grat/dat/merriam_green.dat")
00149     green%name="/home/mrajner/src/grat/dat/merriam_green.dat"
00150     select case (green%dataname)
00151     case("GN")
00152       green%column=[1, 2]
00153     case("GNdt")
00154       green%column=[1, 3]
00155     case("GNdz")
00156       green%column=[1, 4]
00157     case("GNdz2")
00158       green%column=[1, 5]
00159     case("GE")
00160       green%column=[1, 6]
00161     case("GNc")
00162       green%column=[1, 2]
00163     case("G3D")
00164       green%column=[1, 2]
00165     case default
00166       call print_warning( &
00167         "green type not found", &
00168         more=trim(green%dataname), &
00169         error=.true.)
00170     endselect
00171
00172   case ("huang", "/home/mrajner/src/grat/dat/huang_green.dat" )
```

```
00173     green%name="/home/mrajner/src/grat/dat/huang_green.dat"
00174     select case (green%dataname)
00175     case("GN")
00176       green%column=[1, 2]
00177     case("GNdt")
00178       green%column=[1, 3]
00179     case("GNdh")
00180       green%column=[1, 4]
00181     case("GNdz")
00182       green%column=[1, 5]
00183     case default
00184       call print_warning( &
00185         trim(green%dataname) //" not found in " &
00186         // trim(green%name), error=.true.)
00187     endselect
00188
00189   case ("rajner", "/home/mrajner/src/grat/dat/rajner_green.dat")
00190     green%name="/home/mrajner/src/grat/dat/rajner_green.dat"
00191     select case (green%dataname)
00192     case("GN")
00193       green%column=[1, 2]
00194     case("GNdt")
00195       green%column=[1, 3]
00196     case("GNdh")
00197       green%column=[1, 4]
00198     case("GNdz")
00199       green%column=[1, 5]
00200     case default
00201       call print_warning( &
00202         trim(green%dataname) //" not found in " &
00203         // trim(green%name), error=.true.)
00204       call print_warning(green%dataname //"not found in " // green%name, &
00205         error=.true.)
00206     endselect
00207   endselect
00208
00209   if(green%column(1).ne.0 .and. green%column(2).ne.0) then
00210     allocate(tmp(max(green%column(1), green%column(2))))
00211     lines = 0
00212     open (newunit =fileunit, file=green%name, action="read", status="old")
00213     do
00214       call skip_header(fileunit)
00215       read (fileunit, *, iostat = io_status) tmp
00216       if (io_status == iostat_end) exit
00217       lines = lines + 1
00218     enddo
00219
00220     allocate (green%distance(lines))
00221     allocate (green%data(lines))
00222     rewind(fileunit)
00223     lines = 0
00224     do
00225       call skip_header(fileunit)
00226       lines = lines + 1
00227       read (fileunit, *, iostat = io_status) tmp
00228       if (io_status == iostat_end) then
00229         close(fileunit)
00230         exit
00231       endif
00232       green%distance(lines) = tmp(green%column(1))
00233       green%data(lines)     = tmp(green%column(2))
00234     enddo
00235     deallocate(tmp)
00236   endif
00237
00238   ! file specific
00239   if (green%name.eq."/home/mrajner/src/grat/dat/merriam_green.dat") then
00240     select case(green%dataname)
00241     case("GNdz")
00242       green%data = green%data * 1.e-3
00243     endselect
00244   endif
00245
00246   if (.not.present(print)) then
00247     if (.not.log%sparse) &
00248       write(log%unit, form%i3) &
00249       trim(basename(trim(green%name))), trim(green%dataname), &
00250       "columns:", green%column, &
00251       "lines:", size(green%distance)
00252
00253       if (green%dataname.eq."GNc") then
00254       write(log%unit, form%i3) "gnc loosenes" , gnc_looseness
00255     endif
00256   endif
00257
00258   if (green%columndataname(1).eq."R") then
00259     green%distance=(/ (r2d(green%distance(i)), i=1, size(green%distance)) /)
```

```
00260     write(log%unit, form_63) "conversion: radians --> to degrees"
00261   endif
00262   if (green%columndataname(2).eq."a2f") then
00263     green%data=green%data  / (earth%radius)*1e12 * earth%gravity%mean
00264     write(log%unit, form_63) "conversion: aplo --> to farrell"
00265   endif
00266   if (green%columndataname(2).eq."f2m") then
00267     green%data= &
00268       -green%data * green_normalization("f2m")
00269     write(log%unit, form_63) "conversion: farrell --> to merriam"
00270   endif
00271 end subroutine
00272
00273 ! ========================================================================
00274 !> Unification:
00275 ! ========================================================================
00276 subroutine green_unification ()
00277   use mod_utilities, only: size_ntimes_denser,
      spline_interpolation, d2r
00278   use mod_cmdline,   only: info, moreverbose, ind,
      method3d_compute_reference
00279   use mod_printing
00280   use mod_site, only: site
00281   use mod_aggf, only: aggf
00282
00283   type(green_functions) :: tmpgreen
00284   integer :: i, iinfo, imin, imax, j, ii
00285   integer, allocatable, dimension(:):: which_green, tmp
00286
00287   allocate (green_common(size(info)))
00288   allocate (which_green(size(info)))
00289   allocate (tmp(size(green)))
00290
00291   do iinfo=1, size(info)
00292
00293     if (info(iinfo)%distance%step.eq.0) then
00294       do i=1, size(green)
00295         tmp(i) = count(                                        &
00296           green(i)%distance.le.info(iinfo)%distance%stop       &
00297           .and.green(i)%distance.ge.info(iinfo)%distance%start &
00298           )
00299       enddo
00300
00301       which_green(iinfo) = maxloc(tmp, 1)
00302
00303       imin=minloc( &
00304         abs(green(which_green(iinfo))%distance - info(iinfo)%distance%start), 1
      )-1
00305       imax=minloc( &
00306         abs(green(which_green(iinfo))%distance - info(iinfo)%distance%stop), 1)
      +1
00307
00308       if (imin.lt.1) imin = 1
00309       if (imax.gt.size(green(which_green(iinfo))%distance)) then
00310         imax = size(green(which_green(iinfo))%distance)
00311       endif
00312
00313       if (info(iinfo)%distance%denser.ge.0) then
00314         allocate(tmpgreen%distance(                                  &
00315           size_ntimes_denser(imax-imin+1, info(iinfo)%distance%denser) &
00316           ))
00317
00318         do ii = 1, imax-imin
00319           do j = 1, info(iinfo)%distance%denser
00320             tmpgreen%distance((ii-1)*info(iinfo)%distance%denser+j) = &
00321               green(which_green(iinfo))%distance(imin+ii-1)          &
00322               +(j-1)*(green(which_green(iinfo))%distance(imin+ii)     &
00323               -green(which_green(iinfo))%distance(imin+ii-1))         &
00324               /info(iinfo)%distance%denser
00325           enddo
00326         enddo
00327
00328       else
00329         ! if @DD is negative make distance sparse
00330         allocate(tmpgreen%distance((imax-imin)/-info(iinfo)%distance%denser &
00331           +1+min(1,modulo(imax-imin,-info(iinfo)%distance%denser))))
00332         ii=0
00333         do j=1,imax-imin+1
00334           if (j.eq.imax-imin+1.or.modulo(j-1,info(iinfo)%distance%denser).eq.0)
       then
00335             ii=ii+1
00336             tmpgreen%distance(ii)=green(which_green(iinfo))%distance(j)
00337           endif
00338         enddo
00339       endif
00340
00341       tmpgreen%distance(size(tmpgreen%distance)) = &
```

```
00342         green(which_green(iinfo))%distance(imax)
00343
00344       imin = count(tmpgreen%distance.le.info(iinfo)%distance%start)
00345       imax = size(tmpgreen%distance) - &
00346         count(tmpgreen%distance.ge.info(iinfo)%distance%stop ) + 1
00347
00348       allocate(green_common(iinfo)%distance(imax-imin+1))
00349       green_common(iinfo)%distance =       &
00350         tmpgreen%distance(imin:imax)
00351       green_common(iinfo)%distance(1) =    &
00352         (3/4.*info(iinfo)%distance%start+  &
00353         green_common(iinfo)%distance(2)/4)
00354       green_common(iinfo)%distance(size(green_common(iinfo)%distance)) =      &
00355         (3/4.*info(iinfo)%distance%stop+                                      &
00356         green_common(iinfo)%distance(size(green_common(iinfo)%distance)-1)/4)
00357
00358       allocate(green_common(iinfo)%start(size(green_common(iinfo)%distance)))
00359       allocate(green_common(iinfo)%stop(size(green_common(iinfo)%distance)))
00360
00361       green_common(iinfo)%start=(green_common(iinfo)%distance)
00362
00363       do i =1, size(green_common(iinfo)%distance)
00364
00365         green_common(iinfo)%start(i)=(green_common(iinfo)%distance(i) + &
00366           green_common(iinfo)%distance(i-1) ) / 2.
00367
00368         green_common(iinfo)%stop(i)=(green_common(iinfo)%distance(i) + &
00369           green_common(iinfo)%distance(i+1) ) / 2.
00370
00371       enddo
00372
00373       green_common(iinfo)%start(1)= info(iinfo)%distance%start
00374       green_common(iinfo)%stop(size(green_common(iinfo)%stop)) = &
00375         info(iinfo)%distance%stop
00376       deallocate(tmpgreen%distance)
00377
00378       !@DS =/ 0
00379     else
00380       allocate(green_common(iinfo)%distance( &
00381         ceiling( &
00382         (info(iinfo)%distance%stop - info(iinfo)%distance%start) &
00383         /info(iinfo)%distance%step) &
00384         ))
00385       allocate(green_common(iinfo)%start(size(green_common(iinfo)%distance)))
00386       allocate(green_common(iinfo)%stop(size(green_common(iinfo)%distance)))
00387
00388       green_common(iinfo)%start =                &
00389         [                                        &
00390         (info(iinfo)%distance%start +            &
00391         (i-1)*info(iinfo)%distance%step,         &
00392         i=1, size(green_common(iinfo)%distance)) &
00393         ]
00394
00395       green_common(iinfo)%stop = green_common(iinfo)%start(2:)
00396       green_common(iinfo)%stop(ubound(green_common(iinfo)%stop)) = &
00397         info(iinfo)%distance%stop
00398       green_common(iinfo)%distance = &
00399         (green_common(iinfo)%stop + green_common(iinfo)%start)/2
00400     endif
00401
00402     allocate(green_common(iinfo)%data(size(green_common(iinfo)%distance), size(
      green)))
00403     allocate(green_common(iinfo)%dataname(size(green)))
00404
00405     do i = 1,  size(green_common(iinfo)%data, 2)
00406       call  spline_interpolation(          &
00407         green(i)%distance,                 &
00408         green(i)%data,                     &
00409         size(green(i)%distance),           &
00410         green_common(iinfo)%distance,      &
00411         green_common(iinfo)%data(:, i),     &
00412         size(green_common(iinfo)%distance) &
00413         )
00414       where( &
00415           green_common(iinfo)%distance.gt.green(i)%distance(size(green(i)
      %distance)) &
00416           .or.green_common(iinfo)%distance.lt.green(i)%distance(1) &
00417           )
00418         green_common(iinfo)%data(:, i)=0
00419       end where
00420
00421       green_common(iinfo)%dataname(i) = green(i)%dataname
00422
00423       if(green_common(iinfo)%dataname(i) == "G3D") then
00424         if (method3d_compute_reference) then
```

```
00425           do ii=1,size(green_common(iinfo)%data(:,i))
00426             green_common(iinfo)%data(ii,i) =                  &
00427               aggf(                                           &
00428               psi    = d2r(green_common(iinfo)%distance(ii)), &
00429               dz     = info(iinfo)%height%step,               &
00430               zmin   = info(iinfo)%height%start,              &
00431               zmax   = info(iinfo)%height%stop,               &
00432               method = "standard"                             &
00433               )
00434           enddo
00435         endif
00436       endif
00437
00438     enddo
00439   enddo
00440
00441 end subroutine
00442
00443
00444 ! ========================================================================
00445 !> Perform convolution
00446 !!
00447 !! \date 2013-03-15
00448 !! \author M. Rajner
00449 ! ========================================================================
00450 subroutine convolve(site, date)
00451   use mod_constants
00452   use iso_fortran_env
00453   use mod_site, &
00454     only : site_info, local_pressure_distance
00455   use mod_cmdline
00456   use mod_utilities, &
00457     only: d2r, r2d, datanameunit, mmwater2pascal, countsubstring
00458   use mod_spherical
00459   use mod_data
00460   use mod_date, only : dateandmjd
00461   use mod_polygon
00462   use mod_printing
00463   use mod_normalization, only: green_normalization
00464   use mod_aggf, only: aggf
00465   use mod_atmosphere, only: &
00466     standard_pressure, standard_temperature, virtual_temperature
00467   use mod_3d
00468
00469   type(site_info),  intent(in) :: site
00470   type(dateandmjd), intent(in), optional :: date
00471
00472   integer  :: igreen, idist, iazimuth, nazimuth
00473   real(dp) :: azimuth, dazimuth
00474   real(dp) :: lat, lon, area, tot_area, tot_area_used
00475   real(dp) :: val(size(model)), old_val_sp, old_val_rsp
00476   integer  :: i, j, npoints, iheight, nheight
00477   integer(2) :: iok(size(polygon))
00478
00479   real(dp) :: normalize, aux
00480   real(dp), allocatable, dimension(:) :: azimuths, &
00481     heights, pressures, temperatures
00482   logical :: header_p = .true.
00483
00484   ! real(dp) :: h1,h2, v1,v2, p_int !temporary
00485   real(dp) :: rsp
00486   real(dp), dimension(:), allocatable :: result_partial
00487
00488   logical :: first_reduction
00489   first_reduction=.true.
00490
00491
00492   if (transfer_sp%if) then
00493     if (ind%model%hp.eq.0) call print_warning("no @HP with -U", error=.true.)
00494     if (ind%model%h .eq.0) call print_warning("no @H  with -U", error=.true.)
00495   endif
00496
00497   if(.not.allocated(green_common)) then
00498     call green_unification()
00499   endif
00500
00501   val=0
00502
00503   if (site%lp%if) then
00504     do i=1, size(site%lp%date)
00505
00506       if(all(site%lp%date(i, 1:6).eq.date%date(1:6))) then
00507         val(ind%model%sp) = site%lp%data(i)
00508         exit
00509       endif
00510
00511       val(ind%model%sp) = sqrt(-1.)
```

```
00512       if(i.eq.size(site%lp%date)) &
00513         call print_warning("date not found in @LP")
00514     enddo
00515   endif
00516
00517   if (.not. allocated(result)) then
00518     if (any(green%dataname.eq."GE").and.inverted_barometer &
00519       .and. non_inverted_barometer) then
00520       allocate(result(size(green)+1))
00521     else
00522       allocate(result(size(green)))
00523     endif
00524   endif
00525   if(.not.allocated(result_partial)) allocate(result_partial(size(result)))
00526
00527   npoints       = 0
00528   area          = 0
00529   tot_area      = 0
00530   tot_area_used = 0
00531
00532   result = 0
00533   rsp    = 0
00534
00535   do igreen = 1, size(green_common)
00536     do idist = 1, size(green_common(igreen)%distance)
00537       if (allocated(azimuths)) deallocate (azimuths)
00538
00539       if (info(igreen)%azimuth%step.eq.0) then
00540         nazimuth =                                                    &
00541           (info(igreen)%azimuth%stop-info(igreen)%azimuth%start)/360 *  &
00542           max(int(360*sin(d2r(green_common(igreen)%distance(idist)))), 100) * &
00543           info(igreen)%azimuth%denser
00544         if (nazimuth.eq.0) nazimuth=1
00545         dazimuth= (info(igreen)%azimuth%stop-info(igreen)%azimuth%start)/
      nazimuth
00546       else
00547         dazimuth = info(igreen)%azimuth%step
00548         nazimuth= (info(igreen)%azimuth%stop-info(igreen)%azimuth%start)/
      dazimuth
00549       endif
00550
00551       ! calculate area using spherical formulae
00552       area = spher_area(                          &
00553         d2r(green_common(igreen)%start(idist)), &
00554         d2r(green_common(igreen)%stop(idist)),  &
00555         d2r(dazimuth),                          &
00556         radius=earth%radius,                    &
00557         alternative_method=.true.)
00558
00559       ! normalization according to Merriam (1992)
00560       normalize= 1e8 / &
00561         (green_normalization("m", psi = d2r(green_common(igreen)%distance(idist
      ))))
00562
00563       allocate(azimuths(nazimuth))
00564       azimuths = [(info(igreen)%azimuth%start + (i-1) * dazimuth, i= 1,
      nazimuth)]
00565
00566       do iazimuth  = 1, nazimuth
00567         azimuth = azimuths(iazimuth)
00568
00569         npoints = npoints + 1
00570         tot_area=tot_area+area
00571
00572         ! get lat and lon of point
00573         call spher_trig &
00574           (d2r(site%lat), d2r(site%lon), &
00575           d2r(green_common(igreen)%distance(idist)), d2r(azimuth), lat, lon,
      domain=.true.)
00576
00577         ! read polygons
00578         if (ind%polygon%e.ne.0 .or. ind%polygon%n.ne.0) then
00579           do i =1, size(polygon)
00580             if (polygon(i)%if) then
00581               call chkgon(r2d(lon), r2d(lat), polygon(i), iok(i))
00582             endif
00583           enddo
00584         endif
00585
00586         ! get LS
00587         if (ind%model%ls.ne.0.and.inverted_barometer) then
00588           call get_value(                        &
00589             model(ind%model%ls),                  &
00590             lat    = r2d(lat),                    &
00591             lon    = r2d(lon),                    &
00592             val    = val(ind%model%ls),           &
00593             level  = 1,                           &
```

```
00594             method = info(igreen)%interpolation, &
00595             date   = date%date                   &
00596             )
00597         endif
00598
00599         if (iok(1).eq.1 & .and. int(val(ind%model%ls)).eq.1) then
00600           tot_area_used = tot_area_used +area
00601         endif
00602
00603         ! GE, GN, ...
00604         if (any([&
00605           ind%green%gn,   ind%green%ge, ind%green%gg, &
00606           ind%green%gndt, ind%green%gnc, ind%green%gegdt, &
00607           ind%green%g3d &
00608           ].ne.0) &
00609           ) then
00610
00611           if ( &
00612             ind%model%sp.ne.0.and.(model(ind%model%sp)%if &
00613             .or.model(ind%model%sp)%if_constant_value) &
00614             ) then
00615
00616             ! get SP
00617             if (.not.(site%lp%if                       &
00618               .and.green_common(igreen)%distance(idist) &
00619               .lt.local_pressure_distance)) then
00620               call get_value(                                                &
00621                 model(ind%model%sp), r2d(lat), r2d(lon), val(ind%model%sp), &
00622                 level=1,                                                    &
00623                 method = info(igreen)%interpolation,                        &
00624                 date=date%date)
00625             endif
00626             old_val_sp=val(ind%model%sp)
00627
00628             if (.not.isnan(val(ind%model%sp))) then
00629
00630               ! get RSP if given
00631               if (ind%model%rsp.ne.0) then
00632                 call get_value(
      &
00633                   model(ind%model%rsp), r2d(lat), r2d(lon), val(ind%model%rsp),
       &
00634                   level=1, method = info(igreen)%interpolation)
00635               endif
00636               old_val_rsp=val(ind%model%rsp)
00637
00638               if(transfer_sp%if.and..not.all([ind%model%rsp, ind%model%hrsp]
      .ne.0)) then
00639                 call print_warning("@RSP or @HRSP with -U is missing", error=.
      true.)
00640               else
00641                 call get_value( &
00642                   model(ind%model%hrsp), r2d(lat), r2d(lon), val(ind%model%hrsp
      ), &
00643                   level=1, method = info(igreen)%interpolation)
00644               endif
00645
00646               ! get T
00647               if (ind%model%t.ne.0 &
00648                 .and.( &
00649                 transfer_sp%if &
00650                 .or.any(([ &
00651                 ind%green%gndt, &
00652                 ind%green%gegdt, &
00653                 ind%green%gnc, &
00654                 ind%green%g3d &
00655                 ]).ne.0) &
00656                 ) &
00657                 ) then
00658                 call get_value( &
00659                   model(ind%model%t), r2d(lat), r2d(lon), val(ind%model%t), &
00660                   level=1, method=info(igreen)%interpolation, date=date%date)
00661               endif
00662
00663               ! get HP
00664               if (ind%model%hp.ne.0 &
00665                 .and.( &
00666                 transfer_sp%if &
00667                 .or. ind%green%g3d.ne.0 &
00668                 ) &
00669                 ) then
00670                 call get_value( &
00671                   model(ind%model%hp), r2d(lat), r2d(lon), val(ind%model%hp), &

00672                   level=1, method = info(igreen)%interpolation)
00673               endif
00674
```

```
00675               ! get H
00676               if (ind%model%h.ne.0 &
00677                 .and.(              &
00678                 transfer_sp%if      &
00679                 .or.any(([          &
00680                 ind%green%gndt,     &
00681                 ind%green%gndz,     &
00682                 ind%green%gndz2,    &
00683                 ind%green%gndh,     &
00684                 ind%green%gnc,      &
00685                 ind%green%g3d       &
00686                 ]).ne.0)            &
00687                 )                   &
00688                 ) then
00689
00690                 if (optimize.and.green_common(igreen)%distance(idist).gt.3)
      then
00691                   val(ind%model%h)=val(ind%model%hp)
00692                 else
00693                   call get_value( &
00694                     model(ind%model%h), r2d(lat), r2d(lon), val(ind%model%h), &

00695                     level=1, method = info(igreen)%interpolation)
00696                 endif
00697               endif
00698
00699               if (ind%model%sp.ne.0) then
00700                 ! transfer SP if necessary on terrain
00701                 if (transfer_sp%if &
00702                   .and.any([      &
00703                   ind%green%ge,     &
00704                   ind%green%gnc,    &
00705                   ind%green%g3d,    &
00706                   ind%green%gegdt,  &
00707                   ind%green%gg      &
00708                   ].ne.0)           &
00709                   ) then
00710
00711                   val(ind%model%sp) = standard_pressure(           &
00712                     height                   = val(ind%model%h),   &
00713                     h_zero                   = val(ind%model%hp),  &
00714                     p_zero                   = old_val_sp,         &
00715                     method                   = transfer_sp%method, &
00716                     temperature              = val(ind%model%t),   &
00717                     use_standard_temperature = ind%model%t.eq.0,   &
00718                     nan_as_zero              = .false.)
00719
00720                   if(all([ind%model%rsp, ind%model%hrsp].ne.0)) then
00721                     val(ind%model%rsp) = standard_pressure(           &
00722                       height                   = val(ind%model%h),    &
00723                       h_zero                   = val(ind%model%hrsp), &
00724                       p_zero                   = old_val_rsp,         &
00725                       method                   = transfer_sp%method,  &
00726                       temperature              = val(ind%model%t),    &
00727                       use_standard_temperature = ind%model%t.eq.0,    &
00728                       nan_as_zero              = .false.)
00729                     endif
00730                 endif
00731
00732                 if (ind%model%rsp.ne.0) then
00733                   if (.not.
            &
00734                     (site%lp%if
            &
00735                     .and.green_common(igreen)%distance(idist).lt.
      local_pressure_distance &
00736                     .and..not.first_reduction
            &
00737                     )
            &
00738                     ) then
00739
00740                     val(ind%model%sp) = val(ind%model%sp) - val(ind%model%rsp)
00741
00742                     if (first_reduction) first_reduction=.false.
00743
00744                   endif
00745                 endif
00746
00747                 ! if the cell is not over sea and inverted barometer assumption
       was not set
00748                 ! and is not excluded by polygon
00749                 if ((ind%polygon%e.ne.0.and.iok(ind%polygon%e).ne.0).or.(ind
      %polygon%e.eq.0)) then
00750                   !IB or NIB
00751                   if (.not.(ind%model%ls.ne.0.and.inverted_barometer.and.int(
      val(ind%model%ls)).eq.0)) then
```

```
00752                     ! GE
00753                     if (ind%green%ge.ne.0) then
00754                       result(ind%green%ge) = result(ind%green%ge)        &
00755                         + val(ind%model%sp)                              &
00756                         * green_common(igreen)%data(idist, ind%green%ge) &
00757                         * area * normalize
00758                     endif
00759
00760                     ! GEGdt pressure part from Guo 2004
00761                     if (ind%green%gegdt.ne.0) then
00762                       result(ind%green%gegdt) = result(ind%green%gegdt) +   &
00763                         val(ind%model%sp) *                                &
00764                         val(ind%model%t) * 1e-4 *                          &
00765                         green_common(igreen)%data(idist, ind%green%gegdt) * &
00766                         area * normalize
00767                     endif
00768
00769                     ! GG
00770                     if (ind%green%gg.ne.0) then
00771                       aux = mmwater2pascal(val(ind%model%sp), inverted=.true.)
     &
00772                         * area/ (d2r(green_common(igreen)%distance(idist)) *
     &
00773                         earth%radius*1e18)
00774
00775                       result(ind%green%gg) = result(ind%green%gg) +      &
00776                         green_common(igreen)%data(idist, ind%green%gg) * &
00777                         aux * 1e8 ! m s-2 -> microGal
00778                     endif
00779                   endif
00780
00781                   ! ! GE NIB if both IB and NIB wanted
00782                   if (inverted_barometer.and.non_inverted_barometer) then
00783                     if (ind%green%ge.ne.0) then
00784                       result(ubound(result)) = result(ubound(result)) +
     &
00785                         val(ind%model%sp) *                              &
00786                         green_common(igreen)%data(idist, ind%green%ge) * &
00787                         area * normalize
00788                     endif
00789                   endif
00790                 endif
00791
00792                 if (                                                     &
00793                   (ind%polygon%n.ne.0.and.iok(ind%polygon%n).ne.0)     &
00794                   .or.(ind%polygon%n.eq.0)                             &
00795                   ) then
00796
00797                   !3D
00798                   if (method(3)) then
00799
00800                     ! if distance%stop_3d was set restrict computation of 3D to
      this distance
00801                     if(green_common(igreen)%distance(idist).lt.info(igreen)
     %distance%stop_3d) then
00802
00803                       if (ind%model%rsp.eq.0) then
00804                         call print_warning("3D but no RSP", error=.true.)
00805                       endif
00806                       if (ind%model%hrsp.eq.0) then
00807                         call print_warning("3D but no HRSP", error=.true.)
00808                       endif
00809
00810                       if (allocated(heights))      deallocate(heights)
00811                       if (allocated(pressures))    deallocate(pressures)
00812                       if (allocated(temperatures)) deallocate(temperatures)
00813
00814                       if( &
00815                         info(igreen)%height%stop <= max(info(igreen)%height
     %start,val(ind%model%h)) &
00816                         ) then
00817                         cycle
00818                       endif
00819
00820                       nheight=                                           &
00821                         ceiling((info(igreen)%height%stop                &
00822                         -max(info(igreen)%height%start,val(ind%model%h))) &
00823                         /info(igreen)%height%step)
00824
00825                       allocate(heights(nheight))
00826                       allocate(pressures(nheight))
00827                       allocate(temperatures(nheight))
00828
00829                       do iheight=1, nheight
00830                         heights(iheight)=max(info(igreen)%height%start, val(ind
     %model%h)) &
00831                           +(iheight-0.5)*info(igreen)%height%step
```

```
00832                       enddo
00833
00834                       if (.not.allocated(level%height))      allocate (level
      %height(size(level%level)))
00835                       if (.not.allocated(level%temperature)) allocate (level
      %temperature(size(level%level)))
00836                       if (.not.allocated(level%humidity))    allocate (level
      %humidity(size(level%level)))
00837
00838                       do i=1,size(level%level)
00839                         call get_value(
                           &
00840                             model(ind%model%gp), r2d(lat), r2d(lon), level%height
      (i),                  &
00841                             level=level%level(i), method = info(igreen)
      %interpolation, date=date%date)
00842
00843                         if (ind%model%vt.ne.0) then
00844                           call get_value(                          &
00845                             model(ind%model%vt), r2d(lat), r2d(lon), &
00846                             val    = level%temperature(i),           &
00847                             level  = level%level(i),                 &
00848                             method = info(igreen)%interpolation,     &
00849                             date   = date%date                       &
00850                             )
00851                         endif
00852
00853                         if (ind%model%vsh.ne.0) then
00854                           call get_value(                           &
00855                             model(ind%model%vsh), r2d(lat), r2d(lon), &
00856                             val    = level%humidity(i),               &
00857                             level  = level%level(i),                  &
00858                             method = info(igreen)%interpolation,      &
00859                             date   = date%date                        &
00860                             )
00861
00862                           if (.not.isnan(level%humidity(i))) then
00863                             level%temperature(i)= &
00864                               virtual_temperature(level%temperature(i),level
      %humidity(i))
00865                           endif
00866
00867                         endif
00868                       enddo
00869
00870                       i=1
00871                       do while(level%height(i).lt.heights(1).and.i.ne.size(
      level%level))
00872                         i=i+1
00873                       end do
00874
00875                       do iheight=1, nheight
00876
00877                         if (iheight.eq.1) then
00878                           ! h1=val(ind%model%h)
00879                           ! v1=val(ind%model%sp)+val(ind%model%rsp)
00880                           ! h2=level%height(i)
00881                           ! v2=1.e2*dble(level%level(i))
00882
00883                           temperatures(iheight)= &
00884                             level%temperature(i)-6.5e-3*(val(ind%model%h)-val(
      ind%model%hp))
00885
00886                           if (.not.isnan(level%humidity(1))) then
00887                             val(ind%model%t) = &
00888                               virtual_temperature(val(ind%model%t), level
      %humidity(1))
00889                           endif
00890
00891                           pressures(iheight) = standard_pressure(        &
00892                             heights(iheight),                            &
00893                             p_zero=val(ind%model%sp)+val(ind%model%rsp), &
00894                             h_zero=val(ind%model%h), &
00895                             method="standard",                           &
00896                             use_standard_temperature=.true.,             &
00897                             temperature=val(ind%model%t)                 &
00898                             )
00899
00900                         else
00901                           do while(level%height(i+1).lt.heights(iheight).and. i
      .ne.size(level%level))
00902                             i=i+1
00903                           end do
00904
00905                           ! temperature linear interpolation
00906                           if(i.lt.size(level%level)) then
00907                             temperatures(iheight)= &
```

```
00908                                level%temperature(i) &
00909                                + (level%temperature(i+1)-level%temperature(i)) &
00910                                /(level%height(i+1)-level%height(i))*(heights(
      iheight)-level%height(i))
00911                            else
00912                              temperatures(iheight)= &
00913                                level%temperature(i)
00914                            endif
00915
00916                            if(heights(iheight-1).lt.level%height(i).and.(heights
      (iheight).gt.level%height(i))) then
00917                              ! h1=level%height(i)
00918                              ! v1=1.e2*dble(level%level(i))
00919                              ! h2=level%height(i+1)
00920                              ! v2=1.e2*dble(level%level(i+1))
00921
00922                              pressures(iheight) =
           &
00923                                standard_pressure(
           &
00924                                height                    = heights(iheight),
           &
00925                                p_zero                    = 1.e2*dble(level%level(
      i)), &
00926                                h_zero                    = level%height(i),
           &
00927                                method                    = "standard",
           &
00928                                use_standard_temperature = .true.,
           &
00929                                temperature               = temperatures(iheight),
           &
00930                                nan_as_zero               = .true.
           &
00931                                )
00932
00933                            else
00934
00935                              pressures(iheight)=
           &
00936                                standard_pressure(
           &
00937                                height                    = heights(iheight),
           &
00938                                p_zero                    = pressures(iheight-1),
           &
00939                                h_zero                    = heights(iheight-1),
           &
00940                                method                    = "standard",
           &
00941                                use_standard_temperature = .true.,
           &
00942                                temperature               = temperatures(iheight),
           &
00943                                nan_as_zero               = .true.
           &
00944                                )
00945                            endif
00946                          endif
00947
00948                          ! if (i.lt.size(level%level)) then
00949                          ! p_int=exp(dlog(v1)
           +(dlog(v2)-dlog(v1))*(heights(iheight)-h1)/(h2-h1))
00950                          ! if (p_int.gt.1e29)   p_int=0
00951                          ! pressures(iheight)=p_int
00952                          ! endif
00953
00954                          if (method3d(1).or.green_common(igreen)%distance(idist)
      .gt.method3d_refinment_distance) then
00955                            result(ind%green%g3d) = result(ind%green%g3d) &
00956                              + geometry(psi=d2r(green_common(igreen)%distance(
      idist)), h=site%height, z=heights(iheight)) &
00957                              * pressures(iheight)/(temperatures(iheight))  &
00958                              * area * info(igreen)%height%step &
00959                              *(-gravity%constant)*1e8/r_air
00960
00961                          else if (method3d(2)) then
00962                            result(ind%green%g3d) =                          &
00963                              result(ind%green%g3d)                          &
00964                              + potential(                                   &
00965                              psi1=d2r(green_common(igreen)%start(idist)),   &
00966                              psi2=d2r(green_common(igreen)%stop(idist)),    &
00967                              dazimuth=d2r(dazimuth),                        &
00968                              h=site%height,                                 &
00969                              z1= heights(iheight)-info(igreen)%height%step/2, &
00970                              z2= heights(iheight)+info(igreen)%height%step/2  &
00971                              )                                              &
```

```
00972                             * pressures(iheight)/(temperatures(iheight))     &
00973                             *(-gravity%constant)*1e8/r_air
00974                           if (isnan(result(ind%green%g3d)))  then
00975                             ! small distances can cause numerical problems
00976                             result(ind%green%g3d)=0
00977                           endif
00978
00979                         else if (method3d(3)) then
00980                           result(ind%green%g3d) =                             &
00981                             result(ind%green%g3d)                             &
00982                             + cylinder(                                       &
00983                             psi1=d2r(green_common(igreen)%start(idist)),      &
00984                             psi2=d2r(green_common(igreen)%stop(idist)),       &
00985                             dazimuth=d2r(dazimuth),                           &
00986                             h=site%height,                                    &
00987                             z1= heights(iheight)-info(igreen)%height%step/2, &
00988                             z2= heights(iheight)+info(igreen)%height%step/2  &
00989                             )                                                 &
00990                             * pressures(iheight)/(temperatures(iheight))     &
00991                             *(-gravity%constant)*1e8/r_air
00992                         endif
00993
00994
00995                       enddo
00996                     endif
00997                   endif
00998
00999                   !C before GN GNdt etc because it needs SP on H not on site
01000                   if(ind%green%gnc.ne.0) then
01001                     if (           &
01002                       any([        &
01003                       ind%model%sp, &
01004                       ind%model%hp, &
01005                       ind%model%h,  &
01006                       ind%model%t   &
01007                       ].eq.0))      &
01008                       call print_warning("with @GNc you need to give @T @HP @H"
     , error=.true.)
01009
01010                     result(ind%green%gnc) = result(ind%green%gnc)        &
01011                       + val(ind%model%sp)                                &
01012                       * aggf(                                            &
01013                       d2r(green_common(igreen)%distance(idist)),         &
01014                       zmin=val(ind%model%h),                             &
01015                       t_zero=val(ind%model%t),                           &
01016                       h=site%height,                                     &
01017                       dz= gnc_looseness*10.                              &
01018                       *merge(10._dp,                                     &
01019                       merge(0.1_dp,1._dp,                                &
01020                       green_common(igreen)%distance(idist).le.1e-5_dp ), &
01021                       green_common(igreen)%distance(idist).ge.1e-2_dp ), &
01022                       method="standard",                                 &
01023                       predefined=.true.)                                 &
01024                       * area * normalize
01025
01026                     if (.not.quiet) then
01027                       open(unit=output_unit, carriagecontrol='fortran')
01028                       call progress(
        &
01029                         100*igreen*idist
      &
01030                         /(size(green_common(igreen)%distance)*size(green_common
     )), &
01031                         every=1 &
01032                         )
01033                     endif
01034                   endif
01035
01036                   ! transfer SP if necessary on site level
01037                   if (transfer_sp%if &
01038                     .and.any([ &
01039                     ind%green%gn, &
01040                     ind%green%gndt, &
01041                     ind%green%gndz, &
01042                     ind%green%gndz2, &
01043                     ind%green%gndh &
01044                     ].ne.0) &
01045                     ) then
01046                     val(ind%model%sp) = standard_pressure( &
01047                       height=site%height,                 &
01048                       h_zero=val(ind%model%hp),           &
01049                       p_zero=old_val_sp,                  &
01050                       method=transfer_sp%method,          &
01051                       temperature=val(ind%model%t),       &
01052                       use_standard_temperature            &
01053                       = ind%model%t.eq.0,                 &
01054                       nan_as_zero=.false.)
```

```
01055
01056                     if(all([ind%model%rsp, ind%model%hrsp].ne.0)) then
01057                       val(ind%model%rsp) = standard_pressure(          &
01058                         height=site%height,                           &
01059                         h_zero=val(ind%model%hrsp),                   &
01060                         p_zero=old_val_rsp,                           &
01061                         method=transfer_sp%method,                    &
01062                         temperature=val(ind%model%t),                 &
01063                         use_standard_temperature                      &
01064                         = ind%model%t.eq.0,                           &
01065                         nan_as_zero=.false.)
01066                     endif
01067                     if(ind%model%rsp.ne.0) val(ind%model%sp) = val(ind%model%sp
     ) - val(ind%model%rsp)
01068                   endif
01069
01070                   ! GN
01071                   if (ind%green%gn.ne.0) then
01072                     result_partial(ind%green%gn) = &
01073                       val(ind%model%sp) *                             &
01074                       green_common(igreen)%data(idist, ind%green%gn) * &
01075                       area * normalize
01076                     result(ind%green%gn) = &
01077                       result(ind%green%gn) + result_partial(ind%green%gn)
01078                   endif
01079
01080                   ! GNdt
01081                   if (ind%green%gndt.ne.0) then
01082                     if (any(                                            &
01083                       [ind%model%sp, ind%model%t, ind%model%rsp           &
01084                       ].eq.0)) &
01085                       call print_warning("not enough data model for GNdt", &
01086                       error=.true.)
01087                     result_partial(ind%green%gndt) =       &
01088                       val(ind%model%sp)                                 &
01089                       * green_common(igreen)%data(idist, ind%green%gndt)   &
01090                       * (val(ind%model%t)-atmosphere%temperature%standard) &
01091                       *  area * normalize
01092                     result(ind%green%gndt) = result(ind%green%gndt) +       &
01093                       result_partial(ind%green%gndt)
01094                   endif
01095
01096                   ! GNdh
01097                   if (ind%green%gndh.ne.0) then
01098                     if (any(                                            &
01099                       [ &
01100                       ind%model%sp, ind%model%h, ind%model%rsp           &
01101                       ].eq.0)) &
01102                       call print_warning("not enough data model for GNdh", &
01103                       error=.true.)
01104                     result_partial(ind%green%gndh) = &
01105                       val(ind%model%sp)                                 &
01106                       * green_common(igreen)%data(idist, ind%green%gndh)   &
01107                       * (val(ind%model%h)-site%height) &
01108                       *  area * normalize
01109                     result(ind%green%gndh) = result(ind%green%gndh) +       &
01110                       result_partial(ind%green%gndh)
01111                   endif
01112
01113                   ! GNdz
01114                   if (ind%green%gndz.ne.0) then
01115                     if (any(                                            &
01116                       [ &
01117                       ind%model%sp, ind%model%h, ind%model%rsp           &
01118                       ].eq.0)) &
01119                       call print_warning("not enough data model for GNdz", &
01120                       error=.true.)
01121                     result_partial(ind%green%gndz) =  +       &
01122                       val(ind%model%sp)                                 &
01123                       * green_common(igreen)%data(idist, ind%green%gndz)   &
01124                       * (val(ind%model%h)-site%height) &
01125                       *  area * normalize
01126                     result(ind%green%gndz) = result(ind%green%gndz) +       &
01127                       result_partial(ind%green%gndz)
01128                   endif
01129
01130
01131                   ! GNdz2
01132                   if (ind%green%gndz2.ne.0) then
01133                     if (any(                                            &
01134                       [                                                 &
01135                       ind%model%sp, ind%model%h, ind%model%rsp          &
01136                       ].eq.0))                                          &
01137                       call print_warning("not enough data model for GNdz2", &
01138                       error=.true.)
01139
01140                     result_partial(ind%green%gndz2) =
```

```
          &
01141                       val(ind%model%sp)
          &
01142                       * green_common(igreen)%data(idist, ind%green%gndz2)
          &
01143                       * ( (val(ind%model%h)-site%height)
          &
01144                       /(earth%radius * d2r(green_common(igreen)%distance(idist)
      )))**2 &
01145                       *  area * normalize
01146
01147                     result(ind%green%gndz2) = result(ind%green%gndz2) &
01148                       + result_partial(ind%green%gndz2)
01149                   endif
01150
01151                   ! reference 2D for 3D method
01152                   if (ind%green%g3d.ne.0) then
01153
01154                     if(green_common(igreen)%distance(idist).lt.info(igreen)
      %distance%stop_3d) then
01155                       if (info(igreen)%height%start.gt.1) then
01156                         rsp = rsp                                       &
01157                           + standard_pressure(                          &
01158                           height = info(igreen)%height%start,           &
01159                           h_zero = val(ind%model%hrsp),                 &
01160                           p_zero = val(ind%model%rsp),                  &
01161                           method = "standard")                          &
01162                           * green_common(igreen)%data(idist, ind%green%g3d) &
01163                           * area * normalize
01164                       else
01165                         rsp = rsp                                       &
01166                           + val(ind%model%rsp)                          &
01167                           * green_common(igreen)%data(idist, ind%green%g3d) &
01168                           * area * normalize
01169                       endif
01170
01171                     else
01172                       result(ind%green%g3d) =         &
01173                         result(ind%green%g3d)         &
01174                         + sum(result_partial,         &
01175                         mask=(                        &
01176                         green%dataname.eq."GN"        &
01177                         .or.green%dataname.eq."GNdt"  &
01178                         .or.green%dataname.eq."GNdz"  &
01179                         .or.green%dataname.eq."GNdz2" &
01180                         .or.green%dataname.eq."GNdh"  &
01181                         ))
01182                     endif
01183
01184                   endif
01185
01186                 endif
01187               endif
01188             else
01189               result=sqrt(-1.)
01190             endif
01191           elseif(ind%model%ewt.eq.0) then
01192             call print_warning("@SP is required with -M2D -G", error=.true.)
01193           endif
01194         endif
01195
01196         ! surface loads from EWT
01197         if (
                      &
01198           ind%green%gr.ne.0
                      &
01199           .or.ind%green%ghn.ne.0
                      &
01200           .or.ind%green%ghe.ne.0
                      &
01201           ) then
01202           if ((ind%polygon%e.ne.0.and.iok(ind%polygon%e).ne.0).or.(ind%polygon
      %e.eq.0)) then
01203             if (.not.(ind%model%ls.ne.0.and.inverted_barometer.and.int(val(ind
      %model%ls)).eq.0)) then
01204               call get_value(
                        &
01205                 model(ind%model%ewt), r2d(lat), r2d(lon), val(ind%model%ewt),
                        &
01206                 level=1, method = info(igreen)%interpolation, date=date%date)
01207               aux = (val(ind%model%ewt))  *
                        &
01208                 area/d2r(green_common(igreen)%distance(idist)) *
                        &
01209                 1./earth%radius/1e12* 1e3 ! m -> mm
01210               if (isnan(aux)) aux = 0
01211               if (ind%green%gr.ne.0) then
```

```
01212                 result(ind%green%gr) = result(ind%green%gr) +       &
01213                   green_common(igreen)%data(idist, ind%green%gr) &
01214                   * aux
01215
01216                 if (ind%green%ghn.ne.0) then
01217                   result(ind%green%ghn) = result(ind%green%ghn) +      &
01218                     green_common(igreen)%data(idist, ind%green%ghn) * &
01219                     aux * (-cos(d2r(azimuth)))
01220                 endif
01221                 if (ind%green%ghe.ne.0) then
01222                   result(ind%green%ghe) = result(ind%green%ghe) +      &
01223                     green_common(igreen)%data(idist, ind%green%ghe) * &
01224                     aux * (-sin(d2r(azimuth)))
01225                 endif
01226               endif
01227             endif
01228           endif
01229         endif
01230
01231         ! moreverbose point: -L@p
01232         if(ind%moreverbose%p.ne.0) then
01233           if (header_p.and. output%header) then
01234             if(size(green_common).gt.1) &
01235               write(moreverbose(ind%moreverbose%p)%unit, "(a2, x$)") "i"
01236
01237             write(moreverbose(ind%moreverbose%p)%unit, &
01238               '(a8, 8a13, $)')                          &
01239               "name", "lat", "lon",                  &
01240               "distance", "azimuth",                 &
01241               "lat", "lon",                          &
01242               "area", "totarea"
01243
01244             if (result_component) then
01245               write(moreverbose(ind%moreverbose%p)%unit, &
01246                 '(a13, $)')                 &
01247                 (trim(green(i)%dataname), &
01248                 i=lbound(green, 1),       &
01249                 ubound(green, 1)          &
01250                 )
01251             endif
01252
01253             if (result_total) then
01254               if (method(2)) then
01255                 write(moreverbose(ind%moreverbose%p)%unit, &
01256                   '(a13, $)') "G2D_t"
01257               endif
01258               if (method(3)) then
01259                 write(moreverbose(ind%moreverbose%p)%unit, &
01260                   '(a13, $)') "G3D_t"
01261               endif
01262             endif
01263
01264             if (.not.moreverbose(ind%moreverbose%p)%sparse) then
01265               write(moreverbose(ind%moreverbose%p)%unit,
    &
01266                 '(<size(model)>a12)', advance='no' )                          &
01267                 (trim(model(i)%dataname), i=lbound(model, 1), ubound(model, 1))
01268             endif
01269
01270             if (size(iok).gt.0) then
01271               write(moreverbose(ind%moreverbose%p)%unit, &
01272                 '(<size(iok)>(a3, i1))'),               &
01273                 ("ok", i, i =1, ubound(iok, 1))
01274             else
01275               write(moreverbose(ind%moreverbose%p)%unit, *)
01276             endif
01277             header_p=.false.
01278           endif
01279           if (                                          &
01280             .not.moreverbose(ind%moreverbose%p)%sparse    &
01281             .or.                                          &
01282             (moreverbose(ind%moreverbose%p)%sparse        &
01283             .and.(azimuth==azimuths(ubound(azimuths, 1))) &
01284             )                                             &
01285             ) then
01286
01287             if(size(green_common).gt.1) &
01288               write(moreverbose(ind%moreverbose%p)%unit, "(i2, x$)") igreen
01289
01290             write(moreverbose(ind%moreverbose%p)%unit,         &
01291               '(a8, 6' // output%form //',2 en13.3, $)'),       &
01292               site%name, site%lat, site%lon,              &
01293               green_common(igreen)%distance(idist), azimuth, &
01294               r2d(lat), r2d(lon), area, tot_area
01295
01296             if (result_component)                          &
01297               write(moreverbose(ind%moreverbose%p)%unit, &
```

```
01298               '(' // output%form //'$)'),                &
01299               (result(i), i =1, size(result))
01300
01301             if (result_total) then
01302               if (method(2)) then
01303                 write(moreverbose(ind%moreverbose%p)%unit, &
01304                   '(' // output%form //'$)'), &
01305                   sum(result, &
01306                   mask=( &
01307                   green%dataname.eq."GN" &
01308                   .or.green%dataname.eq."GE" &
01309                   .or.green%dataname.eq."GNdt" &
01310                   .or.green%dataname.eq."GNdz" &
01311                   .or.green%dataname.eq."GNdz2" &
01312                   .or.green%dataname.eq."GNdh" &
01313                   ))
01314               endif
01315               if (method(3)) then
01316                 write(moreverbose(ind%moreverbose%p)%unit, &
01317                   '(' // output%form //'$)'), &
01318                   sum(result, &
01319                   mask=( &
01320                   green%dataname.eq."G3D" &
01321                   .or.green%dataname.eq."GE" &
01322                   ))
01323               endif
01324
01325             endif
01326             if (.not.moreverbose(ind%moreverbose%p)%sparse) then
01327               do i=1, size(val)
01328                 call get_value(                          &
01329                   model(i), r2d(lat), r2d(lon), val(i), &
01330                   level=1,                               &
01331                   method = info(igreen)%interpolation,  &
01332                   date=date%date)
01333               enddo
01334               write(moreverbose(ind%moreverbose%p)%unit, &
01335                 '(<size(model)>en12.2, $)') val
01336             endif
01337             if (size(iok).gt.0) then
01338               write(moreverbose(ind%moreverbose%p)%unit, &
01339                 '(<size(iok)>(i4))'), iok
01340             else
01341               write(moreverbose(ind%moreverbose%p)%unit, * )
01342             endif
01343           endif
01344         endif
01345
01346         ! moreverbose auxilary to draw: -L@a
01347         if(ind%moreverbose%a.ne.0) then
01348           call printmoreverbose(
          &
01349              d2r(site%lat), d2r(site%lon), d2r(azimuth), d2r(dazimuth), &
01350              d2r(green_common(igreen)%start(idist)),                    &
01351              d2r(green_common(igreen)%stop(idist))                      &
01352              )
01353         endif
01354       enddo
01355     enddo
01356   enddo
01357
01358
01359   if (ind%green%g3d.ne.0) &
01360     result(ind%green%g3d)=result(ind%green%g3d) - rsp
01361
01362   ! results to output
01363   if (result_component) write (output%unit, "(" // output%form // '$)') result
01364   if (result_total) then
01365     if (method(2)) then
01366       write(output%unit, &
01367         '(' // output%form //'$)'), &
01368         sum(result, &
01369         mask=( &
01370         green%dataname.eq."GN" &
01371         .or.green%dataname.eq."GE" &
01372         .or.green%dataname.eq."GNdt" &
01373         .or.green%dataname.eq."GNdz" &
01374         .or.green%dataname.eq."GNdz2" &
01375         .or.green%dataname.eq."GNdh" &
01376         ))
01377     endif
01378     if (method(3)) then
01379       write(output%unit, &
01380         '(' // output%form //'$)'), &
01381         sum(result, &
01382         mask=( &
01383         green%dataname.eq."G3D" &
```

```
01384         .or.green%dataname.eq."GE" &
01385         ))
01386     endif
01387   endif
01388
01389   ! summary: -L@s
01390   if (ind%moreverbose%s.ne.0) then
01391     if (output%header) write(moreverbose(ind%moreverbose%s)%unit, '(2a8, 3a12)'
      ) &
01392       "station", "npoints", "area", "area/R2", "t_area_used"
01393     write(moreverbose(ind%moreverbose%s)%unit, '(a8, i8, 3en12.2)') &
01394       site%name, npoints, tot_area, tot_area/earth%radius**2, tot_area_used
01395   endif
01396
01397   ! green values : -L@g
01398   if(ind%moreverbose%g.ne.0) then
01399     do i = 1, size(green_common)
01400       if (output%header) &
01401         write(moreverbose(ind%moreverbose%g)%unit, '(a3,100a14)') &
01402         "nr", "distance", "start", "stop", "data", "di(j)-di(j-1)"
01403       do j=1,size(green_common(i)%distance)
01404         write(moreverbose(ind%moreverbose%g)%unit, '(i3,f14.6, 100f14.7)'), &
01405           j, green_common(i)%distance(j), &
01406           green_common(i)%start(j), &
01407           green_common(i)%stop(j), &
01408           green_common(i)%data(j,:), &
01409           green_common(i)%distance(j)-green_common(i)%distance(j-1)
01410       enddo
01411     enddo
01412   endif
01413 end subroutine
01414
01415 ! ==============================================================================
01416 !> returns lat and lon of spherical trapezoid
01417 !! \date 2013.07.03
01418 !! \author Marcin Rajner
01419 ! ==============================================================================
01420 subroutine printmoreverbose (latin, lonin, azimuth, azstep,
      distancestart, distancestop)
01421   use mod_spherical, only : spher_trig
01422   use mod_cmdline,   only : moreverbose, ind
01423   use mod_utilities, only : r2d
01424
01425   real(dp), intent(in) :: azimuth, azstep, latin, lonin
01426   real(dp) ::  lat, lon, distancestart, distancestop
01427
01428   call spher_trig(latin, lonin, distancestart, azimuth - azstep/2, lat, lon)
01429   write(moreverbose(ind%moreverbose%a)%unit, '(8f12.6)'), r2d(lat), r2d(lon)
01430   call spher_trig(latin, lonin, distancestop, azimuth - azstep/2, lat, lon)
01431   write(moreverbose(ind%moreverbose%a)%unit, '(8f12.6)'), r2d(lat), r2d(lon)
01432   call spher_trig(latin, lonin, distancestop, azimuth + azstep/2, lat, lon)
01433   write(moreverbose(ind%moreverbose%a)%unit, '(8f12.6)'), r2d(lat), r2d(lon)
01434   call spher_trig(latin, lonin, distancestart, azimuth + azstep/2, lat, lon)
01435   write(moreverbose(ind%moreverbose%a)%unit, '(8f12.6)'), r2d(lat), r2d(lon)
01436   write(moreverbose(ind%moreverbose%a)%unit, '(">")')
01437 end subroutine
01438
01439 ! ==============================================================================
01440 !! \date 2013-07-02
01441 !! \author M. Rajner
01442 !! \warning input spherical distance in radian
01443 !!
01444 !! method:
01445 !!   default see equation in Rajnerdr
01446 !!   spotl   see \cite spotl manual
01447 !!   olssson see \cite olsson2009
01448 !!
      ==============================================================================
01449 function green_newtonian (psi, h, z, method)
01450   use mod_constants, only: earth, gravity
01451   use mod_normalization, only: green_normalization
01452   real(dp) :: green_newtonian
01453   real(dp), intent (in) :: psi
01454   real(dp), intent (in), optional :: h
01455   real(dp), intent (in), optional :: z
01456   character(*), optional :: method
01457   real(dp) :: h_, z_, eps, t
01458   if (present(h)) then
01459     h_=h
01460   else
01461     h_=0.
01462   endif
01463   if (present(z)) then
01464     z_=z
01465   else
01466     z_=0.
01467   endif
```

```
01468   if (present(method) &
01469     .and. (method.eq."spotl" .or. method.eq."olsson")) then
01470     if(method.eq."spotl") then
01471       eps = h_/ earth%radius
01472       green_newtonian =                                        &
01473         1. /earth%radius**2                                    &
01474         *(eps + 2. * (sin(psi/2.))**2 )                        &
01475         /((4.*(1.+eps)* (sin(psi/2.))**2 + eps**2)**(3./2.)) &
01476         * gravity%constant                                     &
01477         * green_normalization("f",psi=psi)
01478       return
01479     else if (method.eq."olsson") then
01480       t = earth%radius/(earth%radius +h_)
01481       green_newtonian =                        &
01482         1 / earth%radius**2 * t**2 *           &
01483         (1. - t * cos(psi) ) /                &
01484         ( (1-2*t*cos(psi) +t**2 )**(3./2.) ) &
01485         * gravity%constant                     &
01486         * green_normalization("f",psi=psi)
01487       return
01488     endif
01489   else
01490     green_newtonian =                                            &
01491       ((earth%radius + h_) - (earth%radius + z_) * cos(psi))     &
01492       / ((earth%radius + h_)**2 + (earth%radius + z_)**2         &
01493       -2*(earth%radius + h_)*(earth%radius + z_)*cos(psi))**(3./2.)
01494
01495     green_newtonian = green_newtonian &
01496       * gravity%constant / earth%gravity%mean  * green_normalization("m", psi=
    psi)
01497     return
01498   endif
01499 end function
01500 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

```
00001 !
      ================================================================================
00002 !> \file
00003 !
      ================================================================================
00004 module mod_normalization
00005   implicit none
00006
00007 contains
00008 ! ================================================================================
00009 ! ================================================================================
00010 function green_normalization(method, psi)
00011   use mod_constants, only: pi, earth, gravity, dp
00012   use mod_utilities, only: d2r
00013   real(dp):: green_normalization
00014   character(*) :: method
00015   real(dp), optional :: psi
00016
00017   if (method.eq."f2m") then
00018     green_normalization = &
00019         1e-3 &
00020         / earth%gravity%mean  * earth%radius * 2 * pi * (1.- cos(d2r(1._dp)))
00021   else if (method.eq."m") then ! merriam normalization
00022     green_normalization =  &
00023         psi * 1e15 * earth%radius**2 * 2 * pi * (1.- cos(d2r(1._dp)))
00024   else if (method.eq."f") then ! farrell normalization
00025     green_normalization =  &
```

```
00026         psi * 1e18 * earth%radius
00027   endif
00028 end function
00029
00030 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

```
00001 !
      ================================================================================
00002 !> \file
00003 !! Some routines to deal with inclusion or exclusion of polygons
00004 !!
00005 !! \author M.Rajner
00006 !! \date 2012-12-20
00007 !! \date 2013-03-19
00008 !!    added overriding of poly use by command line like in \cite spotl
00009 !
      ================================================================================
00010 module mod_polygon
00011   use mod_constants, only : dp
00012
00013   implicit none
00014   !-----------------------------------------------------
00015   ! polygons
00016   !-----------------------------------------------------
00017   type polygon_data
00018     logical :: use
00019     real(dp), allocatable , dimension (:,:) :: coords
00020   end type
00021
00022   type polygon_info
00023     integer :: unit
00024     character(:), allocatable  :: name
00025     character(len=25) :: dataname
00026     type(polygon_data), dimension (:), allocatable :: polygon
00027     logical :: if
00028     ! global setting (+|-) which override this in polygon file
00029     character(1):: pm
00030   end type
```

```
00031   type(polygon_info) , allocatable, dimension (:) :: polygon
00032
00033 contains
00034 ! ================================================================================
00035 !> This subroutine parse polygon information from command line entry
00036 !!
00037 !! \author M. Rajner
00038 !! \date 2013.05.20
00039 ! ================================================================================
00040 subroutine parse_polygon (cmd_line_entry)
00041   use mod_printing
00042   use mod_cmdline
00043   use mod_utilities, only: file_exists
00044   type(cmd_line_arg),intent(in):: cmd_line_entry
00045   integer :: i
00046
00047   if (allocated(polygon)) then
00048     call print_warning("repeated")
00049     return
00050   endif
00051
00052   allocate(polygon(size(cmd_line_entry%field)))
00053   do i=1, size(cmd_line_entry%field)
00054   polygon(i)%name=cmd_line_entry%field(i)%subfield(1)%name
00055   if(i.gt.1.and.cmd_line_entry%field(i)%subfield(1)%name.eq."") then
00056     polygon(i)%name= polygon(i-1)%name
00057   endif
00058   polygon(i)%dataname=cmd_line_entry%field(i)%subfield(1)%dataname
00059   write(log%unit, form%i2), 'polygon file:' , polygon(i)%name
00060   if (file_exists((polygon(i)%name))) then
00061     polygon(i)%if=.true.
00062     if(cmd_line_entry%field(i)%subfield(2)%name.eq."+" &
00063       .or.cmd_line_entry%field(i)%subfield(2)%name.eq."-" ) then
00064       polygon(i)%pm = cmd_line_entry%field(i)%subfield(2)%name
00065       write(log%unit, form%i3) , "global override:", polygon(i)%pm
00066     endif
00067     call read_polygon(polygon(i))
00068   else
00069     stop 'file do not exist. Polygon file PROBLEM'
00070   endif
00071 enddo
00072
00073 end subroutine
00074 !
      ================================================================================
00075 !> Reads polygon data
00076 !!
00077 !! inspired by spotl \cite Agnew97
00078 !
      ================================================================================
00079
00080 subroutine read_polygon (polygon)
00081
00082   use, intrinsic :: iso_fortran_env
00083   use mod_utilities, only: skip_header
00084   use mod_printing
00085
00086   type(polygon_info) :: polygon
00087   integer :: i , j , number_of_polygons , nvertex
00088   character (1)  :: pm
00089
00090   if (polygon%if) then
00091     ! polygon file
00092     open (newunit = polygon%unit , action="read", file=polygon%name )
00093
00094     ! first get the number of polygon
00095     call skip_header(polygon%unit)
00096     read (polygon%unit , * ) number_of_polygons
00097     allocate (polygon%polygon(number_of_polygons))
00098
00099     ! loop over all polygons in file
00100     do  i=1, number_of_polygons
00101       call skip_header(polygon%unit)
00102       read (polygon%unit, * ) nvertex
00103       allocate (polygon%polygon(i)%coords(nvertex, 2 ))
00104       call skip_header(polygon%unit)
00105       read (polygon%unit, * ) pm
00106       if (pm.eq."+") polygon%polygon(i)%use=.true.
00107       if (pm.eq."-") polygon%polygon(i)%use=.false.
00108       ! override file +|- with global given with command line
00109       if (polygon%pm.eq."+") polygon%polygon(i)%use=.true.
00110       if (polygon%pm.eq."-") polygon%polygon(i)%use=.false.
00111       do j = 1 , nvertex
00112         call skip_header(polygon%unit)
00113         ! lon lat , checks while reading
00114         read (polygon%unit, * ) polygon%polygon(i)%coords(j,1:2)
00115         if ( polygon%polygon(i)%coords(j,1).lt.-180. &
```

```
00116           .or.polygon%polygon(i)%coords(j,1).gt.360.  &
00117           .or.polygon%polygon(i)%coords(j,2).lt.-90.  &
00118           .or.polygon%polygon(i)%coords(j,2).gt. 90. ) then
00119           write (error_unit , form_63) "Somethings wrong with coords in polygon
     file"
00120           polygon%if=.false.
00121           return
00122           elseif( polygon%polygon(i)%coords(j,1).lt.0. ) then
00123           polygon%polygon(i)%coords(j,1) = polygon%polygon(i)%coords(j,1) + 360
    .
00124         endif
00125       enddo
00126     enddo
00127     close (polygon%unit)
00128     ! print summary to log file
00129     write (log%unit, form_63) "name:", trim(polygon%name)
00130     write (log%unit, form_63) "number of polygons:" , size (polygon%polygon)
00131     do i = 1 , size (polygon%polygon)
00132       if (polygon%pm.eq."+".or.polygon%pm.eq."-") write (log%unit, form_63) &
00133         "Usage overwritten with command line option", polygon%pm
00134       write (log%unit, form_63) "use [true/false]:" , &
00135         polygon%polygon(i)%use
00136       write (log%unit, form_63) "number of coords:" , &
00137         size (polygon%polygon(i)%coords(:,1))
00138     enddo
00139   endif
00140
00141 end subroutine
00142
00143 !
     ============================================================================
00144 !> Check if point is in closed polygon
00145 !!
00146 !! From spotl \cite Agnew97
00147 !! adopted to \c grat and Fortran90 syntax
00148 !! From original description
00149 !!  returns iok=0 if
00150 !!     1. there is any polygon (of all those read in) in which the
00151 !!        coordinate should not fall, and it does
00152 !!             or
00153 !!     2. the coordinate should fall in at least one polygon
00154 !!        (of those read in) and it does not
00155 !!    otherwise returns iok=1
00156 !! \author D.C. Agnew \cite Agnew96
00157 !! \author adopted by Marcin Rajner
00158 !! \date 2013-03-04
00159 !!
00160 !! The ilustration explain exclusion idea\n
00161 !! \image latex /home/mrajner/src/grat/doc/figures/polygon_ilustration.pdf
     "capt" width=\textwidth
00162 !! \image html /home/mrajner/src/grat/doc/figures/polygon_ilustration.png
00163 !
     ============================================================================
00164 subroutine chkgon (rlong , rlat , polygon , iok)
00165   real(dp),intent (in) :: rlong, rlat
00166   integer :: i, ianyok
00167   integer(2) , intent (out) :: iok
00168   real(dp) :: rlong2
00169   type(polygon_info) , intent (in) :: polygon
00170
00171   !  ! Check first if we need to use this soubroutine
00172   if (size(polygon%polygon).eq.0) then
00173     iok=0
00174     return
00175   endif
00176
00177   if(rlong.gt.180) rlong2 = rlong - 360.
00178   ! loop over polygons
00179   do i=1,size(polygon%polygon)
00180     ! loop twice for elastic and newtonian
00181     ! polygon is one we should not be in
00182     if(.not.polygon%polygon(i)%use) then
00183       if (  if_inpoly(rlong  ,rlat,polygon%polygon(i)%coords).ne.0 &
00184         .or.if_inpoly(rlong2 ,rlat,polygon%polygon(i)%coords).ne.0 ) then
00185       iok=0
00186       return
00187     endif
00188   endif
00189 enddo
00190 ianyok=0
00191 ! polygon is one we should be in; test to see if we are, and if so set
00192 ! iok to 1 and return
00193 do i=1,size(polygon%polygon)
00194   if(polygon%polygon(i)%use) then
00195     ianyok = ianyok+1
00196     if (  if_inpoly(rlong  ,rlat,polygon%polygon(i)%coords).ne.0 &
00197       .or.if_inpoly(rlong2 ,rlat,polygon%polygon(i)%coords).ne.0 ) then
```

```
00198     iok=1
00199     return
00200   endif
00201 endif
00202   enddo
00203   ! not inside any polygon%polygons; set iok to 0 if there are any we should
      have
00204   ! been in
00205   iok = 1
00206   if(ianyok.gt.0) iok = 0
00207   return
00208 end subroutine
00209
00210 !
      ================================================================================
00211 !! taken from spotl \cite Agnew97
00212 !! \par oryginal comment:
00213 !!  Rewritten by D. Agnew from the version by Godkin and Pulli,
00214 !!  in BSSA, Vol 74, pp 1847-1848 (1984)
00215 !! adopted and slightly modified M. Rajner
00216 !! cords is x, y (lon, lat) 2 dimensional array
00217 !
      ================================================================================
00218 integer function if_inpoly(x,y,coords)
00219   use mod_constants, only: dp, dp
00220   real(dp) ,allocatable , dimension (:,:) , intent (in) :: coords
00221   real(dp) , intent (in) :: x , y
00222   integer :: i , isc
00223   ! Returns 1 if point at (x,y) is inside polygon whose nv vertices
00224   ! Returns 0 if point is outside
00225   ! Returns 2 if point is on edge or vertex
00226
00227   if_inpoly = 0
00228   do  i=1, size(coords(:,1))-1
00229     isc = ncross( &
00230       coords(i,1)   - x,  &
00231       coords(i,2)   - y,  &
00232       coords(i+1,1) - x,  &
00233       coords(i+1,2) - y )
00234     !  on edge - know the answer
00235     if(isc.eq.4) then
00236       if_inpoly = 2
00237       return
00238     endif
00239     if_inpoly = if_inpoly + isc
00240   enddo
00241   ! check final segment
00242   isc = ncross( &
00243     coords(size(coords(:,1)) , 1 ) - x , &
00244     coords(size(coords(:,2)) , 2 ) - y , &
00245     coords(1 , 1 ) - x , &
00246     coords(1 , 2 ) - y )
00247   if(isc.eq.4) then
00248     if_inpoly = 2
00249     return
00250   endif
00251   if_inpoly = if_inpoly + isc
00252   if_inpoly = if_inpoly/2
00253   ! convert to all positive (a departure from the original)
00254   if_inpoly = iabs(if_inpoly)
00255   return
00256 end function
00257
00258 !
      ================================================================================
00259 !> \brief finds whether the segment from point 1 to point 2 crosses
00260 !!   the negative x-axis or goes through the origin (this is
00261 !!   the signed crossing number)
00262 !!
00263 !!     return value       nature of crossing
00264 !!         4               segment goes through the origin
00265 !!         2               segment crosses from below
00266 !!         1               segment ends on -x axis from below
00267 !!                          or starts on it and goes up
00268 !!         0               no crossing
00269 !!        -1               segment ends on -x axis from above
00270 !!                          or starts on it and goes down
00271 !!        -2               segment crosses from above
00272 !!
00273 !! taken from spotl \cite Agnew97
00274 !! slightly modified
00275 !
      ================================================================================
00276 integer function ncross(x1,y1,x2,y2)
00277   real(dp) , intent(in) :: x1 , y1, x2 , y2
00278   real(dp) :: c12 , c21
00279
```

```
00280   ! all above (or below) axis
00281   if(y1*y2.gt.0) then
00282     ncross = 0
00283     return
00284   endif
00285
00286   c12 = x1*y2
00287   c21 = x2*y1
00288
00289   ! through origin
00290   if(c12.eq.c21.and.x1*x2.le.0.) then
00291     ncross = 4
00292     return
00293   endif
00294
00295   ! touches +x axis; crosses +x axis; lies entirely on -x axis
00296   if(   (y1.eq.0.and.x1.gt.0)    &
00297     .or.(y2.eq.0.and.x2.gt.0) &
00298     .or.((y1.lt.0).and.(c12.gt.c21)) &
00299     .or.((y1.gt.0).and.(c12.lt.c21)) &
00300     .or.(y1.eq.0.and.y2.eq.0.and.x1.lt.0.and.x2.lt.0)) &
00301     then
00302     ncross = 0
00303     return
00304   endif
00305
00306   ! cross axis
00307   if(y1.ne.0.and.y2.ne.0) then
00308     if(y1.lt.0) ncross = 2
00309     if(y1.gt.0) ncross = -2
00310     return
00311   endif
00312   ! one end touches -x axis - goes which way?
00313   if(y1.eq.0) then
00314     if(y2.lt.0) ncross = -1
00315     if(y2.gt.0) ncross = 1
00316   else
00317     ! y2=0 - ends on x-axis
00318     if(y1.lt.0) ncross = 1
00319     if(y1.gt.0) ncross = -1
00320   endif
00321   return
00322 end function
00323
00324 end module
00325
00326 !\appendix
00327 ! \chapter{Polygon}
00328 !  This examples show how the exclusion of~selected polygons works
00329 !  \begin{figure}[htb]
00330 !    \includegraphics[width=0.5\textwidth]{../mapa1}
00331 !    \caption{If only excluded polygons (red area) are given
00332 !    all points falling in~it will be excluded (red points) all other
00333 !    will be included}
00334 !  \end{figure}
00335 !  \begin{figure}
00336 !    \includegraphics[width=0.5\textwidth]{../mapa2}
00337 !    \caption{If at least one included are are given
00338 !    (green area) than all points which not fall into included area will
00339 !    be excluded}
00340 !  \end{figure}
00341 !  \begin{figure}
00342 !    \includegraphics[width=0.5\textwidth]{../mapa3}
00343 !    \caption{If there is overlap of~polygons the exclusion has higher
00344 !    priority}
00345 !  \end{figure}
00346 !  \chapter{Interpolation}
00347 !  \begin{figure}
00348 !  \input{/home/mrajner/src/grat/doc/interpolation_ilustration.tex}
00349 !  \caption{Interpoloation}
00350 !  \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

```
00001 ! ============================================================================
00002 !> \file
00003 !! \date 2013-01-09
00004 !! \author M. Rajner
00005 ! ============================================================================
00006 program value_check
00007   use mod_cmdline
00008   use mod_parser
00009   use mod_data
00010   use mod_date
00011   use mod_site
00012   use mod_constants,  only: dp, R_air, earth
00013   use mod_polygon,    only: read_polygon, chkgon, polygon
00014   use mod_atmosphere, only: standard_pressure,
      standard_temperature, geop2geom
00015   use mod_utilities,  only: d2r
00016
00017   implicit none
00018   real (dp) , allocatable , dimension(:) :: val
00019   real (dp)  :: cpu(2), sh
00020   integer    :: i, ii, j ,start, imodel, iprogress = 0
00021   integer(2) :: iok
00022   integer(2) :: ilevel, start_level
00023
00024
00025   call cpu_time(cpu(1))
00026
00027   call intro(                              &
00028     program_calling   = "value_check",      &
00029     accepted_switches = "VFoShvIDLPRqwHMJ&!", &
00030     version           = "beta",             &
00031     cmdlineargs       = .true.              &
00032     )
00033
00034   ! for progress bar
00035   if (output%unit.ne.output_unit.and..not.quiet) open (unit=output_unit,
      carriagecontrol='fortran')
00036
00037   allocate (val(size(model)))
00038
00039   start=0
00040   if (size(date).gt.0) then
00041     start=1
00042     ! print header
00043     if (output%header) then
00044       if (.not.output%prune) then
00045         write (output%unit , '(a10,1x,a14,1x)' , advance = "no" ) "#mjd",
      "date"
00046       endif
00047     endif
00048   endif
00049
00050   ! print header
00051   if (output%header.and.size(site).gt.0) then
00052     if (.not.output%prune) then
00053       write (output%unit, '(a8,2a10$)') "name", "lat", "lon"
00054       if (output%height) then
00055         write (output%unit, '(a10$)') "height"
00056       endif
00057     endif
00058     if (output%level) then
00059       write (output%unit, '(a6$)') "level"
00060     endif
00061   endif
00062
```

```
00063   do i = 1, size(model)
00064     if (output%header) then
00065       if (model(i)%dataname.eq."custom") then
00066         write (output%unit,'(a6,"@custom")', advance='no') trim(model(i)%name)
00067       else
00068         write (output%unit,'(a13)', advance='no') trim(model(i)%dataname)
00069       endif
00070     endif
00071   enddo
00072   if(output%header) write(output%unit, *)
00073
00074   do j = start, size(date)
00075     do i = 1 , size(model)
00076       if (model(i)%if) then
00077         if (model(i)%autoload  &
00078           .and. &
00079           .not.( &
00080           model(i)%autoloadname.eq."ERA" &
00081           .and.(any(model(i)%dataname.eq.["GP","VT","VSH"])) &
00082           )) then
00083
00084           if ( &
00085             (j.eq.1 &
00086             .or. .not. date(j)%date(1).eq.date(j-1)%date(1) &
00087             ) &
00088             ) then
00089             call model_aliases(model(i), year=date(j)%date(1))
00090           endif
00091
00092         else if (model(i)%autoload) then
00093
00094           if ( &
00095             (j.eq.1 &
00096             .or. .not.( &
00097             date(j)%date(1).eq.date(j-1)%date(1) &
00098             .and.date(j)%date(2).eq.date(j-1)%date(2)) &
00099             ) &
00100             ) then
00101
00102             call model_aliases( &
00103               model(i), year=date(j)%date(1), month=date(j)%date(2))
00104           endif
00105         endif
00106
00107         if (allocated(date).and.model(i)%exist) then
00108           call get_variable(model(i), date = date(j)%date)
00109         elseif(model(i)%exist) then
00110           call get_variable(model(i))
00111         endif
00112
00113       endif
00114     enddo
00115
00116     ! print only dates if no site given
00117     if (j.gt.0 .and. size(site).lt.1) then
00118       if (dryrun) then
00119         write (output%unit , '(i4.4,5(i2.2),$)') date(j)%date
00120         if (j.lt.size(date)) write (output%unit , '(", ",$)')
00121       else
00122         write (output%unit , '(f10.3,1x,i4.4,5(i2.2))'  ) date(j)%mjd , date(j)
    %date
00123       endif
00124     endif
00125
00126     if (level%all.and..not.allocated(level%level)) then
00127       allocate(level%level(size(model(1)%level)))
00128       level%level=model(1)%level
00129     endif
00130
00131     if (size(level%level).lt.1) then
00132       start_level=0
00133     else
00134       start_level=1
00135     endif
00136
00137     do ilevel=start_level, size(level%level)
00138       do i = 1 , size(site)
00139         iprogress = iprogress + 1
00140
00141         ! add time stamp if -D option was specified
00142         if (j.gt.0) then
00143           if (.not.output%prune) then
00144             write (output%unit , '(f10.3,1x,i4.4,5(i2.2),1x)' , advance = "no"
    ) date(j)%mjd , date(j)%date
00145           endif
00146         endif
00147
```

```
00148         ! if this point should not be used (polygon) leave as zero
00149         if (allocated(polygon).and.polygon(1)%if) then
00150           call chkgon(site(i)%lon, site(i)%lat, polygon(1), iok)
00151         else
00152           iok=1
00153         endif
00154
00155         imodel = 0
00156         do ii = 1 , size (model)
00157           imodel = imodel + 1
00158           if (model(ii)%if.or.model(ii)%if_constant_value) then
00159             if (iok.eq.1) then
00160               if (j.eq.0) then
00161                 call get_value(model(ii), site(i)%lat, site(i)%lon, val(imodel)
      , &
00162                   method=info(1)%interpolation, level=level%level(ilevel))
00163               else
00164                 call get_value(model(ii), site(i)%lat, site(i)%lon, val(imodel)
      , &
00165                   method=info(1)%interpolation, date=date(j)%date, level=level
      %level(ilevel))
00166               endif
00167             else
00168             endif
00169             if (model(ii)%dataname.eq."LS") val(ii)=int(val(ii))
00170
00171           else if (model(ii)%dataname.eq."custom") then
00172             if(ilevel.eq.1) sh=val(ind%model%vsh)
00173             call customfile_value( &
00174               what  = model(imodel)%name, &
00175               sp    = val(ind%model%sp), &
00176               t     = val(ind%model%t), &
00177               hp    = val(ind%model%hp), &
00178               sh    = sh, &
00179               gp    = val(ind%model%gp), &
00180               vsh   = val(ind%model%vsh), &
00181               vt    = val(ind%model%vt), &
00182               level = level%level(ilevel), &
00183               val   = val(imodel), &
00184               rho   = any(model%name.eq."RHO") &
00185               )
00186           else
00187             val(imodel) = sqrt(-1.)
00188           endif
00189         enddo
00190
00191         if (.not.output%prune) then
00192           write (output%unit , '(a8,2f10.4$)') site(i)%name, site(i)%lat, site(
      i)%lon
00193           if (output%height) then
00194             write (output%unit, '(f10.3$)') site(i)%height
00195           endif
00196         endif
00197
00198         if (output%level.and. allocated(level%level)) then
00199           write (output%unit, '(i6$)') level%level(ilevel)
00200         elseif(output%level) then
00201           write (output%unit, '(i6$)') ilevel
00202         endif
00203
00204         write (output%unit , "("//output%form//'$)') val
00205
00206         if (output%unit.ne.output_unit.and..not.quiet) then
00207           call cpu_time(cpu(2))
00208
00209           call progress(                                  &
00210             100*iprogress/(max(size(date),1)              &
00211             *max(size(site),1)*max(size(level%level),1)), &
00212             cpu(2)-cpu(1)                                 &
00213             )
00214         endif
00215         if (size(val).gt.0) write (output%unit , *)
00216       enddo
00217     enddo
00218   enddo
00219
00220   if (ind%moreverbose%d.ne.0) then
00221     do i=1, size(model)
00222       do j=1, size(model(i)%time)
00223         write (moreverbose(ind%moreverbose%d)%unit, '(g0,1x,i4,5i2.2)') &
00224           model(i)%time(j), model(i)%date(j,:)
00225       enddo
00226     enddo
00227   endif
00228
00229   if (ind%moreverbose%j.ne.0) then
00230     do i = 1, size(model)
```

```
00231       do j = 1, size(model(i)%level)
00232         write (moreverbose(ind%moreverbose%j)%unit, '(i5)') &
00233           model(i)%level(j)
00234       enddo
00235     enddo
00236   endif
00237
00238   call cpu_time(cpu(2))
00239   if (output%unit.ne.output_unit.and..not.quiet) then
00240     call progress(                                  &
00241       100*iprogress/(max(size(date),1)              &
00242       *max(size(site),1)*max(size(level%level),1)), &
00243       cpu(2)-cpu(1),                                &
00244       every=1                                       &
00245       )
00246     close(output_unit)
00247   endif
00248   write(log%unit, '(/,"Execution time:",1x,f16.9," seconds")') cpu(2)-cpu(1)
00249   write(log%unit, form_separator)
00250 end program
```

Chapter 10

Example Documentation

10.1 example_aggf.f90

```
00001 ! ========================================================================
00002 !! This program shows some example of using AGGF module
00003 !!
00004 !! \author Marcin Rajner
00005 !! \date 20121108
00006 ! ========================================================================
00007 program example_aggf
00008   use mod_atmosphere
00009   use mod_constants, only: dp
00010   use mod_utilities
00011   use mod_printing, only: log
00012   implicit none
00013   real(dp) :: cpu(2)
00014
00015
00016   call cpu_time(cpu(1))
00017   call standard1976('/home/mrajner/src/grat/examples/standard1976.dat')
00018   call compare_fels_profiles(
      '/home/mrajner/src/grat/examples/compare_fels_profiles.dat')
00019   call simple_atmospheric_model("/home/mrajner/dr/rysunki/simple_approach.dat")
00020   call green_newtonian_compute( &
00021     ["green_newtonian_olsson.dat","green_newtonian_spotl.dat",
      "green_newtonian.dat"])
00022   call admit_niebauer("/home/mrajner/src/grat/examples/admit_niebauer.dat")
00023   call aggf_thin_layer("/home/mrajner/src/grat/examples/aggf_thin_layer.dat")
00024   call compute_tabulated_green_functions(
      '/home/mrajner/src/grat/dat/rajner_green_full.dat'  , method="full"       , predefined=.false.)
00025   call compute_tabulated_green_functions(
      '/home/mrajner/src/grat/dat/rajner_green_rough.dat' , predefined=.false., rough=.true.)
00026   call compute_tabulated_green_functions(
      '/home/mrajner/src/grat/dat/rajner_green_simple.dat', method="simple"     , predefined=.false.)
00027   call compute_tabulated_green_functions(
      '/home/mrajner/src/grat/dat/rajner_green.dat'       , predefined=.false. )
00028   call aggf_resp_fels_profiles(
      '/home/mrajner/src/grat/examples/aggf_resp_fels_profiles.dat')
00029   call mass_vs_height('/home/mrajner/src/grat/examples/mass_vs_height.dat')
00030   call aggf_resp_hmax('/home/mrajner/src/grat/examples/aggf_resp_zmax.dat')
00031   call aggf_resp_dz('/home/mrajner/src/grat/examples/aggf_resp_dz.dat')
00032   call aggf_resp_t('/home/mrajner/src/grat/examples/aggf_resp_t.dat')
00033   call aggf_resp_h('/home/mrajner/src/grat/examples/aggf_resp_h.dat')
00034
00035   call cpu_time(cpu(2))
00036   print '("Total time: ",f8.3,x,"[s]")', cpu(2)-cpu(1)
00037
00038 contains
00039 ! ========================================================================
00040 !> Mass of atmosphere respect to height
00041 ! ========================================================================
00042 subroutine mass_vs_height (filename)
00043   use, intrinsic:: iso_fortran_env
00044   use mod_utilities, only: file_exists
00045   use mod_constants, only : dp, pi, earth, R_air
00046   use mod_atmosphere
00047   character(*), intent (in), optional:: filename
00048   real(dp) :: max_height,dh, percent
00049   real(dp), allocatable, dimension(:):: mass, height
00050   integer::i,j,file_unit
00051
00052   if (present(filename)) then
```

```
00053     if (file_exists(filename)) return
00054     open ( &
00055       newunit = file_unit, &
00056       file    = filename,  &
00057       action  = 'write'  &
00058       )
00059   else
00060     file_unit = output_unit
00061   endif
00062   write(*,*), "mass_vs_height ---> ",filename
00063
00064   max_height=50000.
00065   dh=10
00066
00067   allocate(height(int(max_height/dh)+1))
00068   allocate(mass(size(height)))
00069   do i =1,size(height)
00070     height(i) = dh*(i-1)
00071     mass(i) = standard_pressure( &
00072       height(i), &
00073       method="standard", &
00074       use_standard_temperature=.true., &
00075       nan_as_zero=.true.) &
00076       / (r_air * standard_temperature(height(i)))
00077   enddo
00078
00079   do i =0,50000,1000
00080     percent=0
00081     do j = 1, size(height)
00082       if (height(j).le.dble(i)) percent=percent+mass(j)
00083     enddo
00084     percent = percent / sum(mass)  * 100.
00085     write(file_unit, '(i6,2f19.9,es10.3)' ), i, percent, &
00086       100-(earth%radius+dble(1))**2 &
00087       * standard_pressure(dble(i),method="standard", use_standard_temperature=.
      true.) &
00088       / standard_gravity(dble(i))&
00089       /earth%radius**2/standard_pressure(dble(0),method="standard") *
      standard_gravity(dble(0))*100
00090   enddo
00091 end subroutine
00092
00093 ! ========================================================================
00094 !> Reproduces data to Fig.~3 in \cite Warburton77
00095 !!
00096 !! \date 2013-03-18
00097 !! \author M. Rajner
00098 !!
00099 ! ========================================================================
00100 subroutine simple_atmospheric_model (filename)
00101   use, intrinsic:: iso_fortran_env
00102   use mod_utilities, only: file_exists
00103   use mod_constants
00104   use mod_aggf, only:simple_def, bouger
00105
00106   real(dp) :: r ! km
00107   integer :: file_unit
00108   character(*), intent(in), optional:: filename
00109   real(dp) :: h =9.
00110
00111   if (present(filename)) then
00112     if (file_exists(filename)) return
00113     open ( &
00114       newunit = file_unit, &
00115       file    = filename,  &
00116       action  = 'write'    &
00117       )
00118   else
00119     file_unit = output_unit
00120   endif
00121
00122   write(*,*), "simple_atmospheric_model ---> ",filename
00123
00124   do r = 0., 25*8
00125     write (file_unit, *) &
00126       r, &
00127       -100*bouger(h=h,r=r)/(earth%gravity%mean*h) * 1e8, & !conversion to
     microGal
00128       -simple_def(r) * 1e8
00129   enddo
00130 end subroutine
00131
00132 ! ========================================================================
00133 !> Compute AGGF and derivatives
00134 !!
00135 !! \author M. Rajner
00136 !! \date 2013-03-18
```

```
00137 ! ========================================================================
00138 subroutine compute_tabulated_green_functions ( &
00139     filename, method, dz, &
00140     predefined,fels_type, rough)
00141   use mod_constants, only: dp
00142   use mod_aggf,     only: aggf, aggfd
00143   use mod_green,     only: green
00144   use mod_utilities, only: d2r, file_exists
00145   use mod_atmosphere
00146
00147   integer :: i, file_unit
00148   character(*), intent(in) :: filename
00149   real(dp), optional :: dz
00150   character(*), optional :: fels_type
00151   character(*), optional :: method
00152   logical, optional, intent(in) :: predefined, rough
00153
00154   if (file_exists(filename)) then
00155     return
00156   else
00157     print '(a,a)', "compute_tabulated_green_functions --> ", trim(filename)
00158   endif
00159
00160   call get_green_distances
00161
00162   open (                &
00163     newunit = file_unit, &
00164     file    = filename,  &
00165     action  = 'write'    &
00166     )
00167
00168   !print header
00169   write (file_unit,*) '# This is set of AGGF computed using module ', &
00170     'aggf from grat software'
00171   write (file_unit,*) '# Normalization according to Merriam92'
00172   write (file_unit,*) '# Marcin Rajner'
00173   write (file_unit,*) '# For detail see www.geo.republika.pl'
00174   write (file_unit,'(10(a23))')                        &
00175     '#psi[deg]',                                        &
00176     'GN[microGal/hPa]'     , 'GN/dT[microGal/hPa/K]' , &
00177     'GN/dh[microGal/hPa/m]', 'GN/dz[microGal/hPa/m]'
00178
00179   do i= 1, size(green(1)%distance)
00180     write(file_unit, '(13f15.6)'), &
00181       green(1)%distance(i), &
00182       aggf(d2r(green(1)%distance(i)), method=method, dz=dz
          , predefined=predefined, fels_type=fels_type, rough=rough), &
00183       aggfd(d2r(green(1)%distance(i)), method=method, dz=dz, aggfdt=.true.
          , predefined=predefined, fels_type=fels_type, rough=rough), &
00184       aggf(d2r(green(1)%distance(i)), method=method, dz=dz,
    first_derivative_h=.true., predefined=predefined, fels_type=fels_type, rough=rough), &
00185       aggf(d2r(green(1)%distance(i)), method=method, dz=dz,
    first_derivative_z=.true., predefined=predefined, fels_type=fels_type, rough=rough)
00186   enddo
00187   close(file_unit)
00188 end subroutine
00189
00190 ! ========================================================================
00191 !> Compare different vertical temperature profiles impact on AGGF
00192 ! ========================================================================
00193 subroutine aggf_resp_fels_profiles (filename)
00194   use mod_constants, only: dp
00195   use mod_aggf,  only: aggf
00196   use mod_green, only: green
00197   character (len=255), dimension (6) :: fels_types
00198   integer :: i, j, file_unit
00199   character(*), intent(in), optional :: filename
00200
00201   if (present(filename)) then
00202     if (file_exists(filename)) return
00203     open ( newunit = file_unit, &
00204       file =filename, &
00205       action  = 'write' )
00206   else
00207     file_unit = output_unit
00208   endif
00209   print *, "aggf_resp_fels_profiles -->", filename
00210
00211   ! Get the spherical distances from Merriam92
00212   call get_green_distances()
00213
00214   ! ! All possible optional arguments for standard_temperature
00215   fels_types = (/ &
00216     "US1976"             , "tropical",   &
00217     "subtropical_summer" , "subtropical_winter" , &
00218     "subarctic_summer"   , "subarctic_winter"     &
00219     /)
```

```
00220   ! print header
00221   write (file_unit, '(100(a20))') &
00222     'psi', (trim(fels_types(i)), i = 1, size(fels_types))
00223
00224   ! print results
00225   do i = 1, size(green(1)%distance)
00226     write(file_unit, '(<size(fels_types)+1>f20.5)'), &
00227       green(1)%distance(i), &
00228       (aggf( &
00229       d2r(green(1)%distance(i)), &
00230       method="standard", &
00231       fels_type=fels_types(j)), j=1,size(fels_types) &
00232       )
00233   enddo
00234   close(file_unit)
00235 end subroutine
00236
00237
00238 ! ============================================================================
00239 !> Compare different vertical temperature profiles
00240 !!
00241 !! Using tables and formula from \cite Fels86
00242 !! \author M. Rajner
00243 !! \date 2013-03-19
00244 ! ============================================================================
00245 subroutine compare_fels_profiles (filename)
00246   use iso_fortran_env
00247   use mod_utilities, only: file_exists
00248   use mod_constants, only: dp
00249   use mod_atmosphere, only : standard_temperature
00250   character (len=255), dimension (6) :: fels_types
00251   real (dp) :: height
00252   integer :: i, file_unit, i_height
00253   character(*), intent (in),optional:: filename
00254
00255   ! All possible optional arguments for standard_temperature
00256   fels_types = (/ "US1976"             , "tropical",   &
00257     "subtropical_summer" , "subtropical_winter" , &
00258     "subarctic_summer"   , "subarctic_winter"    /)
00259
00260   if (present(filename)) then
00261     if (file_exists(filename)) return
00262     open ( newunit = file_unit, &
00263       file =filename, &
00264       action  = 'write' )
00265   else
00266     file_unit = output_unit
00267   endif
00268
00269   print *, "compare_fels_profiles --->", filename
00270
00271   ! Print header
00272   write (file_unit, '(100(a20))' ) &
00273     'height', ( trim( fels_types(i) ), i = 1, size (fels_types) )
00274
00275   ! Print results
00276   do i_height = 0, 70, 1
00277     height=dble(i_height)
00278     write ( file_unit, '(f20.3$)'), height
00279     do i = 1, size (fels_types)
00280       write (file_unit, '(f20.3$)'),  standard_temperature(height*1000,
      fels_type=fels_types(i))
00281     enddo
00282     write ( file_unit, *)
00283   enddo
00284   close(file_unit)
00285 end subroutine
00286
00287 ! ============================================================================
00288 !> Computes AGGF for different site height (h)
00289 ! ============================================================================
00290 subroutine aggf_resp_h (filename)
00291   use mod_green, only: green
00292   use mod_aggf, only: aggf
00293   real(dp) :: heights(6)
00294   character(*), intent(in), optional :: filename
00295   integer :: file_unit, i, ii, j
00296   real(dp) :: aux
00297
00298   if (present(filename)) then
00299     if (file_exists(filename)) return
00300     open ( newunit = file_unit, &
00301       file =filename, &
00302       action  = 'write' )
00303   else
00304     file_unit = output_unit
00305   endif
```

```
00306   print *, "aggf_resp_h --->", filename
00307
00308   call get_green_distances()
00309
00310   heights=[0.,1.,10.,100.,1000.,10000.]
00311
00312
00313   write (file_unit, "(a12,6(x,'h',f0.0))") "distance", heights(1:6)
00314   do i =1, size (green(1)%distance)
00315     ! denser sampling
00316     do ii = 0,8
00317       aux  = green(1)%distance(i) + ii * (green(1)%distance(i+1) - green(1)
      %distance(i)) / 9.
00318       if (aux.gt.0.2 ) exit
00319       write (file_unit, '(F12.6$)'), aux
00320       do j =  1, size(heights)
00321         write (file_unit,'(f12.4,1x,$)') aggf(d2r(aux), method="standard",
      h=heights(j))
00322       enddo
00323       write (file_unit,*)
00324     enddo
00325   enddo
00326   close (file_unit)
00327 end subroutine
00328
00329 ! ========================================================================
00330 !> This computes AGGF for different surface temperature
00331 !!
00332 !! \author M. Rajner
00333 !! \date 2013-03-18
00334 ! ========================================================================
00335 subroutine aggf_resp_t (filename)
00336   use mod_green, only: green
00337   ! use mod_constants, only : dp, atmosphere
00338   use mod_aggf, only : aggf
00339   real(dp), dimension(:,:), allocatable :: results
00340   integer :: i, j
00341   character(*), intent(in), optional :: filename
00342   integer :: file_unit
00343   real(dp) :: temperatures(3)
00344
00345   if (present(filename)) then
00346     if (file_exists(filename)) return
00347     open ( newunit = file_unit, &
00348       file =filename, &
00349       action  = 'write' )
00350   else
00351     file_unit = output_unit
00352   endif
00353   call get_green_distances()
00354
00355   allocate(results(size(green(1)%distance), 3))
00356
00357   temperatures=[0., 15., -45]
00358
00359   write(file_unit, '(4a12)') "distance","T0+0", "T0+15", "T0-45"
00360   do i = 1, size(green(1)%distance)
00361     write(file_unit, '(f12.5$)') green(1)%distance(i)
00362     do j=1, size(temperatures)
00363       write(file_unit, '(f12.5$)') &
00364         aggf(d2r(green(1)%distance(i)), method="standard", t_zero=
      temperatures(j))
00365     enddo
00366     write(file_unit, *)
00367   enddo
00368   close (file_unit)
00369 end subroutine
00370
00371 ! ========================================================================
00372 !> \brief This computes AGGF for different height integration step
00373 ! ========================================================================
00374 subroutine aggf_resp_dz (filename)
00375   use mod_green
00376   use mod_aggf, only: aggf
00377   real(dp), dimension(:,:), allocatable :: results
00378   real(dp), dimension(:), allocatable :: dzs
00379
00380   integer :: file_unit, i, j
00381   character(*), intent (in), optional:: filename
00382
00383   if (present(filename)) then
00384     if (file_exists(filename)) return
00385     open ( newunit = file_unit, &
00386       file =filename, &
00387       action  = 'write' )
00388   else
00389     file_unit = output_unit
```

```
00390   endif
00391
00392   call get_green_distances()
00393
00394   allocate(dzs(5))
00395   dzs=(/ 0.01, 0.1, 1., 10., 100./)
00396
00397   allocate (results(size(green(1)%distance(1:29)),size(dzs)))
00398   results = 0.
00399
00400   do i = 1, size (results(:,1))
00401     do j=1,size(dzs)
00402       results(i,j)=i+j
00403       results(i,j)=aggf(d2r(green(1)%distance(i)), &
00404         method="standard", &
00405         dz=dzs(j))
00406     enddo
00407     ! compute relative errors from column 2 for all dz with respect to column 1
00408     results(i,2:) = abs((results(i,2:) - results(i,1)) / results(i,1)*100.  )
00409   enddo
00410
00411   write(file_unit, '(a14,<size(dzs)>f14.4)') "psi_dz", dzs
00412   write(file_unit, '(f14.5,<size(dzs)>e14.4)') &
00413     (green(1)%distance(i), results(i,:), i=1,size(results(:,1)))
00414   close(file_unit)
00415 end subroutine
00416
00417 ! ============================================================================
00418 !> \brief This computes standard atmosphere parameters
00419 !!
00420 !! It computes temperature, gravity, pressure, pressure (simplified formula)
00421 !! density for given height
00422 ! ============================================================================
00423 subroutine standard1976(filename)
00424   use, intrinsic :: iso_fortran_env
00425   use mod_utilities, only: file_exists
00426   use mod_constants, only : dp, R_air
00427   use mod_atmosphere, only: &
00428     standard_temperature, standard_pressure, &
00429     standard_gravity,     standard_density
00430   integer :: file_unit
00431   character(*), intent (in), optional:: filename
00432   real(dp) :: height
00433
00434   if (present(filename)) then
00435     if (file_exists(filename)) return
00436     open ( newunit = file_unit, &
00437       file =filename, &
00438       action  = 'write' )
00439   else
00440     file_unit = output_unit
00441   endif
00442
00443   print *, "standard atmosphere --->", filename
00444   ! print header
00445   write ( file_unit, '(6(a15))' ) &
00446     'height', 'T', 'g', 'p', 'rho'
00447   do height=0.,68000., 1000
00448     ! print results to file
00449     write( file_unit,'(5f15.5, e12.3)'), &
00450       height/1000.,                      &
00451       standard_temperature(height),      &
00452       standard_gravity(height),          &
00453       standard_pressure(height, method="standard")/100.,      &  ! --> hPa
00454       standard_pressure(height, method="standard") &
00455       /(r_air*standard_temperature(height))
00456   enddo
00457   close( file_unit )
00458 end subroutine
00459
00460 ! ============================================================================
00461 !> \brief This computes relative values of AGGF for different atmosphere
00462 !! height integration
00463 ! ============================================================================
00464 subroutine aggf_resp_hmax (filename)
00465   use mod_utilities, only: file_exists, logspace, d2r
00466   ! use mod_constants, only : dp
00467   use mod_aggf, only : aggf
00468   real (dp), dimension (2) :: psi
00469   real (dp), dimension (:), allocatable :: heights
00470   real (dp), dimension (:,:), allocatable :: results
00471   integer :: file_unit, n, i, j
00472   character(*), intent (in), optional:: filename
00473
00474   if (present(filename)) then
00475     if (file_exists(filename)) return
00476     open ( newunit = file_unit, &
```

```
00477       file =filename, &
00478       action  = 'write' )
00479   else
00480     file_unit = output_unit
00481   endif
00482
00483   print *, "standard atmosphere ---> ", filename
00484   psi=(/0.0001, 10 /)
00485
00486   n=90
00487   allocate(heights(n))
00488
00489   heights= logspace(real(1e-1,dp), real(60000,dp),n)
00490
00491   allocate (results(size(heights), size(psi)))
00492   results=0
00493
00494   do j=1, size(heights)
00495     do i = 1, size(psi)
00496       results(j,i) =aggf(d2r(psi(i)),method="standard", zmax=heights(j))
00497     enddo
00498   enddo
00499   do i = 1, size(psi)
00500     results(:,i)=results(:,i)/results(size(heights),i) * 100. ! in %
00501   enddo
00502
00503   write(file_unit, '(a14,SP,100f14.5)' ),"#heght\psi", (psi(j), j= 1,size(psi))
00504   do i=1, size (results(:,1))
00505     write(file_unit, '(100f14.4)' ) heights(i)/1000, (results(i,j), j = 1, size
      (psi) )
00506   enddo
00507   close(file_unit)
00508 end subroutine
00509
00510 ! ================================================================================
00511 ! ================================================================================
00512 subroutine aggf_thin_layer (filename)
00513   use, intrinsic:: iso_fortran_env
00514   use mod_constants, only: dp, pi
00515   use mod_aggf, only: GN_thin_layer
00516   use mod_utilities, only: d2r, file_exists
00517   use mod_green
00518
00519   integer :: file_unit, i
00520   character(*), intent (in), optional:: filename
00521
00522   if (file_exists(filename)) return
00523
00524   call get_green_distances()
00525
00526   write(*,*), "aggf_thin_layer ---> ",filename
00527   if (present(filename)) then
00528     open (newunit = file_unit, &
00529       file =filename, &
00530       action  = 'write' )
00531   else
00532     file_unit = output_unit
00533   endif
00534   do i = 1, size (green(1)%distance)
00535     write(file_unit,*) green(1)%distance(i), green(1)%data(i), &
00536       gn_thin_layer(d2r(green(1)%distance(i)))
00537   enddo
00538 end subroutine
00539
00540 ! ================================================================================
00541 ! ================================================================================
00542 subroutine admit_niebauer(filename)
00543   use mod_constants
00544   use mod_utilities
00545   real(dp) :: a
00546   real(dp) :: theta
00547   real(dp) :: b, f
00548   character(*), intent(in) :: filename
00549   integer::iun
00550
00551   if (file_exists(filename)) return
00552   print *, "admit_niebauer ---> ", filename
00553
00554   open (newunit=iun, file=filename, action = 'write')
00555
00556   f=earth%radius/9500
00557   do theta=0.5, 180, 0.01
00558     b= 2*f*sin(d2r(theta/2))
00559     a= 2*pi * gravity%constant / earth%gravity%mean* &
00560       (1 - b/(2*f) -1/b + 2/f)
00561     write(iun, *), theta, a *1e10
00562   enddo
```

```
00563 end subroutine
00564
00565 ! ============================================================================
00566 !> compute green newtonian function
00567 ! ============================================================================
00568 subroutine green_newtonian_compute(filenames)
00569   use mod_utilities, only: file_exists
00570   use mod_green
00571   use mod_utilities, only: logspace, d2r
00572   integer:: iun, n, i, j, k
00573   real (dp), allocatable, dimension(:) :: psi, h
00574   character(12), allocatable, dimension(:) :: column_name
00575   character(*),  optional :: filenames(3)
00576   character(20) :: method
00577   character(40) :: prefix
00578
00579   prefix="/home/mrajner/src/grat/examples/"
00580
00581   iun = 6
00582
00583   n = 9 * 50
00584   allocate(psi(n))
00585   psi = logspace(real(1e-6,dp), real(180,dp),n)
00586
00587   allocate(h(11))
00588   h = [0., 1., 10., 100., 1000., 10000., -1., -10., -100., -1000., -10000.]
00589
00590   allocate(column_name(size(h)))
00591   write(column_name, '(f0.0)' ) (h(i),i=1,11)
00592
00593   do k =1,3
00594     if (file_exists(trim(prefix)//trim(filenames(k)))) cycle
00595     print *, "green_newtonian_compute ---> ", trim(prefix)//trim(filenames(k))
00596     open (newunit=iun, file=trim(prefix)//filenames(k), action = 'write')
00597
00598     method = filenames(k)(17:index(filenames(k),".")-1)
00599     write(iun, '(a12,<size(h)>a12)') "#psi", ( "h"//trim(column_name(i)), i = 1
      , 11)
00600     write(iun, '(<size(h)+1>en12.2)'), (psi(i), &
00601       (green_newtonian(d2r(psi(i)), h= h(j), method = method), j=1,size(h)), &
00602       i=1,size(psi))
00603     close(iun)
00604   enddo
00605 end subroutine
00606
00607 ! ============================================================================
00608 ! ============================================================================
00609 subroutine get_green_distances()
00610   use mod_green
00611   if (allocated(green)) deallocate(green)
00612   allocate (green(1))
00613   green(1)%name="merriam"
00614   green(1)%column=[1, 2]
00615   green(1)%dataname="GN"
00616   call read_green(green(1),print=.false.)
00617 end subroutine
00618 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
```